Sponzors

DIMATIA

Program committee

Nikhil Bansal, IBM Watson Research Center
Sanjoy Baruah, University of North Carolina
Christoph Dürr, Ecole Polytechnique
Leah Epstein, University of Haifa
Monaldo Mastrolilli, IDSIA
Rolf Möhring, Technische Universität Berlin (chair)
Chris Potts, University of Southampton
Andreas Schulz, Massachusetts Institute of Technology
Jiří Sgall, Charles University (cochair)
Frits Spieksma, K. U. Leuven
Maxim Sviridenko, IBM Watson Research Center
Steef van de Velde, Erasmus University Rotterdam
Gerhard Woeginger, Technische Universiteit Eindhoven
Guochuan Zhang, Zhejiang University

Organizing committee

Anna Kotešovcová
Jana Kratochvílová
Dušan Knop
Marek Krčál
Ondřej Pangrác
Tomáš Ebenlendr
Ondřej Zajíček
Tomáš Dzetkulič
Jiří Sgall (chair)

Preface

This volume contains abstracts of talks presented at the 10th Workshop on Models and Algorithms for Planning and Scheduling Problems (MAPSP 2011), held from June 19 to June 24, 2011, in Nymburk, Czech Republic.

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

The abstracts in this volume are: 5 invited talks by Yossi Azar, Bert Zwart, Roman Barták, David Shmoys, and Ralf Borndörfer plus 81 contributed talks chosen out of 88 submissions. Of these 81 contributed talks, 10 were chosen as plenary talks and the remaining 71 were split into three parallel tracks. Each submission was reviewed by at least three program committee members.

We thank to all sponsors of MAPSP 2011. Sponsors include the company Gurobi Optimization (http://www.gurobi.com), the research center DIMATIA Charles University (http://dimatia.mff.cuni.cz), and Czech research grants MSM0021620838, IAA100190902 and ITI-1M0545 (http://iti.mff.cuni.cz).

We are very grateful to the members of the program committee, external referees, and the members of the organizing committee. Special thanks go to Ondřej Pangrác for designing and maintaining the website, to Marek Krčál and Dušan Knop for preparing this booklet, to Jan Kratochvíl Jr. for designing the poster, and to Sebastian Stiller for designing the MAPSP logo. Most of all, we thank Conforg, s.r.o. (http://www.conforg.cz) – Anna Kotěšovcová and Jana Kratochvílová – for running all the local arrangements and registration.

June 2011

Rolf Möhring
Jiří Sgall

Monday Morning

9:10 – 10:10	**Invited speaker:** **Yossi Azar** *Fast approximation algorithms for submodular optimization problems* [p. 1]		
	Track A	Track B	Track C
10:50 – 11:10	Fabian Kuhn and **Monaldo Mastrolilli** *Vertex Cover in Graphs with Locally Few Colors and Precedence Constrained Scheduling with Few Predecessors* [p. 49]	Sheng Yu, **Prudence W.H. Wong** and Yinfeng Xu *Online Scheduling of Linear Deteriorating Jobs on Parallel Machines* [p. 58]	Stefan Heinz and **Jens Schulz** *Explanation Algorithms for Cumulative Scheduling* [p. 67]
11:15 – 11:35	Andreas Schranzhofer, **Jian-Jia Chen** and Lothar Thiele *Timing Predictability for Resource Sharing Multicore Systems - Challenges and Open Problems* [p. 52]	Jason A. D. Atkin, Edmund K. Burke and **Stefan Ravizza** *A Statistical Approach for Taxi Time Estimation at London Heathrow Airport* [p. 61]	**Zdeněk Baumelt**, Přemysl Šůcha and Zdeněk Hanzálek *A Genetic Algorithm for A Nurse Rerostering Problem* [p. 70]
11:40 – 12:00	**Ruslan Sadykov** and François Vanderbeck *Machine scheduling by column-and-row generation on the time-indexed formulation* [p. 55]	**Csanád Imreh** and Tamás Németh *Parameter learning in online scheduling algorithms* [p. 64]	Marjan van den Akker, Roland Geraerts, **Han Hoogeveen** and Corien Prins *Path planning in games* [p. 73]

Monday Afternoon

15:30 – 16:00	Plenary talk: **Vincenzo Bonifaci** and Alberto Marchetti-Spaccamela *Feasibility Analysis of Sporadic Real-Time Multiprocessor Task Systems* [p. 20]		
16:00 – 16:30	Plenary talk: **Elisabeth Günther**, Marco Lübbecke and Rolf Möhring *Challenges in Scheduling when Planning the Ship Traffic on the Kiel Canal* [p. 23]		
	Track A	Track B	Track C
16:45 – 17:05	Christophe Lenté, Mathieu Liedloff, Ameur Soukhal and **Vincent T'Kindt** *Exponential-time algorithms for scheduling problems* [p. 76]	**Florian Dahms**, Katharina Beygang and Sven O. Krumke *Online algorithms and bounds for the Train Marshalling Problem* [p. 88]	**Mohamed Marouf**, Laurent George and Yves Sorel *Schedulability analysis for a combination of preemptive strict periodic tasks and sporadic tasks* [p. 100]
17:10 – 17:30	**Vitaly Strusevich** and Kabir Rustogi *Enchanced models of single machine scheduling with a deterioration effect and maintenance activities* [p. 79]	**Sofie Coene** and Frits Spieksma *The lockmaster's problem* [p. 91]	**Enrico Bini** *Mapping real-time applications over multi-processors* [p. 103]
17:35 – 17:55	**Tomáš Ebenlendr** and Jiří Sgall *A lower bound on deterministic online algorithms for scheduling on related machines without preemption* [p. 82]	Wei Yu and **Guochuan Zhang** *Improved approximation algorithms for routing shop scheduling* [p. 94]	**Philippe Thierry**, Laurent George and Jean-François Hermant *Real-time scheduling analysis for ARINC-based virtualized systems* [p. 106]
18:00 – 18:20	**Łukasz Jeż** *One to Rule Them All: a General Randomized Algorithm for Buffer Management with Bounded Delay* [p. 85]	**Grzegorz Pawlak**, Mateusz Cichenski, Mateusz Jarus and Slawomir Walkowski *The road traffic model for the car factory logistics* [p. 97]	**Liliana Cucu-Grosjean** *Probabilistic analysis of periodic real-time tasks with random execution times on identical processors* [p. 110]

Tuesday Morning

9:00 – 10:00	**Invited speaker:** **Bert Zwart** *Scheduling and queueuing: Optimality under rare events and heavy loads* [p. 4]		
	Track A	Track B	Track C
10:40 – 11:00	**Sebastián Marbán**, Cyriel Rutten and Tjark Vredeveld *Learning in stochastic machine scheduling* [p. 113]	**Marin Bougeret**, Pierre-Francois Dutot and Denis Trystram *Using oracles for the design of efficient approximation algorithms* [p. 125]	**Stéphane Dauzère-Pérès**, Sadia Azem and Riad Aggoune *A Column Generation Approach for the Job-Shop Scheduling Problem with Availability Constraints* [p. 137]
11:05 – 11:25	**Rodrigo Carrasco**, Garud Iyengar and Clifford Stein *Energy aware scheduling: minimizing total energy cost and completion time by α-points and α-speeds* [p. 116]	José Verschae and **Andreas Wiese** *On the Configuration-LP for Scheduling on Unrelated Machines* [p. 128]	**Bastian Bludau and Karl-Heinz Küfer** *A two-machine flow shop problem consisting of a discrete processor and a batch processor under uncertainty* [p. 140]
11:30 – 11:50	S. Anand, Naveen Garg and **Nicole Megow** *Meeting deadlines: How much speed suffices?* [p. 119]	Jessica Chang, Hal Gabow and **Samir Khuller** *Scheduling Jobs in Parallel for Energy Savings* [p. 131]	Tomáš Zajíček and **Přemysl Šůcha** *Accelerating a Flow Shop Scheduling Algorithm on the GPU* [p. 143]
11:55 – 12:15	S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow and **L. Stougie** *Mixed-criticality scheduling* [p. 122]	Murat Fırat, **Cor Hurkens** and Gerhard Woeginger *Vehicle refueling with limited resources* [p. 134]	M.S. Barketau, M.Y. Kovalyov, **Maciej Machowiak** and J. Węglarz *Scheduling malleable tasks with arbitrary processing speed functions* [p. 145]

Tuesday Afternoon

15:30 – 16:00	Plenary talk: Kirk Pruhs and **Clifford Stein** *How To Schedule When You Have To Buy Your Energy* [p. 26]		
16:00 – 16:30	Plenary talk: Richard Cole, **José Correa**, Vasilis Gkatzelis, Vahab Mirrokni and Neil Olver *Inner Product Spaces for MinSum Coordination Mechanisms* [p. 29]		
	Track A	Track B	Track C
16:45 – 17:05	Leah Epstein and **Elena Kleiman** *On the quality and complexity of Pareto equilibria in the Job Scheduling Game* [p. 148]	Tao Li and **Joel Wein** *Allocating Resources in Clouds of Game Servers* [p. 159]	**Karin Thörnblad**, Michael Patriksson, Ann-Brith Strömberg and Torgny Almgren *Mathematical modelling of a real flexible job shop in aero engine component manufacturing* [p. 171]
17:10 – 17:30	**Ruben Hoeksma** and Marc Uetz *The Price of Anarchy for Related Machine Scheduling* [p. 151]	Joanna Berlińska and **Maciej Drozdowski** *Scheduling and performance of divisible MapReduce applications* [p. 162]	Aldar Dugarzhapov and **Alexander Kononov** *An exact polynomial time algorithm for the preemptive mixed shop problem with two unit operations per job* [p. 174]
17:35 – 17:55	**Eyjólfur Ingi Ásgeirsson** and Pradipta Mitra *Performance of Distributed Game Theoretic Algorithms for Single Slot Scheduling in Wireless Networks* [p. 153]	**Zdeněk Hanzálek**, Claire Hanen and Přemysl Šůcha *Cyclic Scheduling - New Application and Concept of Core Precedences* [p. 165]	**Sergey Sevastyanov** and Bertrand M.T. Lin *Efficient enumeration of optimal and approximate solutions of the two-machine flow-shop problem* [p. 177]
18:00 – 18:20	**Thierry Garaix**, Christian Artigues and Cyril Briand *Fast minimum float computation in activity networks under interval uncertainty* [p. 156]	Kan Fang, **Nelson Uhan**, Fu Zhao and John Sutherland *Flow Shop Scheduling for Sustainable Manufacturing* [p. 168]	**Roman Čapek**, Přemysl Šůcha and Zdeněk Hanzálek *Scheduling with Alternative Process Plans and the Total Weighted Tardiness Criterion* [p. 180]

Wednesday

9:00 – 10:00	**Invited speaker:** **Roman Barták** *Modelling and Solving Scheduling Problems using Constraint Programming* [p. 8]
10:40 – 11:10	Plenary talk: **Alexander Souza**, Antonios Antoniadis, Falk Hüffner, Pascal Lenzner and Carsten Moldenhauer *Balanced Interval Coloring* [p. 32]
11:10 – 11:40	Plenary talk: **Tim Nonner** *Clique Clustering yields a PTAS for max-Coloring Interval Graphs* [p. 35]

Thursday Morning

9:00 – 10:00	**Invited speaker:** **David Shmoys** *Strong LP Formulations and Primal-Dual Approximation Algorithms* [p. 15]		
	Track A	Track B	Track C
10:40 – 11:00	Leah Epstein and **Asaf Levin** *An AFPTAS for due date scheduling with related machines of general costs* [p. 183]	**Robert Davis** *Quantifying the Sub-optimality of Uniprocessor Fixed Priority Scheduling* [p. 195]	**Ammar Oulamara** and Ameur Soukhal *Scheduling serial batching machine with two competitive agents* [p. 207]
11:05 – 11:25	**Klaus Jansen**, Lars Prädel, Ulrich M. Schwarz and Ola Svensson *Faster approximation algorithms for scheduling with fixed jobs* [p. 186]	**Gurulingesh Raravi**, Björn Andersson and Konstantinos Bletsas *Intra-Type Migrative Scheduling of Implicit-Deadline Sporadic Tasks on Two-Type Heterogeneous Multiprocessor* [p. 198]	Frédéric Guinand, Amine Mahjoub and **Eric Sanlaville** *M machine scheduling under uncertainties on machine availabilities* [p. 210]
11:30 – 11:50	Alexander Grigoriev, **Alexander Kononov** and Maxim Sviridenko *Logarithmic-approximations for the relocation problem* [p. 189]	**Sleman Saliba**, Iiro Harjunkoski and Matteo Biondi *Production Optimization and Scheduling in a Steel Plant: Hot Rolling Mill* [p. 201]	**Hana Rudová** and Pavel Troubil *Media Streams Planning* [p. 213]
11:55 – 12:15	Neil Dobbs, Tomasz Nowicki, **Maxim Sviridenko** and Grzegorz Swirszcz *Approximating the Joint Replenishment Problem* [p. 192]	**Vincenzo Bonifaci**, Gianlorenzo D'Angelo, Alberto Marchetti-Spaccamela, Suzanne van der Ster and Leen Stougie *Mixed-criticality scheduling of sporadic task systems* [p. 204]	**Trivikram Dokka**, Ioannis Mourtos and Frits Spieksma *Fast separation algorithms for three-index assignment problems* [p. 216]

Thursday Afternoon

15:30 – 16:00	Plenary talk: Tobias Brunsch, Heiko Roeglin, **Cyriel Rutten** and Tjark Vredeveld *Smoothed performance guarantees of local optima for multiprocessor scheduling* [p. 38]		
16:00 – 16:30	Plenary talk: **Natalia Shakhlevich** *Divisible Load Scheduling to Minimize the Computation Time and Cost* [p. 41]		
	Track A	Track B	Track C
16:45 – 17:05	Ralf Borndörfer, Guillaume Sagnol and **Elmar Swarat** *An Integrated Vehicle Routing and Duty Roster Planning of Toll Control Inspectors* [p. 219]	**Dirk Briskorn** and Malte Fliedner *The train positioning problem* [p. 231]	Tanujit Dey, David Phillips and **Patrick Steele** *Minimizing predicted air travel delay* [p. 242]
17:10 – 17:30	**Cor Hurkens**, Murat Fırat and Alexandre Laugier *Stability in multi-skill workforce assignments: complexity analysis and stable assignments polytope* [p. 222]	Stanley P.Y. Fung, **Chung Keung Poon** and Duncan K.W. Yung *Online Scheduling of Unit-Length Intervals on Parallel Machines* [p. 233]	**Yasmina Seddik**, Christophe Gonzales and Safia Kedad-Sidhoum *Solving the one-machine scheduling problem with cumulative payoffs* [p. 245]
17:35 – 17:55	**Marco Schulze** and Jürgen Zimmermann *Scheduling of underground mining processes* [p. 225]	**Clemens Thielen**, Sven Krumke and Stephan Westphal *Interval Scheduling on Related Machines: Complexity and Online Algorithms* [p. 236]	**Alexander Kozlov** *To the conjecture on the minimum number of migrations in the optimal schedule for the Pm | pmtn(delay=d) | Cmax problem* [p. 248]
18:00 – 18:20	Dries Goossens and **Frits Spieksma** *Breaks, cuts, and patterns* [p. 228]	Denis Trystram, **Frédéric Wagner**, Haifeng Xu and Guochuan Zhang *New Lower Bounds for Online Multi-threaded Paging Problem* [p. 239]	**Hesham Alfares** and Ahmad Zaid *Particle swarm optimization algorithm for workforce job rotation scheduling* [p. 250]

Friday

9:00 – 10:00	**Invited speaker:** **Ralf Borndörfer** *Scheduling problems and algorithms in traffic and transport* [p. 17]		
	Track A	Track B	Track C
10:40 – 11:00	**Leah Epstein** and Asaf Levin *Optimal robust algorithms for preemptive scheduling* [p. 253]		
11:05 – 11:25	Attila Benkő, **György Dósa** and Xin Han *Reassignment models in online scheduling on two related machines* [p. 256]		
11:30 – 12:00	Plenary talk: **Rene Sitters** *Efficient algorithms for average completion time scheduling* [p. 44]		
12:00 – 12:30	Plenary talk: Nikhil Bansal and **Kirk Pruhs** *The Geometry of Scheduling* [p. 46]		

Table of contents

Invited Talks

Yossi Azar 1
Fast approximation algorithms for submodular optimization problems

Bert Zwart 4
Scheduling and queueuing: Optimality under rare events and heavy loads

Roman Barták 8
Modelling and Solving Scheduling Problems using Constraint Programming

David B. Shmoys 15
Strong LP Formulations and Primal-Dual Approximation Algorithms

Ralf Borndörfer 17
Scheduling problems and algorithms in traffic and transport

Plenary Talks

Vincenzo Bonifaci, Alberto Marchetti-Spaccamela 20
Feasibility Analysis of Sporadic Real-Time Multiprocessor Task Systems

Elisabeth Günther, Marco E. Lübbecke, Rolf H. Möhring 23
Challenges in Scheduling when Planning the Ship Traffic on the Kiel Canal

Kirk Pruhs, **Cliff Stein** 26
How To Schedule When You Have To Buy Your Energy

Richard Cole, **José R. Correa**, Vasilis Gkatzelis, Vahab Mirrokni, Neil Olver . . 29
Inner Product Spaces for MinSum Coordination Mechanisms

Antonios Antoniadis, Falk Hüffner, Pascal Lenzner, Carsten Moldenhauer,
Alexander Souza 32
Balanced Interval Coloring

Tim Nonner 35
Clique Clustering yields a PTAS for max-Coloring Interval Graphs

Tobias Brunsch, Heiko Röglin, **Cyriel Rutten**, Tjark Vredeveld 38
Smoothed performance guarantees of local optima for multiprocessor scheduling

Natalia V. Shakhlevich 41
Divisible Load Scheduling to Minimize the Computation Time and Cost

René Sitters 44
Efficient algorithms for average completion time scheduling

Nikhil Bansal, **Kirk Pruhs** 46
The Geometry of Scheduling

Regular Talks

Fabian Kuhn, **Monaldo Mastrolilli** 49
Vertex Cover in Graphs with Locally Few Colors and Precedence Constrained Scheduling with Few Predecessors

Andreas Schranzhofer, **Jian-Jia Chen**, Lothar Thiele 52
Timing Predictability for Resource Sharing Multicore Systems - Challenges and Open Problems

Ruslan Sadykov, François Vanderbeck 55
Machine scheduling by column-and-row generation on the time-indexed formulation

Sheng Yu, **Prudence W. H. Wong**, Yinfeng Xu 58
Online Scheduling of Linear Deteriorating Jobs on Parallel Machines

Jason A. D. Atkin, Edmund K. Burke, **Stefan Ravizza** 61
A Statistical Approach for Taxi Time Estimation at London Heathrow Airport

Csanád Imreh, Tamás Németh 64
Parameter learning in online scheduling algorithms

Stefan Heinz, **Jens Schulz** 67
Explanation Algorithms for Cumulative Scheduling

Zdeněk Bäumelt, Přemysl Šůcha, Zdeněk Hanzálek 70
A Genetic Algorithm for A Nurse Rerostering Problem

Marjan van den Akker, Roland Geraerts, **Han Hoogeveen**, Corien Prins 73
Path planning in games

Christophe Lenté, Mathieu Liedloff, Ameur Soukhal, **Vincent T'kindt** 76
Exponential-time algorithms for scheduling problems

Vitaly A. Strusevich, Kabir Rustogi 79
Enchanced models of single machine scheduling with a deterioration effect and maintenance activities

Tomáš Ebenlendr, Jiří Sgall 82
A lower bound on deterministic online algorithms for scheduling on related machines without preemption.

Łukasz Jeż 85
One to Rule Them All: a General Randomized Algorithm for Buffer Management with Bounded Delay

Katharina Beygang, **Florian Dahms**, Sven O. Krumke 88
Online algorithms and bounds for the Train Marshalling Problem

Sofie Coene, Frits C.R. Spieksma 91
The lockmaster's problem

Wei Yu, **Guochuan Zhang** 94
Improved approximation algorithms for routing shop scheduling

Grzegorz Pawlak, Mateusz Cichenski, Mateusz Jarus, Slawomir Walkowski . . . 97
The road traffic model for the car factory logistics

Mohamed Marouf, Laurent George, Yves Sorel 100
Schedulability analysis for a combination of preemptive strict periodic tasks and sporadic tasks

Enrico Bini 103
Mapping real-time applications over multiprocessors

Philippe Thierry, Laurent George, Jean-François Hermant 106
Real-time scheduling analysis for ARINC-based virtualized systems

Liliana Cucu-Grosjean 110
Probabilistic analysis of periodic real-time tasks with random execution times on identical processors

Sebastián Marbán, Cyriel Rutten, Tjark Vredeveld 113
Learning in stochastic machine scheduling

Rodrigo A. Carrasco, Garud Iyengar, Cliff Stein 116
Energy aware scheduling: minimizing total energy cost and completion time by α-points and α-speeds

S. Anand, Naveen Garg, **Nicole Megow** 119
Meeting deadlines: How much speed suffices?

S. K. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow, **L. Stougie** 122
Mixed-Criticality Scheduling

Marin Bougeret, Pierre-Francois Dutot, Denis Trystram 125
Using oracles for the design of efficient approximation algorithms

José Verschae, **Andreas Wiese** 128
On the Configuration-LP for Scheduling on Unrelated Machines

Jessica Chang, Harold Gabow, **Samir Khuller** 131
Scheduling Jobs in Parallel for Energy Savings

Murat Fırat, **Cor Hurkens**, Gerhard J. Woeginger 134
Vehicle refueling with limited resources

Sadia Azem, Riad Aggoune, **Stéphane Dauzère-Pérès** 137
A Column Generation Approach for the Job-Shop Scheduling Problem with Availability Constraints

Bastian Bludau, Karl-Heinz Küfer 140
A two-machine flow shop problem consisting of a discrete processor and a batch processor under uncertainty

Tomáš Zajíček, **Přemysl Šůcha** 143
Accelerating a Flow Shop Scheduling Algorithm on the GPU

M.S. Barketau, M. Y. Kovalyov, **M. Machowiak**, J. Węglarz 145
Scheduling malleable tasks with arbitrary processing speed functions

Leah Epstein, **Elena Kleiman** 148
On the quality and complexity of Pareto equilibria in the Job Scheduling Game

Ruben Hoeksma, Marc Uetz 151
The Price of Anarchy for Related Machine Scheduling

Eyjólfur Ingi Ásgeirsson, Pradipta Mitra 153
Performance of Distributed Game Theoretic Algorithms for Single Slot Scheduling in Wireless Networks

Thierry Garaix, Christian Artigues, Cyril Briand 156
Fast minimum float computation in activity networks under interval uncertainty

Tao Li, **Joel Wein** 159
Allocating Resources in Clouds of Game Servers

Joanna Berlińska, **Maciej Drozdowski** 162
Scheduling and performance of divisible MapReduce applications

Zdeněk Hanzálek, Claire Hanen, Přemysl Šůcha 165
Cyclic Scheduling - New Application and Concept of Core Precedences

Kan Fang, **Nelson A. Uhan**, Fu Zhao, John W. Sutherland 168
Flow Shop Scheduling for Sustainable Manufacturing

Karin Thörnblad, Michael Patriksson, Ann-Brith Strömberg, Torgny Almgren . 171
Mathematical modelling of a real flexible job shop in aero engine component manufacturing

Aldar Dugarzhapov, **Alexander Kononov** 174
An exact polynomial time algorithm for the preemptive mixed shop problem with two unit operations per job

Sergey Sevastyanov, Bertrand M.T. Lin 177
Efficient enumeration of optimal and approximate solutions of the two-machine flow-shop problem

Roman Čapek, Přemysl Šůcha, Zdeněk Hanzálek 180
Efficient enumeration of optimal and approximate solutions of the two-machine flow-shop problem

Leah Epstein, **Asaf Levin** 183
An AFPTAS for due date scheduling with related machines of general costs

Klaus Jansen, Lars Prädel, Ulrich M. Schwarz, Ola Svensson 186
Faster approximation algorithms for scheduling with fixed jobs

Alexander Grigoriev, **Alexander Kononov**, Maxim Sviridenko 189
Logarithmic-approximations for the relocation problem

Neil Dobbs, Tomasz Nowicki, **Maxim Sviridenko**, Grzegorz Swirszcz 192
Approximating the Joint Replenishment Problem

Robert I. Davis 195
Quantifying the Sub-optimality of Uniprocessor Fixed Priority Scheduling

Gurulingesh Raravi, Björn Andersson, Konstantinos Bletsas 198
Intra-Type Migrative Scheduling of Implicit-Deadline Sporadic Tasks on Two-Type Heterogeneous Multiprocessor

Matteo Biondi, Iiro Harjunkoski, **Sleman Saliba** 201
Production Optimization and Scheduling in a Steel Plant: Hot Rolling Mill

Vincenzo Bonifaci, Gianlorenzo D'Angelo, Alberto Marchetti-Spaccamela, Suzanne van der Ster, Leen Stougie 204
Mixed-Criticality Scheduling of Sporadic Task Systems

Ammar Oulamara, Ameur Soukhal 207
Scheduling serial batching machine with two competitive agents

Frédéric Guinand, Amine Mahjoub, **Eric Sanlaville** 210
M machine scheduling under uncertainties on machine availabilities

Hana Rudová, Pavel Troubil 213
Media Streams Planning

Trivikram Dokka, Ioannis Mourtos, Frits C.R. Spieksma 216
Fast separation algorithms for three-index assignment problems

Ralf Borndörfer, Guillaume Sagnol, **Elmar Swarat** 219
An Integrated Vehicle Routing and Duty Roster Planning of Toll Control Inspectors

Cor Hurkens, Murat Fırat 222
Stability in multi-skill workforce assignments: complexity analysis and stable assignments polytope

Marco Schulze, Jürgen Zimmermann 225
Scheduling of underground mining processes

Dries R. Goossens, **Frits C.R. Spieksma** 228
Breaks, cuts, and patterns

Dirk Briskorn, Malte Fliedner 231
The train positioning problem

Stanley P.Y. Fung, **Chung Keung Poon**, Duncan K.W. Yung 233
Online Scheduling of Unit-Length Intervals on Parallel Machines

Clemens Thielen, Sven O. Krumke, Stephan Westphal 236
Interval Scheduling on Related Machines: Complexity and Online Algorithms

Denis Trystram, **Frédéric Wagner**, Haifeng Xu, Guochuan Zhang 239
New Lower Bounds for Online Multi-threaded Paging Problem

Tanujit Dey, David Phillips, **Patrick Steele** 242
Minimizing predicted air travel delay.

Yasmina Seddik, Christophe Gonzales, Safia Kedad-Sidhoum 245
Solving the one-machine scheduling problem with cumulative payoffs

Alexander Kozlov 248
To the conjecture on the minimum number of migrations in the optimal schedule for the $Pm \,|\, pmtn(delay = d) \,|\, C_{max}$ *problem*

Hesham K. Alfares, Ahmad A. Zaid 250
Particle swarm optimization algorithm for workforce job rotation scheduling

Leah Epstein, Asaf Levin 253
Optimal robust algorithms for preemptive scheduling

György Dósa, Attila Benkő, Xin Han 256
Reassignment models in online scheduling on two related machines

Fast approximation algorithms for submodular optimization problems

Yossi Azar *

1 Submodular functions and optimization problems

We consider three submodular optimization problems. For each of these problems we provide a different fast, combinatorial approximation algorithm. For the first problem, *r*anking with rubmodular valuations, our algorithm provides the best possible approximation. For the second problem, *m*aximization submodular function under linear packing constraints, we match the best approximation provided by non-combinatorial algorithm. For the third problem, *s*ubmodular Max-SAT, we get a non-trivial approximation in linear time. We first define submodular function. Then in the each section we describe one problem.

Let $f : 2^{[m]} \rightarrow R$ be a set function, where $[m] = \{1, 2, \ldots, m\}$. The function f is *submodular* iff $f(S) + f(T) \geq f(S \cup T) + f(S \cap T)$, for all $S, T \subseteq [m]$. An alternative definition of submodularity is through the property of decreasing marginal values. Given a function $f : 2^{[m]} \rightarrow R$ and a set $S \subseteq [m]$, the function f_S is defined by $f_S(j) = f(S \cup \{j\}) - f(S)$. The value $f_S(j)$ is called the incremental marginal value of element j to the set S. The *decreasing marginal values* property requires that $f_S(j)$ is non-increasing function of S for every fixed j. Formally, it requires that $f_S(j) \geq f_T(j)$, for all $S \subseteq T$ and $j \in [m] \setminus T$. Since the amount of information necessary to convey an arbitrary submodular function may be exponential, we assume a value oracle access to the function. A *value oracle* for f allows us to query about the value of $f(S)$ for any set S. Throughout this abstract, whenever we refer to submodular functions, we shall also imply *normalized* and *monotone* functions. Specifically, we assume that a submodular function f also satisfies $f(\emptyset) = 0$ and $f(S) \leq f(T)$ whenever $S \subseteq T$.

2 Ranking with Submodular Valuations

An instance of this problem consists of a ground set $[m]$, and a collection of n monotone submodular set functions $f^1, \ldots, f^n$, where each $f^i : 2^{[m]} \rightarrow R_+$ and a weight vector $w \in R^n_+$. The objective is to find a linear ordering of the ground set elements that minimizes the weighted cover time of the functions. The cover time of a function is the minimal number of elements in the prefix of the linear ordering that form a set whose corresponding function value is greater than some predetermined threshold. More precisely, the objective is to find a linear ordering $\pi : [m] \rightarrow [m]$ that minimizes

*azar@tau.ac.il. Blavatnik School of Computer Science, Tel-Aviv University, Tel-Aviv 69978, Israel.

$\sum_{i=1}^{n} w_i c_i$, where c_i is the cover time of function f^i, defined as the minimal index for which $f^i(\{\pi(1), \ldots, \pi(c_i)\}) \geq 1$. Here, $\pi(t)$ stands for the element scheduled at time t according to the linear ordering π. The unit threshold is chosen without loss of generality. The motivation for the problem comes from web search ranking. Each user type has a Submodular relevance function and the The goal is to order the result items in a way that minimizes the average effort of the user types to see a critical mass.

We design (see [2]) an $O(\ln(1/\epsilon))$-approximation algorithm where $\epsilon = \min\{f_S^i(j) > 0\}$ is the smallest non-zero marginal value that any function may gain from some element. We note that elements can have a marginal value of zero. Our algorithm orders the ground set elements using an *adaptive residual updates scheme*, which iteratively selects an element that has a maximal marginal contribution with respect to an appropriately defined residual cover of the functions.

Theorem 1. *The adaptive residual updates algorithm constructs a linear ordering for the ranking with submodular valuations problem whose induced cost is no more than* $O(\ln(1/\epsilon))$ *times the optimal one.*

Some related work can be found in [4, 5, 8].

3 Maximization Submodular Function under Linear Packing Constraints

The input of this problem consists of a matrix $A \in [0,1]^{m \times n}$, a vector $b \in [1, \infty)^m$, and a monotone submodular set function $f : 2^{[n]} \to R_+$. The objective is to find a set S that maximizes $f(S)$ subject to $Ax_S \leq b$. Here, x_S stands for the characteristic vector of the set S. We note that the restrictions on the entries of A and b are without loss of generality since arbitrary non-negative packing constraints can be reduced to the above form by first eliminating any element j for which there is some constraint i such that $A_{ij} > b_i$, and then scaling the input. A well-studied special setting of our problem is when the objective function f is *linear*, namely, there is a weight vector $c \in R_+^n$ such that $f(S) = \sum_{j \in S} c_j$. This special setting captures the class of packing integer programs, which models several fundamental combinatorial optimization problems, including maximum independent set, hypergraph matching, and disjoint paths.

Our main result (see [1])is a multiplicative updates algorithm for maximizing a monotone submodular function subject to any number of linear packing constraints. The approximation ratio matches the best known performance guarantee even for the special case where the function is linear. More precisely, let $W = \min\{b_i/A_{ij} : A_{ij} > 0\}$ be the *width* of the packing constraints, we attain the following result.

Theorem 2. *There is a deterministic polynomial-time algorithm that attains an approximation ratio of* $\Omega(1/m^{1/W})$ *for maximizing a monotone submodular function under linear packing constraints.*

Some related work can be found in [6, 7, 9–12].

4 Submodular Max-SAT

An input instance of submodular Max-SAT consists of a set $V = \{v_1, \ldots, v_n\}$ of boolean variables, and a collection $C = \{c_1, \ldots, c_m\}$ of clauses, where each clause is a disjunction

of literals over the variables in V. Let $f : C \to R_+$ be a monotone submodular function over the clauses. Given an assignment $\eta : V \to \{\mathsf{True}, \mathsf{False}\}$, we denote by $C(\eta) \subseteq C$ the subset of clauses satisfied by η. The objective is to find an assignment η that maximizes $f(C(\eta))$ over all possible assignments. We note that the classical Max-SAT problem is obtained as a special case when f is an additive function. An *additive* function can be represented as a sum of weights. We design a proportional select algorithm (see [3]), which processes the variables in an arbitrary order (online), and assign them a value randomly proportion to the marginal contribution assignment.

Theorem 3. *Algorithm proportional select achieves an expected competitive ratio of* $2/3$ *for the online submodular Max-SAT problem.*

References

[1] Y. Azar and I. Gamzu. Efficient submodular function maximization under linear packing constraints. 2010. Manuscript available at http://arxiv.org/abs/1007.3604.

[2] Y. Azar and I. Gamzu. Ranking with submodualr intents. In *Proceedings 22nd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1070–1079, 2011.

[3] Y. Azar, I. Gamzu, and R. Roth. Submodular max-sat. 2011. Manuscript.

[4] Y. Azar, I. Gamzu, and X. Yin. Multiple intents re-ranking. In *Proceedings 41st Annual ACM Symposium on Theory of Computing*, pages 669–678, 2009.

[5] N. Bansal, A. Gupta, and R. Krishnaswamy. A constant factor approximation algorithm for generalized min-sum set cover. In *Proceedings 21st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1539–1545, 2010.

[6] P. Briest, P. Krysta, and B. Vöcking. Approximation techniques for utilitarian mechanism design. In *Proceedings 37th ACM Symposium on Theory of Computing*, pages 39–48, 2005.

[7] C. Chekuri, J. Vondrák, and R. Zenklusen. Dependent randomized rounding for matroid polytopes and applications. 2010. Manuscript available at: `http://arxiv.org/abs/0909.4348`.

[8] U. Feige, L. Lovász, and P. Tetali. Approximating min sum set cover. *Algorithmica*, 40(4):219–234, 2004.

[9] N. Garg and J. Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM J. Comput.*, 37(2):630–652, 2007.

[10] A. Kulik, H. Shachnai, and T. Tamir. Maximizing submodular set functions subject to multiple linear constraints. In *Proceedings 20th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 545–554, 2009.

[11] P. Raghavan and C. D. Thompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–374, 1987.

[12] A. Srinivasan. Improved approximation guarantees for packing and covering integer programs. *SIAM J. Comput.*, 29(2):648–670, 1999.

Scheduling and queueuing: Optimality under rare events and heavy loads

Bert Zwart *

1 Queues and scheduling disciplines

We review scheduling policies for the GI/GI/1 queue, i.e., the single server queue with renewal arrivals and i.i.d. service times, and use V_π to denote the stationary sojourn time under policy π. We focus on policies π that satisfy the following three conditions:

1. π is ***work-conserving***: the scheduling policy always has the server working at speed 1 whenever work is present in the system.

2. π is ***non-anticipative***: a scheduling decision at time t does not depend on information about customers that arrive beyond time t. (We do allow the scheduler to use the sizes of jobs on and after arrival.)

3. π is ***non-learning***: the scheduling decisions cannot depend on information about previous busy periods. That is, a scheduling decision on a sample path cannot change when the history before the current busy period is changed.

The first two assumptions are standard and allow a policy to exploit detailed information, such as past and/or remaining service requirements of individual jobs. The third condition is formulated in such a way that a scheduling discipline cannot be driven by data from the (distant) past. It is non-standard, but is satisfied by all common policies. The third condition is important, because it creates a setting in which the scheduler is not aware of the job size distribution.

We additionally introduce some notation: denote a generic job size by B and its mean by β, a generic interarrival time by A, the arrival rate by λ, and the load by $\rho = \lambda\beta < 1$. Importantly, under these conditions, V_π is a.s. finite.

2 Tail optimality

The major focus of the paper is how to choose π such that the sojourn time tail $P(V_\pi > t)$ converges to 0 as fast as possible as $t \to \infty$. That is, we are interested in scheduling disciplines that avoid long sojourn times in an optimal way. Motivated by this, we define a notion of optimality of scheduling policies with respect to the sojourn time tail.

*CWI Amsterdam, bertz@cwi.nl

Definition 1. A scheduling discipline π_0 is **weakly tail-competitive** for a class $\mathcal{P}$ of interarrival time distributions and job size distributions, if

$$\limsup_{t\to\infty} \frac{P(V_{\pi_0} > t)^{1+\tau}}{P(V_\pi > t)} < \infty \tag{1}$$

holds for every $\tau > 0$, every $P \in \mathcal{P}$ and every work-conserving, non-anticipative, non-learning scheduling policy π. π_0 is called **tail-competitive** if the same property holds for $\tau = 0$, and **strongly tail-competitive** if additionally the lim sup is bounded by 1 for $\tau = 0$.

Insight into the optimality of scheduling disciplines can be obtained from the following two simple lower bounds, which are independent of the scheduling discipline:

$$P(V_\pi > t) \geq P(B > t), \tag{2}$$

$$P(V_\pi > t) \geq \frac{1}{E[N]} P(C_{max} > t). \tag{3}$$

Here C_{max} is the maximum amount of work in the system during a busy cycle and N the number of jobs arriving during a busy cycle. An approach for proving optimality of π_0 is to analyze the tail behavior of V_{π_0}, and then to compare with the tail behavior of C_{max} or B. We now review existing results on the tail behavior of $P(V_{\pi_0} > t)$ for several choices of π_0.

3 Review of results for specific scheduling disciplines

We focus on two specific classes of job size distributions: light-tailed and heavy-tailed. We say that a job size B is *light-tailed* if $\Phi(\theta) = E[\exp\{\theta B\}] < \infty$ for some $\theta > 0$. For *heavy-tailed* job sizes, we consider the class of *regularly varying distributions*, which have $P(B > t) = L(t)t^{-\alpha}$ where L is a slowly varying function (i.e., $L(ax)/L(x) \to 1$ as $x \to \infty$ for every $a > 0$) and $\alpha > 1$ is a constant. The Pareto distribution is an important special case.

Light tails

We focus on FCFS and (preemptive) LCFS. For FCFS, we write $V_\pi = V_F$ and for LCFS we set $V_\pi = V_L$. Let Φ_A be the MGF of A and set $\Psi(\theta) = -\Phi_A^{-1}(1/\Phi(\theta))$. (Note that $\Psi(\theta) = \lambda(\Phi(\theta) - 1)$ if the interarrival time distribution is exponential with rate λ). $\Psi(\theta)$ is strictly convex if either A or B is non-deterministic. Now, we can state the large deviations results for FCFS and LCFS:

$$\lim_{t\to\infty} \frac{-\log P(V_F > t)}{t} = \gamma_F := \sup\{\theta : \Psi(\theta) - \theta \leq 0\}, \tag{4}$$

$$\lim_{t\to\infty} \frac{-\log P(V_L > t)}{t} = \gamma_L := \sup_{\theta\geq 0}\{\theta - \Psi(\theta)\}. \tag{5}$$

From the strict convexity of $\Psi(\theta) - \theta$, and the fact that $\Psi'(0) = \rho$, it follows that

$$\gamma_L < (1-\rho)\gamma_F. \tag{6}$$

This inequality shows that, for light tails, FCFS is better at preventing large sojourn times than LCFS. Indeed, FCFS is known to maximize the decay rate. In our setting,

this implies weak optimality. Optimality of FCFS can be guaranteed under the condition that Cramérs condition holds, i.e., if $\Phi_A(-\gamma_F)\Phi(\gamma_F) = 1$ and $\Phi'(\gamma_F) < \infty$. In this case, it is known that $P(C_{max} > t) \sim KP(V_F > t)$ for a constant K. Combining this with (3) it follows that, if Cramérs condition is satisfied, then

$$\limsup_{t\to\infty} \frac{P(V_F > t)}{P(V_\pi > t)} < \infty \tag{7}$$

for any scheduling discipline π.

In contrast to the optimality of γ_F, the decay rate γ_L is the smallest possible decay rate. To see this, note that V_π is by definition stochastically smaller than the total time to emptiness when starting from steady state, just after an arrival (i.e., a residual busy period). The decay rate of this random variable equals γ_L.

Interestingly, many other common policies (including PS and SRPT under some additional assumptions) have been shown to have decay rate equal to γ_L. The intuition behind all these policies is that a large sojourn time is caused by a large service requirement. In addition, the corresponding customer will leave the system after a long busy period of small customers.

Heavy tails

Under regularly varying job sizes and general interarrival times, the following results hold:

$$P(V_F > x) \sim \frac{\rho}{1-\rho}\frac{1}{\alpha-1}tP(B > t), \tag{8}$$

$$P(V_L > t) \sim E[N]P(B > t(1-\rho)), \tag{9}$$

$$P(V_{PS} > t) \sim P(V_{SRPT} > t) \sim P(B > t(1-\rho)), \tag{10}$$

where $f(x) \sim g(x)$ denotes $\lim_{x\to\infty} f(x)/g(x) = 1$.

There are two important observations about these results that we would like to highlight. First, since $P(B > t(1-\rho)) \sim (1-\rho)^{-\alpha}P(B > t)$, PS, SRPT and PLCFS are within a constant of optimal. Second, notice that FCFS has a sojourn time tail that is one degree heavier than optimal. In fact, the sojourn time tail of FCFS is as heavy as possible, up to a constant factor. The same holds for all other non-preemptive policies. The reason is that, under any non-preemptive policy, a job of size x will cause of the order x other customers to wait for a long time. This leads to a lower bound of the order $xP(B > x)$.

4 Impossibility

The previous section reveals a clear dichotomy between the scheduling policies that perform well under light-tailed and heavy-tailed job size distributions. FCFS is weakly tail-competitive under light-tailed job sizes, but is far from optimal under heavy-tailed job sizes; whereas the opposite is true for LCFS, SRPT and PS. This motivates the question: does there exist a scheduling policy that is weakly tail-competitive across all job size distributions? It turns out that the answer is negative:

Theorem 2. *There does not exist a work conserving, non-anticipative, and non-learning scheduling policy π that is weakly tail-competitive for any $\mathcal{P}$ that contains all P having a job size distribution that is either light-tailed or regularly varying with $\alpha > 2$.*

Intuition behind the theorem will be provided during the talk. We will also show that it is possible to exploit information about the first two moments of the job size distribution to develop tail-robust scheduling disciplines.

5 Queues under heavy load

The single server queue is stable if and only if $\rho < 1$. A key question is on the behavior of V_π for fixed π as $\rho \to 1$. For many service disciplines, in particular, for all non-preemptive disciplines,

$$E[V_\pi] = O(1/(1-\rho)), \qquad \rho \to 1.$$

Time permitting, we will review these results as well, and also give a more detailed account on the behavior of the expected sojourn time under SRPT, where typically

$$E[V_{SRPT}] = o(1/(1-\rho)), \qquad \rho \to 1.$$

It turns out that the exact rate of growth depends in a detailed fashion on the tail of the service time distribution.

References

[1] Boxma, O., B. Zwart. 2007. Tails in scheduling. *Performance Evaluation Review* **34**, 13–20.

[2] A. Wierman, B. Zwart. 2009. Is tail-optimal scheduling possible? Under submission.

[3] M. Lin, A. Wierman, B. Zwart. 2010. Heavy-traffic analysis of mean response time under Shortest Remaining Processing Time. Under submission.

[4] Nair, J., A. Wierman, B. Zwart. 2010. Tail-robust scheduling via Limited Processor Sharing. *Performance Evaluation* **67**, 978–995.

Modelling and Solving Scheduling Problems using Constraint Programming

Roman Barták *

1 Introduction

Constraint programming [11] is a technology for solving combinatorial optimisation problems using solving approach that combines search and inference. The idea is that users formulate the problem to be solved as a so called constraint satisfaction problem and a generic constraint solver produces a solution to this constraint satisfaction problem. Constraint satisfaction problem consists of a finite set of variables, each variable is annotated by a set of possible values called a domain, and finally there is a set of constraints that specify allowed combinations of values to be assigned to the variables. As constraint can be any relation of logical, arithmetical, or combinatorial nature, CP gives a big flexibility in formulating the problems as constraint satisfaction problems. However, as we will explain later, different formulations lead to different levels of inference which may dramatically influence efficiency of problem solving. Hence problem modelling is critical in using constraint satisfaction techniques.

In this paper we focus on constraint models for scheduling problems. Scheduling is a prominent application area for constraint satisfaction thanks to natural formulation of scheduling problems as constraint satisfaction problems [2] including side constraints that frequently appear in real-life problems. Moreover, there exist many inference techniques for various scheduling constraints that can be naturally exploited by constraint solvers. Last but not least, the search strategy of constraint solvers can be influenced by users so it is easy to include existing scheduling heuristics. Probably the main advantage of *constraint-based scheduling* is its modularity and flexibility. Rather than focusing on a single algorithm solving efficiently a particular class of problems, CP provides a universal solving approach where many existing solving techniques can naturally be combined.

To understand the differences between models one requires knowledge of how constraint satisfaction works. Hence, we will start with some background on constraint satisfaction in general. Then we will look at some specific constraints for modelling scheduling problems, such as resource constraints. We will conclude with practical notes on using constraint satisfaction techniques in scheduling based on experience from real-life projects realised for Visopt and ManOPT companies.

*bartak@ktiml.mff.cuni.cz. Charles University in Prague, Faculty of Mathematics and Physics, Malostranské náměstí 2/25, 118 00 Praha 1, Czech Republic.

2 Constraint satisfaction at glance

Constraint satisfaction problem (CSP) is a triple (X, D, S), where X is a finite set of variables x_i, D is a set of domains for the variables, for each x_i we have a finite set of possible values $D_i \in D$, and C is a set of constraints. Constraint c_j is a relation over a subset of variables $Y_j \subseteq X$ (Y_j is called a *scope* of the constraint) which defines allowed tuples of values for variables in Y_j. We say that an instantiation σ of variables satisfies the constraint c if restriction of σ to the variables in the scope of c gives an allowed tuple. A feasible solution to a CSP is an instantiation of all variables satisfying all the constraints. Sometimes, a CSP is accompanied by an objective function that maps instantiations of variables to numbers. Then we may look for a feasible solution to a CSP that minimizes or maximizes the value of the objective function. Such a problem is called a *Constraint Optimisation Problem* (COP).

Constraint satisfaction problems can be solved by search, namely by *backtracking* algorithm that takes a not-yet-instantiated variable and tries to assign a value from variable's domain. If the obtained partial instantiation of variables does not violate any constraint, the algorithm goes to a next variable, otherwise it backtracks (tries a different value and if no value remains in the domain then returns to the previously instantiated variable). The algorithm stops when all variables are instantiated, then we obtained a feasible solution, or when no other option remains, then the problem has no solution. The search algorithm can be guided by variable and value ordering heuristics recommending which variable should be instantiated first and in which order the values should be tried. Obviously in the worst case this algorithm has exponential time complexity which is not surprising as CSPs belong among NP-hard problems.

Though search is a general mechanism to solve any CSP, the real power horse behind the efficiency of constraint solvers is using inference that prunes the search tree. Sometimes pruning is so strong that search is not necessary or it is very "shallow" and the problem can be solved merely by inference. To demonstrate the type of inference used in constraint solvers let us assume constraint $A < B$ and $D_A = \{2, 3, 4\}$, $D_B = \{1, 2, 3\}$. Obviously, values 3 and 4 can never be assigned to A to satisfy the constraint as there is no compatible value in D_B. Similarly values 1 and 2 cannot be used for B as they have no support - a compatible value - in D_A. These values can be filtered out from domains of variables A and B without loss of any solution. In general, inference in most constraint solvers is realised using a technique known as *arc consistency* that works as follows. Each constraint has a filtering procedure that removes values from the domains of variables in the constraint's scope that violate the constraint. This can be realised for any constraint c and current domains of variables D_i using the following rule:

$$\forall j \in scope(c) : D'_j \leftarrow ((\times_{i \in scope(c)} D_i) \cap c) \downarrow D_j.$$

Briefly speaking we select the tuples from c that contain only values from the current domains of variables and we project these tuples back to the domains to obtain the values supported by the constraint. If we know the semantics of the constraint, it is possible to implement the filtering procedure in a more efficient way. For example, for constraint $A < B$ we can filter out from D_A all values x such that $x \geq \max(D_B)$ rather than blindly checking pairs (x, y) for compatibility with the constraint. This is the way how a special

solving algorithm can be integrated into a constraint solver - it is "enough" to convert the algorithm into a filtering procedure. When the filtering procedures are defined, they are applied to domains of variables repeatedly until a fix point is reached. By the fix point we mean that no procedure deletes a value from any domain so we reached a state where each constraint is locally consistent. For e binary constraints where the size of domains is d the optimal time complexity to make the problem arc consistent is $O(ed^2)$ [14]. Note that the overall consistency algorithm can be implemented in incremental way so when a domain of a variable is changed, for example by the search algorithm, then only the filtering algorithms for constraints affected by this change are waked up. This way, the change is propagated through the network of constraints hence the technique is also called *constraint propagation*.

As we just outlined when solving a CSP search interleaves with constraint propagation by first making the problem arc consistent and then after each search decision propagating the decision to all variables by making the constraints consistent again. This way, it is possible to exploit efficiency of inference and generality of search. Naturally, as inference is realised via local propagation of constraints, the overall efficiency is strongly influenced by the *constraint model* - a formulation of the problem as a constraint satisfaction problem.

3 Constraints in scheduling

Constraint-based scheduling is an approach to solve scheduling problems by converting them to a CSP and then solving that CSP using general constraint satisfaction techniques. Basically it means deciding the variables describing possible options and connecting the variables via constraints. Note that variables and their domains define the search space to be explored by the search algorithm while the constraints define not only the relations between the variables that must hold but also determine the level of inference. Constraints together with the search heuristics are two major assets influencing efficiency of problem solving.

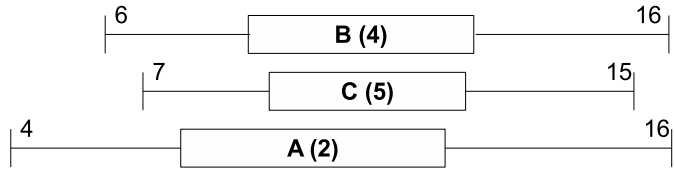
Scheduling deals with the allocation of activities to scarce resources and to limited time. Hence the main decision variable is the start time S_a of each activity a. The domain of this variable consists of time points when the activity can start. Frequently the domain is represented as an interval and filtering algorithms shrink this interval (see below). Sometimes, processing time P_a of activity and its end time E_a may be defined as variables too, for example, if processing time depends on resource to which the activity is allocated. For interruptible or elastic activities an orthogonal approach might be more appropriate - each activity a is represented by a set of Boolean variables $X_{a,t}$ such that variable $X_{a,t}$ describes whether activity a runs at time t or not. If resource allocation is part of the problem, then a resource variable R_a for each activity describes to which resource the activity is allocated. In problems where planning is integrated with scheduling and the system should also decide about which activities will be included in the final schedule [3,4], a Boolean validity variable may be used to describe whether the activity is included or not. Note finally that it is possible to combine several types of variables within a single model to simplify description of constraints. Also, not all the variables must be the decision variables whose values are decided by the search algorithm.

Some auxiliary variables may be used just to simplify specification of constraints and the values of these variables will be set purely be inference. For example the end time of the activity can be inferred from the start time, if the processing time is constant.

Naturally, the choice of variables influences how the constraints can be expressed. There could be simple arithmetic constraints describing the temporal relations. For example $S_a + P_a = E_a$ describes the relation between the start time, the processing time, and the end time of activity a. Temporal inference is usually efficient as simple temporal problems are tractable [10]. More complex inference techniques are behind the resource constraints. We can describe the unary resource simply as a set of disjunctive constraints specifying that no two activities allocated to the same resource overlap in time. These constraints can also be easily extended if resource allocation is part of the problem as the following constraint shows:

$$S_a + P_a \leq S_b \vee S_b + P_b \leq S_a \vee R_a \neq R_b.$$

However, the above disjunctive constraints do not infer a lot and a more global view is necessary. As scheduling is a premium application area of constraint programming, many special constraints (filtering algorithms) were proposed for modeling resources [1,2,9,16]. We will sketch here only the main idea of one of the most frequently used inference techniques for unary resource constraints - edge finding [1]. Assume a situation from the following figure which shows three activities A, B, C with their processing times and time windows:



The disjunctive unary resource constraints as presented above are consistent and do not shrink the time window. However, assume the situation that activity A is not processed first. Then the processing of activities can start earliest at time 6 (the earliest start time of activity B), but as the activities need 11 (2+5+4) time units for processing, obviously they cannot all finish before time 16 which is the latest completion of all three activities. The conclusion is that activity A must be processed first which means that activities B and C must be processed after A. As their latest completion time is 16 and they need 9 time units for processing, we can shrink the time window of A to interval $\langle 4, 7\rangle$. The above deduction can be described by the following inference rule:

$$est(\Omega) + \sum_{X \in \Omega \cup \{A\}} P_X > lct(\Omega \cup \{A\}) \Rightarrow E_A \leq \min\{lct(\Omega') - \sum_{X \in \Omega'} P_X \mid \Omega' \subseteq \Omega\}$$

where Ω is a set of activities that does not contain activity A, $est(\Omega)$ is the earliest start time of activities in set Ω (the minimal time in the domains of start time variables) and similarly $lct(\Omega)$ is the latest completion time. It may seem that to achieve strongest inference one needs to explore all possible sets Ω, but this can be done in time $O(n.log\ n)$, where n is the number of activities [9, 16].

So far we focused on one aspect of constraint-based scheduling, which was the design of constraint models and integration of special inference rules into the model. The second important aspect is flexibility of search procedure that makes it easy to use specific search strategies such as well known EDD rule [12]. Note that during search, it is not necessary to explicitly instantiate the variables but it is possible to do search by splitting the search space via adding new constraints. For example precedence constraint posting [8] is a search method that branches on the ordering constraints between activities that cannot overlap in time such as activities allocated to a unary resource. For example if activities a and b are allocated to the same unary resource and their order is not yet decided then we can split the search space by posting constraint $S_a + P_a \leq S_b$ in one search branch and constraint $S_b + P_b \leq S_a$ in the other search branch. To select which activities should be ordered first and which order should be tried first the heuristics based on slack [15] can be used.

In summary, constraint-based scheduling is a flexible approach for modelling and solving scheduling problems. It naturally treats any type of constraints that appear in real-life problems and it allows integration of special solving algorithms via the inference procedures behind the constraints as well as using scheduling strategies within the search algorithm.

4 Practical experiences

From the previous text, it should be clear that constraint-based scheduling is more about modelling and less about algorithms. In fact, when solving the real-life problems the role of formal scheduling model is even higher than it may seem from the academic perspective. The scheduling concepts such as activities, temporal and resource constraints are not directly present in business environments. For example, enterprise systems operate with concepts such as bill of material, workflow, and demand so before solving the scheduling problem it is necessary to formulate the problem itself. In other words, it is necessary to translate the concepts used in business practice to scheduling concepts that can be used to formulate the scheduling problem [6]. Moreover, the type of underlying scheduling problem [7] is not always obvious from the original concepts, so it is dangerous to rely on a specific scheduling algorithm. The reason is that a change in data may change dramatically the scheduling problem to be solved. This is where constraint-based scheduling can help because it is based on a "universal" model of the scheduling problem.

We have applied constraint-based scheduling techniques in two real-life projects: Visopt ShopFloor system and MAK€ tool. Though both projects shared the same vision - providing an easy-to-use tool where the user models the enterprise and the system automatically optimizes production - they were applied to different production environments and they also differ dramatically in the used scheduling model.

Visopt ShopFloor [3] addressed large complex enterprises such as food and chemical companies. Though applied to process industries the scheduling system used discrete scheduling using batches. The scheduling engine was designed around the concept of a dynamic CSP where variables and constraints were added during the problem solving. The reason was that the system was actually doing integrated planning and scheduling

and based on the description of enterprise, the system generated activities to satisfy the demands and allocated the activities to available resources. Briefly speaking the scheduling model was relatively close to the description of enterprise and it directly included many special constraints appearing in practice. The system used a very detailed universal scheduling model with many features, but such a model is hard to maintain. Moreover, some practical features still needed to be translated to the concepts used in the scheduling model. This experience together with the observation that many concepts implemented in the Visopt ShopFloor scheduling engine were actually never used in practice brought us to the idea of developing a "light" scheduling model covering only the core concepts and "compiling" other concepts to the core concepts. This idea was materialized in the scheduling engine for MAK€ tool.

MAK€ is a performance prediction and optimisation tool for small and medium enterprises marketed by ManOPT Systems. Similarly to Visopt ShopFloor it uses Enterprise Modeller to describe the production environment and from obtained data it automatically generates a scheduling model [6]. As mentioned above, the scheduling engine in MAK€ is based on a completely different concept. Though it still solves integrated planning and scheduling problems where it is necessary to select the activities to satisfy the demands, it uses a classical CSP. All possible activities are modelled in the system and connected using the temporal network with alternatives (TNA) [4] which is the core concept to describe optional process routes. To select the activities for the final schedule we use the validity variables as described in the previous section. We developed special inference techniques to work with optional activities in TNA [5] and actually this project initiated a new area of research on inference techniques for resource constraints with optional activities [16]. These novel techniques are used in recent versions of ILOG CP Optimizer [13].

To summarize our experience, we believe that formal problem modelling and (re-) formulation techniques are critical to bring the scheduling technology to the fingers of regular users. These techniques form a bridge between advanced scheduling techniques on one side and real-life problems on the other side. Constraint-based scheduling seems to be a good integration platform in this endeavour as it is flexible enough to model various constraints and to absorb specific solving algorithms.

5 Acknowledgements

The author is supported by the Czech Science Foundation under the contract P202/10/1188.

References

[1] Baptiste, P. and Le Pape, C (1996). *Edge-finding constraint propagation algorithms for disjunctive and cumulative scheduling.* Proceedings of the Fifteenth Workshop of the U.K. Planning Special Interest Group (PLANSIG).

[2] Baptiste, P.; Le Pape, C.; Nuijten, W. (2001). *Constraint-based Scheduling: Applying Constraints to Scheduling Problems*. Kluwer Academic Publishers, Dordrecht.

[3] Barták, R. (2002). *Visopt ShopFloor: On the edge of planning and scheduling*. Proceedings of CP2002, LNCS 2470, pp. 587-602, Springer-Verlag.

[4] Barták, R. and Čepek, O. (2007). *Temporal Networks with Alternatives: Complexity and Model*. Proceedings of the Twentieth International Florida AI Research Society Conference (FLAIRS), pp. 641-646, AAAI Press.

[5] Barták, R.; Čepek, O.; Hejna, M. (2008). *Temporal Reasoning in Nested Temporal Networks with Alternatives*. Recent Advances in Constraints, LNAI 5129, pp. 17-31, Springer-Verlag.

[6] Barták R.; Little J.; Manzano O.; and Sheahan C. (2010). *From Enterprise Models to Scheduling Models: Bridging the Gap*. Journal of Intelligent Manufacturing, Volume 21, Number 1, pp. 121-132, Springer Verlag.

[7] Brucker, P. (2001). *Scheduling algoritms*. Springer Verlag.

[8] Cesta, A.; Oddi, A.; and Smith S.F. (2000). *Iterative Flattening: A Scalable Method for Solving Multi-Capacity Scheduling Problems*. In Proceedings of the National Conference on Artificial Intelligence (AAAI), pp. 742-747, AAAI Press.

[9] Carlier, J. and Pinson, E. (1994). *Adjustment of heads and tails for the job-shop problem*. European Journal of Operational Research 78(2), pp. 146-161.

[10] Dechter, R.; Meiri, I.; Pearl, J. (1991). *Temporal Constraint Networks*, Artificial Intelligence 49, pp. 61-95.

[11] Dechter, R. (2003). *Constraint Processing*. Morgan Kaufmann.

[12] Jackson, J.R. (1955). *Scheduling a production line to minimize maximum tardiness*. Research report 43. Management Science Research Project, University of California. Los Angeles.

[13] Laborie P. (2009). *IBM ILOG CP Optimizer for Detailed Scheduling Illustrated on Three Problems*. Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. LNCS 5546, pp. 148-162, Springer Verlag.

[14] Mohr, R. and Henderson, T.C. (1986). *Arc and path consistency revised*. Artificial Intelligence 28, pp. 225-233.

[15] Smith, S.F. and Cheng, Ch.-Ch. (1993). *Slack-Based Heuristics For Constraint Satisfaction Scheduling*. Proceedings of the National Conference on Artificial Intelligence (AAAI), pp. 139-144. AAAI Press.

[16] Vilím, P.; Barták, R.; Čepek, O. (2005). *Extension of O(n log n) filtering algorithms for the unary resource constraint to optional activities*. Constraints, Volume 10, Number 4, pp. 403-425. Springer Verlag.

Strong LP Formulations and Primal-Dual Approximation Algorithms

David B. Shmoys *

The state of the art of the design and analysis of approximation algorithms for NP-hard discrete optimization has advanced significantly over the past two decades; furthermore, the most prevalent approach has been to rely on the strength of natural linear programming relaxations. Work in computational integer programming over the same time period has shown the power of adding combinatorially-motivated valid inequalities, but this has not been employed often in the design of approximation algorithms. We will show how this approach can be applied in the design and analysis of primal-dual approximation algorithms. We will present several recent approximation results along these lines.

Carr, Fleischer, Leung, and Phillips [3] initiated a very interesting line of research by introducing a class of *knapsack-cover* inequalties for a simple minimum-cost knapsack problem: in this problem, we are given a set of items, each with a given cost and a given size, along with a demand that needs to be satisfied; the objective is to select a subset of items of total size that is at least the demand, of minimum total cost. The natural 0-1 integer program has an unbounded integrality gap. The knapsack-cover inequalities of Carr et al. strengthen the LP relaxation in the following way: for each subset of items A, one considers the residual demand assuming that all of the items in A have been selected for inclusion in the knapsack; then each other item has an effective size that is the minimum of its true size and the residual demand, and the new constraint states that the total effective size that we select from the remaining items must be sufficient to meet the residual demand. Carr et al. gave a rounding approach that yielded a 2-approximation algorithm (via a clever application of the ellipsoid method). Carnes and Shmoys [2] give a (surprisingly simple) primal-dual analogue of this LP-rounding result, and we shall explain this result in detail.

In fact, the minimum-cost knapsack problem is equivalent to the scheduling problem (in the notation of Graham, Lawler, Lenstra, and Rinnooy Kan [5]) $1|d_j = D|\sum w_j U_j$, that is, the problem of minimizing the total weight of late jobs on a single machine, where all jobs have a common due date. Recently, Bansal and Pruhs [1] gave the first constant approximation algorithm for a vast generalization of this: $1||\sum f_j$, provided the cost functions are nonnegative (in addition to the standard assumption that they be nondecreasing). The algorithm of Bansal and Pruhs finds a feasible solution of cost within a factor of 16 of the optimum. Cheung and Shmoys [4] have recently generalized the construction of Carnes and Shmoys to give a direct primal-dual approximation algorithm for $1||\sum f_j$ that, for any $\epsilon > 0$, can find in polynomial time, a solution of cost

*shmoys@cs.cornell.edu. School of Operations Research & Information Engineering, Department of Computer Science, Cornell University, 231 Rhodes Hall, Ithaca, NY, USA 14853

within a factor of $2 + \epsilon$ of optimal. One surprising aspect of this result is that it is easy to further generalize it to allow a machine that can run at variable speed according to a prespecified speed function $s(t)$. We shall outline the main ideas behind these results.

References

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

[2] T. Carnes and D. B. Shmoys. Primal-dual schema for capacitated covering problems. In *Proceedings of the 13th Conference on Integer Programming and Combinatorial Optimization*, number 5035 in Lecture Notes in Computer Science, pages 288–302, 2008.

[3] R. D. Carr, L. Fleischer, V. J. Leung, and C. A. Phillips. Strengthening integrality gaps for capacitated network design and covering problems. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 106–115, 2000.

[4] M. Cheung and D. B. Shmoys. A primal-dual approximation algorithm for min-sum scheduling problems. Submitted.

[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. *Annals of Discrete Mathematics*, 5:287–326, 1979.

Scheduling problems and algorithms in traffic and transport

Ralf Borndörfer *

1 Introduction

Traffic and transport is one of the classical application areas of scheduling models and algorithms. Important problems, in particular, about vehicle and crew scheduling, are nowadays well understood, and powerful mathematical optimization methods are available, which have already become an industry standard in public and air transport. Such successes motivate research to extend optimization approaches to further applications. I will discuss four areas with potential in this respect, namely,

- railway scheduling,
- robust scheduling,
- service design.

Each of them features challenging, important scheduling problems, which are currently under investigation in the research community. The discussion is in terms of examples; for a general overview and pointers to the literature see the survey article [2].

2 Traffic and transport scheduling

Traffic and transport scheduling is about guiding flows of vehicles, crews, passengers, and goods through a network of some sort. Problems involving only local dependencies between successive trips can be dealt with using network flows, problems involving constraints on individual paths or rotations using set partitioning approaches. Including further requirements calls for an extension of these methods.

2.1 Railway scheduling

Railway systems differ from public and air transport networks in two main points, namely, in train composition, i.e., the combination of different vehicles, and in the capacity restrictions imposed by the movement of the trains on a track system. Because of these two characteristics, railway scheduling problems have a special, "railway combinatorics" flavor. A major problem is that decomposition or simplification approaches often fail, because railways systems lack sufficient degrees of freedom in order to recombine solutions of individual problems. On the other hand, integrated models on a detailed level become quickly intractable for scenarios of practically relevant sizes.

*borndoerfer@zib.de. Zuse-Institute Berlin, Takustr. 7, 14195 Berlin, Germany.

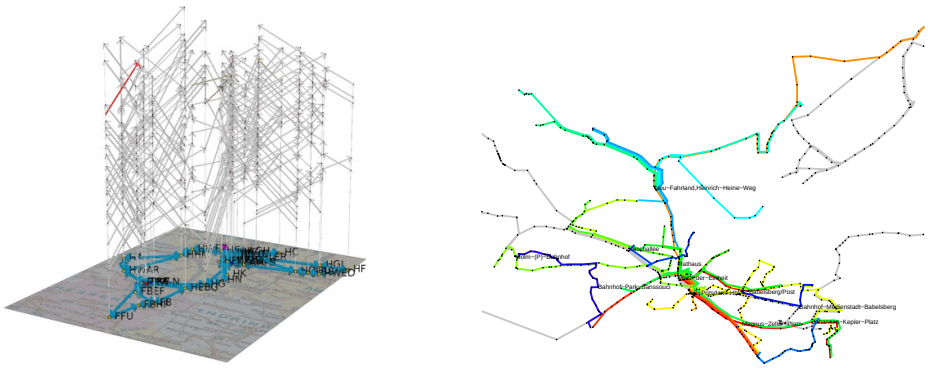


Figure 1: Optimized railway timetable (left) and line plan (right).

One way to approach this problem is to develop micro-macro transformation methods, that dynamically adjust a model's level of detail as needed, namely, to a high accuracy at the bottlenecks, and to a low precision where it can be afforded, similar to adaptive mesh-generation in finite element methods. I will illustrate such an approach at the example of the track allocation or railway timetabling problem, see also [5, 6].

Investigating the combinatorics of railways also leads to new types of models. I will discuss how the coordinated planning of trips that repeat over a week in order to maximize the regularity of the schedule can be expressed in terms of a hypergraph assignment model.

2.2 Robust scheduling

The best plan is worth little if it can't be implemented in practice, and this is even more true for highly optimized plans, which leave little safety margins. Safeguarding against disruptions is therefore major topic in railway and airline scheduling. The main goal is to come up with robust schedules, i.e., schedules that can absorb a certain amount of operational fluctuations. By placing buffers at appropriate places, delays are supposed to diminish, instead of building up.

The two main approaches are to use stochastic optimization in order to minimize expected delays, based on input disruption probabilities derived from historical data, and to control the worst-case over some set of input scenarios in a robust optimization approach. Of course, they can also be combined.

I will discuss an optimization approach to robust tail assignment that minimizes the expected propagated delay along aircraft rotations. The method is based on a numerical approximation of propagated convolution integrals. It is computationally efficient and combines with standard column generation methods, see [1] for more details.

2.3 Service design

Vehicle and crew scheduling have important optimization potentials, but after all, they can only operate within the limits of a given infrastructure. Much larger effects could be expected from optimizing the system itself. With the notable exception of air traffic,

mathematics does not yet play much of a role here. This is remarkable, because understanding and controlling the correlation between supply and demand is the key to increasing a systems attractivity (or minimize its loss, if costs have to be saved). Service design must balance costs and utilities, i.e., it is naturally multi-criterial, and it must match the use of resources with the passenger response, i.e., it is highly integrated, or even behavioral, such that, ultimately, bi-level and game-theoretical methods must be used.

I will discuss an example in line planning in public transport, integrating line and passenger routing, see also [3, 4] and an application dealing with fare evader controls on the German highway system; here, of course, the goal is to minimize the "usage".

References

[1] Ralf Borndörfer, Ivan Dovica, Ivo Nowak, and Thomas Schickinger. Robust tail assignment. ZIB Report 10-08, ZIB, Takustr. 7, 14195 Berlin, 2010. Submitted 31.05.2010, accepted 07.07.2010 as one of two finalist papers for the Anna Valicek Medal of the 50th Annual Conference of the Airline Group of the International Federation of Operational Research Societies (AGIFORS 2010), and awarded 23.09.2010 with the Anna Valicek Medal in silver (over bronze).

[2] Ralf Borndörfer, Martin Grötschel, and Ulrich Jäger. Planning Problems in Public Transit. In Martin Grötschel, Klaus Lucas, and Volker Mehrmann, editors, *Production Factor Mathematics*, pages 95–122. acatech and Springer, Berlin Heidelberg, 2010.

[3] Ralf Borndörfer, Martin Grötschel, and Marc E. Pfetsch. A column-generation approach to line planning in public transport. *Transportation Science*, 41(1):123–132, February 2007. ZIB Report 05-18 available at `http://opus.kobv.de/zib/volltexte/2005/852/`.

[4] Ralf Borndörfer, Marika Neumann, and Marc E. Pfetsch. The line connectivity problem. In Bernhard Fleischmann, Karl Heinz Borgwardt, Robert Klein, and Axel Tuma, editors, *Operations Research Proceedings 2008*, pages 557–562. Springer Verlag, Berlin, 2009. ZIB Report 08-31 available at `http://opus.kobv.de/zib/volltexte/2008/1117/`.

[5] Ralf Borndörfer and Thomas Schlechte. Models for railway track allocation. In Christian Liebchen, Ravindra K. Ahuja, and Juan A. Mesa, editors, *Proceeding of the 7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS 2007)*, Dagstuhl, Germany, 2007. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.

[6] Ralf Borndörfer, Thomas Schlechte, and Steffen Weider. Railway track allocation by rapid branching. In Thomas Erlebach and Marco Lübbecke, editors, *Proceedings of the 10th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*, volume 14 of *OpenAccess Series in Informatics (OASIcs)*, pages 13–23, Dagstuhl, Germany, 2010. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ZIB Report 10-11.

Feasibility Analysis of Sporadic Real-Time Multiprocessor Task Systems*

Vincenzo Bonifaci (Speaker) † Alberto Marchetti-Spaccamela ‡

The *sporadic task model* is a model of recurrent processes in hard real-time systems that has received great attention in the last years (see for example [1, 3] and references therein). A sporadic task $\tau_i = (C_i, D_i, P_i)$ is characterized by a worst-case execution time C_i, a relative deadline D_i, and a minimum interarrival separation P_i. Such a sporadic task generates a potentially infinite sequence of jobs: each job arrives at an unpredictable time, after the minimum separation P_i from the last job of the same task has elapsed; it has an execution requirement less than or equal to C_i and a deadline that occurs D_i time units after its arrival time. A sporadic task system $\mathcal{T}$ is a collection of such sporadic tasks. Since the actual interarrival times can vary, there are infinitely many job sequences that may be generated by $\mathcal{T}$.

We are interested in designing algorithms that tell us when a given sporadic task system can be feasibly scheduled on a set of $m \geq 1$ identical processors, where we allow any job to be interrupted and resumed later on another processor at no penalty. The problem can be formulated in several ways:

- *Feasibility*: is it possible to feasibly schedule on m processors any job sequence that can be generated by $\mathcal{T}$?
- *Online feasibility*: is there an online algorithm that can feasibly schedule on m processors any job sequence that can be generated by $\mathcal{T}$?
- *Schedulability*: does the given online algorithm Alg feasibly schedule on m processors any job sequence that can be generated by $\mathcal{T}$?

It is useful to remark that, despite their apparent similarity, the feasibility problem and the online feasibility problem are in fact distinct, as shown recently by [7].

We study the three above problems in the context of sporadic multiprocessor task systems, and we provide new results for each of them.

For the feasibility problem, we give the first correct test, thus answering [3, Open Problem 3]. The test has high complexity, but it has the interesting consequence that a job sequence that witnesses the infeasibility of a task system $\mathcal{T}$ has without loss of generality length at most doubly exponential in the bitsize of $\mathcal{T}$.

We then give the first correct test for the online feasibility problem. The test has exponential time complexity and is constructive: if a system is deemed online feasible,

*A version of this work appeared in the 18th European Symposium on Algorithms [5].

†bonifaci@mpi-inf.mpg.de. Max-Planck-Institut für Informatik, Campus E1.4, 66123 Saarbrücken, Germany.

‡alberto@dis.uniroma1.it. Department of Computer and Systems Science, Via Ariosto 25, 00185 Rome, Italy.

then an optimal online scheduling algorithm can also be constructed, in the same time bound. Moreover, this optimal algorithm is without loss of generality *memoryless*: its decisions depend only on the current (finite) state and not on the entire history up to the decision point.

For the schedulability problem, we provide a general schedulability test showing that the schedulability of a system by any memoryless algorithm can be tested in polynomial space. This improves a result of Baker and Cirinei [2], that provided an exponential space test for essentially the same class of algorithms.

All the above results, that are derived for constrained-deadline systems where $D_i \leq P_i$ for all i, can be extended to the arbitrary-deadline case in which deadlines may exceed periods, at the expense of increasing some of the complexity bounds.

We finally consider the issue of discrete time schedules versus continuous time schedules. The above results are derived with the assumption that the time line is divided into indivisible time slots and preemptions can occur only at integral points, that is, the schedule has to be discrete. In a continuous schedule, time is not divided into discrete quanta and preemptions may occur at any time instant. We show that in a sporadic task system (with integer input parameters and integer arrival times) a discrete schedule exists whenever a continuous schedule does, thus showing that the discrete time assumption is without loss of generality. Such an equivalence was known for the single processor setting [4]; however, the proof relied on the optimality of the EDF algorithm and thus did not extend to multiprocessor systems. In fact, this equivalence was also listed among the relevant open problems in real-time scheduling [3, Open Problem 5].

Our main conceptual contribution is to show how the feasibility problem, the online feasibility problem and the schedulability problem can be cast as the problem of deciding the winner in certain games of infinite duration played on a finite graph. We then use tools from the theory of games to decide who has a winning strategy. In particular, in the case of the feasibility problem we have a game of imperfect information where one of the players does not see the moves of the opponent, a so-called *blindfold game* [8]. This can be reformulated as a one-player (i.e., solitaire) game on an exponentially larger graph and then solved via a reachability algorithm. However, a technical complication is that in our model a job sequence and a schedule can both have infinite length, which when the system is feasible makes the construction of a feasible schedule challenging. We solve this complication by an application of König's Infinity Lemma from graph theory [6, p. 200]. This is the technical ingredient that, roughly speaking, allows us to reduce the job sequences from infinite length to finite length and ultimately to obtain the equivalence between continuous and discrete schedules.

The power of our new approach is its generality: it can be applied to all three problems, and at the same time it yields proofs that are not technically too complicated. We hope that this approach might be useful to answer similar questions for other real-time scheduling problems.

References

[1] T. P. Baker and S. K. Baruah. Schedulability analysis of multiprocessor sporadic task systems. In S. H. Son, I. Lee, and J. Y.-T. Leung, editors, *Handbook of Real-Time and Embedded Systems*, chapter 3. CRC Press, 2007.

[2] T. P. Baker and M. Cirinei. Brute-force determination of multiprocessor schedulability for sets of sporadic hard-deadline tasks. In E. Tovar, P. Tsigas, and H. Fouchal, editors, *Proc. 11th Conf. on Principles of Distributed Systems*, volume 4878 of *Lecture Notes in Computer Science*, pages 62–75. Springer, 2007.

[3] S. K. Baruah and K. Pruhs. Open problems in real-time scheduling. *Journal of Scheduling*, 13(6):577–582, 2009.

[4] S. K. Baruah, L. E. Rosier, and R. R. Howell. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Systems*, 2(4):301–324, 1990.

[5] V. Bonifaci and A. Marchetti-Spaccamela. Feasibility analysis of sporadic real-time multiprocessor task systems. In M. D. Berg and U. Meyer, editors, *Proc. 18th European Symp. on Algorithms*, volume 6347 of *Lecture Notes in Computer Science*, pages 230–241. Springer, 2010. To appear in *Algorithmica*.

[6] R. Diestel. *Graph theory*. Springer, Heidelberg, 3rd edition, 2005.

[7] N. Fisher, J. Goossens, and S. K. Baruah. Optimal online multiprocessor scheduling of sporadic real-time tasks is impossible. *Real-Time Systems*, 45(1):26–71, 2010.

[8] J. H. Reif. The complexity of two-player games of incomplete information. *Journal of Computer and System Sciences*, 29(2):274–301, 1984.

Challenges in Scheduling when Planning the Ship Traffic on the Kiel Canal

Elisabeth Günther (Speaker) [*] Marco E. Lübbecke [†] Rolf H. Möhring [‡]

1 Introduction

The Kiel Canal connects the North and Baltic seas and is ranked among the world's three major canals. In fact, in terms of traffic, it is the busiest artificial waterway worldwide. In a billion Euro project, the German Federal Waterways and Shipping Administration plans to enlarge the canal during the coming years. This project is about contributing to well-founded advises on how the enlargement can be optimally done. In order to evaluate various construction possibilities and enlargement strategies it is indispensable to first provide an accurate model for the ship traffic and designing an algorithm which (ideally optimally) controls it.

The problem very roughly is as follows. There is bi-directional ship traffic on the canal; there are several locks at both ends. Ships are classified in different size categories. Passing and overtaking is allowed only if the sizes of the two ships do not exceed a given threshold which depends on the meeting point. If otherwise a *conflict* occurs, ships have to wait at designated, capacitated places, the *sidings*. The objective is to minimize the total passage time, including lock and siding waiting times. The overall scheduling is currently done by teams of experienced planners, for the locks and for the sidings. In this abstract we concentrate on the latter problem, but both are treated in an integrated way during the project.

2 Ship Traffic Control

We are given an interval $C \subset \mathbb{R}$ as *canal* partitioned into a set of intervals $\mathcal{E} = (e_i)_{i=1,\dots,m}$ with $\dot{\cup}_{i=1}^{m} e_i = C$ as *segments* where some special segments $\mathcal{T} \subset \mathcal{E}$ are called *sidings*. Furthermore, a set of requests $R = \{(v_i, r_i, s_i, t_i, w_i, \ell_i, b_i, B_i) \mid i \in S\}$ is given that corresponds to *ships* $S = \{1, \dots, n\}$ with maximum *velocity* v_i, *release date* r_i, *start* and *target positions* $s_i \neq t_i \in C$ somewhere in the canal, ship dimensions like *width* w_i and *length* ℓ_i and finally *waiting bounds* per siding b_i and in total B_i. Ship $i \in S$ is called *updirected* when $t_i > s_i$ and *downdirected* otherwise. The specified velocity of a ship $i \in S$ and the given length of a segment $e \in \mathcal{E}$ define the transit time τ_{ie} of ship i along segment e assuming constant full speed along each segment like the manual planners do.

[*] eguenth@math.tu-berlin.de. Technische Universität Berlin, Institut für Mathematik, Germany.

[†] marco.luebbecke@rwth-aachen.de. RWTH Aachen University, Chair of Operations Research, Germany.

[‡] rolf.moehring@tu-berlin.de. Technische Universität Berlin, Institut für Mathematik, Germany.

For each segment $e \in \mathcal{E}$ we are given a set $C_e \subseteq S \times S$ of ship pairs having a *conflict* on segment e. These are at the one hand side those ships with opposite travel directions that are not allowed to pass each other on that segment. Therefore, a decision on who is passing this segment first must be made. But also for all ships traveling in the same directions an order must be decided. Thus, all same directed ships have a conflict.

To define a solution we need to determine feasible departure times d_{ie} specifying when each ship $i \in S$ is leaving a segment $e \in \mathcal{E}$. Due to space limitations feasibility will be defined exactly only for a relaxed problem by the following conditions:

$$d_{i,e_{-_s}} + \tau_{,e} = d_{i,e} \qquad \forall i \in S, e \in \mathcal{E} \setminus \mathcal{T} \tag{1}$$

$$d_{i,t_{-_i}} + \tau_{i,t} + w_{i,t} = d_{i,t} \qquad \forall i \in S, t \in \mathcal{T} \tag{2}$$

$$z_{i,j,e} = 1 \;\Rightarrow\; d_{i,e} + \Delta(i,j,e) \leq d_{j,e} \qquad \forall e \in \mathcal{E} \setminus \mathcal{T}, (i,j) \in C_e \tag{3}$$

$$z_{i,j,e} = 0 \;\Rightarrow\; d_{j,e} + \Delta(j,i,e) \leq d_{i,e} \qquad \forall e \in \mathcal{E} \setminus \mathcal{T}, (i,j) \in C_e \tag{4}$$

$$\underline{d}_{i,e} \leq d_{i,e} \leq \overline{d}_{i,e} \qquad \forall i \in S, e \in \mathcal{E} \tag{5}$$

$$w_{i,t} \geq 0 \qquad \forall i \in S, t \in \mathcal{T} \tag{6}$$

$$z_{i,j,e} \in \{0,1\} \qquad \forall e \in \mathcal{E} \setminus \mathcal{T}, (i,j) \in C_e \tag{7}$$

All conditions concerning passing of and waiting in sidings are missing here. Equations (1) and (2) guarantee that departure times can be traveled with the given velocities and that waiting, represented by non negative variables $w_{i,t}$ in (6), only occurs in sidings. Segment e_{-_i} is the segment that must be passed before e when traveling in direction of ship $i \in S$. Each departure time variable is bounded from below by $\underline{d}_{i,e}$ and from above by $\overline{d}_{i,e}$ that are defined by the release dates, the waiting bounds and the travel times (5). The binary variables $z_{i,j,e}$ of (7) decide how to deal with two conflicting ships on a segment. Depending on this decisions one of the two conditions (3) and (4) must be respected. The used delta $\Delta(i,j,e)$ depends on the ship dimensions, the maximum allowed speed, the travel directions and the start and target positions of the two conflicting ships as well as the special safety distance regulations. It is not necessarily symmetric and can be a negative value.

The goal is to minimize the total waiting time $\sum_{i \in S, t \in \mathcal{T}} w_{i,t}$. This problem is shown to be NP-hard. Solutions are visualized in exactly the same way the planners are used to see them, in a distance-time diagram of the canal, see Figure 1.

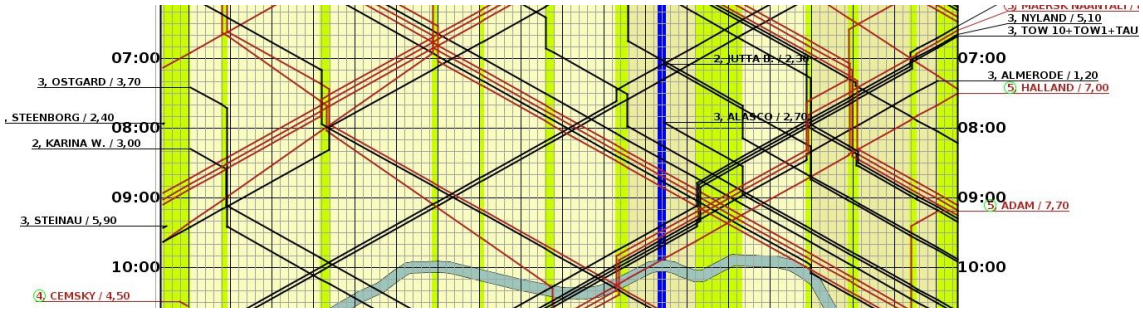


Figure 1: Example of a distance-time diagram showing one of our solutions.

3 Challenges in Scheduling

The mentioned binary variables of the above model define a "sequence" for the conflicting ships on each segment. In this sense, we are talking here about scheduling decisions that must be made for each segment and form the hard part of this model (forgetting about the difficulties arising within the sidings). There are two main differences to classical scheduling which cause problems for the known standard techniques. First, the relation of having a conflict is not transitive and hence, the resulting "sequence" is no total order. Second, there is no designated "processing time." The time period, for which a segment is blocked by a ship for another one really depends on the properties of both and even is not symmetric.

Even when restricting to special cases where these difficulties do not occur we get interesting scheduling problems. Consider the case of one segment surrounded by to sidings where all ships have a conflict on that segment and same velocity. Then the main task is to decide when to change the currently active travel direction on this segment, because every switch of travel direction induces lots of waiting time. The ships must be grouped into batches with setup costs. Hence, this problem can be interpreted as a two family batch scheduling problem with release dates, [1] and [3]. Under further restrictions inducing identical "processing times" this can be solved by dynamic programming in polynomial time. Considering more than one segment yields problems of job shop scheduling character.

It is also worth mentioning that there are certain similarities to train scheduling on a single track line [2], but also here the big problem of non conflicting ships occurs.

4 An Algorithm

The main goal of the project was to develop a software that can do simulations in lots of distinct settings to answer questions arising in the enlargement process. Therefore, our algorithm must cover all important real world conditions. This is implemented by an elaborate procedure for dynamic routing with time windows extending the techniques of [4]. Once these feasible and realistic solutions can be produced, it was embedded in a local search procedure that considers the scheduling decisions on the segments to improve solutions. To respect the online character this is done in a rolling horizon manner. Since the officers in charge of the enlargement were satisfied and impressed by the produced solutions, our software is a perfect tool for simulations.

References

[1] A. Allahverdi, C.T. Ng, T.C.E. Cheng, and M. Y. Kovalyov. A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187(3):985 – 1032, 2008.

[2] A. Caprara, M. Fischetti, and P. Toth. Modeling and solving the train timetabling problem. *Operations Research*, 50(5):851–861, 2002.

[3] C. N. Potts and M. Y. Kovalyov. Scheduling with batching: A review. *European Journal of Operational Research*, 120(2):228 – 249, 2000.

[4] R.H. Möhring, E. Köhler, E. Gawrilow, and B. Stenzel. Conflict-free real-time AGV routing. In H. Fleuren, D. den Hertog, and P. Kort, editors, *Operations Research Proceedings 2004*, pages 18–24, Berlin, 2005. Springer-Verlag.

How To Schedule When You Have To Buy Your Energy

Kirk Pruhs * Cliff Stein (Speaker) †

As the price of server hardware has remained relatively stable, energy cost becomes one of the primary components in the total cost of ownership for computer server systems in data centers [2]. A commonly used power management technique is speed scaling, changing the speed of the processor. As the dynamic power used by a processor is approximately the cube of the speed of the processor (this is called the cube-root rule for CMOS based processors [3, 5]), even a modest reduction in speed can have a dramatic impact on power. Researchers at Google reported an approximately twenty percent energy savings from implementing the following reactive strategy: When the workload of a processor was light, the speed was scaled down, and when most processors were at maximum speed, some less time critical tasks were suspended, to be restarted when the system was not so heavily loaded [4].

Scheduling problems related to speed scaling and power management naturally have competing dual objectives: some quality of service (QoS) objective, and some power related objective. By now there are many tens of papers on speed scaling in the algorithmic literature (and many more in the general computer science literature). Roughly speaking, all of the formal problems considered in the algorithmic speed scaling literature fall into one of two categories. The first type of problem turns one of the QoS or power objectives into a constraint, and optimizes the other objective. An example is minimizing the total flow time subject to the constraint that the energy used doesn't exceed an energy bound representing the energy stored in a battery. The second type of problem optimizes the sum of the QoS and power objectives. An example of this type of problem is minimizing the sum of energy used and total flow time.

In this paper, we introduce a new class of speed scaling problems, which makes the monetary cost of energy more explicit, and we provide algorithmic results for a particular problem in this class. We assume that the scheduler is aware of the income obtainable from finishing particular jobs by particular times, and is aware of the cost of energy. We then naturally assume that the scheduler's goal is to maximize profit, which is the aggregate income minus the aggregate energy cost. One can easily formulate many natural problems within this framework, depending on how one formalizes income and energy costs (and also, of course, depending on the processor and job environments). Here we consider a rather general model for the income of jobs: We assume that there is an non-negative non-increasing income function $I_i(t)$ associated with each job i that specifies the income that is obtained if the job is finished at time t. And we consider the most natural and simple model for energy costs: We assume a fixed cost per unit of energy.

We now explain the job and machine environments that we consider in this paper. Jobs arrive over time at the data center consisting of m identical speed-scalable proces-

*kirk@cs.putt.edu. Computer Science Department. University of Pittsburgh, Pittsburgh, PA, USA
†cliff@ieor.columbia.edu. Department of IEOR, Columbia University, New York, NY, USA.

sors. There is an arbitrary power function $P(s)$ that specifies the power when a processor is run at speed s. Job i arrives at time r_i, with known work/size w_i, and known income function $I_i(t)$. The online scheduler must decide, at each time, which job to run on each processor, and at what speed to run each processor. We allow preemption and migration, that is, jobs can be suspended at any time, and restarted from the point of suspension at a later time, possibly on a different machine. Recall that our objective is to maximize the income from the scheduled jobs minus the total energy costs.

The standard measure of goodness for an online algorithm is competitiveness, which in this setting is, roughly speaking, the worst-case, over all possible inputs, of the relative error between the optimal profit and the profit achieved by the online algorithm. One generally seeks algorithms that are competitive, that is, where this relative error is bounded. The motivation for seeking competitive algorithms is that if the online algorithm achieves very little profit, then it must be because great profit was not achievable, and not because the algorithm was at fault.

Our Results

The most obvious first concern that arises when seeking a competitive algorithm for this problem is that one can imagine a situation where the online algorithm does not achieve a positive profit, even though a positive profit is achievable, immediately killing any hope of a competitive algorithm. We start by observing, that this situation cannot occur, that is, that there is a simple online algorithm that achieves a positive profit if it is possible to do so. Unfortunately, we show that, in some sense, this result is the best positive result possible for the competitive ratio by showing that the competitive ratio can not be bounded by any function of the number of jobs. In contrast, we show that, if the online algorithm has $(1+\epsilon)$-speed augmentation, which in this setting means that if a processor can run at power P and speed s, then the online algorithm can run the processor at power P and speed $(1+\epsilon)s$. We then give an online scheduling algorithm that we show is $O(\frac{1}{\epsilon^3})$-competitive in terms of profit. Using standard terminology one could say that this algorithm is a scalable scheduling algorithm, that is, it is $(1+\epsilon)$-speed $O(1)$-competitive. Intuitively, scalable algorithms can handle almost the same load as optimal.

We now give an overview of the development of our scalable algorithm. The first key idea is that of a critical speed function $\hat{s}_i(t)$, which, for job i, specifies the fastest speed that the adversary can run job i and still obtain a non-negative profit if the job completes at time t. When a job i is released, the online algorithm determines whether to admit the job, and if the job is admitted, determines a deadline d_i for the job. Whenever an admitted job i is run by the online algorithm, it will be run at speed slightly faster than the critical speed for its deadline, $\hat{s}_i(d_i)$. Fixing the speed for a job defines a density for the job, which is roughly the profit that will be obtained by the job if it is completed at its deadline divided by the time that the job must be run to be completed. Intuitively, the online algorithm always picks the highest density jobs to run. Also intuitively, when a job is released, the online algorithm sets the deadline to be the time where it will obtain maximum profit from this job, assuming that in the future no more jobs arrive and that the highest density jobs will be run at their critical speeds.

To show that the online algorithm is scalable, we show that the profit obtained by the online algorithm is a constant fraction of the profit of the jobs that the online algorithms

admits, and that the profit of these admitted jobs is a constant fraction of the optimal profit. In order to accomplish the latter goal, we show that there is a near optimal schedule OPT', that, with modest speed augmentation, is $O(1)$-competitive in terms of profit with the optimal schedule, and OPT' has the property that it runs each job i at speed approximately equal to the critical speed of the job for the completion time $\hat{s}_i(C_i^O)$. of that job in the optimal schedule OPT. Thus OPT' is still nearly optimal, but is structurally similar to the online schedule in that jobs are run at their critical speeds. A priori, it is not clear that such a schedule OPT' exists since a job i may be run at very different speeds in OPT' and in the online schedule. In other words, $\hat{s}_i(C_i^O)$ and $\hat{s}_i(d_i)$ may be very different, since there is no reason that the completion time in the optimal schedule, C_i^O, and the deadline set by the online algorithm, d_i, need be similar.

The income model in our paper was considered in [1], a scalable algorithm for maximizing income on a single fixed speed processor was given. Our algorithm and analysis necessarily generalize the results in [1] as we have multiple processors instead of a single processor, our processors are speed scalable instead of fixed speed, and we have profit as the objective instead of income. The fact that the processors are speed scalable creates complications because the algorithm and analysis in [1] use the fact that the processing time for a job is fixed. The objective of profit also creates complications because the algorithm and analysis in [1] use the fact that income is monotonic in time, which isn't true for profit.

References

[1] Nikhil Bansal, Ho-Leung Chan, and Kirk Pruhs. Competitive algorithms for due date scheduling. In *ICALP*, pages 28–39, 2007.

[2] Luiz André Barroso. The price of performance. *ACM Queue*, 3(7):48–53, 2005.

[3] David M. Brooks, Pradip Bose, Stanley E. Schuster, Hans Jacobson, Prabhakar N. Kudva, Alper Buyuktosunoglu, John-David Wellman, Victor Zyuban, Manish Gupta, and Peter W. Cook. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 20(6):26–44, 2000.

[4] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. Power provisioning for a warehouse-sized computer. In *International Symposium on Computer Architecture*, pages 13–23, 2007.

[5] Trevor Mudge. Power: A first-class architectural design constraint. *Computer*, 34(4):52–58, 2001.

Inner Product Spaces for MinSum Coordination Mechanisms *

Richard Cole † José R. Correa (Speaker) ‡ Vasilis Gkatzelis §
Vahab Mirrokni ¶ Neil Olver ‖

Traditionally, work in operations research has focused on finding globally optimal solutions for optimization problems. In tandem, computer scientists have long studied the effects of a lack of different kinds of resources, mainly the lack of computational resources in optimization. In designing massive decentralized systems, the *lack of coordination* among different participating agents has become an important consideration. This issue is typically addressed through distributed algorithms in which a central authority designs mechanisms (protocols) specifying the rules of the game, with the goal that the independent and selfish choices of the users result in a socially desirable outcome. To measure the performance of these algorithms, the global objective function (social cost) is evaluated at equilibrium points for selfish users. For games, probably the most accepted such measure is the *price of anarchy* [10], the worst case ratio of the social cost at a Nash equilibrium to that at a social optimum; the same measure can be used for coordination mechanisms; sometimes we call this their *approximation factor* to highlight that this is a distinct measure.

The by now standard approach to bound the price of anarchy (PoA) when social cost is taken to be the sum of individual costs works as follows [11]. First, the social cost is bounded by using the equilibrium conditions, noting that an individual is better off at equilibrium than she would be if she unilaterally changed her strategy to the one she would use in a centralized optimum. Second, the actual (weighted) sum of player costs is also upper bounded, using an appropriately chosen inequality, by a linear combination of the social cost of the equilibrium and the social cost of an optimal solution.

In this paper we establish a methodology to deal with the second step of this proof scheme. Our method interprets the sum in the second step as an inner product on a suitable space. Then, we apply the Cauchy-Schwartz inequality in the chosen inner product space, and go back to the original space by applying a minimum norm distortion inequality. Many of the existing results employ a special case of this approach in which the costs can be expressed in terms of quadratic polynomials to which the Cauchy-Schwartz inequality can be applied directly without the need for an intermediate inner product space. We apply our new method in the context of scheduling jobs on unrelated machines. Our method elucidates the hidden structure in the games we consider. Once

*Full version appears in [4]. Supported by grants NSF CCF0830516 and FONDECYT 1090050.
†cole@cims.nyu.edu. Courant Institute, New York University, N.Y.
‡jcorrea@dii.uchile.cl. Departamento de Ingeniería Industrial, Universidad de Chile.
§gkatz@cims.nyu.edu. Courant Institute, New York University, N.Y.
¶mirrokni@google.com. Google Research, New York, N.Y.
‖olver@math.mit.edu. Department of Mathematics, MIT.

the framework has been set up, our proofs become short and elegant, thus we anticipate that this method may prove useful elsewhere too.

Specifically, we consider the classic problem of scheduling n jobs on m unrelated machines from a game theoretic perspective. In this situation, job j takes time p_{ij} if processed on machine i, and also has an associated weight w_j. Although the goal is to minimize the weighted sum of completion times of jobs, we consider the *scheduling game* in which each job is a fully informed player wanting to minimize its individual completion time, while each machine announces a policy which it will follow in processing the jobs it is assigned. Our goal is to choose the policy so as to minimize the approximation ratio of the actual costs under this policy to the optimal costs obtainable under any policy. To this end, several approaches imposing incentives on self-interested agents have been proposed, including some using monetary transfers [2,5,8], and others enforcing strategies on a fraction of users as a Stackelberg strategy [16]. Ultimately one could also apply a VCG mechanism to achieve social efficiency. The main drawback of these methods is the need for global knowledge of the system. A different approach, and the focus of our paper, uses coordination mechanisms, which only require local computations.

More formally, a coordination mechanism [3] is a set of *local policies*, one per machine, specifying how the jobs assigned to that machine are scheduled. Here, *local* means that a machine's schedule must be a function only of the jobs it is assigned (and their processing times on all machines), allowing the policy to be implemented in a distributed fashion. We actually study *strongly* local policies, meaning that the schedule of any machine i is a function only of the processing times p_{ij}, weights w_j and IDs of jobs assigned to it. It will also be useful to restrict attention to policies that always use the full capacity of a machine, and release jobs immediately upon completion. We call such policies *prompt*.

Employing our new technique, we develop the first constant-factor approximate coordination mechanisms for the selfish machine scheduling problem for unrelated machines. We start by studying Smith's Rule [15], in which machines process jobs in increasing order of their processing time to weight ratio. Here the space that appropriately fits our method is L^2 and a norm distortion inequality is in fact not needed. We prove that the approximation factor for this policy is exactly 4, improving upon a result by Correa and Queyranne [6]. We also show that this is the best possible among all deterministic and non-preemptive strongly local coordination mechanisms, assuming the prompt property.

The constant approximation ratio for the weighted sum of completion times is in sharp contrast to the known super-constant inapproximability results for coordination mechanisms for the makespan function [1,9] (e.g., an $\Omega(m)$ lower bound for the shortest-first coordination mechanism). In fact, it is still open whether there is a coordination mechanism with a constant approximation ratio for the makespan function.

Next, we go beyond the approximation ratio of 4 using preemptive[1] and randomized mechanisms. First, we consider a preemptive policy, generalizing that of Dürr and Thang [7], in which each machine splits its processing capacity among its assigned jobs in proportion to their weights. We uncover a connection of this policy to Smith's Rule, allowing us to apply a similar proof strategy, but yielding a significantly improved approximation factor of 2.618. On the other hand, we prove that with anonymous jobs, no set of deterministic prompt policies, be they preemptive or not, can achieve a factor better than 2.5. To break this new barrier we consider a policy in which jobs are randomly, but non-uniformly, ordered, based on their processing time to weight ratio. Under this

[1]By preemption we mean that the computation of a job is suspended and, implicitly, resumed later.

policy the appropriate space has to be carefully chosen and uses a rather nonstandard inner product, induced by a Hilbert matrix, whose i, j entry equals $1/(i+j-1)$. A norm distortion inequality is then needed to relate the norm in this space to the original cost of an optimal schedule, leading to yet another improvement in the approximation factor to 2.134. Moreover, we show a lower bound of 5/3 for this policy.

Finally, inspired by our preemptive mechanism we design a new *combinatorial* $(2+\epsilon)$-approximation algorithm for optimizing the weighted sum of completion times on unrelated machines. This improves on the approximation factor of our mechanisms and complements the known non-combinatorial constant-factor approximation algorithms: a linear programming based $\frac{3}{2}+\epsilon$-approximation algorithm [12], and the best currently known factor of $\frac{3}{2}$ based on a convex quadratic relaxation [13, 14].

References

[1] Y. Azar, K. Jain, and V.S. Mirrokni. (almost) optimal coordination mechanisms for unrelated machine scheduling. In *SODA*, pages 323–332, 2008.

[2] I. Caragiannis, C. Kaklamanis, and P. Kanellopoulos. Taxes for linear atomic congestion games. *ACM Transactions on Algorithms (to appear)*.

[3] G. Christodoulou, E. Koutsoupias, and A. Nanavati. Coordination mechanisms. *Theor. Comput. Sci.*, 410(36):3327–3336, 2009.

[4] R. Cole, J.R. Correa, V. Gkatzelis, V. Mirrokni, and N. Olver. Inner Product Spaces for MinSum Coordination Mechanisms. To appear in *STOC* 2011.

[5] R. Cole, Y. Dodis, and T. Roughgarden. How much can taxes help selfish routing? *J. Comput. Syst. Sci.*, 72(3):444–467, 2006.

[6] J.R. Correa and M. Queyranne. Efficiency of equilibria in restricted uniform machine scheduling with minsum social cost. 2010 (manuscript).

[7] C. Dürr and N.K. Thang. Non-clairvoyant scheduling games. In *SAGT*, pages 135–146, 2009.

[8] L. Fleischer, K. Jain, and M. Mahdian. Tolls for heterogeneous selfish users in multicommodity networks and generalized congestion games. In *FOCS*, pages 277–285, 2004.

[9] L. Fleischer and Z. Svitkina. Preference-constrained oriented matching. In *ANALCO*, pages 56–65, 2010.

[10] E. Koutsoupias and C. Papadimitriou. Worst-case equilibria. In *STACS*, pages 404–413, 1999.

[11] T. Roughgarden. Intrinsic robustness of the price of anarchy. In *STOC*, pages 513–522, 2009.

[12] A.S. Schulz and M. Skutella. Scheduling unrelated machines by randomized rounding. *SIAM J. Discrete Math.*, 15(4):450–469, 2002.

[13] J. Sethuraman and M.S. Squillante. Optimal scheduling of multiclass parallel machines. In *SODA*, pages 963–964, 1999.

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

[15] W. Smith. Various optimizers for single stage production. *Naval Res. Logist. Quart.*, 3(1-2):59–66, 1956.

[16] H. von Stackelberg. Marktform und Gleichgewicht. *Springer-Verlag*, 1934. English translation entitled *The Theory of the Market Economy*.

Balanced Interval Coloring

Antonios Antoniadis Falk Hüffner Pascal Lenzner
Carsten Moldenhauer Alexander Souza (Speaker) *

We consider the following load balancing problem: We are given a set $\mathcal{I} = \{I_1, \ldots, I_n\}$ of tasks, where each task is represented by an interval $I = [\ell, r] \in \mathcal{I}$ with starttime ℓ and endtime r. Furthermore, we are given k servers and have to assign the tasks to the servers as evenly as possible. That is, we want to minimize the maximal difference of the numbers of tasks processed by any two servers over all times.

We formalize this in terms of an interval coloring problem: We are given a set $\mathcal{I} = \{I_1, \ldots, I_n\}$ of n intervals on the real line and a set $K = \{1, \ldots, k\}$ of k colors. A *k-coloring* is a mapping $\chi : \mathcal{I} \to K$. For a fixed k-coloring χ and a point $x \in \mathbb{R}$, let $c_i(x)$ denote the number of intervals containing x that have color i in χ. Define the *imbalance* of χ at x by

$$\mathrm{imb}(x) = \max_{i,j \in K} |c_i(x) - c_j(x)|. \tag{1}$$

In words, this is the maximum difference in the size of color classes at point x. The *imbalance* of χ is given by $\mathrm{imb}(\chi) = \max_{x \in \mathbb{R}} \mathrm{imb}(x)$.

These definitions yield the following minimization problem:

Minimum Imbalance Interval k-Coloring
Instance: A set of intervals $\mathcal{I}$.
Task: Find a k-coloring χ with minimal $\mathrm{imb}(\chi)$.

We call a k-coloring with imbalance at most one *balanced*. Observe that if the number of intervals intersecting at some point is not divisible by k, then imbalance at least one is unavoidable. On the other hand, if the number of intersecting intervals is divisible by k, then no coloring having imbalance one exists. Thus, if a balanced coloring exists, its imbalance is minimal.

The questions considered in this paper are outlined as follows:

(i) Is there always a balanced k-coloring?

(ii) If so, is it possible to construct a balanced k-coloring in polynomial time?

(iii) If we consider arcs of a circle (instead of intervals), do balanced k-colorings always exist?

(iv) How is the situation if intervals arrive online?

*Institut für Informatik, Humboldt-Universität zu Berlin, D-10099 Berlin, Germany, {antoniad,hueffner,lenzner,moldenha,souza}@informatik.hu-berlin.de

(v) If d-dimensional boxes (instead of intervals) are considered, can the existence of a balanced k-coloring be decided in polynomial time?

The problem has close connections to discrepancy theory; see Doerr [4] and Matoušek [6] for introductions to the field. Let $H = (X, U)$ be a hypergraph consisting of a set X of vertices and a set $U \subseteq 2^X$ of hyperedges. Analogous to the previous definitions, a k-coloring is a mapping $\chi : X \to K$, and the imbalance $\mathrm{imb}(\chi)$ is the largest difference in size between two color classes over all hyperedges. The *discrepancy* problem is to determine the smallest possible imbalance, i. e., $\mathrm{disc}(H) = \min_{\chi: X \to K} \mathrm{imb}(\chi)$.

Hence our problem is to find the discrepancy of the hypergraph $H = (I, U)$, where U is the family of all maximal subsets of intervals intersecting at some point. It turns out that this hypergraph has totally unimodular incidence matrix, which is useful because de Werra [9] proved that balanced k-colorings exist for hypergraphs with totally unimodular incidence matrix. However, the proof in [9] is only partially constructive: A balanced k-coloring is constructed by iteratively solving the problem of balanced 2-coloring on hypergraphs with totally unimodular incidence matrix, for which no algorithm was given in [9].

Further related work in discrepancy theory mostly considers hypergraph coloring with two colors and often from existential, rather than algorithmic perspective. For an arbitrary hypergraph H with n vertices and m hyperedges, the bound $\mathrm{disc}(H) \leq \sqrt{2n \ln(2m)}$ for 2-coloring follows with the probabilistic method; see also [4]. For $m \geq n$, Spencer [7] proved the stronger result $\mathrm{disc}(H) = O(\sqrt{n \log(m/n)})$, which is in particular interesting for $m = O(n)$. If each vertex is contained in at most t edges, the 2-coloring bound $\mathrm{disc}(H) = O(\sqrt{t} \log n)$ was shown by Srinivasan [8] and the bound $\mathrm{disc}(H) \leq 2t-1$ by Beck and Fiala [2]. Biedl et al. [3] improved the bound to $\mathrm{disc}(H) \leq \max\{2t-3, 2\}$ for 2-colorings and established $\mathrm{disc}(H) \leq 4t - 3$ for general k-colorings. They also showed that it is NP-complete to decide the existence of balanced k-colorings for hypergraphs with $t \geq \max\{3, k-1\}$ and $k \geq 2$.

Bansal [1] recently gave efficient algorithms that achieve 2-color imbalances similar to [7, 8] up to constant factors. In particular, an algorithm yields $\mathrm{disc}(H) = O(\sqrt{n} \log(2m/n))$ matching the result of Spencer [7] if $m = O(n)$. Furthermore, $\mathrm{disc}(H) = O(\sqrt{t} \log n)$ complies with the non-constructive result of Srinivasan [8]. For general $k > 2$, Doerr and Srivastav [5] gave a recursive method constructing k-colorings from (approximative) 2-colorings.

Our Contributions. We contribute the following answers to the above questions:

(i) Balanced k-colorings exist for any set $\mathcal{I}$ of intervals, i. e., question (i) can *always* be answered in the affirmative. We establish this by showing that our hypergraph H has totally unimodular incidence matrix and then applying a result of de Werra [9]. This also follows independently from our algorithmic results below.

(ii) We present an $O(n \log n)$ time algorithm for finding a balanced 2-coloring, thereby establishing a constructive result for intervals. Furthermore, we give an $O(n \log n + kn \log k)$ algorithm for finding a balanced k-coloring. This is an improvement in time complexity, since the construction of de Werra [9] combined with our algorithm for 2-coloring only yields $O(n \log n + k^2 n)$. We also note that our algorithm works for any hypergraph with incidence matrix having the consecutive-ones property.

(iii) If we consider arcs of a circle instead of intervals, balanced k-colorings do not exist in general. However, we give an algorithm achieving imbalance at most two with the same time complexity as in the interval case.

(iv) In an online scenario, in which we learn intervals over time, the imbalance of *any* online algorithm can be made arbitrarily high.

(v) For d-dimensional boxes, it is NP-complete to decide if a balanced k-coloring exists for any $d \geq 2$ and any $k \geq 2$. Our reduction is from Not-All-Equal 3SAT. This result clearly implies NP-hardness of the respective minimization problem.

References

[1] N. Bansal. Constructive algorithms for discrepancy minimization. In *Proc. 51st FOCS*. IEEE Computer Society, 2010. To appear. Also in arXiv:1002.2259v4.

[2] J. Beck and T. Fiala. "Integer making" theorems. *Discrete Applied Mathematics*, 3(1):1–8, 1981.

[3] T. C. Biedl, E. ÄŇenek, T. M. Chan, E. D. Demaine, M. L. Demaine, R. Fleischer, and M.-W. Wang. Balanced k-colorings. *Discrete Mathematics*, 254(1–3):19–32, 2002.

[4] B. Doerr. *Integral Approximation*. Habilitationsschrift, Christian-Albrechts-UniversitÃd't zu Kiel, 2005.

[5] B. Doerr and A. Srivastav. Multicolour discrepancies. *Combinatorics, Probability and Computing*, 12:365–399, 2003.

[6] J. Matoušek. *Geometric Discrepancy: An Illustrated Guide*, volume 18 of *Algorithms and Combinatorics*. Springer, 1999.

[7] J. Spencer. Six standard deviations suffice. *Transactions of the American Mathematical Society*, 289(2):679–706, 1985.

[8] A. Srinivasan. Improving the discrepancy bound for sparse matrices: Better approximations for sparse lattice approximation problems. In *Proc. 8th SODA*, pages 692–701. ACM-SIAM, 1997.

[9] D. de Werra. Equitable colorations of graphs. *Revue FranÃğaise d'Informatique et de Recherche opÃl'rationnelle*, R-3:3–8, 1971.

Clique Clustering yields a PTAS for max-Coloring Interval Graphs

Tim Nonner (Speaker) *

Coloring a given graph $G = (V, E)$ is a classical NP-hard problem in combinatorial optimization. One reason why graph coloring has been studied so extensively is the fact that many practical problems in scheduling and planning can be formulated in such a way. The arguably simplest example is that the nodes V represent tasks which need to be partitioned into *color classes* of pairwise non-conflicting tasks, where a conflict between two tasks is indicated by an edge in E connecting them. All tasks in one color class may share a common resource, and hence minimizing the number of color classes also minimizes the number of needed resources. It is natural to assume that each task requires a resource during a given time interval, and hence two task conflict if their time intervals intersect. This results in an *interval graph* and we may think of the nodes in V as intervals in this case. It is folklore that an optimal coloring of a given interval graph can be found in polynomial time using the *first-fit strategy*: sort the intervals according to their left endpoints, and then iteratively assign colors according to this ordering. This drastically differs from general graphs, where finding an optimal coloring is hard to approximate even within $n^{1-\epsilon}$ for any $\epsilon > 0$, unless NP $\subseteq$ ZPP [5].

However, coloring interval graphs fails to express non-uniform resource requirements. For instance, a resource might be a buffer and the tasks memory requests of different size that need to be buffered during a given time interval [12]. In this case, a buffer used by different non-conflicting requests needs to be large enough to hold any such request. We can model this extension by assigning a weight $w_I \in \mathbb{R}^+$ to each interval $I \in V$ that represents the size of the corresponding request, and hence the buffer assigned to a color class C of requests needs to have at least size $\max_{I \in C} w_I$. Thus, finding an optimal coloring in this context is called *max-coloring interval graphs*: partition a given interval graph $G = (V, E)$ into color classes $C_1, C_2, \ldots, C_k$ such that $\sum_{i=1}^{k} \max_{I \in C_i} w_I$ is minimized. The classical problem of coloring interval graphs is contained as a special case by using uniform weights.

1 Previous work

Unfortunately, the first-fit strategy does not work for max-coloring interval graphs, but it has been shown by Pemmaraju, Raman, and Varadarajan [12] that this problem is NP-hard. A simplified proof was given by the same authors in [11] with a reduction from coloring circular arc graphs, a superclass of interval graphs. They also conjectured APX-hardness, but this conjecture is no longer valid [13]. Finally, they presented an

*tno@zurich.ibm.com. IBM Research - Zurich, Business Optimization Group, Saeumerstrasse 4, 8803 Rueschlikon, Switzerland.

2-approximation algorithm for max-coloring interval graphs [12], and Pemmaraju and Raman [10] later on showed that any graph class that admits an α-approximation algorithm for coloring also admits an 4α-approximation algorithm for max-coloring. Recall here than an *α-approximation algorithm* yields a solution in polynomial whose cost is at most α times the cost of an optimal solution. Hence, since it is a well-known fact that perfect graphs, a superclass of interval graphs, can be colored in polynomial time [6], this yields a 4-approximation algorithm for max-coloring perfect graphs. Epstein and Levin [3] improved this factor from 4 to e. On the other hand, after a line of improvements [4,7,8], Kavintha and Mestre [9] presented an algorithm for max-coloring paths, a subclass of interval graphs, which requires only time $\mathcal{O}(n+S(n))$, where $S(n)$ is the time to sort the node weights. A polynomial-time approximation scheme (PTAS) for trees is known [1], i.e., there is a $(1+\epsilon)$-approximation algorithm for any $\epsilon > 0$, but max-coloring bipartite graphs is APX-hard [2], i.e., there is no PTAS unless P = NP. However, in the context of the original formulation of this problem in buffer management, interval graphs remain the most relevant graph class, since they are easy to describe, but yet powerful enough to model temporal conflicts.

2 Contributions

Closing a gap which has been open for years, we settle the approximation complexity of max-coloring interval graphs by presenting a PTAS. Our main building block, which we call *clique clustering*, is to group intervals in clusters in order to trade the overlap structure for accuracy such that only a logarithmic number of clusters intersect at each point. We think that this exponential drop is of general interest and likely to find application in other interval based problems.

References

[1] Evripidis Bampis, Alexander Kononov, Giorgio Lucarelli, and Ioannis Milis. Bounded max-colorings of graphs. In *Proceedings of the 21st International Symposium on Algorithms and Computation (ISAAC'10)*, pages 353–365, 2010.

[2] Dominique de Werra, Marc Demange, Bruno Escoffier, Jérôme Monnot, and Vangelis Th. Paschos. Weighted coloring on planar, bipartite and split graphs: Complexity and approximation. *Discrete Applied Mathematics*, 157(4):819–832, 2009.

[3] Leah Epstein and Asaf Levin. On the max coloring problem. In *Proceedings of the 5th International Workshop on Approximation and Online Algorithms (WAOA'07)*, pages 142–155, 2007.

[4] Bruno Escoffier, Jérôme Monnot, and Vangelis Th. Paschos. Weighted coloring: further complexity and approximability results. *Inf. Process. Lett.*, 97(3):98–103, 2006.

[5] Uriel Feige and Joe Kilian. Zero knowledge and the chromatic number. *J. Comput. Syst. Sci.*, 57(2):187–199, 1998.

[6] M. Gröschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag, 1988.

[7] D. J. Guan and Xuding Zhu. A coloring problem for weighted graphs. *Inf. Process. Lett.*, 61(2):77–81, 1997.

[8] Magnús M. Halldórsson and Hadas Shachnai. Batch coloring flat graphs and thin. In *Proceedings of the 11th Scandinavian Workshop on Algorithm Theory (SWAT'08)*, pages 198–209, 2008.

[9] Telikepalli Kavitha and Julián Mestre. Max-coloring paths: Tight bounds and extensions. In *Proceedings of the 20th International Symposium on Algorithms and Computation (ISAAC'09)*, pages 87–96, 2009.

[10] Sriram V. Pemmaraju and Rajiv Raman. Approximation algorithms for the max-coloring problem. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP'05)*, pages 1064–1075, 2005.

[11] Sriram V. Pemmaraju, Rajiv Raman, and Kasturi Varadarajan. Max-coloring and online coloring with bandwidths on interval graphs. *ACM Transactions on Algorithms*, accepted for publication.

[12] Sriram V. Pemmaraju, Rajiv Raman, and Kasturi Varadarajan. Buffer minimization using max-coloring. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'04)*, pages 562–571, 2004.

[13] Rajiv Raman. Personal communication. 2010.

Smoothed performance guarantees of local optima for multiprocessor scheduling

Tobias Brunsch* Heiko Röglin† Cyriel Rutten (Speaker) ‡
Tjark Vredeveld §

1 Introduction

In this talk, we consider *smoothed performance guarantees of local optima* for the following scheduling problem. Given is a set J of n jobs, each of which needs to be processed without preemption on one of m machines. A machine can process at most one job at a time and all jobs and machines are available at time 0. The goal is to schedule the jobs in such a way that the *makespan* is minimized, i.e., we want the last job to complete as early as possible. The time p_{ij} it takes for a job j ($j \in J$) to be fully processed on a machine i depends on the machine environment. In this talk, we consider two machine environments. The first one is the one of *uniform parallel machines*, also known as related machines: each job j has a given processing requirement $p_j \geq 0$ and each machine has a speed $s_i > 0$. We assume machines to be indexed according to their speeds where machine 1 is the machine having maximum speed. The processing time for job j on machine i is $p_{ij} = p_j/s_i$. The second machine environment that we consider is the one of *restricted identical machines*: a job j is only allowed to be processed on a subset $\mathcal{M}_j \subseteq \{1, \ldots, m\}$ of the machines and when processed on one of its elegible machines, the processing time of job j is p_j. That is, $p_{ij} = p_j$ if $i \in \mathcal{M}_j$ and $p_{ij} = \infty$ for $i \notin \mathcal{M}_j$. Both versions are known to be strongly NP-hard [3].

One way to find approximate solutions is *local search*. Local search methods iteratively search through the set of feasible solutions. Starting from an initial solution, a local search procedure moves from a feasible solution to a neighboring solution until some stopping criteria are met. A *neighborhood function* defines for each feasible solution a set of solutions which are in some sense close to it. This set is called a *neighborhood*. The choice of a suitable neighborhood function has an important influence on the performance of local search. The simplest form of local search is *iterative improvement*, also called local improvement. This method iteratively chooses a better solution in the neighborhood of the current solution and terminates when no better solution is found. The final solution is called a *local optimum*.

An algorithm is called a ρ-approximation algorithm if it is guaranteed to deliver a solution that has value at most ρ times the optimal solution value; the value ρ is called

*brunsch@cs.uni-bonn.de. Department of Computer Science, University of Bonn, Germany
†heiko@roeglin.org. Department of Computer Science, University of Bonn, Germany
‡c.rutten@maastrichtuniversity.nl. Maastricht University, The Netherlands
§t.vredeveld@maastrichtuniversity.nl. Maastricht University, The Netherlands

the *performance guarantee*. The goal of *worst-case analysis of local search* is to determine the performance guarantee local optima with respect to a neighborhood. Also the time to find such a local optimum by an iterative improvement procedure is of interest.

Smoothed analysis was introduced by Spielman and Teng [6] as a hybrid between average-case and worst-case analysis to explain the success of algorithms that are known to work well in practice while presenting poor worst-case performance. The basic idea is to randomly perturb the initial input instances and to analyze the performance of the algorithm on the perturbed instances. Given an input instance $\check{I}$, we denote by $N(\check{I})$ the set of instances that are obtainable by smoothing the instance according to some probability distribution f. The *smoothed performance guarantee* is defined as

$$\rho = \sup_{\check{I}} \mathbb{E}_{I \xleftarrow{f} N(\check{I})} \left[\frac{\mathsf{alg}(I)}{\mathsf{opt}(I)} \right].$$

Smoothing model. In our smoothing model, only the processing requirements are smoothed. W.l.o.g. we assume that these processing requirements are between 0 and 1. To smooth the processing requirements, we use an additive smoothing model. Let $\check{p}_j$ denote the original processing requirement. Then given a smoothing parameter $\phi \geq 1$, the smoothed processing requirements are defined by

$$p_j = \begin{cases} \frac{1}{2\phi} + \epsilon_j & \text{if } p_j < \frac{1}{2\phi}, \\ \check{p}_j + \epsilon_j & \text{if } \frac{1}{2\phi} \leq \check{p}_j \leq 1 - \frac{1}{2\phi}, \\ 1 - \frac{1}{2\phi} + \epsilon_j & \text{if } \check{p}_j > 1 - \frac{1}{2\phi}, \end{cases}$$

where ϵ_j is drawn uniformly at random from the interval $[\frac{-1}{2\phi}, \frac{1}{2\phi}]$.

2 Neighborhood

Before discussing the neighborhoods, we first describe our representation of a schedule. As the sequence in which the jobs are processed does not influence the makespan of a schedule for a given assignment of the jobs to the machines, we represent a schedule by such an assignment. This is equivalent to a partitioning of the set of jobs into m disjoint subsets $M_1, \ldots, M_m$, where M_i is the set of jobs scheduled on machine i. Given a schedule $\sigma = (M_1, \ldots, M_m)$ the load of machine i is the total processing time of its jobs: $L_i(\sigma) = \sum_{j \in M_i} p_{ij}$.

In this talk, we consider *jump* and *lex-jump* optimal solutions. Given a solution $\sigma = (M_1, \ldots, M_m)$, we obtain a jump neighbor by selecting a job j, scheduled on a machine i, and a machine $h \neq i$ on which J is not scheduled. The neighbor, $\sigma' = (M'_1, \ldots, M'_m)$ is obtained by moving job j to machine h, i.e., $M'_i = M_i \setminus \{j\}$, $M'_h = M_h \cup \{j\}$, and $M'_\ell = M_\ell$ for $\ell \neq i, h$. We say that a schedule σ is *jump optimal* if no jump decreases the makespan or the number of critical machines without increasing the makespan.

The lex-jump neighborhood has the same neighborhood function as the jump neighborhood. The only difference is in what is considered a better solution. From the definition of jump optimal solutions, it is clear that a jump neighbor can be improving only if a job from one of the critical machines is moved. For the lex-jump neighborhood, we say that a neighbor σ' of schedule σ, in which job j is moved from machine i to machine h, is an improving neighbor if $L_h(\sigma') < L_i(\sigma)$, or equivalently $L_h(\sigma) + p_{hj} < L_h(\sigma)$. That

is, the vector of non-increasingly sorted machine loads of σ' is lexicographically smaller than that of schedule σ. Hence, we call a local optimum a lex-jump optimal solution. Note that by considering jobs as selfish agents that want to minimize the load of the machine on which it is scheduled, a lex-jump optimal schedule can be seen as a Nash equilibrium.

The tight (worst-case) performance guarantee for jump optimal schedules is $(1 + \sqrt{4m-3})/2$ on related machines [2, 5] and also on restricted identical machines [4], and $1/2 + \sqrt{1/4 + (m-1)s_1/s_m}$ for restricted related machines [4]. The tight (worst-case) performance guarantee for jump optimal schedules is $\Theta(\log m / \log\log m)$ [1,7] on related machines as well as on restricted identical machines [4] and is $\Theta(\log S / \log\log S)$ [4] on restricted related machines, where $S = \sum_{i \in M} s_i$.

3 Our results

For the related machines environment, we show that, when smoothing according to the model described in the introduction with parameter ϕ, the smoothed performance guarantee for jump-optimal solutions is $2 + 3.7\phi$ and we also give a lower bound of $\Omega(\phi)$ on the smoothed performance guarantee, showing that our analysis is tight up to a constant factor. For the lex-jump neighborhood we have an upper and lower bound on the smoothed performance guarantee of $\Theta(\phi)$.

Finally, for the restricted machines environment, we show that smoothing does not help. For any setting having restricted machines, and for reasonably small values of ϕ, we can construct (smoothed) instances reaching the worst-case performance guarantees (without smoothing) up to a constant factor.

References

[1] B. Awerbuch, Y. Azar, Y. Richter, and D. Tsur, *Tradeoffs in worst-case equilibria*, Theoretical Computer Science **361** (2006), 200–209.

[2] Y. Cho and S. Sahni, *Bounds for list schedules on uniform processors*, SIAM Journal on Computing **9** (1980), 91–103.

[3] M.R. Garey and D.S. Johnson, *Computers and intractability: A guide to the theory of NP-completeness*, W.H. Freeman and Company, New York, 1979.

[4] D. Recalde, C. Rutten, P. Schuurman, and T. Vredeveld, *Local search performance guarantees for restricted related parallel machine scheduling*, LATIN 2010: Theoretical Informatics, LNCS, vol. 6034, Springer, Berlin, 2010, pp. 108–119.

[5] P. Schuurman and T. Vredeveld, *Performance guarantees of local search for multiprocessor scheduling*, Informs Journal on Computing **19** (2007), no. 1, 52–63.

[6] D.A. Spielman and S.H. Teng, *Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time*, Journal of the ACM **51** (2004), no. 3, 385–463.

[7] B. Vöcking, *Selfish load balancing*, Algorithmic Game Theory (N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani, eds.), Cambridge University Press, New York, NY, USA, 2007.

Divisible Load Scheduling to Minimize the Computation Time and Cost

Natalia V. Shakhlevich (Speaker)*

Parallel computer systems have given rise to new scheduling models that go beyond the classical scheduling theory. While in a traditional scheduling model a task can be processed by one machine at a time, a new feature of multiprocessor computations is the ability to split tasks into several parts and to process them simultaneously by different processors, see, e.g., [2, 6]. An additional feature of modern Grid computing and cloud computing systems is the introduction of the cost factor, see, e.g. [1, 4, 7]. This study is motivated by the lack of theoretical research in the area and some inaccuracies which can be found in the earlier research.

We consider the network model described in [5]. There is a set $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$ of m processors connected via a bus type communication medium. One processor of the set $\mathcal{P}$ is selected as a master processor to receive a divisible load of size τ and to divide it into portions of size $\alpha_1\tau$, $\alpha_2\tau$,..., $\alpha_m\tau$, $\sum_{k=1}^{m} \alpha_k = 1$, which are then transmitted to other processors from $\mathcal{P}$ to perform required computations.

The processors have different computation speeds and for each processor $P_k \in \mathcal{P}$ the inverse of the speed w_k is given. This implies that the load of size $\alpha_k\tau$ allocated to processor P_k requires computation time $\alpha_k w_k \tau$.

If P_1 is selected as a master processor and the transmission sequence is P_2, P_3, ..., P_m, then P_1 can start processing its own load of size $\alpha_1\tau$ at time 0 and at the same time it can start transmitting the relevant portions of the load first to P_2, then to P_3, etc., until the last portion is transmitted to P_m, see Fig. 1. If z is the time needed to transmit the whole load of size τ, then the communication time for transmitting the portion $\alpha_k\tau$ to processor P_k is $\alpha_k z$.

With the selected transmission order, processor P_1 completes its portion of computation at time

$$T_1 = \alpha_1 w_1 \tau. \tag{1}$$

Processor P_k, $2 \le k \le m$, receives its portion of the load at time $\sum_{i=2}^{k} \alpha_i z$ and immediately after that it can start computation, which takes $\alpha_k w_k \tau$ time. Thus processor P_k completes its portion of the load at time

$$T_k = \sum_{i=2}^{k} \alpha_i z + \alpha_k w_k \tau.$$

The finish time T of the load is defined as the maximum completion time among all processors,

$$T = \max_{1 \le k \le m} \{T_k\}.$$

*N.Shakhlevich@leeds.ac.uk. School of Computing, University of Leeds, Leeds, LS2 9JT, U.K.

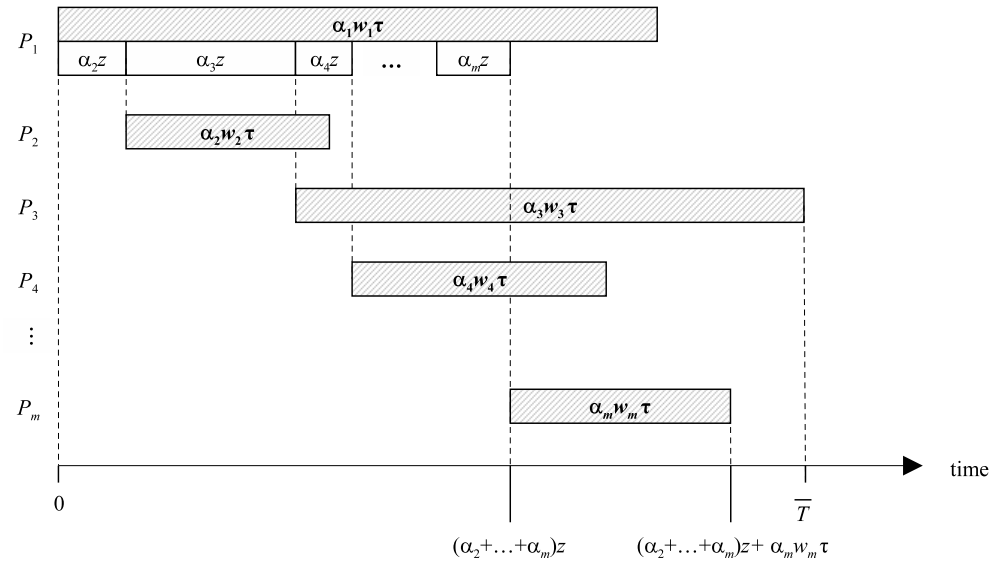


Figure 1: An example of a schedule with master processor P_1 and transmission sequence $P_2, \ldots, P_m$

It is assumed in the described scenario that the master processor can perform data transmission and computation simultaneously. This usually happens if the processor is equipped with an additional front-end co-processor which takes care of all data transfer so that the master processor can perform computation as any other processor of the network. In the absence of a front-end co-processor, the master processor performs data transmission first and only after that it can start computing its portion of the load. In the latter scenario, Fig. 1 should be modified so that for processor P_1 the box "$\alpha_1 w_1 \tau$" is moved immediately after "$\alpha_m z$", and formula (1) should be replaced by

$$T_1 = \sum_{i=2}^{m} \alpha_i z + \alpha_1 w_1 \tau. \tag{2}$$

Processing the load in accordance with the load distribution α_1, α_2,..., α_m incurs computation cost which depends on processors' costs. Following the notation from [5], we denote the cost of using processor $P_k \in \mathcal{P}$ during one time unit by c_k so that the cost of performing the portion of the load $\alpha_k w_k \tau$ by processor P_k is $c_k \alpha_k w_k \tau$. The overall cost of using all processors P is therefore

$$K = \sum_{k=1}^{m} c_k \alpha_k w_k \tau.$$

Thus a schedule S is given by

- the transmission sequence with the first processor of the sequence selected as a master processor

  and

- the load distribution α_1, α_2,..., α_m with $\sum_{k=1}^{m} \alpha_k = 1$.

The quality of a scheduled is measured in terms of the two characteristics: maximum completion time T and computation cost K. As a solution of a bicriteria problem we accept the set of Pareto optimal points defined by the break-points of the so-called efficiency frontier. In a pair of the associated single criterion problems, one of the objectives is bounded while the other one is to be minimized.

We perform a systematic analysis of the model with a fixed number of processors and develop an algorithm for solving the bicriteria problem together with its two single-criterion counterparts. This study leads to formulating some important properties of the general case with an arbitrary number of processors. We demonstrate that the earlier research [5] has a number of limitations and leads to open questions. Some assumptions result in incorrect major conclusions. In particular, it is generally assumed in [5] that the load should be distributed so that all processors complete their portions simultaneously, while as we demonstrate, there often exists a dominating schedule with non-simultaneous finishing times of the processors. Moreover, fixing the processor sequence in the non-decreasing order of the cost/speed characteristic,

$$c_1 w_1 \leq c_2 w_2 \leq \cdots \leq c_m w_m,$$

may be appropriate only for Pareto-optimal solutions with relatively large deadlines; optimal schedules for tight deadlines may have a different order of processors.

References

[1] R. Buyya, D. Abramson and S. Venugopal (2005) The grid economy, *Proceedings of the IEEE* 93, 698-714.

[2] M. Drozdowski (2009) *Scheduling for Parallel Processing*, Springer, London.

[3] M. Drozdowski, M. Lawenda (2005) The combinatorics in divisible load scheduling, *Foundations of Computing and Decision Sciences* 30, 297-308.

[4] S. Kumar, K. Dutta and V. Mookerjee (2009) Maximizing business value by optimal assignment of jobs to resources in grid computing, *European Journal of Operational Research* 194, 856-872.

[5] J. Sohn, T.G. Robertazzi and S. Luryi (1998) Optimizing computing costs using divisible load analysis, *IEEE Trans. Parallel and Distributed Systems* 9, 225-234.

[6] T.G. Robertazzi (2003) Ten reasons to use divisible load theory, *IEEE Computer* 36, 63-68.

[7] J. Yu, R. Buyya and K. Ramamohanarao, Workflow Scheduling Algorithms for Grid Computing. In: F. Xhafa, A. Abraham. eds, *Metaheuristics for Scheduling in Distributed Computing Environments*, Springer, Berlin, Germany, 2008.

Efficient algorithms for average completion time scheduling

René Sitters (Speaker)*

1 Introduction

We analyze the competitive ratio of algorithms for minimizing (weighted) average completion time on identical parallel machines and prove that the well-known shortest remaining processing time algorithm (SRPT) is 5/4-competitive w.r.t. the average completion time objective. For weighted completion times we give a deterministic algorithm with competitive ratio $1.791 + o(1)$. This ratio holds for preemptive and non-preemptive scheduling. These results were presented earlier in [7]

The shortest remaining processing time (SRPT) algorithm is a well-known and simple online procedure for preemptive scheduling of jobs. It produces an optimal schedule on a single machine with respect to the average completion time objective [6].This is not true when SRPT is applied to parallel machines. The best known upper bound on its competitive ratio was 2 [5] until recently (SODA2010), Chung et al. [2] showed that the ratio is at most 1.86. Moreover, they show that the ratio is not better than $21/19 > 1.105$.

The SRPT algorithm has a natural generalization to the case where jobs have given weights. Unfortunately, our proof does not carry over to this case. No algorithm is known to have a competitive ratio less than 2. Remarkably, even for the offline problem, the only ratio less than 2 results from the approximation scheme given by Afrati et al. [1]. A deterministic online algorithm for the preemptive case is given by Megow and Schulz [4] and for the non-preemptive case by Correa and Wagner [3].On the single machine, no non-preemptive online algorithm can be better than 2 competitive [8] but it was unknown if the same is true for parallel machines. We give a simple online algorithm that runs in $O(n \log n)$ time and has competitive ratio $1.791 + o(1)$, i.e., it drops down to 1.791 for $m \to \infty$.

The SRPT algorithm:

Let $t = 1$. Repeat:
If there are more than m jobs available for slot t, then process m jobs in slot t that have the shortest remaining processing times among all available jobs. Otherwise, process all available jobs. Let $t = t + 1$.

Theorem 1. *SRPT is* 5/4*-competitive for minimizing total completion time on identical machines.*

* rsitters@feweb.vu.nl. Department of Econometrics and Operations Research, Free University, Amsterdam.

Algorithm ONLINE(ϵ):

INPUT: Instance $I = \{(p_j, w_j, r_j) \mid j = 1 \dots n\}$.

(i) Let $I' = \{(p'_j, w'_j, r'_j) \mid j = 1 \dots n\}$ with $p'_j = p_j, w'_j = w_j$ and $r'_j = r_j + \epsilon p_j$.

(ii) Apply *non-preemptive* WSPT to I' on a fast single machine (a machine that runs m times faster). Let s_j be the start time of job j.

(iii) Each job j is placed at time s_j on one of the parallel machines as early as possible (but not before s_j).

Theorem 2. *With $\epsilon = 1/\sqrt{m}$, algorithm* ONLINE(ϵ) *is δ_m-competitive for minimizing total weighted completion time, where $\delta_m = (1 + 1/\sqrt{m})^2(3e - 2)/(2e - 2)$. The ratio holds for preemptive and non-preemptive scheduling on m identical parallel machines.*

References

[1] F. Afrati, E. Bampis, C. Chekuri, D. Karger, C. Kenyon, S. Khanna, I. Milis, M. Queyranne, M. Skutella, C. Stein, and M. Sviridenko, *Approximation schemes for minimizing average weighted completion time with release dates*, FOCS '99, 1999, pp. 32–44.

[2] C. Chung, T. Nonner, and A. Souza, *SRPT is 1.86-competitive for completion time scheduling*, Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (Austin, Texas), 2010, pp. 1373–1388.

[3] J.R. Correa and M.R. Wagner, *LP-based online scheduling: from single to parallel machines*, Mathematical Programming **119** (2009), 109–136.

[4] N. Megow and A.S. Schulz, *On-line scheduling to minimize average completion time revisited*, Operations Research Letters **32** (2004), 485–490.

[5] C. Phillips, C. Stein, and J. Wein, *Minimizing average completion time in the presence of release dates, networks and matroids; sequencing and scheduling*, Mathematical Programming **82** (1998), 199–223.

[6] L. Schrage, *A proof of the optimality of the shortest remaining processing time discipline*, Operations Research **16** (1968), no. 3, 687–690.

[7] R.A. Sitters, *Efficient algorithms for average completion time scheduling*, Proc. 14th Int. Conf. Integer Programming and Combinatorial Optimization, 2010, pp. 411–423.

[8] A.P.A. Vestjens, *On-line machine scheduling*, Ph.D. thesis, Department of Mathematics and Computing Science, Technische Universiteit Eindhoven, Eindhoven, the Netherlands, 1997.

The Geometry of Scheduling

Nikhil Bansal * Kirk Pruhs (Speaker) †

We consider the following general offline scheduling problem:

General Scheduling Problem (GSP): The input consists of a collection of n jobs, and for each job j a positive integer release time r_j, a positive integer size p_j, and a cost or weight function $w_j(t) \geq 0$ for each $t > r_j$ (we are purposely not precise about how these weight functions are represented in the input). Jobs are to be scheduled preemptively on one processor after their release times. If job j completes at time t, then a cost of $\sum_{s=r_j+1}^{t} w_j(t)$ is incurred. The scheduling objective is to minimize the total cost, $\sum_{j=1}^{n} \sum_{s=r_j+1}^{C_j} w_j(t)$, where C_j is the completion time of job j.

This general problem generalizes several natural scheduling problems, for example:

Weighted Flow Time: If $w_j(t) = w_j$, where w_j is some fixed weight associated with job j, then the objective is weighted flow time.

Flow Time Squared: If $w_j(t) = 2(t - r_j) - 1$, then the objective is the sum of the squares of the flow times.

Weighted Tardiness: If $w_j(t) = 0$ for t not greater than some deadline d_j, and $w_j(t) = w_j$ for t greater than d_j, then the objective is weighted tardiness.

In general, this problem formulation can model any cost objective function that is the sum of arbitrary cost functions for individual jobs, provided these cost functions are non-decreasing, i.e. it cannot hurt to finish a job earlier. Despite much interest, large gaps remain in our understanding for even basic flow time based scheduling objectives. For example, for weighted flow time, the best known approximation ratios achievable by polynomial-time algorithms are essentially no better than the poly-logarithmic competitive ratios achievable by online algorithms. For weighted tardiness, and flow time squared, no nontrivial approximation ratios were previously known to be achievable. While in contrast, for all of these three problems, even the possibility of a polynomial time approximation scheme (PTAS) has not been ruled out.

The main contribution of the paper of the paper [2] that we discuss here is the design and analysis of a randomized $O(\log \log nP)$-approximation algorithm for GSP, where P is the maximum job size. In the special case when all the release times are 0, we obtain an $O(1)$-approximation algorithm. Let $W = \max_{j,t} w_j(t)$ be the maximum value attained by any weight function. The running time of our algorithm is polynomial in n, $\log P$ and $\log W$, provided that we can in polynomial time determine the times when a weight function doubles. This is polynomial in the input size if the input must contain an explicit representation of the largest possible weight.

*nikhil@us.ibm.com. IBM T. J. Watson Research Center, Yorktown Heights, NY 10598 USA.

†kirk@cs.pitt.edu. Computer Science Department, University of Pittsburgh, Pittsburgh, PA 15260 USA. Supported in part by NSF grants CNS-0325353, IIS-0534531, and CCF-0830558, and an IBM Faculty Award.

The primary insight to obtain these results is to view the scheduling problem geometrically. The initial step is to show that GSP can be reduced (with only a constant factor loss in the approximation ratio) to the following geometric set-cover problem that we call R2C:

Definition of the R2C Problem: The input consists of a collection of $\mathcal{P}$ points in two dimensional space, and for each point $p \in \mathcal{P}$ an associated positive integer demand d_p. Each point $p \in \mathcal{P}$ is specified by its coordinates (x_p, y_p). Further the input contains a collection $\mathcal{R}$ of axis-parallel rectangles, each of them abutting the y-axis. That is, each rectangle $r \in \mathcal{R}$ has the form $(0, x_r) \times (y_r^1, y_r^2)$. In addition, each rectangle $r \in \mathcal{R}$ has an associated positive integer capacity c_r and positive integer weight w_r. The goal is to find a minimum weight subset $S \subset \mathcal{R}$ of rectangles, such that for each point $p \in \mathcal{P}$, the total capacity of rectangles covering p is at least d_p, that is, $\sum_{r \in \mathcal{R}: p \in \mathcal{R}} c_r \geq d_p$.

Job sizes will be mapped to rectangle capacities in our reduction, so we will also use P to denote the largest capacity of any rectangle. Our algorithm for R2C starts with the natural linear programming (LP) relaxation of the problem, strengthened by adding the so-called knapsack cover inequalities. To round this LP solution, our algorithm then proceeds in a way that is by now standard (see for example [1]) in the applications of knapsack cover inequalities. In the terminology of [1], we reduce the problem to rounding an LP solution for the so-called *priority* set cover version of the problem and in addition several set multi-cover problems. These resulting problems are simpler as they are uncapacitated.

In particular we proceed as follows. The algorithm first picks rectangles that are selected by the LP solution to a significant extent (i.e. $x_r \geq \beta$, for some fixed constant β), and then considers the *residual* solution. The knapsack cover inequalities guarantee that remaining LP variables for a feasible solution to the residual instance. Since all variables $x_r \leq \beta$ in this solution, the capacities and demands can be rounded to powers of 2, and the variables can be scaled by a constant factor, so that each point's demand is covered several times over.

Points are then classified as heavy or light depending on whether or not the optimal LP solution extensively covers the point with rectangles whose capacity is larger than the demand of the point. We reduce the problem of covering the heavy points by rectangles with higher capacity to the geometric cover problem R3U defined below. We show that the instances of R3U that we obtain have boundaries with low union complexity. In particular, the boundary of the union of any k objects has a complexity of $O(k \log P)$. Using Varadarajan's quasi-uniform sampling technique [3] for approximating weighted set cover on geometric instances with low union complexity, one can obtain a covering that is an $O(\log \log P)$-approximation to fractional cover specified by the LP solution.

Definition of the R3U Problem: The input consists of a collection of $\mathcal{P}$ points in three dimensional space. Each point $p \in \mathcal{P}$ is specified by its coordinates (x_p, y_p, z_p). Further the input contains a collection $\mathcal{R}$ of axis-parallel right cuboids each of them abutting the xy and yz coordinate planes. That is, each right cuboid $r \in \mathcal{R}$ has the form $(0, x_r) \times (y_r^1, y_r^2) \times (0, z_r)$. In addition, each right cuboid $r \in \mathcal{R}$ has an associated positive integer weight w_r. The goal is to find a minimum weight subset $S \subset \mathcal{R}$ of cuboids such that each point $p \in \mathcal{P}$ is covered by at least one cuboid.

We reduce the problem of covering the light points to $\log P$ different instances, one

for each possible job size, of the weighted geometric multi-cover problem R2M defined below. We then show how to use the local ratio technique to obtain a solution for each instance of R2M that is $O(\log \log nP)$-approximate with the cost in the optimal LP solution for jobs of this size. Combining these solutions for various sizes implies a solution for covering all light points with cost $O(\log \log nP)$ times the LP cost.

Definition of the R2M Problem: The input consists of a collection of $\mathcal{P}$ points in two dimensional space, and for each point $p \in \mathcal{P}$ an associated positive integer demand d_p. Each point $p \in \mathcal{P}$ is specified by its coordinates (x_p, y_p). Further the input contains a collection $\mathcal{R}$ of axis-parallel rectangles, each of them abutting the y-axis. That is, each rectangle $r \in \mathcal{R}$ has the form $(0, x_r) \times (y_r^1, y_r^2)$. In addition, each rectangle $r \in \mathcal{R}$ has an associated positive integer weight w_r. The goal is to find a minimum weight subset $S \subset \mathcal{R}$ of rectangles, such that for each point $p \in \mathcal{P}$, the number of rectangles covering p is at least d_p.

References

[1] Deeparnab Chakrabarty, Elyot Grant and Jochen Könemann. On Column-Restricted and Priority Covering Integer Programs. *IPCO*, 2010.

[2] Nikhil Bansal and Kirk Pruhs. The Geometry of Scheduling. *FOCS*, 2010.

[3] K. R. Varadarajan. Weighted geometric set cover via quasi-uniform sampling. *STOC*, 2010.

Vertex Cover in Graphs with Locally Few Colors and Precedence Constrained Scheduling with Few Predecessors

Fabian Kuhn * Monaldo Mastrolilli (Speaker) †

Vertex Cover is one of the most studied problems in combinatorial optimization: Given a graph $G = (V, E)$ with weights w_i on the vertices, find a subset $V' \subseteq V$, minimizing the objective function $\sum_{i \in V'} w_i$, such that for each edge $(u, v) \in E$, at least one of u and v belongs to V'. Vertex cover cannot be approximated within a factor of 1.3606 [5], unless P=NP. Moreover, if the Unique Game Conjecture holds, Khot and Regev [13] show that vertex cover is hard to approximate within any constant factor better than 2. On the other side several simple 2-approximation algorithms are known. Hochbaum [11] uses the natural linear program relaxation and a threshold rounding approach to obtain better than 2 approximation algorithms when a k-coloring of the graph is given as input. This yields a $(2 - 2/k)$-approximation for the minimum weighted vertex cover problem. For graphs with degree bounded by d, this directly leads to a $(2 - 2/d)$-approximation. This basic approach has been considerably improved (for sufficiently large d) by Halperin [9]. The improvement is obtained by replacing the linear program relaxation with a stronger semidefinite program relaxation, and using a fundamental result by Karger, Motwani and Sudan [12]. The algorithm in [9] achieves a performance ratio of $2 - (1 - o(1))\frac{2 \ln \ln d}{\ln d}$, which improves the previously known [8] ratio of $2 - \frac{\ln d + O(1)}{d}$. Under the Unique Game Conjecture, Austrin, Khot and Safra [3] have recently proved that it is NP-hard to approximate vertex cover in bounded degree graphs to within a factor $2 - (1 + o(1))\frac{2 \ln \ln d}{\ln d}$. This exactly matches the algorithmic result of Halperin [9] up to the $o(1)$ term.

Problem $1|prec|\sum w_j C_j$ is a classical scheduling problem and it is defined as the problem of scheduling a set $N = \{1, \ldots, n\}$ of n jobs on a single machine, which can process at most one job at a time. Each job j has a processing time p_j and a weight w_j, where p_j and w_j are nonnegative integers. Jobs also have precedence constraints between them that are specified in the form of a *partially ordered set* (*poset*) $\mathbf{P} = (N, P)$, consisting of the set of jobs N and a partial order i.e. a reflexive, antisymmetric, and transitive binary relation P on N, where $(i, j) \in P$, whenever $i \neq j$, implies that job i must be completed before job j can be started. The goal is to find a non-preemptive schedule which minimizes $\sum_{j=1}^{n} w_j C_j$, where C_j is the time at which job j completes in the given schedule. Problem $1|prec|\sum w_j C_j$ is a classical and fundamental problem in scheduling theory with precedence constraints [14]. Its complexity certainly depends on the poset complexity: indeed, it can be efficiently solved when there are no precedence constraints or when the precedence constraints are not "very complicated", namely when

*fabian.kuhn@usi.ch. Faculty of Informatics, University of Lugano (USI), 6904 Lugano, Switzerland.

†monaldo@idsia.ch. Dalle Molle Institute for Artificial Intelligence (IDSIA), 6928 Manno, Switzerland.

the *dimension* of the poset (see [16]) is at most two [1,4]. More generally, the dimension of the input poset has been established to be an important parameter for the approximability of the problem [1, 2], with the lower the (fractional) dimension the better is the approximation ratio. Unfortunately, in the general case, recognizing the (fractional) dimension of a poset is hard even to approximate [10]. Another natural parameter of partial orders is given by the poset in- or out-degree [7], namely the job maximum number of predecessors or successors, respectively. One of the first NP-complete proofs [15] for $1|prec|\sum w_jC_j$ shows that the problem remains strongly NP-hard even if every job has at most two predecessors (or successors) in the poset. In [2], the authors present a $(2-2/\max\{\Delta,2\})$-approximation algorithm, where $\Delta-1$ is the minimum between the in- and the out-degree of the input poset. The latter gives a "good" approximation algorithm when the poset has a "small" Δ.

Overview of the Results In 1986 Erdős et. al. [6] defined the local chromatic number of a graph as the minimum number of colors that must appear within distance 1 of a vertex. For any fixed $\Delta \geq 2$, they presented graphs with arbitrarily large chromatic number that can be colored so that: (i) no vertex neighborhood contains more than Δ different colors, and (ii) adjacent vertices from two color classes form an induced subgraph that is complete and bipartite, i.e. a biclique.

We investigate the weighted vertex cover problem in graphs when a locally bounded coloring is given as input. This generalizes in a very natural vein the vertex cover problem in bounded degree graphs to a class of graphs with arbitrarily large chromatic number. Assuming the Unique Game Conjecture, we provide a tight characterization. More precisely, we prove that it is UG-hard to improve the approximation ratio of $2-2/(\Delta+1)$ if only condition (i), but not (ii), holds for the given coloring. A matching upper bound is also provided. Vice versa, when both the above two properties (i) and (ii) hold, we present a randomized approximation algorithm with performance ratio of $2-\Omega(1)\frac{\ln\ln\Delta}{\ln\Delta}$. This matches (up to the constant factor in the lower order term) the known inapproximability result [3] for the special case of bounded degree graphs. The provided approximation algorithm builds on the approximation algorithm for vertex cover in bounded degree graphs provided in [9], which first solves an SDP-relaxation of the problem to obtain a "vector coloring" that is then turned into a vertex cover via a randomized geometric rounding procedure by using a fundamental result by Karger, Motwani and Sudan [12]. However, the rounding approach in [12] does not generalize to the class of graphs considered here (actually, the rounding procedure and analysis in [12] are strongly based on the assumption that the graph has "few", i.e. $O(n)$ edges). To prove the claimed result we provide a novel and more general rounding scheme.

Moreover, we show that when both the above two properties (i) and (ii) hold, the obtained result finds a natural application in the classical scheduling problem known as $1|prec|\sum w_jC_j$. In a series of recent papers [1, 2, 4] it was established that this scheduling problem is a special case of the minimum weighted vertex cover in graphs $G_{\mathbf{P}}$ of incomparable pairs defined in the dimension theory of partial orders. We show that $G_{\mathbf{P}}$ satisfies properties (i) and (ii) where $\Delta-1$ is the maximum number of predecessors (or successors) of each job in the poset $\mathbf{P}$ that defines the precedence constraints set.

References

[1] C. Ambühl and M. Mastrolilli. Single machine precedence constrained scheduling is a vertex cover problem. *Algorithmica*, 53(4):488–503, 2009.

[2] C. Ambühl, M. Mastrolilli, N. Mutsanas, and O. Svensson. Scheduling with precedence constraints of low fractional dimension. In *IPCO*, pages 130–144, 2007.

[3] P. Austrin, S. Khot, and M. Safra. Inapproximability of vertex cover and independent set in bounded degree graphs. In *IEEE Conference on Computational Complexity*, pages 74–80, 2009.

[4] J. R. Correa and A. S. Schulz. Single machine scheduling with precedence constraints. *Mathematics of Operations Research*, 30(4):1005–1021, 2005.

[5] I. Dinur and S. Safra. On the hardness of approximating minimum vertex cover. *Annals of Mathematics*, 162(1):439–485, 2005.

[6] P. Erdös, Z. Füredi, A. Hajnal, P. Komjáth, V. Rödl, and Á. Seress. Coloring graphs with locally few colors. *Discrete Mathematics*, 59(1-2):21–34, 1986.

[7] Felsner and Trotter. On the fractional dimension of partially ordered sets. *DMATH: Discrete Mathematics*, 136:101–117, 1994.

[8] M. M. Halldórsson and J. Radhakrishnan. Greed is good: Approximating independent sets in sparse and bounded-degree graphs. *Algorithmica*, 18(1):145–163, 1997.

[9] E. Halperin. Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs. *SIAM J. Comput.*, 31(5):1608–1623, 2002.

[10] R. Hegde and K. Jain. The hardness of approximating poset dimension. *Electronic Notes in Discrete Mathematics*, 29:435–443, 2007.

[11] D. S. Hochbaum. Efficient bounds for the stable set, vertex cover and set packing problems. *Discrete Applied Mathematics*, 6:243–254, 1983.

[12] D. R. Karger, R. Motwani, and M. Sudan. Approximate graph coloring by semidefinite programming. *J. ACM*, 45(2):246–265, 1998.

[13] S. Khot and O. Regev. Vertex cover might be hard to approximate to within 2-epsilon. *J. Comput. Syst. Sci.*, 74(3):335–349, 2008.

[14] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. Sequencing and scheduling: Algorithms and complexity. In S. C. Graves, A. H. G. Rinnooy Kan, and P. Zipkin, editors, *Handbooks in Operations Research and Management Science*, volume 4, pages 445–552. North-Holland, 1993.

[15] J. K. Lenstra and A. H. G. Rinnooy Kan. The complexity of scheduling under precedence constraints. *Operations Research*, 26:22–35, 1978.

[16] W. T. Trotter. *Combinatorics and Partially Ordered Sets: Dimension Theory*. Johns Hopkins Series in the Mathematical Sciences. The Johns Hopkins University Press, 1992.

Timing Predictability for Resource Sharing Multicore Systems - Challenges and Open Problems

Andreas Schranzhofer Jian-Jia Chen (Speaker) * Lothar Thiele †

1 Introduction

In modern computer systems, multiprocessor systems on chip (MPSoCs) and multicore platforms have been widely applied. These commercial off-the-shelf (COTS) products possess significantly increased computational performance in the average case and reduced cost and time-to-market properties that are increasingly appealing for applications in the avionic and automotive industry. Demand for computational resources is growing with the ascend of concepts such as fly-by-wire and assisted driving. However, such concepts rely on guarantees on the worst-case response time (WCRT). In addition, MPSoCs employ shared resources, such as memory or I/O peripherals, for performing communication and data exchange between cores. Multiple processing elements competing for access to a shared resource yields contention and as a result, significantly increased WCRT.

MPSoCs with shared resources have been studied by Schliecker et al. [3] using event models and an iterative approach to estimate the worst-case execution time (WCET). Pellizzoni et al. [1] have studied systems with shared resources and proposed methods to analyze the worst-case delay. In [1], we propose sequentially executing superblocks to constitute tasks. Superblocks are specified by their upper bound on access requests to a shared memory and their maximum required computation time. Superblocks are constructed by execution blocks either by static analysis of a program or by the designers efforts to manually arrange memory accesses at the beginning and the end of the execution blocks. Different arbitration policies on a shared memory (FP, FCFS, RR) are analyzed and the worst-case delay suffered by a task due to the interference on the shared memory is computed, using a dynamic programming approach.

Other works focus on arbitration policies, that eliminate interference. Rosen et al. [2] use Time Division Multiple Access (TDMA) for accessing a bus. A task is modeled as dedicated communication phase at its beginning and end, and an execution phase in between. Static analysis is used to derive all feasible execution traces of a task and to derive the WCET thereof. We propose a task model in [4], where the number of accesses to a shared resource is only known as an upper bound for different time windows. The analysis of the WCRT is based on maximizing the cumulative time a task waits for time slots. Besides the task model with dedicated phases for communication and

*Department of Informatics, Karlsruhe Institute of Technology (KIT), Germany Email: jian-jia.chen@kit.edu

†Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Switzerland Email: {schranzhofer,thiele}@tik.ee.ethz.ch

execution, our algorithm in [4] is also applicable to tasks modeled as a single general phase, without dedicated communication. Following that, we introduce different models of accessing shared resources and models of execution in [5]. As a conclusion, separation of communication and execution is crucial for designing multicore resource sharing systems, but excessive time-triggering increases their WCRT. Moreover, for systems with an adaptive arbiter on the shared resource, i.e., an arbiter that is composed of a static and a dynamic arbitration segment, we show how to derive a WCRT bound by using dynamic programming in [6].

2 Challenges and Open Problems

The problem of deriving the WCRT is hard and as shown in our previous work, so far tight results can only be derived for very restrictive task models, see [4–6]. In this section, we would like to provide the challenges we have faced for analyzing the worst-case response time and some possible solutions. Deriving the worst-case trace is not trivial when a pending access to the shared resource blocks the execution on the processing element. Consider a multicore platform with multiple tasks competing for a shared resource, and no synchronization among those tasks. Furthermore, consider another task, whose WCRT shall be computed. Then it is not clear, which interference pattern by other tasks results in the WCRT of the task under analysis. Even though the results we have derived in [4–6] are safe for bounding the worst-case response time, the tightness of the analysis is still questionable.

In the case of static arbitration, when interference is eliminated, this problem is avoided and the worst-case response time is derived by maximizing the stall time in between time slots. In the other case, when arbitration on the shared resource follows a dynamic scheme, deriving the worst-case interference is a major issue. Our work in [1] and the work by Schliecker et al. in [3], approximate the interference as the sum of possible resource accesses by other tasks in a particular time window. This is pessimistic, since these tasks can also be interfered with and therefore the actual interference might be much smaller. So far, a tight algorithm would require to produce all possible interfering traces, which is of exponential complexity. Limiting the number of interfering traces is one potential direction of research to compute tight WCRTs for systems with shared resources and dynamic arbitration policies thereon. This is achieved by assuming not only an upper bound on resource accesses for a task or superblock, but also a lower bound. This way, the number of feasible traces is reduced significantly, since the freedom to postpone resource accesses is restricted. A major issue for this approach is the derivation of an aggregated interference for the task under analysis, which has to be derived from the access patterns of all the other tasks.

Another line of research follows a similar approach as our work in [4, 5], namely the elimination of interference by isolating the accesses by multiple tasks to the shared resource from each other. This can be achieved by using servers to arbitrate the shared resource. Constant Bandwidth Server (CBS) and Total Bandwidth Server (TBS) can be reduced to a TDMA arbitration, in case interference can be assumed to be present at all times. Otherwise, these arbitration policies result in lower WCRTs. Besides applying these server arbitration policies, it is unclear how an optimal server would look like. How a server should be constructed, such that a minimized (optimal) WCRT can be achieved, is an open question. Another issues is the combined optimality criterion for

multiple servers, serving multiple tasks. Satisfying real-time guarantees is a condition that has to be satisfied for feasibility, and therefore cannot serve as optimality criterion. This criterion has to consider properties such as WCRT or a combination of the WCRTs for multiple tasks in relation to their respective deadline.

All these approaches consider a given task to processing element allocation. However, taking interference between tasks into consideration during the mapping process, opens another possible direction of research for designing predictable resource sharing multicore systems. As an example, consider a set of tasks that communicate over a shared memory. Distributing these tasks over a large amount of processing elements results in a lot of communication requests over the shared memory. Concentrating these tasks on as few processing elements as possible, on the other hand, reduces the amount of communication over the shared resource. Conclusively the amount of potential interferences is reduced as well.

3 Conclusion

Contention on shared resources results in significantly increased worst-case response time (WCRT) guarantees, which are required for automotive and avionic applications. We show in our work that for general models of execution and general arbitration policies, this problem is hard and upper bounds on the WCRT are not tight. Using more restrictive models of execution, along with a static arbitration on the shared resource, we derive an efficient algorithm to compute a tight bound on the WCRT. We propose new analysis approaches to determine the worst-case interference by enriching the description of tasks with a lower bound on resource access requests. Then we propose the usage of servers to eliminate interference among tasks. Considering interference in the design process, when deciding on the mapping, is a viable way to limit interference and to eventually obtain a predictable resource sharing MPSoC.

References

[1] R. Pellizzoni, A. Schranzhofer, J.-J. Chen, M. Caccamo, and L. Thiele. Worst case delay analysis for memory interference in multicore systems. In *Proceedings of DATE*, pages 741–750, 2010.

[2] J. Rosen, A. Andrei, P. Eles, and Z. Peng. Bus access optimization for predictable implementation of real-time applications on multiprocessor systems-on-chip. In *RTSS*, pages 49–60, 2007.

[3] S. Schliecker, M. Negrean, and R. Ernst. Bounding the shared resource load for the performance analysis of multiprocessor systems. In *DATE*, pages 759–764, 2010.

[4] A. Schranzhofer, J.-J. Chen, and L. Thiele. Timing analysis for tdma arbitration in resource sharing systems. In *Proceedings of RTAS*, pages 215–224, 2010.

[5] A. Schranzhofer, R. Pellizzoni, J.-J. Chen, L. Thiele, and M. Caccamo. Worst-case response time analysis of resource access models in multi-core systems. In *Proceedings of DAC*, 2010.

[6] A. Schranzhofer, R. Pellizzoni, J.-J. Chen, L. Thiele, and M. Caccamo. Timing analysis for resource access interference on adaptive resource arbiters. In *Proceedings of RTAS*, April 2011.

Machine scheduling by column-and-row generation on the time-indexed formulation

Ruslan Sadykov (Speaker) * François Vanderbeck †

1 Introduction

We consider the minimum cost scheduling of jobs $j \in J = \{1, \ldots, n\}$ with processing times $p_j \in I\!N$, on a single machine, a single job at a time, with no preemption. Let $T \geq \sum_j p_j$ be the length of the planning horizon. Period t represents time interval $[t-1, t)$ for $t = 1, \ldots T$. We assume a generic cost function: the inputs allow us to compute values c_{jt} representing the cost of processing job j starts at the ouset of period t.

One of the approaches to solve this problem uses the following time-indexed Integer Programming formulation. Let a binary variable z_{jt}, $j \in J$, $t = 1, \ldots, T$, equals to one if job j starts at the outset of period t. Let also job 0 with processing time 1 model the machine idle time. The computationally most efficient such time-indexed formulation is the so-called "flow" reformulation [1]:

$$[R] \equiv \min \Big\{ \sum_{jt} c_{jt}\, z_{jt} : \quad \sum_{t=1}^{T-p_j+1} z_{jt} = 1 \; \forall j \in J, \quad \sum_{j=0}^{n} z_{j1} = 1, \\ \sum_{j=0}^{n} (z_{jt} - z_{j,t-p_j}) = 0 \; \forall t \in \{2, \ldots, T\}, \quad z_{jt} \in \{0,1\} \; \forall j, t \Big\}$$

where the first group of constraints models the assignment of each job to a time period, while the others enforce the "one-job-at-a-time" restriction. The formulation has nT variables and $(n+T)$ constraints (note that T is pseudo-polynomial in the input size).

The linear programming (LP) relaxation of this formulation is known to produce very tight lower bounds. However, its size becomes impractical for instances with a large time horizon. One of the methods to overcome this difficulty is to apply the column generation approach based on the totally uni-modular subsystem formed by the one-job-at-a-time constraints, as done by [5]. This method consists in defining a reformulation:

$$[M] \equiv \min \Big\{ \sum_{g \in G} c^g\, \lambda_g : \sum_{g \in G} \sum_{t=1}^{T-p_j+1} z^g_{jt}\, \lambda_g = 1 \; \forall j \in J, \sum_{g \in G} \lambda_g = 1, \; \lambda_g \in \{0,1\} \; \forall g \in G \Big\}$$

where G is the set of "pseudo-schedules" (in which each job does not necessarily appears exactly once), vector z^g, scalar c^g define the associated solution and cost for a solution

*Ruslan Sadykov@inria.fr. INRIA Bordeaux — Sud-Ouest, 351 cours de la Libération, 33405 Talence, France

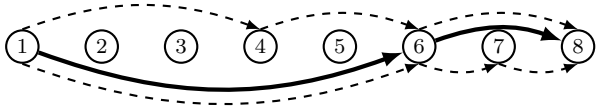
†fv@math.u-bordeaux1.fr. Université Bordeaux I, 351 cours de la Libération, 33405 Talence, France

$g \in G$. The LP relaxation of [M] is solved by column generation. The pricing subproblem can be modeled as the search for a shortest path: $z^* = \arg\min\{\sum_{jt}(c_{jt} - \pi_j)\, z_{jt} : \sum_{j=0}^{n} z_{j1} = 1,\ \sum_{j=0}^{n}(z_{jt} - z_{j,t-p_j}) = 0\ \forall t > 1, \quad z_{jt} \in \{0,1\}\ \forall j,t\}$, where π_j is a dual solution to the linear relaxation of [M]. Thus, each pseudo-schedule defines a path in a graph whose nodes represent periods and where a job j is represented by arcs $(t, t+p_j)$, and idle times by arcs $(t, t+1)$.

2 Column-and-row generation approach

An alternative approach is a column-and-row generation for the LP relaxation of [R]. The method is reviewed in [4]. Variables z are generated dynamically, not one at the time, but by lots. To do it, we solve the above pricing subproblem (where π is the dual solution of the assignment constraints of [R]), and add to [R] the components of its solution z^* with a negative reduced cost in the LP relaxation of [R] along with the flow conservation constraints that are binding for that solution. The components of z^* with a non-negative reduced cost are stored in the column pool and added to [R] on one of the subsequent iterations if their reduced cost becomes negative. In [4], we indeed showed that either the current LP value of [R] is optimal, or some components of z^* must have a negative reduced cost in the LP relaxation of [R]. Therefore, this column-and-row generation approach solves the LP relaxation of [R] after a finite number of iterations.

Compare to a standard column generation approach for [M], the interest of this alternative approach is to allow for the recombination of previously generated pricing problem solutions, and thus to accelerate the convergence. To illustrate what is meant by recombination, we picture below two pseudo-schedules (dashed) and a new pricing problem solution z^* (bold) that can be obtained by recombining these two without the need to explictly generate it through pricing.



Note that such recombination is not feasible in [M] where the only feasible solutions are those defined by the convex combinations of previously generated columns. Such column-and-row generation approach applies to any problem admitting a decomposition in which the subproblem is solved by the shortest path problem or, more generally, by the min-cost flow problem or by dynamic programming.

3 Computational results

We performed a computational comparison of three approaches on the same computer: solving the LP relaxation of [R] directly using *Cplex 12.1*; solving the LP relaxation of [M] by standard column generation; and solving the LP relaxation of [R] by column-and-row generation. Column[-and-row] generation algorithms were implemented using BaPCod — a generic Branch-and-Price code developed by the INRIA RealOpt team in Bordeaux. A problem-specific implementation is likely to produce better results.

The three approaches were tested on instances with 25, 50, and 100 jobs and the total weighted tardiness objective function. The test instances were generated using

the procedure from [3] which is the most used in the literature. The objective is to minimize the total weighted tardiness. Processing times of jobs are uniformly distributed in interval $[1, 100]$. For each n, we generated 25 instances, each for different pairs of two parameters, varying the relative range of due dates and the average tardiness factor.

The results are presented in Table 1. "*cpu*" is the solution time (in seconds), "*it*" is the number of iterations in the column[-and-row] generation procedure, "*sp*" is the number of calls to the pricing subproblem solver, and "*%z*"is the percentage of z variables generated in the column-and-row generation approach (from the total number of z variables in [R]). The column-and-row generation approach outperforms the other two. Moreover, its advantage increases with the increase of n.

	Cplex for [R]	Column generation for [M]			Column-and-row generation for [R]			
n	*cpu*	*it*	*sp*	*cpu*	*it*	*sp*	*%z*	*cpu*
25	11.2	343	343	2.1	208	69	5.8%	1.5
50	153.0	1270	1270	39.4	339	106	4.5%	16.9
100	2233.0	8784	8784	2891.5	466	139	4.5%	169.1

Table 1: Computational results

4 Perspectives

Our further research agenda is (*i*) to combine the column-and-row generation with an enumeration algorithm to solve the scheduling problem to optimality; (*ii*) to check whether the combination of the column-and-row generation approach with a cutting plane method is computationally advantageous; and (*iii*) to speed-up the column-and-row generation using standard stabilization techniques for column generation and variable fixing based on reduced cost (as in [2]). We also plan to experiment this column-and-row generation approach on the arc-time indexed formulation in which each binary variable z_{ijt} determines whether job i immediately precedes job j at time moment t. LP relaxation of this formulation generates even better lower bounds [2].

References

[1] Yunpeng Pan and Leyuan Shi. On the equivalence of the max-min transportation lower bound and the time-indexed lower bound for single-machine scheduling problems. *Mathematical Programming*, 110:543–559, 2007.

[2] Artur Pessoa, Eduardo Uchoa, Marcus Poggi de Aragão, and Rosiane Rodrigues. Exact algorithm over an arc-time-indexed formulation for parallel machine scheduling problems. *Mathematical Programming Computation*, 2:259–290, 2010.

[3] Chris N. Potts and Luk N. Van Wassenhove. A Branch and Bound Algorithm for the Total Weighted Tardiness Problem. *Operations Research*, 33(2):363–377, 1985.

[4] Ruslan Sadykov and François Vanderbeck. Column generation for extended formulations. Working paper, http://hal.inria.fr/inria-00539870/en/, 2011.

[5] J.M. van den Akker, C.A.J. Hurkens, and M.W.P. Savelsbergh. Time-indexed formulations for machine scheduling problems: Column generation. *INFORMS Journal on Computing*, 12(2):111–124, 2000.

Online Scheduling of Linear Deteriorating Jobs on Parallel Machines

Sheng Yu* Prudence W. H. Wong (Speaker)† Yinfeng Xu‡

1 Introduction

Scheduling a set of jobs with fixed processing times is a classical problem [13]. Yet, there are numerous situations that the processing time increases (deteriorates) as the start time increases, e.g., maintenance or cleaning schedule, fire fighting, steel production and financial management [10, 11]. The study of minimizing makespan of deteriorating jobs first focused on a single machine, with linear deterioration [2] and non-linear deterioration [7]. Since then, the problem has attracted a lot of attention (see e.g., [1, 4, 6]).

A job J_j with *linear deterioration* has a processing time $p_j = a_j + b_j s_j$, where $a_j \geq 0$ is the "normal" processing time, $b_j > 0$ is the deteriorating rate, and s_j is the start time. Linear deterioration is said to be *simple* if $a_j = 0$, i.e., $p_j = b_j s_j$. In this case, to avoid trivial solution, it is natural to assume that the start time of the first job is $t_0 > 0$ since a start time of zero implies that the processing time of all jobs is zero.

It is observed in [7] that, when all jobs are available at time 0, $1 \,|\, p_j = a_j + b_j s_j \,|\, C_{\max}$ can be solved optimally by scheduling in ascending order of a_j/b_j. For $1 \,|\, p_j = b_j s_j \,|\, C_{\max}$, the makespan is independent of the job processing order [11]. The problem becomes NP-hard and strongly NP-hard on two machines and m machines, respectively (c.f. the complexity when $p_j = a_j$ [5]). The problem $P2 \,|\, p_j = b_j s_j \,|\, C_{\max}$ is NP-hard [9, 12]. FPTAS have been proposed for $Pm \,|\, p_j = a_j + b_j s_j \,|\, C_{\max}$ [8] and $Pm \,|\, p_j = b_j s_j \,|\, C_{\max}$ [14].

Online algorithms have only been studied for $Pm \,|\, p_j = b_j s_j, \text{online-list} \,|\, C_{\max}$ [3], in which a job has to be scheduled before the next job can be seen. They showed that the competitive ratio of List Scheduling (LS) is $(1 + b_{\max})^{\frac{m-1}{m}}$ and this is the best possible.

Our contributions. We first consider $Pm \,|\, p_j = a_j + bs_j, \text{online-list} \,|\, C_{\max}$ and show that Round Robin (RR) is α-competitive and no on-line algorithm is better than $(2 - \frac{1}{\alpha})$-competitive, where $\alpha = \frac{a_{\max}}{a_{\min}}$. Furthermore, when jobs have release times, we study a class of "reasonable" algorithms including LS that do not idle when there are available jobs. For $Pm \,|\, p_j = b_j s_j, r_j, \text{online} \,|\, C_{\max}$, we show that these algorithms achieve a competitive ratio of $(1 + b_{\max})^{2(1-\frac{1}{m})}$ and no online algorithm is better than $(1 + b_{\max})^{\frac{1}{2}}$. We also show that the competitive ratio of Round Robin (RR) can be unbounded.

Notations and problem definition. We are to schedule non-preemptively a set of jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ onto machines $M_1, M_2, \ldots, M_m$. For J_j, we denote by r_j and p_j the release and processing time, respectively,. where $p_j = a_j + b_j s_j$ and s_j is the start time of J_j. Let $b_{\max} = \max_{1 \leq j \leq n}\{b_j\}$, $a_{\max} = \max_{1 \leq j \leq n}\{a_j\}$, $a_{\min} = \min_{1 \leq j \leq n}\{a_j\}$,

*School of Management, Xi'an Jiaotong University, a.sheng@stu.xjtu.edu.cn
†Department of Computer Science, University of Liverpool, pwong@liverpool.ac.uk
‡School of Management, Xi'an Jiaotong University, yfxu@mail.xjtu.edu.cn

and $\alpha = \frac{a_{\max}}{a_{\min}}$. Consider a schedule S. For $1 \le j \le n$, the completion time of job J_j in S is denoted by $c_j(S)$. For any $1 \le k \le m$, the number of jobs and the set of jobs dispatched to M_k by S is denoted by $n^k(S)$ and $\mathcal{J}^k(S)$. The makespan of machine M_k and schedule S are denoted by $C^k_{\max}(S)$ and $C_{\max}(S)$, respectively. The objective of the problem is to minimize the makespan of the schedule produced.

2 Varying deteriorating rates $p_j = b_j s_j$

We consider the problem $Pm \,|\, p_j = b_j s_j, r_j, \text{online} \,|\, C_{\max}$, i.e., $a_j = 0$. In this setting, for any job J_j, $c_j = s_j + b_j s_j = s_j(1 + b_j)$. We first show some lower bounds (Theorem 1) (note that (ii) still holds even under the condition without release times). Then we study the performance of "reasonable" online algorithms (RA) that do not idle when there are available jobs. LS is reasonable but RR is not. We observe that if a RA schedule does not idle at all, its makespan is bounded (Lemma 2) and then we extend the idea and show that RA is $(1 + b_{\max})^{2(1-\frac{1}{m})}$-competitive (Theorem 3).

Theorem 1. *For* $Pm \,|\, p_j = b_j s_j, r_j, \text{online} \,|\, C_{\max}$, (i) *no deterministic online algorithm is better than* $(1 + b_{\max})^{\frac{1}{2}}$*-competitive;* (ii) *the competitive ratio of* RR *is unbounded.*

Lemma 2. *If in the* RA *schedule, every machine is not idle at all, then* $\frac{C_{\max}(\text{RA})}{C_{\max}(\text{OPT})} \le (1 + b_{\max})^{1-\frac{1}{m}}$.

Proof. (Sketch.) For simplicity, we assume the first m jobs have $r_j = 1$. Let $\mathcal{J}^k(\text{OPT})$ be the set of jobs dispatched to M_k by the optimal schedule OPT. Then, $C^k_{\max}(\text{OPT}) \ge \prod_{j:J_j \in \mathcal{J}^k(\text{OPT})}(1 + b_j)$, and thus, $C_{\max}(\text{OPT}) \ge (\prod_{j:J_j \in \mathcal{J}}(1 + b_j))^{\frac{1}{m}}$. Let ℓ be the index of a job with completion time equals to $C_{\max}(\text{RA})$ and p be the machine M_p to which RA dispatches J_ℓ. As there is no idle time in RA schedule $s_\ell(\text{RA}) \le (\prod_{j:J_j \in \mathcal{J}-\{J_\ell\}}(1+b_j))^{\frac{1}{m}}$. As a result, $C_{\max}(\text{RA}) = s_\ell(\text{RA})(1 + b_\ell) \le C_{\max}(\text{OPT})(1 + b_{\max})^{1-\frac{1}{m}}$. □

Theorem 3. *For* $Pm \,|\, p_j = b_j s_j, r_j, \text{online} \,|\, C_{\max}$, RA *is* $(1 + b_{\max})^{2(1-\frac{1}{m})}$*-competitive.*

3 Fixed deteriorating rate $p_j = a_j + b\ s_j$

In the online-list model, when a job is given, the online algorithm has to dispatch the job to a machine and specify the period of time to process the job before the next job is given. Suppose $\mathcal{J}^k(S) = \{J_{k,1}, J_{k,2}, \cdots, J_{k,n^k}\}$. The completion time of the job $J_{k,j}$ is $c_{k,j} = \sum_{1 \le i \le j} a_{k,i}(1 + b)^{j-i}$. Therefore, on a particular machine, the optimal schedule is to schedule jobs in increasing order of normal processing time a_j. We give some lower bounds (Theorem 4) and an upper bound of RR (Theorem 5).

Theorem 4. *Consider* $Pm \,|\, p_j = a_j + bs_j, \text{online-list} \,|\, C_{\max}$. (i) *No online algorithm is no better than* $(2-\frac{1}{\alpha})$*-competitive. (ii) Both* LS *and* RR *are no better than* α*-competitive.*

Proof. (Sketch.) (i) The adversary releases m jobs with the same normal processing time a. If an algorithm dispatches any two of these jobs to the same machine, the adversary stops releasing jobs. Otherwise, the adversary releases the $(m + 1)$-th job with normal processing time being $a(2 + b)$. In either case, the ratio is no better than $(2 - \frac{1}{\alpha})$.

(ii) We release $2qm$ jobs for some positive integer q. The job list contains qm jobs with $a_{\max}$ followed by those qm jobs with $a_{\min}$. Both LS and RR assign for each machine q jobs with $a_{\max}$ followed by q jobs with $a_{\min}$ while the optimal algorithm schedules the jobs with $a_{\min}$ first. The ratio is arbitrarily close to α by setting q as a large integer. □

Theorem 5. RR *is at most* α*-competitive for* $Pm \,|\, p_j = a_j + bs_j, \text{online-list} \,|\, C_{\max}$.

Proof. (Sketch.) On a single machine, no matter what order the jobs are scheduled, the makespan is no more than α times the optimal as long as the machine is not idle. We further observe that in any schedule, the maximum number of jobs on a machine is at least $\lceil \frac{n}{m} \rceil$ while in RR, the maximum number is at most $\lceil \frac{n}{m} \rceil$. Then we can show that RR is α-competitive. □

References

[1] B. Alidaee and N. K. Womer. Scheduling with time dependent processing times: Review and extensions. *Journal of the Operational Research Society*, 50(7):711–720, 1999.

[2] S. Browne and U. Yechiali. Scheduling deteriorating jobs on a single processor. *Operations Research*, 38(3):495–498, 1990.

[3] M. B. Cheng and S. J. Sun. A heuristic MBLS algorithm for the two semionline parallel machine scheduling problems with deterioration jobs. *Journal of Shanghai University*, 11(5):451–456, 2007.

[4] T. C. E. Cheng, Q. Ding, and B. M. T. Lin. A concise survey of scheduling with time-dependent processing times. *European Journal of Operational Research*, 152:1–13, 2004.

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

[6] S. Gawiejnowicz. *Time-Dependent Scheduling*. Springer-Verlag, Berlin, 2008.

[7] J. N. D. Gupta and S. K. Gupta. Single facility scheduling with nonlinear processing times. *Computers and Industrial Engineering*, 14(4):387–393, 1988.

[8] L. Y. Kang and C. T. Ng. A note on a fully polynomial-time approximation scheme for parallel-machine scheduling with deteriorating jobs. *International Journal of Production Economics*, 109:108–184, 2007.

[9] A. Kononov. Scheduling problems with linear increasing processing times. In e. a. Zimmermann U, editor, *Operations Research Proceedings 1996. Selected Papers of the Symposium on Operations Research (SOR 96)*, pages 208–212, Berlin, 1997. Springer.

[10] A. S. Kunnathur and S. K. Gupta. Minimizing the makespan with late start penalties added to processing times in a single facility scheduling problem. *European Journal of Operation Research*, 47(1):56–64, 1990.

[11] G. Mosheiov. Scheduling jobs under simple linear deterioration. *Computers and Operations Research*, 21(6):653–659, 1994.

[12] G. Mosheiov. Multi-machine scheduling with linear deterioration. *INFOR: Information Systems and Operational Research*, 36(4):205–214, 1998.

[13] M. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Prentice-Hall, Upper Saddle River, 2002.

[14] C. R. Ren and L. Y. Kang. An approximation algorithm for parallel machine scheduling with simple linear deterioration. *Journal of Shanghai University*, 11(4):351–354, 2007.

A Statistical Approach for Taxi Time Estimation at London Heathrow Airport

Jason A. D. Atkin * Edmund K. Burke * Stefan Ravizza (Speaker) *

1 Introduction and Problem Description

With a predicted year on year increase in the number of flights [5], increasing attention will need to be paid to the environmental effects of air transportation, in addition to an increasing focus upon maintaining and improving on-time performance at airports [3]. These challenges will face both airlines and airports in the near future and improved taxi time predictions (which is the aim of the model discussed here) will be an important tool for handling them.

The purpose of the ground movement problem is to guide aircraft around the surface of the airport to arrive at their destinations in a timely (and potentially more environmentally friendly) manner. From an optimisation point of view, the ground movement problem can be considered to be one of the most important airside operations at an airport [2], since it links together several other important problems, such as runway sequencing (for arrivals and/or departures) and gate assignment. The aims of this combined routing and scheduling problem are usually to reduce the overall taxi time, reduce the fuel burn, arrive at the destination by a target time, and/or to absorb any necessary waiting time or delays in preferred positions (e.g. before the engines are started). It has to guarantee conflict-free routes for each aircraft throughout the movement and, therefore, usually has to coordinate the movement of multiple aircraft around the surface at once. A comprehensive review of the problem can be found in [2].

In previous research, it has been difficult to quantify the benefits of new ground movement approaches compared with the status quo at airports, since it can be hard to distinguish between the effects of the variability in taxi times and the benefits of the improved routing and sequencing. Research is badly needed to understand and quantify the variability in taxi times, and moreover, to develop models which accurately predict taxi times from more measurable but influential factors. In particular, if predicted taxi times are to be used in a decision support tool for the ground movement problem, which itself incorporates the effects of re-routing and queuing delays, the predicted taxi times must not already include the effects of these. The production of a model to quantify and eliminate these delays has not, however, been simple.

London Heathrow Airport is one of the busiest international airports in the world, despite the fact that it operates with only two runways and (for noise control reasons) is restricted to using only a single runway at a time for departures. There has been

*jaa@cs.nott.ac.uk, ekb@cs.nott.ac.uk, smr@cs.nott.ac.uk. School of Computer Science, University of Nottingham, Jubilee Campus, Nottingham, NG8 1BB, UK.

considerable research and innovation of late into automated support for the various operations at the airport, in terms of both information sharing and advisory tools.

The aim of this research is to consider the variability in taxi times, both including and excluding the runway queue times, to determine the degree to which they can be predicted and to develop functions to predict them, where possible, by utilising information which will be available in advance.

2 Ground Movement Model and Statistical Analysis

Our research consists of an innovative combination of a ground movement model and a statistical multiple linear regression approach, to improve the predictability of taxi times. Advance predictions for departure taxi times have long been known to be useful and Idris et al. [4] performed a statistical analysis of departing aircraft at a North American airport. More recent research by Atkin et al. [1] used the model which we will present to show the benefits from considering both departures and arrivals at the same time, due to their interactions, and of including more accurate models of the airport layout and current mode of operations. The work analysed the effects of a number of potential taxi-time-influencing factors at two European hub airports: Stockholm-Arlanda Airport in Sweden and Zurich Airport in Switzerland. We now extend this research to consider the applicability to London Heathrow, where departure queue delays are relatively unpredictable for individual aircraft. We compare and contrast both the generated models and the accuracy of the predictions at the different airports.

We will discuss the model which has been developed for this work, explaining both the ground movement model which has been used and the statistical analysis which uses it. We will then present and discuss the potential influencing factors for taxi speeds and the importance which was determined for each, utilising real recorded data from London Heathrow Airport, supplied by NATS Ltd. The goal has been to present a model which is as practical (requiring the least information) and easy to interpret (so that the effects of influencing factors can be understood and validated by the problem domain experts) as possible, while maintaining a high level of accuracy.

We will not only discuss and highlight the significant factors and the way in which they have been identified, but will also consider the ways in which the model has been verified, such as the results from leave-one-out cross validation, which shows not only the quality of the model as a whole, but also the variability in predicting new observations.

3 Applicability and Importance

Advance taxi time predictions are important for a number of reasons. Better taxi time predictions can be used to improve take-off time predictions (by considering the push back times plus taxi times) or on-stand time predictions (for arrivals, where the landing time has already been predicted), both of which are of increasing importance at airports, [3]. They can also be used to enable engine start-up to be delayed (moving waiting time back from the runway queues, with engines running, to the stands, before they are started) to reduce fuel burn and pollution.

Moreover, the developed model links the ground movement paths which are taken, to the ground movement speeds. This enables the results to be fed into any ground movement decision support tool to improve the reliability of the taxi speed/time predictions.

Such decision support tools, which aim to optimise the ground movement problem, need taxi time predictions for isolated aircraft (since the algorithms usually explicitly consider the interactions of aircraft, introducing delays or re-routing as appropriate) along specific routes, and sufficient examples are not usually directly available in historic data. The utilised statistical approach provides the opportunity to disregard the effects of other aircraft on the airport surface, meeting this objective.

Of course, improved predictions mean that less slack will be needed to allow for taxi time uncertainty, allowing more efficient movement prediction or planning.

Acknowledgements This research was funded by EPSRC (The Engineering and Physical Sciences Research Council). The authors wish to thank NATS Ltd., for providing the real data about aircraft movement which made it possible to consider Heathrow in this research.

References

[1] J.A.D. Atkin, E.K. Burke, M.H. Maathuis, and S. Ravizza. *A combined statistical approach and ground movement model for improving taxi time estimations at airports.* (Submitted).

[2] J.A.D. Atkin, E.K. Burke, and S. Ravizza (2010). *The airport ground movement problem: Past and current research and future directions.* Proceedings of the 4th International Conference on Research in Air Transportation (ICRAT), Budapest, Hungary.

[3] Eurocontrol (2010). *Airport CDM Implementation Manual.* Technical Report, Version 3.1, Eurocontrol. Available at: http://www.euro-cdm.org/library _eurocontrol_implementation.php

[4] H.R. Idris, J.P. Clarke, R. Bhuva, and L. Kang (2002). *Queuing model for taxi-out time estimation.* Air Traffic Control Quarterly 10(1):1–22.

[5] SESAR (2006). *Milestone Deliverable D1 - Air Transport framework: The current Situation.* Technical Report, Edition 3, Eurocontrol. Available at: http://www.eurocontrol.int/sesar/public/standard_page/documentation.html

Parameter learning in online scheduling algorithms

Csanád Imreh (Speaker) * Tamás Németh †

1 Introduction

In on-line computation an algorithm must make its decisions based only on past events without secure information on future. Such algorithms are called on-line algorithms. On-line algorithms have many applications in different areas such as computer science, economics and operations research. Typically, the quality of an online algorithm is assessed via competitive analysis. An online minimization algorithm is C-competitive if the algorithm cost is never more than C times the optimal cost. Detailed information about competitive analysis can be found in [3].

On the other hand in some cases the algorithm which has the best competitive ratio does not work well in average cases or on real data sets. In [1] the problem of online scheduling algorithms to minimize makespan on identical machines is considered on real data sets, and it is shown, that some algorithms with good competitive ratio have a poor average performance.

In this talk we present a technique which might be used to improve the performance of online algorithms in the average case and/or real data. Several online algorithms can be considered as a member of a parametrized class of algorithms, with choosing the parameter which minimizes the competitive ratio. In these cases one can define an algorithm which works in phases and tries to learn the best value of the parameter. At each phase the algorithm investigates the known part of the input and determines the value of the parameter which gives the best result on this part. Then this value is used in the next phase. In average case such learning algorithms can often give better results.

2 The investigated models

2.1 Scheduling with rejection

The problem of scheduling with rejection is defined in [2]. In this model, it is possible to reject the jobs. The jobs are characterized by a processing time p_j and a penalty w_j. The goal is to minimize the makespan of the schedule for the accepted jobs plus the sum of the penalties of the rejected jobs. In the online case a 2.618-competitive algorithm is given for arbitrary number of machines. This algorithm is called Reject Total Penalty (RTP). One basic idea in scheduling with rejection is to compare the penalty and the load (processing time divided by the number of machines) of the job, and reject the job

*cimreh@inf.u-szeged.hu. Institute of Informatics University of Szeged, Árpád tér 2, H-6720 Szeged, Hungary.

†tnemeth@inf.u-szeged.hu. Institute of Informatics University of Szeged, Árpád tér 2, H-6720 Szeged, Hungary.

in the case when the penalty is smaller. This greedy algorithm can make a bad decision when the number of machines is large and this makes possible to accept large jobs with small loads. RTP handles these jobs more carefully as follows:

Algorithm RTP(α)

- 1. *Initialization.* Let $R := \emptyset$.
- 2. When job j arrives
  - (i) If $w_j \leq \frac{p_j}{m}$, then reject.
  - (ii) Let $r = \sum_{i \in R} w_i + w_j$. If $r \leq \alpha \cdot p_j$, then reject job j, and set $R = R \cup \{j\}$.
  - (iii) Otherwise, accept j and schedule it by LIST

Considering the competitive ratio the problem of scheduling with rejection on identical machines is completely solved in the general case, where no further restrictions are given on the jobs, algorithm RTP is an optimal online algorithm in the sense that it achieves the smallest possible competitive ratio. On the other hand algorithm RTP is a parametrized algorithm depending on the parameter α, its parameter learning extension is defined in [7]. The basic idea of this algorithm can be described as follows:

Algorithm PAROLE (PHASE i)

- At the beginning of phase i, use algorithm CHOOSE to find a new parameter α_i.
- Perform $RTP(\alpha_i)$ on the arrived part of the input. Change set R.
- Use $RTP(\alpha_i)$ for the jobs arriving during the phase.

Here CHOOSE is a subrutin which looks for the value α_i where the cost $RTP(\alpha_i)(I)$ is minimal for the known part of input.

2.2 Scheduling on two sets of identical machines

In [5] the scheduling problem on the following machine environment is investigated. In the problem there are two sets P and S containing k and m identical machines. Each job j has two different processing times p_j and s_j, depending on which set of machines to schedule it. This machine environment models the situation where we have two types of machines. For the jobs we have to choose the set where we want to schedule it, and then we have to schedule it on one of the machines of the set. Let C_P and C_S denote the makespans (maximal completion time) achieved at the sets of machines. Two different objective functions are considered .

If the goal is to minimize the maximum completion time, then the objective function is the maximum of the makespans ($\max C_P$,C_S). This problem is called Maximum Two Sets Scheduling. On the other hand if the sets are independent (the resources used by them are not common) and each set of machines has a cost proportional to the makespan, then we have to minimize the sum ($C_P + C_S$) of the makespans.

In [5] a generalization of the RTP algorithm is presented for the solution of the problem, here we present a generalization of the parameter learning algorithm from [7].

2.3 Other applications

We also note that the idea of learning parameter in online algorithms can be used in other areas as well. In the dynamic data acknowledgment problem (see [4]) the idea of parameter learning is very effective it decreases the average cost by 20 % (see [6]).

Acknowledgment

This study was partially supported by the TÁMOP-4.2.1/B-09/1/KONV-2010-0005 program of the Hungarian National Development Agency.

References

[1] S. Albers and B. Schröder, An Experimental Study of Online Scheduling Algorithms *ACM Journal of Experimental Algorithms*, article 3, 2002.

[2] Bartal, Y., Leonardi, S., Marchetti-Spaccamela, A., Sgall, J., Stougie, L.: Multiprocessor scheduling with rejection, *SIAM Journal on Discrete Mathematics*, **13**, 64–78, 2000.

[3] A. Borodin and R. El-Yaniv, *Online Computation and Competitive Analysis* Cambridge University Press, Cambridge 1998.

[4] D. R. Dooly, S. A. Goldman, and S. D. Scott, On-line analysis of the TCP acknowledgment delay problem, *Journal of ACM* **48(2)** 243–273, 2001.

[5] Cs. Imreh, Scheduling problems on two sets of identical machines, *Computing*, **70**, 277–294, 2003.

[6] Cs. Imreh and T. Németh, Parameter learning algorithm for the online data acknowledgment problem, *Optimization Methods and Software*, to appear 2011 DOI: 10.1080/10556788.2010.544313

[7] T. Németh and Cs. Imreh, Parameter learning online algorithm for multiprocessor scheduling with rejection, *Acta Cybernetica* **19(1)**, 125–133, 2009.

Explanation Algorithms for Cumulative Scheduling *

Stefan Heinz † Jens Schulz (Speaker) ‡

1 Introduction

In cumulative scheduling, conflict analysis is one of the key ingredients to solve these problems efficiently, see [2, 5, 7]. Thereby, the computational complexity of explanation algorithms that 'explain' infeasibilities or bound changes (see definition below) plays an important role. Their role is even more substantial when we are faced with a backtracking system where explanations need to be constructed on the fly.

In this talk we present complexity results for computing minimum-size explanations for the propagation algorithms time-tabling, edge-finding, and energetic reasoning. Due to the hardness results, we present optimal and heuristic approaches to deliver explanations. Our computational results show that minimum-size explanations drastically decrease the number of nodes in a branch-and-bound tree search and reduce the computation times by one-half.

2 Problem Description

In cumulative scheduling we are given a set of jobs that require a certain amount of resources. In our case, the resources are renewable with a constant capacity and each job is non-interruptible with a fixed processing time and demand request for one or several resources. A resource can be, for example, a group of workers with the same specialization, a set of machines, or entities like power supply. Additionally, each job has an earliest start time and a latest completion time. The goal is to find a feasible start time for each job such that the available capacities of the resources are respected at any point in time.

Cumulative scheduling problems have been tackled with techniques from constraint programming (CP), integer programming, or satisfiability testing (SAT). Hybrid approaches have been developed which combine methods from these areas. Currently, the best results are reported by a hybrid solver which uses CP and SAT techniques [7]. However, there are still instances with 60 jobs and four cumulative constraints published in the PSPLib [6] that resist to be solved to proven optimality.

*Supported by the DFG Research Center Matheon *Mathematics for key technologies* in Berlin.

†heinz@zib.de Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany

‡jschulz@math.tu-berlin.de Technische Universität Berlin, Institut für Mathematik, Straße des 17. Juni 136, 10623 Berlin, Germany

3 Conflict Analysis

Various exact approaches use a branch-and-bound approach to solve these $\mathcal{NP}$-hard problems. Among them are SAT solvers that make explicit use of infeasible subproblems. They perform non-chronological backtracking or derive *no-goods* to speed up the search. As can be seen in recent publications, this *conflict analysis* plays an important role to solve cumulative scheduling problems efficiently [2, 5, 7].

Conflict analysis takes place under the following circumstances. During branch-and-bound search the lower and upper bounds of variables are updated by various propagation algorithms or simply by branching decisions. A subproblem becomes infeasible, e.g., because the lower bound of the start time variable of some job can be updated to a value larger than the current upper bound, because the resource demand of all jobs for some interval is larger than the available capacity, or because a relaxation, such as the linear programming relaxation, becomes infeasible. Analyzing such infeasibilities means to *explain* them and learn from them. An *explanation* of an infeasibility or of a bound update is a set of lower and/or upper bounds of variables that, whenever they occur in that combination, lead to an infeasible state or to that bound update. During conflict analysis the goal is to learn further constraints to detect similar infeasible subproblems earlier. To this end, cuts in a so-called *conflict graph* are computed. This graph consists of the initial explanation, which states a reason for the infeasibility. During the process of the conflict analysis this graph is extended by explanations for bound changes which are part of previous bound explanations. In the end each variable bound in that graph is connected via edges to the elements of its explanation. For a detailed description of the conflict analysis we refer to [1].

When considering propagation algorithms in cumulative scheduling, these explanations are in general not unique and bear a huge potential to optimize the solvers behavior.

The task of an *explanation algorithm* is to *explain* bound changes or infeasible states, i.e., state a set of variable bounds that lead to the deduction, under certain objective criteria. Minimum-size explanations are such a well-suited objective function [4]. Since for each variable multiple bound changes may have been discovered, not only the current bounds can be part of the explanation, but also earlier bound changes. This technique is called *bound-widening* and leads to multiple alternatives per variable that can be reported. Here, theoretical questions about the complexity of computing optimal explanations arise.

4 Results

We show that it is possible to compute in polynomial time minimum-size explanations for bound changes which result from energetic reasoning and edge-finding. In case of time-tabling, we prove that an important special case is already weakly $\mathcal{NP}$-hard. To this end, we establish a relation to *unsplittable flow problems on the path* [3]. We evaluate different heuristic approaches and an exact mixed integer programming approach to explain bound changes derived by that algorithm. Using these minimum-size explanations pays off in total compared to using faster but weaker explanation algorithms. In the context of bound-widening, the problems all become $\mathcal{NP}$-hard.

Overall we experience a huge reduction in the number of nodes and in the computation times when using conflict analysis. Figure 1 visualizes the results of [4]. These

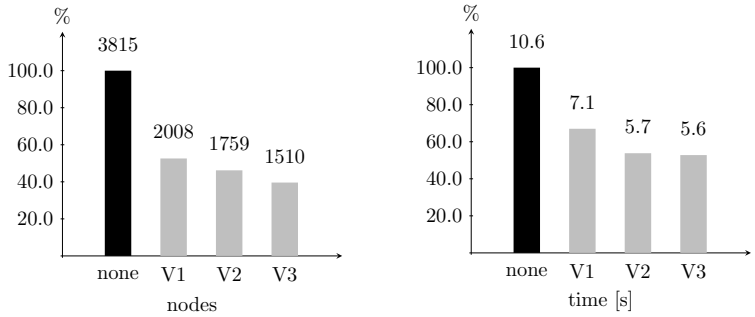


Figure 1: Comparison of the average number of nodes and average running times for 60 non-trivial and solvable instances from PSPLib `J60` are shown. More sophisticated methods (variants V1 to V3) decrease the number of nodes and the average running times. More detailed results and the different variants can be found in [4].

computational results also reveal the strength of bound-widening techniques, where a widening of minimum-size explanations decreases the running time and the number of nodes additionally.

References

[1] Tobias Achterberg. Conflict analysis in mixed integer programming. *Discrete Optimization*, 4(1):4–20, 2007.

[2] Timo Berthold, Stefan Heinz, Marco E. Lübbecke, Rolf H. Möhring, and Jens Schulz. A constraint integer programming approach for resource-constrained project scheduling. In Andrea Lodi, Michela Milano, and Paolo Toth, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2010)*, volume 6140 of *LNCS*, pages 313–317. Springer, 2010.

[3] Paul Bonsma, Jens Schulz, and Andreas Wiese. A constant factor approximation algorithm for unsplittable flow on paths. *CoRR*, abs/1102.3643, 2011.

[4] Stefan Heinz and Jens Schulz. Explanations for the cumulative constraint: An experimental study. In Panos M. Pardalos and Steffen Rebennack, editors, *Experimental Algorithms (SEA 2011)*, volume 6630 of *Lecture Notes in Computer Science*, pages 400–409. Springer, 2011.

[5] Andrei Horbach. A boolean satisfiability approach to the resource-constrained project scheduling problem. *Annals of Operations Research*, 181:89–107, 2010.

[6] PSPLib. Project Scheduling Problem LIBrary. `http://129.187.106.231/psplib/`.

[7] Andreas Schutt, Thibaut Feydy, Peter Stuckey, and Mark Wallace. Explaining the cumulative propagator. *Constraints*, pages 1–33, 2010. 10.1007/s10601-010-9103-2.

A Genetic Algorithm for A Nurse Rerostering Problem

Zdeněk Bäumelt (Speaker) * Přemysl Šůcha * Zdeněk Hanzálek *

1 Introduction

This paper is focused on a *Nurse Rerostering Problem* (NRRP) that is tackled every day in hospitals in case of some unpredicted circumstances (e.g. sick leaves of the nurses). Consequently, it is necessary to move the shift of the absent nurse to someone else in the roster. However, NRRP cannot be usually solved by the same methods used for another well known *Nurse Rostering Problem* (NRP), since each change of the original roster may lead to collisions with the nurses' already planned free-time activities. Therefore, the main objective is to solve NRRP respecting the minimum changes in the original roster. Moreover, the roster has to satisfy a given set of constraints to preserve the structure of the original roster.

2 Problem Statement

The NRRP discussed in this paper is defined as follows. Let E be a *set of the nurses*, D be a *set of the days* and S be a *set of the shifts*. Subsequently, let R^0 be a binary matrix standing for an *original roster* (the NRP output) of a size $E \times D \times S$, i.e. $R_{ijs} = 1$ denotes that shift s is assigned to nurse i on day j. Similarly, a *modified roster* is defined by R. Afterwards, let A be a binary *matrix of absence* of a size $E \times D \times S$, i.e. the fact that nurse i is not able to work shift s on day j is expressed by $A_{ijs} = 1$. Furthermore, let Δ^R_{max} be a *maximum number of modifications* in R according to R^0. Consequently, the NRRP is given by a triplet $\{R^0, A, \Delta^R_{max}\}$.

We consider the NRRP containing the shifts {early, late, night, free (day-off), holiday (fixed day-off)}. In order to determine whether shift s_1 can be followed by shift s_2 or not, we define a binary matrix SP of a size $S \times S$. Furthermore, *requested shifts* are stated by a binary matrix RS of a size $S \times D$, i.e. RS_{sj} represents the count of the requested shifts of type s on day j. Finally, one has to distinguish which parts of the roster are fixed and which can be modified. For this purpose, let F be a binary *matrix of the fixation* having a size $E \times D$. We take into account the following assumptions of NRRP: a) all shifts have the same length equal to 8 hours; b) all nurses have the same grades, i.e. all shifts can be assigned to them; c) all nurses have the same workloads.

The objective of the NRRP is to modify R^0 in order to find R that satisfies following hard constraints. The constraint (hc_1) assumes that nurse i is assigned to exactly one shift per day j, i.e. $\sum_s R_{ijs} = 1$ must hold for $\forall i \in E, \forall j \in D$. Let Δ^R be a *number*

*baumezde@fel.cvut.cz, suchap@fel.cvut.cz, hanzalek@fel.cvut.cz. Z. Bäumelt, P. Šůcha, Z. Hanzálek, Department of Control Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, Technická 2, 166 27, Prague 6, Czech Republic

of modifications in R. Thereafter, the constraint (hc_2) guarantees that $\Delta^R = 1/2 \cdot \sum_{i,j,s} \left| R^0 - R \right| \leq \Delta^R_{max}$. The constraint (hc_3) ensures that all fixed shifts in R cannot be changed, i.e. $\forall F_{ij} = 1 \Rightarrow R_{ijs} = R^0_{ijs}, \forall i \in E, \forall j \in D, \forall s \in S$. Moreover, the available nurses have to be taken into account only (hc_4), i.e. $\forall A_{ijs} = 1 \Rightarrow R_{ijs} = 0, \forall i \in E, \forall j \in D, \forall s \in S$. The constraint (hc_5) avoids the forbidden shift precedences, i.e. $\forall SP_{s_1 s_2} = 0 \Rightarrow R_{ijs_1} \wedge R_{i,j+1,s_2} = 0, \forall i \in E, \forall j < |D|, \forall s_1, s_2 \in S$. The last hard constraint (hc_6) expresses that the coverage of shifts has to be met, i.e. $RS_{sj} = \sum_i R_{ijs}, \forall j \in D, \forall s \in S$.

Naturally, NRRP contains the same soft constraints as NRP to keep the original roster. Let Z^Δ be a *multiobjective function*, given by a linear combination of the weighted penalizations of the particular soft constraints, that is minimized. The total workload balancing among nurses (sc_1) is incorporated in Z^Δ by $\sum_i \left| \sum_{j,s} R_{ijs} - \overline{W} \right|$, where $\overline{W}$ is an *average workload* equal to $1/|E| \cdot \sum_{s,j} RS_{sj}$. The constraint (sc_2) takes into account the maximum number of consecutive days-on. Finally, the count of modifications Δ^R is minimized by (sc_3) in order to keep R as close to R^0 as possible.

Most of the approaches dealing with NRP are summarized in [1]. Unfortunately, NRRP receives a much less attention than NRP, though NRRP has to be solved as often as NRP. Several approaches are presented in papers by Moz and Pato. The first paper [2] models NRRP by an integer multicommodity flow represented by a multi-level acyclic network. Another approach based on a genetic algorithm was described in more variations in papers [4] and [3] (contains the proof that NRRP is NP-complete based on the polynomial-time reduction to another well-known NP-complete problem, the three-dimensional matching problem). Different strategies of evolutionary algorithms were discussed in [6] where the results of Moz and Pato were outperformed. Another approach for NRRP based on a recursive search algorithm is presented in [5].

We have observed from our preliminary results that the time complexity of the NRRP approach is crucial. We implemented a recursive search algorithm with the bounded depth of the recursion ($\sim \Delta^R_{max}$) on the CPU. The computational time for instances with $|E| = 50, \Delta^R_{max} = 4$ reached dozens of seconds, while for instances with the same parameters and $\Delta^R_{max} = 8$ it increased to hundreds of seconds, which was the greatest disadvantage of the approach. Moreover, this approach was memory consuming, too. Therefore, our intention is to propose an approach for NRRP eliminating these drawbacks.

3 Basic Concept of Solution

We have decided to solve NRRP by a genetic algorithm (GA) based on an island model of the population (see in [8]). The outline of the GA is following. An *initial population* $\mathcal{P}_0$ containing *individuals* $\mathcal{I}$ is generated. Subsequently, an evaluation of the current population, a selection and a recombination process are performed repeatedly. The stop condition is met when the total count of iterations is reached or the value of Z^Δ is decreased to 0.

Each individual $\mathcal{I}$ of a population $\mathcal{P}$ is defined by $\mathcal{I} = \left\{ M^R, T^R \right\}$ as follows. Let $M^R = \{(i_1, j_1, s_{n_1}), (i_2, j_2, s_{n_2}), \ldots\}$ be a *list of modifications* in R of a length Δ^R_{max}. Each element of M^R corresponds to one *modification* in R with respect to R^0, i.e. a *new shift* $s_n \neq s$ is assigned to nurse i on day j. Furthermore, let $T^R = \{(i_1, i'_1, j_1), (i_2, i'_2, j_2), \ldots\}$ be a *list of exchanges* in R of a length $\lceil 1/2 \cdot \Delta^R_{max} \rceil$. In

this case, each triplet of T^R represents one *exchange* of shifts in R between nurses i and i' on day j. This representation is redundant from the data point of view, but its benefit lies in the efficient performance of GA described in the next paragraph.

The T^R representation is more appropriate for the generation of the initial population $\mathcal{P}_0$. Each individual can be created by filling of the T^R randomly under the policy of some relations in T^R, e.g. for each 2 following exchanges t and $t+1$ in T^R holds that $i_t = i_{t+1} \vee i_t = i'_{t+1} \vee i'_t = i_{t+1} \vee i'_t = i'_{t+1}$. Similarly, the T^R representation is more efficient for the selection and the recombination process. On the contrary, the M^R representation is more effective for the evaluation of all $\mathcal{I}$ in $\mathcal{P}$ and therefore, T^R has to be mirrored to M^R to evaluate the current population. The evaluation is performed over the nurses and days with the modifications only.

4 Contribution

We have introduced one of the NRRP representation, that, to the best of our knowledge, was not presented yet. We want to open a discussion about an efficient representation of the NRRP suitable for a genetic algorithm. We will present the performance of the proposed approach with the described representation of the problem. Furthermore, the achieved results will be compared with our already implemented CPU solution.

References

[1] Burke, E.K., De Causmaecker, P., Vanden Berghe, G., Landeghem, H.V.: The State of the Art of Nurse Rostering. Journal of Scheduling, 7(6), 441-499 (2004)

[2] Moz, M. and Pato, M. V.: Solving the Problem of Rerostering Nurse Schedules with Hard Constraints: New Multicommodity Flow Model, Annals of Operations Research, 128, 179-197 (2004)

[3] Pato, M. V. and Moz, M.: A Genetic Algorithm Approach to a Nurse Rerostering Problem, Computers & Operations Research, 34, 667-691 (2007)

[4] Pato, M. V. and Moz, M.: Solving a Biobjective Nurse Rerostering Problem by Using a Utopic Pareto Genetic Heuristic, Journal of Heuristics, 14, 359-374 (2008)

[5] Kitada, M., Morizawa, K. and Nagasawa, H.: Recursive Search Approach in Nurse Rerostering Following a Sudden Absence of Nurses. In Proceedings of the 10th Asia Pacific Industrial Engineering & Management Systems Conference 2009, 1219-1227 (2009)

[6] Maenhout, B., Vanhoucke, M.: An Evolutionary Approach for the Nurse Rerostering Problem, Computers & Operations Research, In Press, Corrected Proof (2011)

[7] Wong, M.L. and Wong, T.T.: Implementation of Parallel Genetic Algorithms on Graphics Processing Units. In Intelligent and Evolutionary Systems, 187, 197-216 (2009)

[8] Alba, E.: Parallel Metaheuristics: A New Class of Algorithms. Wiley-Interscience (2005)

[9] Talbi, E.G.: Parallel Combinatorial Optimization. Wiley-Interscience (2006)

[10] NVIDIA: CUDA C, Programming Guide, Version 3.1 (2010)

[11] Sanders, J., Kandrot, E.: CUDA by Example: An Introduction to General-Purpose GPU Programming. Addison-Wesley Professional (2010)

[12] Genetic Programming On General Purpose Graphics Processing Units, http://www.gpgpgpu.com/

Path planning in games

Marjan van den Akker [*] Roland Geraerts [†] Han Hoogeveen (Speaker) [‡]
Corien Prins [§]

1 Introduction

Path planning is one of the fundamental problems in games. The path planning problem can be defined as finding a collision-free path, traversed by a unit, between a start and goal position in an environment with obstacles.

The variant we study is the problem of finding paths for one or more *groups* of units, such as soldiers or tanks in a real-time strategy game, all traversing the same (static) environment. Each group has its own start and goal position (or area), and each unit will traverse its own path. The objective is to find the paths that minimize the average arrival times of all units.

Our main contribution is that we propose an efficient solution based on techniques from integer linear programming for the path planning problem with groups. Here we do not care about possible interference of paths; this will be taken care of by a collision-avoidance algorithm. Our solution can be used to handle difficult situations which typically occur near bottlenecks (e.g. narrow passages) in the environment. Initially, we restrict ourselves to the situation in which each unit has the same width and speed; we will address the consequences of relaxing this assumption later. Moreover, we will also discuss how we can use our approach in a game environment.

2 Basic algorithm

To solve the problem, we first construct a *directed* graph that resembles the free space in the environment; we will discuss the issue of undirected edges later. There are several ways to create such a graph. One possibility is to use tiles, as is done in earlier games, but this is considered to lead to unnatural paths. A better alternative is to use a waypoint graph [1] in combination with a navigation mesh [2]. No matter how the graph has been constructed, we determine for each arc in the graph the traversal time as the time it takes to traverse the arc. We further determine its capacity as the number of units that can traverse the arc while walking next to each other.

[*] marjan@cs.uu.nl. Department for Information and Computing Sciences, Utrecht University, P.O. Box 80089, 3508 TB Utrecht, The Netherlands.

[†] roland@cs.uu.nl. Department for Information and Computing Sciences, Utrecht University, P.O. Box 80089, 3508 TB Utrecht, The Netherlands.

[‡] slam@cs.uu.nl. Department for Information and Computing Sciences, Utrecht University, P.O. Box 80089, 3508 TB Utrecht, The Netherlands.

[§] c.r.prins@tue.nl. Department of Mathematics and Computer Science, TU Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands.

Given this graph, the problem of finding a set of optimal paths boils down to a multi-commodity flow problem; see the overview paper by Skutella [3]. If there is only one group of units, then this problem can be solved efficiently by the dynamic flow algorithm by Ford and Fulkerson [4]. Baumann and Skutella [5] have proposed an efficient algorithm to solve the variant in which all units are identical and we have only one origin and several destinations. For the general case, no efficient algorithm is known. We want to present a heuristic that is based on techniques from (integer) linear programming. The basic idea is that we formulate the problem as an integer linear program (ILP), after which we solve the LP-relaxation and convert this to an integral solution. In this way, we make the problem tractable, without loosing too much on quality.

Instead of using variables that indicate for each arc at each time the number of units of group k that traverse this arc (an arc formulation), we use a formulation that is based on *paths* for each origin-destination pair. A path is described by the arcs that it uses and the times at which it enters these arcs. Here we require that the difference in the entering times of two consecutive arcs (i, j) and (j, k) on the path is no less than the traversal time $l(i, j)$ of the arc (i, j); if this difference is larger than $l(i, j)$, then this implies that there is a waiting time at j. Initially, we assume that there is infinite waiting capacity at all vertices. The cost coefficient of a path is equal to the time at which the end-point is reached.

Given the set of possible paths with their characteristics, we introduce for each path s a decision variable x_s corresponding to the number of characters that will follow this path. The goal is to minimize the total cost (total arrival time) subject to

- the constraint that for each origin-destination pair at least the desired number of characters are sent;
- the capacity constraints of the arcs at each time point.

To find out which paths are useful, we solve the LP-relaxation (which we obtain by allowing fractional values of x_s) using column generation. Here the pricing problem boils down to finding a shortest path for each origin-destination pair in the *time expanded* graph. This graph is created from the original graph as follows: we create a set of vertices (v, t) for each original vertex v and each timepoint $t = 0, \ldots, T$, where T is the time-horizon; we put an arc between vertices (v, t_1) and (w, t_2) if $t_2 - t_1$ is equal to the traversal time of the arc (v, w) in the original graph. The length of each arc is equal to the time-difference of the vertices with a correction for the dual multipliers. If we find an improving path, then we add it, together with the set of paths that are obtained from it by shifting the time. Eventually, we solve the LP-relaxation this way.

3 Extensions and applicability in a game

Games (and especially strategy and shooter games) are highly dynamic. This yields a number of game-specific challenges that we have to overcome before our approach can be applied in a game. These challenges are

- It must run in real-time;
- We do not want to use paths that lead to isolated units in a war game;
- The environment may change;

- There can be different types of units (different speed, width);
- It must be integrated with a local rule for collision avoidance.

The real-time aspect makes it impossible to solve the ILP, even if we have restricted the number of variables dramatically. But fortunately there is no need for finding the optimal solution. After we have solved the solution to the LP-relaxation, we can just round it down. Moreover, we do not even have to solve the LP-relaxation to optimality: we can use column generation to improve the current solution, until we run out of time. To avoid that not enough characters reach their target after rounding down, we increase the number of characters that have to be sent. This yields a very beneficial side-effect: we end up with a set of compatible paths, from which we can select the right number in a preprocessing step, in which we can take other criteria into account, like the 'no isolated units' constraint. Since a change in the LP can be solved starting with the former solution our method is well suited to deal with imminent changes of the environment affecting the capacities of the arcs (for example because of a bridge that gets destroyed).

Next to these challenges, there are some extensions to the model that we should incorporate. Release dates and deadlines for the arrival at a given destination are easily incorporated by putting constraints on the time-expanded graph. It is more difficult to model *undirected* edges in the graph. Enforcing the total capacity without any side-constraints is possible, but this leads to an enormous number of additional constraints, especially when the traversal time is big. Therefore, we replace an edge by two arcs with total capacity equal to the capacity of the edge. Currently, we use a constant division of the total capacity, where the capacity division is left to the LP; it is also possible to make this division time-dependent.

References

[1] S. Rabin (2004). *AI Game Programming Wisdom 2*. Charles River Media Inc.

[2] R. Geraerts (2010). Planning short paths with clearance using explicit corridors. *IEEE International Conference on Robotics and Automation*, 1997–2004.

[3] M. Skutella, 2008. *An Introduction to Network Flows Over Time*. http://www.math.tu-berlin.de/coga/publications/techreports/2008/Report-022-2008.xhtml

[4] L. Ford Jr. and D. Fulkerson (1958). Constructing maximal dynamic flows from static flows. *Operations Research 6*, 419–433.

[5] N. Baumann and M. Skutella (2009). Earliest arrival flows with multiple sources. *Mathematics of Operations Research 34*, 499–512.

Exponential-time algorithms for scheduling problems

Christophe Lenté [*] Mathieu Liedloff [†] Ameur Soukhal [‡]
Vincent T'kindt (Speaker) [§]

1 Introduction

This communication deals with scheduling theory and exponential-time algorithms. Most of the scheduling problems dealt with in the literature are intractable problems, *i.e.* $\mathcal{NP}$-hard problems according to complexity theory. Consequently, an optimal solution of such problems can only (unless $\mathcal{P} = \mathcal{NP}$) be achieved by superpolynomial-time algorithms, the design of which has been the matter of a complete part of the literature over the last decades. Usually, the evaluation of the efficiency of such algorithms is conducted using extensive computational experiments and the challenge is to solve instances of size as high as possible. But, theoretically speaking, several fundamental questions remain open: for exponential-time algorithms can we establish stronger conclusions than their non polynomiality in time on the worst case? For instance, is it possible to derive upper bounds on their average complexity or their worst-case complexity? This is one task which is usually performed for optimal algorithm solving polynomially solvable problems, since when we provide an optimal polynomial time algorithm we usually also provide information about the number of steps it requires to compute an optimal solution. Why not for $\mathcal{NP}$-hard problems and exponential time algorithms? This would have the advantage to enable comparisons of such algorithms for a given problem.

But the interest in studying the worst-case time complexity of such algorithms, or even their average complexity, is beyond the simple interest of counting a number of steps. It is related to establishing properties of $\mathcal{NP}$-hard problems. To illustrate this, assume we deal with a $\mathcal{NP}$-hard optimisation problem for which a brute enumeration algorithm would calculate an optimal solution requiring $n!$ steps, with n the size of the input. The question is: can this problem admit an exponential-time algorithm with a worst-case time complexity lower than that of this enumeration algorithm? Can we solve it using, for instance, at most 2^n steps? Having such a property would not only give a property on the expected difficulty of a problem, but also challenge the design of efficient optimal algorithms: their efficiency should be still evaluated via computational experiments,

[*] christophe.lente@univ-tours.fr. Université François-Rabelais de Tours, Laboratoire d'Informatique, 64 avenue Jean Portalis, 37200 Tours, France.

[†] mathieu.liedloff@univ-orleans.fr. LIFO, Université d'Orléans, rue Léonard de Vinci, BP 6759, 45067 Orléans Cedex 2, France.

[‡] ameur.soukhal@univ-tours.fr. Université François-Rabelais de Tours, Laboratoire d'Informatique, 64 avenue Jean Portalis, 37200 Tours, France.

[§] tkindt@univ-tours.fr. Université François-Rabelais de Tours, Laboratoire d'Informatique, 64 avenue Jean Portalis, 37200 Tours, France.

but they would have also to not exceed the upper bound on the worst-case complexity established on the problem.

The problem of designing exponential-time algorithms with bounds on their worst-case complexity has been the matter of growing interest in combinatorial optimization over the last years ([1, 2]). To the best of our knowledge, no such exponential-time algorithm for scheduling problems can be found in the literature, except one provided in [1] on a single machine problem referred to as $1|prec|\sum C_i$ according to the well-known three-field notation of scheduling problems ([3]). In the literature on exponential-time algorithms, several technics can be used to derive worst-case bounds ([4]): *Dynamic Programming*, *Sort & Search*, *Branch-and-Reduce*, *Inclusion-Exclusion*, *Iterative Compression*, ...

2 Worst-case complexities for basic scheduling problems

We consider a bunch of basic scheduling problems for which worst-case time complexities have been established by providing exponential-time algorithms. The obtained results are summarized in table 1. For each problem, we provide its notation, the worst-case time complexity of the "trivial brute enumeration" algorithm and the best worst-case time complexity proposed. By "brute force enumeration" algorithm, we mean an algorithm which enumerates all the solutions to find the optimal one. The symbol *dec* in the notation indicates that the corresponding problems satisfy a particular property of decomposability.

Problem	Enumeration	Proposed complexity
$1\|d_i\|\sum_i w_iU_i$	$O^*(2^n)$	$O^*(\sqrt{2}^n)$
$1\|dec\|f_{max}$	$O^*(n!)$	$O^*(2^n)$
$1\|dec\|\sum f_i$	$O^*(n!)$	$O^*(2^n)$
$P2\|\|C_{max}$	$O^*(2^n)$	$O^*(\sqrt{2}^n)$
$P2\|d_i\|\sum_i w_iU_i$	$O^*(3^n)$	$O^*(\sqrt{3}^n)$
$P3\|\|C_{max}$	$O^*(3^n)$	$O^*(\sqrt[3]{9}^n)$
$P\|dec\|f_{max}$	$O^*((nm)^n)$	$O^*(3^n)$
$P\|dec\|\sum f_i$	$O^*((nm)^n)$	$O^*(3^n)$
$F2\|d_i = d,\ d\ unknown, \sum_i U_i = \epsilon\|d$	$O^*(2^n)$	$O^*(\sqrt{2}^n)$

Table 1: Bounds on the worst-case time complexity for some NP-hard scheduling problems

Most of the results presented in table 1 are obtained by application of dynamic programming, the *Sort & Search* method or dedicated decomposition schemes. During the presentation we will show some of the obtained results and demonstrate the utility of this new research topic in scheduling theory.

References

[1] Woeginger, G. Exact algorithms for NP-hard problems: A survey. M. Junger, G. Reinelt, G. Rinaldi, eds., *Combinatorial Optimization – Eureka, You Shrink!*. Springer, vol 2570, 185–207 (2003).

[2] Woeginger, G. Space and Time Complexity of Exact Algorithms: Some Open Problems. R. Downey, M. Fellows, F. Dehne, eds., *Parameterized and Exact Computation*. Springer, vol 3162, 281–290 (2004).

[3] Graham, R. L., E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics* **5** 287–326 (1979).

[4] Fomin, F., Kratsch, D. Exact Exponential Algorithms. *Springer* (2010).

Enchanced models of single machine scheduling with a deterioration effect and maintenance activities

Vitaly A. Strusevich (Speaker)* Kabir Rustogi†

1 Introduction

We consider single machine scheduling models in which the processing conditions of the machine are subject to deterioration and can be restored, at least partly, by running machine maintenance. At time zero, the machine is in a perfect processing state, and its processing conditions deteriorate as the number of the completed jobs increases. During each *maintenance period* (*MP*) no job processing takes place. Each job $j \in N = \{1, 2, \ldots, n\}$ is associated with its normal processing time p_j, which represents its duration, provided that job j is the first to be processed on the machine under perfect conditions. The maintenance durations are known, but the number of the MPs is decided by the decision-maker. In a particular schedule, the jobs are partitioned into k *groups*, $1 \leq k \leq n$, one before the first MP and one after each of the $k-1$ MPs. The actual processing time of job j may depend on both the group it has been placed into and the position of the job in the processing sequence in that group.

While there are numerous papers that study scheduling problems with deteriorating effects and scheduling problems with machine maintenance, the enhanced models that integrate both phenomena have not been addressed until very recently; see [1] and [2].

In problems that we consider, the objective is to minimize the *makespan* that is equal to the total processing time of all jobs plus the total duration of all MPs. In the most general model, the actual processing time of job j that is sequenced in the position $r \geq 1$ in group x is given by

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

We refer to $g_j^{[x]}(r)$ as a *deterioration factor*. We assume a certain order between the values of the factors $g_j^{[x]}(r)$, so that

$$\begin{aligned} 1 &= g_j^{[1]}(1) \leq g_j^{[1]}(2) \leq \cdots \leq g_j^{[1]}(n-k+1); \\ g_j^{[x-1]}(1) &\leq g_j^{[x]}(1) \leq g_j^{[x]}(2) \leq \cdots \leq g_j^{[x]}(n-k+1) \text{ for each } x, 2 \leq x \leq k, \end{aligned}$$

for each job j and for a fixed number of groups k. These inequalities reflect the fact that the starting conditions of the machine do not get better after the next MP, and

*V.Strusevich@greenwich.ac.uk. School of Computing and Mathematical Sciences, Univeristy of Greenwich, Old Royal Naval College, Park Row, London SE10 9LS, U.K.

†K.Rustogi@greenwich.ac.uk. School of Computing and Mathematical Sciences, Univeristy of Greenwich, Old Royal Naval College, Park Row, London SE10 9LS, U.K.

this holds across all jobs. We denote the general problem to minimize the makespan by $1\left|p_j g_j^{[x]}(r), MP\right| C_{\max}$, provided that the number of the MPs is not known and should be determined by the decision-maker together with an optimal schedule. Problem $1\left|p_j g_j^{[x]}(r), MP\right| C_{\max}$ essentially captures a trade-off between the fast processing of the jobs on a well-maintained machine and the time that is required to guarantee that the machine is in an acceptable condition.

2 Results

If the number of MPs is fixed to be $k-1$, then for an optimal schedule $S^*(k)$ with k groups the makespan is equal to $C(k) = P(k) + T(k)$, where $P(k)$ is the total duration of processing the jobs, and $T(k)$ is the total duration of all MPs.

Job-independent, Group-Independent factors.

In this case, it is assumed that

$$g_j^{[x]}(r) = g(r)$$

for all jobs $j \in N$ and all groups $x, 1 \le x \le n$. The function g is given in the form of an ordered array of numbers such that

$$g(1) = 1 \le g(2) \le \cdots \le g(n).$$

For this model, each MP brings the machine back to the perfect condition, the same as in the beginning of processing.

We give an algorithm for computing $C(k)$ by a direct assignment of each job to a group and to a position in the group. This solves problem $1\,|p_j g(r), MP|\, C_{\max}$ in $O(n^2)$ time.

Recall that a sequence $A(k)$, $1 \le k \le n$, is called *convex* if

$$A(k) \le \frac{1}{2}\left(A(k-1) + A(k+1)\right),\ 2 \le k \le n-1,$$

and is called $V-$shaped if there exists a k_0, $1 \le k_0 \le n$, such that

$$A(1) \ge \cdots \ge A(k_0 - 1) \ge A(k_0) \le A(k_0 + 1) \le \cdots \le A(n).$$

We prove that the sequence $P(k)$, $1 \le k \le n$, is convex. If the sequence $T(k)$, $1 \le k \le n$, is also convex (which happens, e.g., when the maintenance times are monotone) then the sequence $C(k)$, $1 \le k \le n$, appears to be $V-$shaped, and problem $1\,|p_j g(r), MP|\, C_{\max}$ can be solved in $O(n \log n)$ time.

Notice that a special case with $g(r) = r^a$ (job-independent, group-independent polynomial positional deterioration) and equal durations of all MPs has been solved in $O(n^2)$ time by using a so-called group balance principle; see [1].

Job-independent, Group-dependent factors.

In this case, it is assumed that

$$g_j^{[x]}(r) = g^{[x]}(r),\ 1 \le x \le n,$$

for all jobs $j \in N$. The function g is given in the form of a collection of n ordered array of numbers such that

$$\begin{aligned}
1 &= g^{[1]}(1) \le g^{[1]}(2) \le \cdots \le g^{[1]}(n);\\
g^{[1]}(1) &\le g^{[2]}(1) \le g^{[2]}(2) \le \cdots \le g^{[2]}(n-1);\\
&\quad . \quad . \quad .\\
g^{[x-1]}(1) &\le g^{[x]}(1) \le g^{[x]}(2) \le \cdots \le g^{[x]}(n-x+1) \text{ for some } x, 2 < x < n;\\
&\quad . \quad . \quad .\\
g^{[n-1]}(1) &\le g^{[n]}(1).
\end{aligned}$$

For this model it is assumed that after an MP the machine is brought to a condition that is not necessarily perfect, and no better then after the previous MP.

For the resulting problem $1\left|p_j g^{[x]}(r), MP\right| C_{\max}$ we give a dynamic programming algorithm that requires $O(n^3)$ time, as well as an improved algorithm that runs in $O(n^2)$ time and is based on an efficient manipulation with the deterioration factors.

Job-dependent, Group-dependent factors. We solve the resulting general problem $1\left|p_j g_j^{[x]}(r), MP\right| C_{\max}$ in $O(n^4)$ time by reducing it to $O(n)$ dynamically generated rectangular linear assignment problems, each with a cost matrix that contains n row related to the jobs and $O(n)$ columns related to the pairs "group-position". This is a generalization and an improvement of [2], where a special case with $g_j^{[x]}(r) = g_j(r) = r^{a_j}$ (job-dependent, group-independent polynomial positional deterioration) has been solved in $O(n^5)$ time by a rather straightforward use of the linear assignment problem.

We also address the models in which the duration of an MP depends on its start time and derive polynomial-time algorithms.

References

[1] W.H. Kuo and D.L. Yang (2008). Minimizing the makespan in a single-machine scheduling problem with the cyclic process of an aging effect. *J. Oper. Res. Soc.,* 59, 416–420.

[2] C.-L. Zhao and H.-Y. Tang (2010). Single machine scheduling with general job-dependent aging effect and maintenance activities to minimize makespan. *Appl. Math. Model.,*34, 837–841.

A lower bound on deterministic online algorithms for scheduling on related machines without preemption.*

Tomáš Ebenlendr (Speaker) † Jiří Sgall ‡

1 Introduction

We prove a new lower bound of 2.564 on deterministic online algorithms for makespan scheduling on related machines without preemptions. The previous lower bound was 2.438 by Berman et al [2]. They use combinatorial approach with computer based search through the graph of possible states of an algorithm. In contrast, we use an analytical bound on maximal frequency of scheduling jobs.

The currently best algorithms are $3 + \sqrt{8} \approx 5.828$ competitive deterministic and 4.311 competitive randomized one [2]. For an alternative very nice presentation see [1]. The lower bound for randomized algorithms is 2, see [3]. Thus, both in the deterministic and randomized cases, significant gaps remain.

We consider *one-by-one* online scheduling on uniformly related machines. The speed of machine M_i is denoted s_i. Each job is characterized by its processing time p_j takes p_j/s_i time to process on M_i. No preemptions are allowed, i.e., once the job is started it cannot be interrupted and the machine is busy with this job until the job is processed. The objective is to minimize the *makespan* (also called the length of the schedule, or the maximal completion time). The online algorithm sees only the next job from the input sequence and it has to schedule this job before it is given the following job. Note that in this model it is not necessary to specify the starting times of jobs, as any schedule can be trivially converted to the schedule without idle time (gaps), while not increasing the makespan. (Accordingly, this model is often considered as a variant of load balancing.)

Our lower bound is based on an instance where both the machine speeds and the processing times are a geometric sequence of machines, with both sequences having the same common ratio, similarly as in [2, 3]. First, we consider how the algorithm behaves on one of the machines and we upper bound the frequency of scheduling a job on this machine. This bound is a function of the competitive ratio, the common ratio of the geometric sequence, and the speed of the machine. Then we take the sum of these bounds on frequencies over all machines. Any online algorithm has to schedule one job in one step, thus this sum has to be at least 1. Finally, we let the common ratio of the geometric sequence to approach to 1, and obtain our lower bound.

We number the machines as well as the jobs from 0 (to obtain simpler formulas). Thus we have machines $M_0, M_1, \ldots, M_{m-1}$ and jobs $\mathcal{J} = (J_0, J_1, \ldots, J_{n-1})$. We use $\mathcal{J}[j] = (J_0, J_1, \ldots, J_j)$ to denote the input sequence of jobs cut off after J_j.

*The research was supported by project GAUK 166610

†ebik@math.cas.cz. Institute of Mathematics, AS CR, Žitná 25, CZ-11567 Praha 1, Czech Republic.

‡sgall@kam.mff.cuni.cz. Dept. of Applied Mathematics, Faculty of Mathematics and Physics, Charles University, Malostranské nám. 25, CZ-11800 Praha 1, Czech Republic.

Let $\mathcal{J}_i$ be the set of jobs scheduled on machine M_i. The completion time of the machine is then simply the sum of processing times of the jobs scheduled to the machine divided by its speed: $C_i = \frac{1}{s_i} \sum_{j:J_j \in \mathcal{J}_i} p_j$.

2 Lower bound

Our lower bound is proved by an instance with a the geometric sequence of machines, $s_i = \alpha^{-i}$, and a geometric sequence of jobs, $p_i = \alpha^i$, for some $\alpha > 1$. Both sequences have the same length, i.e., $n = m$. The optimal schedule after step t is to schedule the jobs on the machines in the reverse order, i.e., the J_j on machine M_{t-j}. The optimal makespan is $C^*_{\max}(\mathcal{J}[t]) = p_t = \alpha^t$.

The following main lemma gives the bound on the frequency. A technical difficulty is that the bound holds only in a cerain amortized way, so we have to formulate it carefully in terms of the number of jobs completed. The number t_i can be interpreted as the highest possible amortized frequency of scheduling a job to machine M_i with respect to the claimed competitive ratio R.

Lemma 1. *Let* A *be an R-competitive algorithm. Consider the instance described above. Let*

$$t_i \;=\; \log_\alpha \frac{R}{R-\alpha^i} \qquad \text{for } s_i = \alpha^{-i} > R^{-1} \tag{1}$$

Then, for any fixed $\alpha > 1$ and n, the algorithm A *schedules at most $\frac{n}{t_i} + Rs_i + 1$ jobs from the input sequence on machine M_i. Moreover the algorithm schedules at most one job on the machine with speed equal to R^{-1} (if there is any) and no job any slower machine.*

Proof. If $s_i < R^{-1}$, then no job can be scheduled the machine M_i, since if the sequence would end now, the optimal makespan is equal to the size of the last job on input, i.e, $C^*_{\max}(\mathcal{J}[t]) = p_t$. Moreover if $s_i = R^{-1}$ then there can be only one job scheduled on M_i: The same argument now shows that no job is scheduled on M_i before scheduling any job. Thus we assume $s_i > R^{-1}$ from now on.

Let $p'_1, p'_2, \ldots, p'_{n_i}$ be (the processing times of) the jobs in $\mathcal{J}_i$ (i.e., those scheduled on the machine M_i). We can bound p'_j by $\sum_{k=1}^{j} p'_j \le Rs_i p'_j$ because the algorithm is R-competitive and the optimal makespan is p'_j after scheduling this job as the last one. This yields:

$$p'_j \;\ge\; \max\left\{ \frac{\sum_{k=1}^{j-1} p'_k}{Rs_i - 1}, 1 \right\} \quad \text{for } j = 1, 2, \ldots, n_i\,.$$

We define a sequence $(q_j)_{j=1}^{n_i}$ of lower bounds on the processing times from $\mathcal{J}_i$ recursively:

$$q_j \;=\; \max\left\{ \frac{\sum_{k=1}^{j-1} q_k}{Rs_i - 1}, 1 \right\} \quad \text{for } j = 1, 2, \ldots, n_i\,.$$

We can see that $q_j \le p'_j$, as $q_1 = 1 \le p'_1$ and $\sum_{k=1}^{j-1} q_j \le \sum_{k=1}^{j-1} p'_j$ using induction on j.

If $q_j > 1$ then by the definition of q_j we have

$$q_{j+1} \;=\; q_j + \frac{q_j}{Rs_i - 1} \;=\; q_j \frac{Rs_i}{Rs_i - 1} \quad \text{and thus} \quad \log_\alpha \frac{q_{j+1}}{q_j} \;=\; \log_\alpha \frac{Rs_i}{Rs_i - 1} \;=\; t_i\,. \tag{2}$$

Now we bound n_i. We know that $q_{n_i} \le p'_{n_i} \le p_n = \alpha^n$. Using (2) we have at most $n/t_i + 1$ numbers of size $q_j > 1$ in $q_1, \ldots, q_{n_i}$. We also have that $\sum_{j=1}^{\lfloor Rs_i \rfloor} q_j \ge \lfloor Rs_i \rfloor$, thus $q_j > 1$ for any $j > Rs_i$. This gives that there are no more than $n/t_i + 1 + Rs_i$ jobs scheduled to the machine M_i by an R-competitive algorithm. □ □

Theorem 2. *For any R-competitive deterministic algorithm for nonpreemptive scheduling on related machines, the following inequality holds:*

$$1 \le \int_0^1 \frac{\ln(R)}{-\ln(1 - R^{-x})}\, dx\,. \tag{3}$$

This gives $R > 2.564$.

Proof. The algorithm has to schedule all jobs, thus Lemma 1 implies

$$n \;=\; \sum_{i=0}^{n-1} n_i \;=\; \sum_{i=0}^{\lfloor \log_\alpha R \rfloor} n_i \;\le\; (R+1)\lfloor \log_\alpha R \rfloor + \sum_{i=0}^{\lfloor \log_\alpha R \rfloor} \frac{n}{t_i}\;.$$

We can set n arbitrarily large, so that the term $(R+1)\lfloor \log_\alpha R \rfloor$ is negligible. Thus, for any $\varepsilon > 0$, we get:

$$\begin{aligned}
1 - \varepsilon \quad &\le \quad \sum_{i=0}^{\lfloor \log_\alpha R \rfloor} \frac{1}{t_i} \;=\; \sum_{i=0}^{\lfloor \frac{\ln R}{\ln \alpha} \rfloor} \frac{\ln \alpha}{-\ln(1 - \alpha^i R^{-1})} \\
&\le \quad \int_{-1}^{\frac{\ln R}{\ln \alpha}} \frac{\ln \alpha}{-\ln(1 - \alpha^i R^{-1})} di \qquad (4) \\
&= \quad \int_{-\frac{\ln \alpha}{\ln R}}^{1} \frac{\ln R}{-\ln(1 - R^{y-1})} dy \qquad (5) \\
&\xrightarrow{\alpha \to 1} \quad \int_0^1 \frac{\ln R}{-\ln(1 - R^{y-1})} dy\;. \qquad (6)
\end{aligned}$$

In (4) we simply bound the sum by the appropriate integral. We use the fact that the function in the sum can be viewed as a continuous and decreasing function of i. We substitute $i = y\frac{\ln R}{\ln \alpha}$ to get (5). Now we get the desired bound (3) by taking the limits $\varepsilon \to 0$, $\alpha \to 1$, and by substituting $x = 1 - y$ in the integral. The inner function of the integral in (3) is a bounded monotone function of R and x. So we can solve the integration numerically and get the threshold of $R \approx 2.5649877$. □ □

References

[1] A. Bar-Noy, A. Freund, and J. Naor. New algorithms for related machines with temporary jobs. *Journal of Scheduling*, 3:259–272, 2000.

[2] Piotr Berman, Moses Charikar, and Marek Karpinski. On-line load balancing for related machines. *J. Algorithms*, 35:108–121, 2000.

[3] L. Epstein and J. Sgall. A lower bound for on-line scheduling on uniformly related machines. *Operations Research Letters*, 26(1):17–22, 2000.

One to Rule Them All: a General Randomized Algorithm for Buffer Management with Bounded Delay

Łukasz Jeż (Speaker) *

1 Introduction

In this paper, we consider the problem of *buffer management with bounded delay*, introduced by Kesselman et al. [6]. This problem models the behavior of a single network switch responsible for scheduling packet transmissions along an outgoing link as follows. We assume that time is divided into unit-length steps. At the beginning of a time step, any number of packets may arrive at a switch and be stored in its *buffer*. Each packet has a positive weight, corresponding to the packets priority, and a deadline, which specifies the latest time when the packet can be transmitted. Only one packet from the buffer can be transmitted in a single step. A packet is removed from the buffer upon transmission or expiration, i.e., reaching its deadline. The goal is to maximize the *gain*, defined as the total weight of the packets transmitted.

We note that *buffer management with bounded delay* is equivalent to a scheduling problem in which packets are represented as jobs of unit length, with given release times, deadlines and weights; former two restricted to integer values. In this setting, the goal is to maximize the total weight of jobs completed before their respective deadlines.

As the process of managing packet queue is inherently a real-time task, we model it as an *online problem*. This means that the algorithm, when deciding which packets to transmit, has to base its decision solely on the packets which have already arrived at a switch, without the knowledge of the future. To measure the performance of such an algorithm, we use the standard notion of *competitive analysis* [3], which, roughly speaking, compares the gain of the algorithm to the gain of the *optimal solution* on the same instance. We say that a deterministic algorithm is $\mathcal{R}$-competitive if on any instance I its gain is at least $1/\mathcal{R}$ of the optimum (offline) gain on I. Defining competitive ratio for randomized algorithms (considered by us) requires some care [3], and we omit it to due to space constraints. We remark though that two adversary models are considered: *oblivious* and *adaptive*. Our results, which are all upper bounds, hold in the adaptive adversary model unless otherwise stated; this implies that they also hold in the other model. We also remark that in the adaptive adversary model one cannot assume that the optimum/adversary's schedule obeys an earliest deadline first order, while such assumption is usually made in analyses for the other model.

*lje@cs.uni.wroc.pl. Institute of Compurer Science, University of Wrocław, ul. Joliot-Curie 15, PL-50-383 Wrocław, Poland.

2 Our results

The main contribution of this paper is a simple memoryless scale-invariant algorithm Mix-R, which may be viewed as RMix, proposed by Chin et al. [4], with a different probability distribution over pending packets. The competitive ratio of Mix-R is at most $e/(e-1)$ on the one hand, but on the other it is provably better than that for many restricted variants of the problem. Specifically,

Theorem 1. Mix-R *is* $1/\left(1-(1-\frac{1}{N})^N\right)$*-competitive, where* N *is the maximum, over steps, number of packets that are assigned positive probability in a step.*

Note that $1/\left(1-(1-\frac{1}{N})^N\right)$ tends to $e/(e-1)$ from below. The number N can be bounded a priori in certain restricted variants of the problem, thus giving better bounds for them. The key observation allowing to bound N in certain cases is that some packets are dominated by others in the following sense.

We denote a packet of weight w and deadline d by (w,d). We say that a packet $a=(w_a,d_a)$ is *dominated* by a packet $b=(w_b,d_b)$ at time t if at time t both a and b are pending, $w_a \le w_b$ and $d_a \ge d_b$. If one of these inequalities is strict, we say that a is *strictly dominated* by b. We say that packet a is (strictly) dominated whenever there exists a packet b that (strictly) dominates it. By a standard exchange argument one can prove that, wlog, no algorithm (including the optimum/adversary) ever transmits a strictly dominated packet.

It is then easy to see that for instances with *span* (difference between deadline and release time) of any packet bounded by s, or instances with at most s different packet weights, $N \le s$. Thus for either kind of instances Mix-R is $1/\left(1-(1-\frac{1}{s})^s\right)$-competitive. In particular, on 2-bounded instances Mix-R coincides with the previously known optimal 4/3-competitive algorithm Rand [2] for the adaptive adversary model. The number N can also be bounded by 2 in case of *similarly ordered* instances (aka instances with *agreeable deadlines*) and oblivious adversary [5, Lemma 2.7].

In our analysis, we follow the paradigm of modifying the adversary's buffer, introduced by Li et al., cf. [5]. Namely, we assume that in each step precisely the same packets are pending the algorithm and the adversary. Once they both transmit a packet, we modify the adversary's buffer to make it identical with that of the algorithm. This amortized analysis technique leads to a streamlined and intuitive proof.

In both the algorithm and its analysis it is the respective order of packets' deadlines rather than their exact values that matter. Therefore, our results are also applicable to a more general problem of *Collecting Items* [1], in which the algorithm is given precisely that information on deadlines. In fact, Mix-R is the optimum randomized memoryless algorithm for *Collecting Items*, since [1, Theorem 6.3] (stated therein in a weaker, non parameterized way, with proof left out due to space constraints).

Theorem 2. *For every randomized memoryless algorithm for the* Collecting Items *problem, there is an adaptive adversary's strategy using at most* N *different packet weights such that the algorithm's competitive ratio against the strategy is at least* $1/\left(1-(1-\frac{1}{N})^N\right)$*, and at every step the algorithm has at most* N *packets in its queue.*

We note that while Mix-R is a very simple algorithm, it subsumes all previously known randomized algorithms for packet scheduling, with the sole exception of the optimum 1.25-competitive algorithm against oblivious adversary for 2-bounded instances [4]. Our analysis also provides new bounds for several restricted variants of the problem.

Algorithm 1 MIX-R (single step)

if there are no pending packets **then**
    do nothing and proceed to the next step
end if
$m \leftarrow 0$ ▷ counts packets that are not strictly dominated
$n \leftarrow 0$ ▷ counts packets with positive probability assigned
$r \leftarrow 1$ ▷ unassigned probability
$H_0 \leftarrow$ pending packets
$h_0 = (w_0, d_0) \leftarrow$ heaviest packet from H_0
while $H_m \neq \emptyset$ **do**
    $m \leftarrow m + 1$
    $h_m = (w_m, d_m) \leftarrow$ heaviest not strictly dominated packet from H_{m-1}
    $p_{m-1} \leftarrow \min\{1 - \frac{w_m}{w_{m-1}},\ r\}$
    $r \leftarrow r - p_{m-1}$
    if $r > 0$ **then**
        $n \leftarrow n + 1$
    end if
    $H_m \leftarrow \{x \in H_{m-1} \mid x \text{ is not dominated by } h_m\}$
end while
$p_m \leftarrow r$
transmit h chosen from $h_1, \ldots, h_n$ with probability distribution $p_1, \ldots, p_n$
proceed to the next step

References

[1] M. Bienkowski, M. Chrobak, C. Dürr, M. Hurand, A. Jeż, Ł. Jeż, and G. Stachowiak. Collecting weighted items from a dynamic queue. In *Proc. of the 20th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 1126–1135, 2009.

[2] M. Bienkowski, M. Chrobak, and Ł. Jeż. Randomized algorithms for buffer management with 2-bounded delay. In *Proc. of the 6th Workshop on Approximation and Online Algorithms (WAOA)*, pages 92–104, 2008.

[3] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[4] F. Y. L. Chin, M. Chrobak, S. P. Y. Fung, W. Jawor, J. Sgall, and T. Tichý. Online competitive algorithms for maximizing weighted throughput of unit jobs. *Journal of Discrete Algorithms*, 4:255–276, 2006.

[5] Ł. Jeż. Randomized algorithm for agreeable deadlines packet scheduling. In *Proc. of the 27th Symp. on Theoretical Aspects of Computer Science (STACS)*, pages 489–500, 2010.

[6] A. Kesselman, Z. Lotker, Y. Mansour, B. Patt-Shamir, B. Schieber, and M. Sviridenko. Buffer overflow management in QoS switches. *SIAM Journal on Computing*, 33(3):563–583, 2004. Also appeared in *Proc. of the 33rd STOC*, pages 520–529, 2001.

Online algorithms and bounds for the Train Marshalling Problem

Katharina Beygang * Florian Dahms (Speaker) † Sven O. Krumke ‡

1 Introduction

Marshalling and shunting problems play an important role in railway logistics. The sorting process usually takes place in a shunting yard, which can be seen as a set of parallel stacks or queues, that represent the classification tracks. These are connected with an incoming and one outbound track on which railway cars arrive or a properly sorted train can leave.

We consider the problem formulation as it was proposed by Dahlhaus et al. [1]. In the following we will refer to this model as the Train Marshalling Problem (TMP). In the European railway system the coupling and decoupling steps are mostly done manually, due to a lack of electronic couplings, and are therefore time consuming so we want to keep the number of these operations to a minimum. Therefore we will allow only one coupling and decoupling step per car. Furthermore we assume that the precise order of the leaving cars is not of relevance and the only restriction is that all cars of the same destination will appear in one block of the outgoing train. This objective makes sense as in reality railway cars have certain functions and are interchangeable within their field of application.

The model will now be as follows. The incoming train is decoupled and the cars are rolled in to the classification tracks. After the last car was rolled in, the cars from each classification track are pulled out as a whole and then coupled to form the outbound train. The objective will be to find an allocation of the cars to the classification tracks, such that a minimum of tracks is used, while the outbound train is feasible, i.e. the cars are blocked by their respective destinations. We assume that the ordering of the blocks in the train is of no relevance. This additional freedom actually makes the problem difficult, but leads to a possibly huge reduction in used tracks. Fixing the order of blocks would result in a problem that is easy to solve but might use a lot more tracks. In the following the optimal number of classification tracks for a certain problem instance will be denoted by K. See figure 1 for an example of such a sorting process.

Formally TMP can be defined as follows:

*beygang@mathematik.uni-kl.de. Department of Mathematics, University of Kaiserslautern, Paul-Ehrlich-Str. 14, 67663 Kaiserslautern, Germany.

†dahms@or.rwth-aachen.de. Chair of Operations Research, RWTH Aachen University, Templergraben 64, 52062 Aachen, Germany.

‡krumke@mathematik.uni-kl.de. Department of Mathematics, University of Kaiserslautern, Paul-Ehrlich-Str. 14, 67663 Kaiserslautern, Germany.

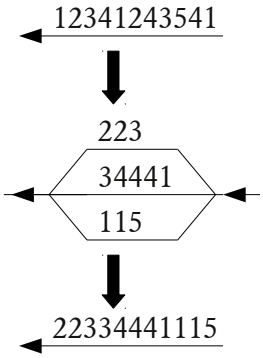


Figure 1: Example of sorting an incoming train with 5 different destinations according to the TMP model

Definition 1. *Let the set $I_n = \{1, 2, \ldots, n\}$ represent the n cars of the incoming train and let $S = \{S_1, \ldots, S_t\}$ be a partition of I_n into t sets that represent the destinations of the cars. Then TMP can be formulated as finding the smallest number K and a permutation $\pi(1), \ldots, \pi(t)$ of the destinations, so that the sequence*

$$(1, \ldots, n)^K = (\underbrace{1, \ldots, n, 1, \ldots, n, \ldots\ldots, 1, \ldots, n}_{K \text{ times}})$$

contains all the elements from $S_{\pi(1)}$ followed by the elements of $S_{\pi(2)}$ and so forth.

Note that each repetition of the sequence $1, \ldots, n$ represents one classification track and π gives the order of the destinations in the outbound train. In the example in figure 1, the destinations correspond to the partitions $S_1 = \{1, 5, 11\}, S_2 = \{2, 6\}, S_3 = \{3, 8\}, S_4 = \{4, 7, 10\}, S_5 = \{9\}$ of I_{11} and π would be the permutation $\pi = (2, 3, 4, 1, 5)$ of the destinations, leading to $K = 3$ classification tracks.

Another problem that arises within this context is the online version of TMP, where we want to assign tracks to the railway cars without knowledge about the order of the cars that will arrive later. Such a model is useful as in practice decisions often need to be made immediately and under incomplete knowledge or uncertainty. Furthermore online algorithms provide robustness in the sorting procedure as they are invariant to interruptions in the schedule and delays which are quite common in railway routine.

2 Bounds on the optimal value

As determining an optimal allocation of the cars to the classification tracks was shown to be NP-hard [1], it is desirable to get some easily calculable bounds on the optimal number of tracks. To achieve this we use some concepts from graph theory. First we associate every problem instance with an interval graph, where every destination corresponds to one interval that lasts from the arrival time of the first car to the arrival time of the last car of the respective destination. Now let ω be the size of a maximum clique in this interval graph, then we have for every incoming train

$$\lceil \frac{\omega + 1}{2} \rceil \leq K \leq \omega$$

These bounds can easily be calculated as one can compute ω for interval graphs in linear time.

Furthermore, we introduce the following new lower bound

Theorem 2. *If one divides the incoming train into two properly separated trains with clique numbers ω_1 and ω_2, we have*

$$K \geq \lceil \frac{\omega_1 + \omega_2}{2} \rceil$$

Even the optimal choice of properly separated sub-trains can be done in polynomial time, by exploiting the structure of maximal cliques in interval graphs.

Using random problem instances drawn uniformly from the set of all instances with a predefined length, this bound turns out to be fairly close to the optimal value.

3 The online problem

In the online version of TMP we assume that the incoming cars are arriving one by one and we need to push cars to the classification tracks before the destination of the next car is revealed. The goal will be to determine an optimal deterministic online algorithm for this model in terms of competitive analysis, i.e. we measure the worst case ratio between the number of tracks the online algorithm uses and the optimal value K.

We require that for each arriving car it is known if it is the last car of its destination, as otherwise there could not be any competitive online algorithm - even when using randomization. This information might be available if we know for example the total number of cars for each destination.

First we derive the following lower bound

Theorem 3. *There is no deterministic online algorithm for TMP that is better than 2-competitive.*

Using the bounds from above we now show that a greedy interval coloring scheme achieves exactly this competitive ratio and is therefore optimal among all deterministic online algorithms.

References

[1] E. Dahlhaus, P. Horak, M. Miller, J.F. Ryan (2000). *The train marshalling problem.* Discrete Applied Mathematics, 103(1-3):41-54.

[2] E. Dahlhaus, F. Manne, M. Miller, J.F. Ryan (2000). *Algorithms for combinatorial problems related to train marshalling.* AWOCA, pages 7-16.

[3] K. Beygang, F. Dahms, S.O. Krumke (2010). *Train Marshalling Problem - Algorithms and Bounds.* technical report: Report in Wirtschaftsmathematik 125, TU Kaiserslautern.

[4] F. Dahms (2010). *Train Marshalling Problems - Algorithms and Complexity.* Diplomarbeit, TU Kaiserslautern.

The lockmaster's problem

Sofie Coene (Speaker) * Frits C.R. Spieksma *

1 Introduction

Transportation of goods by ship, over sea as well as over waterways, has become more and more popular. Here, we focus on transport by inland ships over waterways. The European Commission promotes the better use of inland waterways in order to relieve heavy congested transport corridors. Carriage of goods by inland waterways is an environmentally friendly mode of transport, which can make a significant contribution to sustainable mobility in Europe [1]. Typically, these waterways are interrupted by locks such that higher water levels can be maintained. These locks are a bottleneck for transportation over water and have not been studied very broadly in OR literature. We now continue with the description of a very basic situation that will act as our core problem: the lockmaster's problem. Later we will discuss more realistic extensions. Consider a single lock. Ships coming from upstream, wanting to go downstream, arrive at the lock at given times r_i, $i = 1, \ldots, n_1$. Other ships, coming from downstream, wanting to go upstream, arrive at the lock at given times s_i, $i = 1, \ldots, n_2$. Let $n = n_1 + n_2$, and let T denote the lock-time. We assume that all data are integral. Our goal is to find a feasible lock-strategy that minimizes total waiting time of all ships. In other words, we need to determine at which moments in time the lock should start to go up (meaning at which moment in time ships that are downstream are lifted), and at which moments in time the lock should start to go down (meaning at which moments in time ships that are up are being lowered). Clearly, for such a strategy to be feasible, (i) going-up moments and going-down moments (referred to as moments) should alternate, (ii) consecutive moments should be at least T time-units apart. It is clear that this particular problem is not very realistic: capacity restrictions, weights, multiple locks, are ignored. We will, however, also deal with these issues.

2 A dynamic programming algorithm

The problem described in the introduction is closely related to scheduling a batching machine. When, in the problem described above, we only have downstream going ships, the lock can be seen as a batching machine and the jobs are the arriving ships with release dates and equal processing times (i.e. lock-time T). Following the notation of Baptiste [2] this is problem $1|p - batch, b = n, r_i, p_i = p| \sum w_i C_i$. Baptiste shows that this problem is polynomially solvable. When there are upstream going and downstream going ships, we are dealing with two families of jobs, and only jobs of the same family can

*sofie.coene@econ.kuleuven.be. Research group Operations Research and Business Statistics (ORSTAT), Katholieke Universiteit Leuven, Naamsestraat 69, 3000 Leuven, Belgium.

be together in a batch. Clearly, this is related to batch scheduling with compatibilities between jobs, see e.g. Boudhar [3]. Notice however that in our case processing a batch of one family needs to be alternated by processing a batch containing jobs of the other family; i.e. it is not possible to process two batches of the same family consecutively. We will show that the lockmaster's problem can also be solved in polynomial time by a dynamic programming algorithm.

When is a lock likely to start going up or down? Either upon arrival of a ship or immediately upon arrival of the lock. This suggests that the number of moments the lock starts operating is limited. We introduce a set of moments U at which it is possible to go up. These upmoments are referred to as u_i. Similarly, we introduce a set of moments at which it is possible to go down, the set D. These downmoments are referred to as d_i. Let us define set $S = \{s_i\}$, set $R = \{r_i\}$ and $\Theta = \{0, 2T, 4T, \ldots, 4nT\}$. We use the Minkowski-sum to sum two sets, i.e. the sum of two sets $A = \{a_i\}$ and $B = \{b_i\}$ as follows:

$$A + B = \{a_i + b_j | a_i \in A, b_j \in B\}.$$

Then, bearing this definition in mind, here is a proposal for U and for D:

$$U = (S + \Theta) \cup (R + \Theta + \{T\}),$$

$$D = (R + \Theta) \cup (S + \Theta + \{T\}).$$

We will come back to the cardinality of U and D.

Lemma 1. *There is an optimal lock strategy for the lockmaster's problem whose upmoments are contained in U, and whose downmoments are contained in D.*

Proof. Contradiction. Suppose there is an instance such that each optimal strategy has either an upmoment not in U or a downmoment not in D (these moments are called a failure). Consider that optimal strategy for this instance which has an earliest failure, say at time t. Let us assume for convenience that at time t, the lock went up. Notice that t cannot be equal to an s_i. Consider that moment in time t. Let $\epsilon > 0$ be a very small quantity. There are two possibilities:

(i) at $t - \epsilon$ the lock was waiting to go up. If, in our optimal strategy, there are ships transported up at time t, it cannot have been optimal to wait until t, since no downstream ships arrive at time t (since t is not in S). Hence, there are no ships transported. But then, we need not have waited, and there is an optimal strategy that immediately went up after the last time before t we went down.

(ii) at $t-\epsilon$ the lock was going down. Thus, at $t-T$, the lock started a down-operation. This moment in time is, by assumption, in D. But then it follows that t is in U. Contradiction.

□

Now, let us further analyze U and D.

Lemma 2. *When, for a given instance for the lockmaster's problem, during a time period equal to $4T$ no ships arrive at the lock, the instance can be divided into two instances. The solution can then be found by solving these smaller instances.*

Proof. Due to space limitations we omit the proof here. □

From now on we assume (without loss of generality, due to lemma 2) that each instance of the lockmaster's problem has the property that a ship arrives during any $4T$ interval. This allows us to bound the cardinality of $U \cup D$ to $O(n^2)$.

We now describe a dynamic programming algorithm DP. Let us define $f(u_i, d_j)$ (with $u_i \leq d_j - T$) as the minimal costs of a locking strategy that takes care of all up-requests up to u_i, all down-requests up to d_j, which features an upmoment at time $t = u_i$, which features a downmoment at time $t = d_j$, and such that there are no other up- or downmoments in between u_i and d_j.

Here is a recursion:

$$f(u_i, d_j) = \min_{d_{j'} \leq u_i - T; u_{i'} \leq d_{j'} - T} \{f(u_{i'}, d_{j'}) + \sum_{\ell : u_{i'} \leq s_\ell \leq u_i} (u_i - s_\ell) + \sum_{k : d_{j'} \leq r_k \leq d_j} (d_j - r_k)\}.$$

For this recursion to work we set $u_1 = \min\{s_1, r_1 - T\}$. The optimal value is given by $\min\{f(u_i, d_j) | u_i \geq s_{n_2}, d_j \geq r_{n_1}, u_i \in U, d_j \in D\}$.

To determine the time complexity of DP, we claim the following.

Claim 3. *If $d_j \in D \setminus R$, then the previous upmoment was $d_j - T$.*

Claim 4. *If $d_j \in R$, then the previous upmoment is either $d_j - T$ or the latest $s_i \leq d_j - T$.*

Theorem 5. *DP is a polynomial-time algorithm for the lockmaster's problem.*

Proof. Correctness follows from lemma's 1 and 2 and the following. Given that U and D have cardinality $O(n^2)$, this algorithm will have $O(n^2)$ states, due to the claims above. Computing each state can be done by evaluating $O(n^2)$ states, leading to a total time complexity for this algorithm equal to $O(n^4)$. □

3 Extensions

For the analysis above we chose as an objective to minimize the sum of the waiting times, which is a very natural objective function for this problem. We can, however, deal with other settings using a similar approach. We are able to solve the weighted case and the case with non-uniform locking times for going up and down. We also studied the problem with capacity restrictions, i.e a maximum on the number of boats that can go in the lock together. Finally, we study the problem with multiple locks in series.

References

[1] H. Allaeys, Optimalisering van een sluis (in Dutch) (2010). *Master Thesis, Katholiek Universiteit Leuven.*

[2] P. Baptiste, Batching Identical Jobs (2000). *Mathematical Methods of Operation Research 52, pp. 355-367.*

[3] M. Boudhar, Scheduling a batch processing machine with bipartite compatibility graphs (2003). *Mathematical Methods of Operation Research 57, pp. 513-527.*

Improved approximation algorithms for routing shop scheduling

Wei Yu * Guochuan Zhang (Speaker) †

1 Introduction

We are given an undirected edge-weighted graph $G = (V, E)$, where $V = \{0, 1, 2, \ldots, n\}$ is the vertex set and E is the edge set. There are n jobs, where job j is placed at vertex j for $j = 1, 2, \ldots, n$. We have m machines $M_1, M_2, \ldots, M_m$ which originally stay at vertex 0, Job j consists of m operations $O_{1,j}, O_{2,j}, \ldots, O_{m,j}$, and $O_{i,j}$ should be processed by machine M_i for $p_{i,j}$ time units without any interruption. To process these jobs the machines travel between the vertices at the same speed. Let $t_{j,k}$ be the travel time between vertices j and k, which defines the edge length. Clearly, the network is metric.

A feasible schedule is to plan a routing for each machine and schedule the operations such that at any time each job is processed by at most one machine (the machine must arrive at the job) and each machine processes at most one job. In such a schedule S denote $s_{i,j}(S)$ to be the starting time of operation $O_{i,j}$.

We consider two models. For flow shop the operations $O_{1,j}, O_{2,j}, \ldots, O_{m,j}$ of job j have to be processed in this order, while for open shop the operations of a job may be processed in an arbitrary order. We aim to minimize the makespan. The two problems are, respectively, denoted as $RO||C_{\max}$ and $RF||C_{\max}$. However, there are two slightly different versions with respect to the makespan. For a schedule S, let $C_j(S)$ be the completion time of job j (by which all its operations are done). In the path-version the makespan $C_{\max}(S) = \max_{1 \le j \le n} C_j(S)$, while in the tour-version the makespan $C_{\max}(S) = \max_{1 \le j \le n}\{C_j(S) + t_{j,0}\}$. More precisely, in the path-version we take care of the largest job completion time. In the tour-version we are concerned about the largest time when all machines return to their home, i.e., vertex 0, after the completion of all jobs.

Obviously the well know traveling salesman problem and the flow (shop) scheduling problem are special cases of our models.

For the tour-version of the m-machine problem $RF||C_{\max}$, Averbakh and Berman [1] presented a simple $\max\{\frac{m+1}{2}, \rho\}$-approximation algorithm, given a ρ-approximation tour on the underlying graph. Averbakh et al. [2] dealt with the tour-version of the m-machine problem $RO||C_{\max}$, and gave an $(\frac{m+1}{2} + \rho)$-approximation algorithm by using the same algorithm for $RF||C_{\max}$. Chernykh et al. [3] gave a 13/8-approximation algorithm for

*yuwei2006831@zju.edu.cn. College of Computer Science, Zhejiang University, Hangzhou, 310027, China.

†zgc@zju.edu.cn. College of Computer Science, Zhejiang University, Hangzhou, 310027, China.

$RO2||C_{\max}$. Moreover, they devised an $O(\sqrt{m})$-approximation algorithm for the m-machine case $RO||C_{\max}$ using a job-aggregation idea and the greedy algorithm for the classical open shop problem.

2 Main results

We significantly improve the previous results by showing the following theorems.

Theorem 1. *For any $\epsilon > 0$, there exists an $O(\log m(\log\log m)^{1+\epsilon})$-approximation algorithm for both versions of $RO||C_{\max}$.*

Theorem 2. *There exists an $O(m^{2/3})$-approximation algorithm for both versions of $RF||C_{\max}$.*

Both algorithms transfer the original instance into one with a number of jobs polynomially bounded by the number of machines so that the new instance is relatively easily approximated and there is not much loss by aggregating jobs. For $RO||C_{\max}$ we are inspired by the idea on job-aggregation in [3]. Meanwhile we exchange the roles of the machines and the jobs, and apply an algorithm in [4]. For $RF||C_{\max}$, after properly aggregating the jobs we will borrow the algorithm by Nagarajan et al. [5] for the permutation flow shop problem. In the following we give a sketch of the two improved algorithms.

2.1 Routing open shop

Given an instance I of $RO||C_{\max}$, we first construct an instance I' of $O||C_{\max}$ with the same machines but $n' \leq 2m+1$ jobs. Then we obtain a schedule S' of I' as follows: firstly, we exchange the roles of machines and jobs and assign a same prescribed sequence to indicate the processing order of the operations belonging to each new job, which obtain an instance I'' of $F||C_{\max}$. After that we run the algorithm proposed by Czumaj and Scheideler [4] for a generalized flow shop problem, called an acyclic job shop problem, on I'' to obtain a schedule S'' and transfer it into a schedule S' of I' of equal makespan. By the results in [4], the makespan of S' is at most $O(\log n' \log\log n' \cdot (L' + p'_{\max}))$, where L' and $p'_{\max}$ are the maximum machine load and maximum processing time of I' respectively. By the construction of I', we can bound $L' + p'_{\max}$ within a constant factor of the optimal value of I. Moreover, from the construction of S' we can see that schedule S' satisfies the property that the processing order of jobs on each machine are the same prescribed sequence. This property, which can not be achieved by a simple greedy algorithm, allows us to transfer the schedule S' into a feasible schedule S of I with a makespan increasing by a constant factor of the optimal value of I.

2.2 Routing flow shop

The same idea used in routing open shop can not be applied to routing flow shop, since the schedule may violate the constraints for flow shop, i.e., the operations of the same job should be processed successively by $M_1, M_2, \ldots, M_m$. However, we can combine the idea of job-aggregation and the algorithm for permutation flow shop problem to derive an $O(m^{2/3})$-approximation algorithm, which works as below.

Given an instance I of $RF||C_{\max}$, we first construct an instance I' of $F||C_{\max}$ with the same machine set and $n' \leq 2m^{2/3} + 1$ jobs. Using the algorithm proposed by Nagarajan and Sviridenko [5] for permutation flow shop problem we obtain a permutation schedule S'. The makespan of S' is at most $O(\sqrt{\min\{m, n'\}} \cdot (L' + p'_{\max}))$, where L' and $p'_{\max}$ are the maximum machine load and maximum processing time of I' respectively. By the construction of I', we can bound $L' + p'_{\max}$ within an $O(m^{1/3})$ factor of the optimal value of I. Finally, the property that S' is a permutation schedule, which can not be achieved by the algorithm of Czumaj and Scheideler [4], allows us to transfer S' into a permutation schedule S of I with the makespan bounded by the claimed ratio.

References

[1] I. Averbakh, O. Berman, A simple heuristic for m-machineąą flow-shop and its applications in routing-scheduling problems. *Operations Research* 47 (1999) 165–170.

[2] I. Averbakh, O. Berman, I.D. Chernykh, The routing open-shop problem on a network: complexity and approximation. *European Journal of Operational Research* 173 (2006) 531–539.

[3] I. Chernykh, N. Dryuck, A. Kononov, S. Sevastyanov, The routing open-shop problem: New approximation algorithms. in: E. Bampis, K. Jansen (Eds.), the Proceedings of the 7th Workshop on Approximation and Online Algorithms, in: *Lecture Notes in Computer Science* 5893 (2010) 75–85.

[4] A. Czumaj, C. Scheideler, A new algorithmic approach to the general Lovász local lemma with applications to scheduling and satisfiability problems, In: Proceeding of the 32nd Symposium on Theory of Computing, 38–47, 2000.

[5] V. Nagarajan, M. Sviridenko, Tight bounds for permutation flow shop scheduling. In: A. Lodi, A. Panconesi, and G. Rinaldi (Eds.): the Proceedings of the 13th Conference on Integer Programming and Combinatorial Optimization, in: *Lecture Notes in Computer Science* 5035 (2008) 154–168.

[6] D. Shmoys, C. Stein, J. Wein, Improved approximation algorithms for shop scheduling problems. *SIAM Journal on Computing* 23(3) (1994) 617–632.

The road traffic model for the car factory logistics

Grzegorz Pawlak (Speaker) * Mateusz Cichenski † Mateusz Jarus ‡
Slawomir Walkowski §

1 Introduction

Many aspects of the traffic control system may have influence on production and logistic system in the factory. In the paper the WHAT IF analysis was considered. The purpose of the paper was to prepare the simulation model for the traffic in the urban environment due to the transport organization between the two plants of the car factory. The motivation for this work was generated by the necessity of changing the transportation model and checking the influence of renovation one of the main crossroads in the roads network. The one of the main junction is rebuilding in the shuttle route of internal lorries traffic in the car factory. The observation was made for the several types of the vehicles: delivery final product trucks, suppliers chain trucks, JIT lorries and shuttles between two plants involved in the car factory logistic system. The simulation model has been constructed for the considered area of the city map. The real data has been collected either from the road traffic control operators or the car factory logistic control system. The junction graph model has been proposed and adapted to the traffic model. The main junctions in the areas were modelled according to the real design. The traffic light control program was incorporated into the simulation models. The real traffic density data on the observed area were collected. The goal was to prepare the simulation model for the one hypothetical average working day 24h. The distribution of the traffic was constructed out of the real data set. Also the distribution of the car routing on the each junction have been designed. For several types of the vehicles the routing was determined. For example for the buses and shuttles which routing is deterministic and fixed. The traffic jams were defined and observed during the simulation. Analysis generates several reports aiding the decision maker of the logistic control. Having the simulation tool and the traffic model dedicated to the particular circumstances one can construct the optimization models either to minimize the number of trucks in the shuttle routes or minimize the negative influence for the production continuity. There is also possible to study several scenarios and to observe the traffic jams and congestions and their influence to the trucks circulation.

*grzegorz.pawlak@cs.put.poznan.pl. Institute of Computing Science, Poznan University of Technology, ul. Piotrowo 2, 61-315 Poznan, Poland.

†mcichenski@skno.cs.put.poznan.pl. Institute of Computing Science, Poznan University of Technology, ul. Piotrowo 2, 61-315 Poznan, Poland.

‡mjarus@skno.cs.put.poznan.pl. Institute of Computing Science, Poznan University of Technology, ul. Piotrowo 2, 61-315 Poznan, Poland.

§swalkowski@skno.cs.put.poznan.pl. Institute of Computing Science, Poznan University of Technology, ul. Piotrowo 2, 61-315 Poznan, Poland.

2 Traffic modelling and input data analysis

To achieve the results which can be applied in practice we try to construct the traffic model as close to reality as possible. The whole visualized map consists of multiple segments connected with special markers of different types to create a precise reflection of real routes. Their length is diverse and depends only on the distance between certain characteristic points. Every segment represents a straight section of road which is either just a part of a longer curve or indicates some crossings. It also allows for one-way motion only - to create a second lane another list of segments needs to be created. Map structure is defined in an XML file as a set of tags of different type. The values were derived from the data we gathered from City Roads Authorities (CRA) and Volkswagen car factory. We were able to get sufficient data from CRA about the traffic flow in the area. We also managed to get all the traffic lights programs that were used on the main junctions in the area. To correctly analyze that data we based on the junctions blueprints which were provided by CRA. VW introduced information about the routes of their lorries and the numbers of lorries that cruises between plants. What's more, due to the large number of employees, we had to include the data about buses routes and special buses rented by VW factory for their workers.

There are four main junctions in the area with traffic light system: On those junctions we had data from magnetic loop counters about the number of cars that passed on certain lane. Using that data we could calculate the distributions for the lanes - the probability value that a car will pick that lane when it will have the chance to switch it. We know how many cars go through particular lane from the data. To get the distribution we have to sum up all the cars that went through the road and divide the quantities on each lane by the total number of cars.

3 Software solution

Our goal was to create an extendable tool to simulate and analyze traffic. Moreover, we wanted to provide a visualization that allows the playback of simulated scenarios. Many tools does not provide the functionality to replay exactly the same simulation and most of them also do the simulation and visualization process simultaneously. We decided to split these processes and create independent tool for simulation and independent tool for visualization. With this approach it is possible to play the simulation many times to verify the results. Simulation generates reports with the number of traffic jams, their duration and the time that certain type of vehicle spent driving on its route.

The applications use the concept of time line with frames. Each frame describes the current state of simulation. However, the frames contain only information about the objects that changed their state from previous state. That's why we introduced key frames that contain information about the whole state, which makes it possible to restore the state using key frame i.e. at a given time stamps T_1 the car moved forward on the segment, thus this information will be included in the following time stamps T_2. If the same car had to stop because there was another car in front of it, so it did not change its state thus providing this information in following time stamps T_3 is unnecessary.

4 Computational experiments and reports

Apart from the file for visualization purposes, simulation process produces also report with statistics, which sum up the simulation results. It consists of two main parts: statistics for the factory logistic vehicles and information about traffic jams. In the first part, vehicles are classified according to their type, route and direction. For each class, the following characteristics are calculated and outputted:

- start and end point of the route,
- number of vehicles which drove through the whole route (one physical vehicle may be counted many times if it accomplished many cycles),
- minimum, maximum and average time of driving the route by a single vehicle.

In the second part of the report, each entry of each junction is characterized by statistics of the traffic jams which occurred on the junctions. These statistics contain:

- total time of the traffic jams,
- number of traffic jams,
- average time of a single traffic jam,
- average length of a traffic jam (expressed as a number of vehicles stuck in a traffic jam),
- total number of vehicles stuck in the traffic jams,
- average time spent by factory logistic vehicles in the traffic jams.

5 WHAT IF analysis

We have distinguished three periods during the 24h when the traffic picks. Distinguishing these hours allows us to focus on situations when traffic conditions are really difficult. The shuttle trucks from Volkswagen are in circulation in the same cycle road for the whole day, so concentrating on the time spans described above is crucial and enough for getting the simulation results in worst cases.

We have analyzed seven different pre-defined scenarios of the traffic organization which contains specified road network and distributions of cars in sources and junctions. We have analyzed the influence on the necessary truck number to avoid the production disturbances. One can observe the crucial points and values of parameters having direct influence on the production system. In such complex system where multi type transportation system is connected with the production and with many constrains the simulation and WHAT IF analysis cloud be the practically useful method.

Schedulability analysis for a combination of preemptive strict periodic tasks and sporadic tasks

Mohamed Marouf (Speaker) Laurent George Yves Sorel *

1 Introduction

We consider the problem of scheduling tasks with strict periods combined with sporadic tasks. Both types of task have fixed priorities and are preemptive. For a task with a strict period, the difference between its starting time and its release time must be identical for every job. Tasks with strict periods are typically in charge of controlling the activities of a system (sensor/actuator, feedback control, ect.). The freshness of the information they use and/or the reactivity of the system are constrained. Indeed, for control tasks, it might be important to control their jitters (the difference between the worst case and the minimum response time) to ensure the stability of the control loop. In this paper, we consider for controlled tasks, the solution satisfying property 1 that minimizes the jitter of tasks with strict periods.

Property 1. *For any task with strict period, (i) the start time of any job of the task must be equal to its release time and (ii) the Worst Case Response Time (WCRT) of the task must be equal to its Worst Case Execution Time (WCET).*

In this paper, we provide a sufficient schedulability condition for the schedulability of tasks with strict periods. We show how to define their first release times such that property 1 is met (based on paper [1]). Tasks with strict periods have the same fixed priority, the highest one. Sporadic tasks all have a lower priority than any task with a strict period. We show in this paper how to define the worst case scenario for the schedulability of sporadic tasks in the presence of tasks with strict periods. Then we propose a schedulability condition for sporadic tasks based on the worst case response time computation.

2 Scheduling tasks with strict periods

A task $\tau_i(C_i, T_i, D_i)$ with the strict period T_i is characterized by (i) a first start time S_i^1, (ii) a strict period T_i equal to the deadline D_i such as the start time of the k^{th} job of task τ_i is given by $S_i^k = S_i^1 + (k \cdot T_i)$, and (iii) a WCET $C_i \leq T_i$.

It has been proved in [2] that a set of tasks Γ is schedulable (sufficient condition) if

$$\sum_{i=1}^{n} C_i \leq g \tag{1}$$

where g is the *gcd* of the periods. The start time of each task is given by

*{Mohamed.Marouf, Laurent.George, Yves.Sorel}@inria.fr. INRIA Paris-Rocquencourt, Domaine de Voluceau BP 105, 78153 Le Chesnay Cedex, France.

$$S_1 = 0,\ S_i = \sum_{j=1}^{i-1} C_j,\ \ i \geq 2 \tag{2}$$

In [1] we gave *a local schedulability condition* when a set of tasks Γ do not satisfy condition (1). In order to prove that Γ is schedulable, we first schedule a sub-set of tasks according to condition (1). Then, for each remaining (candidate) task, we apply the following local schedulability condition iteratively, such that this candidate task, added to a sub-set of tasks already scheduled, leads to a schedulable set of tasks. A candidate tasks $\tau_c(C_c, T_c, D_c)$ is thus schedulable if

$$C_c \leq C_i \cdot \delta\left[T_c mod(T_i) \cdot (T_c mod(2g) + T_i mod(2g))\right] \tag{3}$$

where *mod* is the modulo function and δ is the Kronecker symbol: $\delta(i) = \begin{cases} 1\ if\ i = 0 \\ 0\ otherwise \end{cases}$

The start times of a candidate task τ_c are given by

$$\begin{cases} If\ (T_c mod(T_i) = 0)\ then\ S_c^1 = S_i^1 + kg,\ \ 1 \leq k \leq \frac{T_c}{T_i} - 1 \\ If\ (T_i mod(2g) = T_c mod(2g) = 0)\ then\ S_c^1 = S_i^1 + (2k+1)g,\ \ 0 \leq k \leq \frac{T_c}{2g} - 1 \end{cases} \tag{4}$$

Theorem 1. *The scheduling of strict periodic tasks in the time interval* $[kL, (k+1)L]$, $k \in \mathbb{N}^*$, *with start times satisfying equations (2) or (4), is identical to the scheduling obtained in the time interval* $[0, L]$. *L is the LCM (Least Common Multiple) of all the tasks periods.*

Proof According to the conditions (2, 4), $0 \leq S_i^1 < T_i$. The k^{th} start time of a task τ_i is given by $S_i^k = S_i^1 + kT_i$. From the definition of S_i^k, $k \leq 1$, a task released at time S_i^k ends at time $S_i^k + C_i$. Thus, the scheduling of tasks in the time interval $[kL, (k+1)L]$, $k \in \mathbb{N}^*$, is identical to the scheduling obtained in the time interval $[0, L]$, . ⊡

We define the set Ψ which contains all the start times of the strict periodic jobs on a time interval $[0, L[$. Thus, $\Psi = \{S_i^1 + k \cdot T_i,\ k = 0..(\frac{L}{T_i} - 1),\ i = 1..n\}$.

3 Scheduling sporadic tasks

A sporadic task $\tau_i(C_i, T_i, D_i)$ is defined by its WCET C_i, its minimum inter-arrival time T_i and its relative deadline $D_i \leq T_i$. In order to study the schedulability of sporadic tasks when tasks with strict periods have already been scheduled, we have to determine the critical instants for a sporadic job τ_i^j. A critical instant is defined as the instant where a job will have the largest response time if its release time r_i^j is set to it. It has been proved in [3] that a critical instant occurs when the release time of a sporadic job is equal to the release time of a highest priority job. For all sporadic tasks, this results in first releasing all tasks with priority higher than τ_i at the same time as the first release time of τ_i. To consider tasks with strict periods, we have to study all cases of release times $S_i \in \Psi$ following the scenario defined in section 2. However, rather than testing all the start times of Ψ, some start times are useless and will be thus pruned from Ψ according to the following lemma.

Lemma 2. *For a sequence of consecutive executions of strict periodic jobs, only the first release times in the sequence should be considered.*

Thus, if $\exists S_i^k$, $S_j^l \in \Psi$ such that $S_i^k - S_j^l = C_j$, S_i^k is removed from Ψ. Consider Ψ the release times of tasks with strict periods obtained applying equations (2, 4). Consider that task τ_i is first released at time $S \in \Psi$. $W_i(t)$ denotes the sum of the computational requirements at time t (w.r.t time S) of all the tasks with higher priority, plus one execution of τ_i starting from time S. It is given by

$$W_i(t) = C_i + \sum_{\tau_j \in \Gamma_P^{NS}/i} \left\lceil \frac{t}{T_j} \right\rceil C_j + \sum_{\tau_j \in \Gamma^S} \left\lceil \frac{t - s_j}{T_j} \right\rceil C_j \quad (5)$$

where s_j is the relative start time S_j^k according to a release time S of τ_i. s_j is given by

$$s_j = S_j^1 + \left\lceil \frac{S - S_j^1}{T_j} \right\rceil T_j - S \quad (6)$$

The WCRT of τ_i is the solution of $r_i = W(r_i)$ computed by iteration [3].

4 Example

We use the Rate-Monotonic algorithm to schedule the following tasks. Let us consider three tasks with strict periods: $\tau_1(1,4,4)$, $\tau_2(1,6,6)$ and $\tau_3(1,12,12)$. τ_1 and τ_2 satisfy condition (1) and are schedulable with $S_1^1 = 0$, $S_2^1 = 1$. τ_3 and τ_1 satisfy condition (3), thus τ_1 is schedulable with $S_3^1 = 6$. $L = LCM(T_1, T_2, T_3) = 12$ thus $\Psi = \{0, 1, 4, 6, 7, 8\}$. After eliminating the successive jobs according to lemma 2, $\Psi = \{0, 4, 6\}$.
Let $\tau_4(2, 6, 8)$, $\tau_5(2, 12, 12)$ be sporadic tasks to be scheduled.
We have: $W_4(t) = 2 + \sum_{j=1}^{3} \left\lceil \frac{t-s_j}{T_j} \right\rceil C_j$ and $W_5(t) = C_5 + \left\lceil \frac{t}{T_4} \right\rceil C_4 + \sum_{j=1}^{3} \left\lceil \frac{t-s_j}{T_j} \right\rceil C_j$.

S	s_1	s_2	s_3	r_4	r_5
0	0	1	6	4	**12**
4	0	3	2	**6**	8
6	2	1	0	5	**12**

As $r_4 \leq D_4$ and $r_5 \leq D_5$, τ_4 and τ_5 are schedulable, as shown in figure 1.

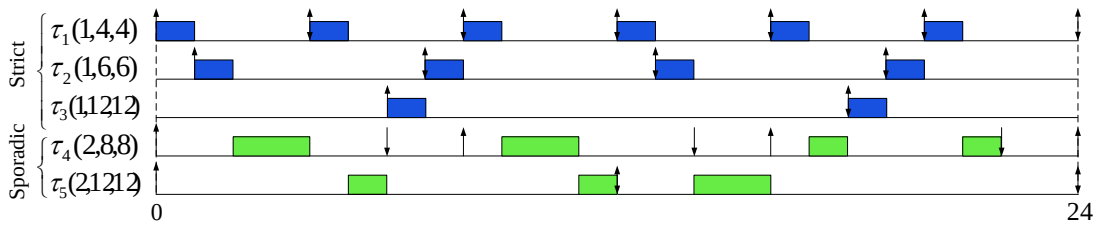


Figure 1: Scheduling diagram of strict periodic and sporadic tasks

References

[1] M. Marouf and Y. Sorel. *Schedulability conditions for non-preemptive hard real-time tasks with strict period*, In Proceedings of 18th International Conference on Real-Time and Network Systems, RTNSâĂŹ10, Toulouse, France, November 2010.

[2] O. Kermia and Y. Sorel. *Schedulability Analysis for Non-Preemptive Tasks under Strict Periodicity Constraints*, In Proceedings of 14th International Conference on Real-Time Computing Systems and Applications, RTCSA'08, Kaohsiung, Taiwan, August 2008.

[3] M. Joseph and P. Pandya. *Finding response times in a real-time system*, British Computer Society Computer Journal, vol. 29, no. 5, pp. 390-395. 1986.

Mapping real-time applications over multiprocessors

Enrico Bini (Speaker)*

1 Motivation

In distributed embedded systems, the execution of an application is often divided in stages, where each stage consumes the output of the preceding one [5, 6]. Applications are then executed over more than one processor (a distributed platform).

When more than one application executes over the same platform, an incorrect behavior of an application (such as requiring more time than expected) can affect the others, potentially leading to serious malfunctioning. To avoid this interference the execution is often confined in resource reservations [4] that provide a view of a processor *with reduced speed*. Following this design practice, the effects of a misbehavior are confined to the erroneous application only, preventing a domino effect over the entire system. Hence from now on, the computing platform is viewed as a set of m processors with speeds $s_1, \ldots, s_m$ less than or equal to 1.

It is then necessary to decide how to partition the application over the available processors and how to assign their speeds.

2 The problem

In short, the problem can be summarized as follows:

given an application,
find a job partitioning and a speed assignment,
such that the cost J is minimized.

An *application* $\mathcal{A}$ is constituted by a set of *tasks* $\mathcal{A} = \{\tau_1, \ldots, \tau_n\}$, each one with *execution requirement* $e_i \geq 0$ (since tasks execute on speed-variable processors then the time needed by τ_i to complete on a processor with speed s_k is e_i/s_k). Tasks are subject to *precedence constraints* that establish a total order among them. Without loss of generality, we assume a task labeling such that $\forall i = 1, \ldots, n-1$, $\tau_i \prec \tau_{i+1}$.

As tasks are activated, they release a *job* that needs to be executed. The j-th job of task τ_i is denoted by $\tau_{i,j}$. Task τ_1 is activated periodically with period p, hence job $\tau_{1,j}$ is released at $(j-1)p$. Because of the precedence constraint, job $\tau_{i,j}$, with $i \geq 2$, is released upon the completion of $\tau_{i-1,j}$. Task τ_n has a deadline d relative to the activation of τ_1, hence the *absolute deadline* of τ_{nj} is at $(j-1)p+d$. A natural constraint on the deadline d is

$$d \geq \sum_{i=1}^{n} e_i \tag{1}$$

*e.bini@sssup.it. Scuola Superiore Sant'Anna, via G. Moruzzi 1, 56124, Pisa, Italy.

The application *utilization*, defined as

$$u = \frac{\sum_{i=1}^{n} e_i}{p} \tag{2}$$

represents the fraction of time required by the application.

The jobs generated by the tasks have to be partitioned over m processors. The number m of processors is a free variable that needs to be assigned. Each processor π_k has speed $s_k \leq 1$. Without loss of generality the speeds are assumed in decreasing order. Both the partitioning and the speeds have to be chosen such that the application deadline is met. A (simple) necessary condition for the feasibility of the speeds, is

$$\sum_{k=1}^{m} s_k \geq u \tag{3}$$

otherwise the m processors do not provide enough computing capacity to the application.

Figure 1 shows an example of a feasible solution (job partitioning on processors and their speeds) for an application composed by four tasks each one with the same execution requirement $e_i = 1$, period $p = 4$, and deadline $d = 10$.

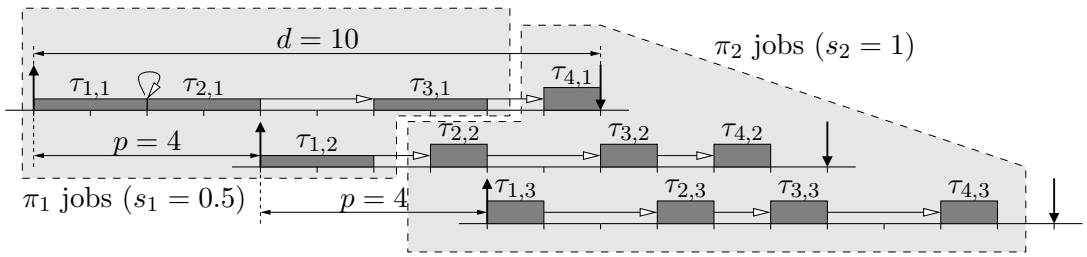


Figure 1: Example of job partitioning over two processors.

Among all the feasible solutions we aim at choosing the one that minimizes a cost J that is a function of the speeds $(s_1, s_2, \ldots, s_m)$. J has a property of being non-decreasing with any s_k and $J(0, \ldots, 0) = 0$. A first example is simply

$$J = \sum_{k=1}^{m} s_k \tag{4}$$

that translates the idea of minimizing the "overall computing power".

In energy minimization problems, the cost J is convex [2]. This property implies that the optimum has balanced values of speeds (in short, $J(s/2, s/2) \leq J(s, 0)$). However, as speeds are interpreted as fractions of computing power provided by a resource reservation, as we do in this paper, it is instead more convenient to have unbalanced speeds. For example, it is more convenient to have one fully dedicated processor (with $s_1 = 1$) rather than two with half speed ($s_1 = s_2 = 0.5$), since in the former case we need to implement fewer reservation mechanisms. This leads to a *concavity* requirement for J that can be formally expressed as follows

$$s_k \geq s_\ell \;\Rightarrow\; \forall \varepsilon \in [0, s_\ell] \quad J(\ldots, s_k, \ldots, s_\ell, \ldots) \geq J(\ldots, s_k + \varepsilon, \ldots, s_\ell - \varepsilon, \ldots) \tag{5}$$

An explicit example of cost with property (5) is

$$J_{\mathsf{frag}} = \max_{\substack{k=1,\ldots,m \\ s_k \neq 0}} \frac{\sum_{j=k}^{m} s_j}{s_k} \tag{6}$$

This expression of cost has been already adopted in a partitioning problem [1]. It is indirectly suggested by Funk et al. who showed, in a different context, that the smaller J_{frag} the higher chances a (Liu and Layland) task set has to be scheduled by global EDF on a platform with speeds $(s_1, \ldots, s_m)$ [3].

We now observe some facts about the problem of minimizing J_{frag}. If $\sum_i e_i \leq p$ then we can partition all jobs on the same processor with speed $s_1 = \frac{\sum_i e_i}{\min\{p,d\}}$ and achieve the minimum cost J_{frag} of 1; If $\sum_i e_i > p$ and $d \leq p$, then the problem is infeasible. Hence we assume

$$\sum_{i=1}^{n} e_i > p, \quad d > p \tag{7}$$

Moreover, let S be the set of a feasible speed assignment. If it exists $S^* \subset S$ with

$$\sum_{s_k \in S^*} s_k \leq 1 \tag{8}$$

then the new set of speeds $\hat{S} = S \setminus S^* \cup \left\{ \sum_{s_k \in S^*} s_k \right\}$ is feasible and, thanks to property (5), has smaller cost J_{frag}. Hence there must exist an optimal solution such that for any pair of speeds (s_k, s_ℓ), we have

$$s_k + s_\ell > 1. \tag{9}$$

References

[1] G. Buttazzo, E. Bini, and Y. Wu, *Partitioning parallel applications on multiprocessor reservations*, in Proceedings of the 22nd Euromicro Conference on Real-Time Systems, Bruxelles, Belgium, 2010, pp. 24–33.

[2] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, *Low-power cmos digital design*, IEEE Journal of Solid-State Circuit, 27 (1992), pp. 473–484.

[3] S. Funk, J. Goossens, and S. Baruah, *On-line scheduling on uniform multiprocessors*, in Proceedings of the 22nd IEEE Real-Time Systems Symposium, London, United Kingdom, Dec. 2001, pp. 183–192.

[4] C. W. Mercer, S. Savage, and H. Tokuda, *Processor capacity reserves: Operating system support for multimedia applications*, in Proceedings of IEEE International Conference on Multimedia Computing and Systems, Boston, MA, U.S.A., May 1994, pp. 90–99.

[5] J. C. Palencia and M. González Harbour, *Schedulability analysis for tasks with static and dynamic offsets*, in Proceedings of the 19th IEEE Real-Time Systems Symposium, Madrid, Spain, Dec. 1998, pp. 26–37.

[6] K. Tindell and J. Clark, *Holistic schedulability analysis for distributed hard real-time systems*, Microprocessing and Microprogramming, 50 (1994), pp. 117–134.

Real-time scheduling analysis for ARINC-based virtualized systems

Philippe Thierry (Speaker) * Laurent George † Jean-François Hermant ‡

1 Introduction

Motivation In the case of partitioned virtualized architectures like avionics ARINC-based systems, multiple virtual machines hosting various task sets with different real-time properties are executed on the same hardware in dedicated time slots. Time slots repartition is defined by an hypervisor so as to ensure time isolation between virtual machines.
The scheduling is hierarchical. We refer to global scheduling the scheduling of time slots by the hypervisor and local scheduling the scheduling of tasks by each virtual machine in its time slot. We consider in this paper a Time Division Multiplexing (TDM) scheduling for the global scheduling of time slots and Earliest Deadline First (EDF) scheduling for local scheduling denoted TDM/EDF. In those architectures, time slots for each virtual machine must be defined depending on the requirements of the associated virtual machine and on the overall system needs.
Defining such time slots can be difficult and depends on the system objectives. Such objectives can be limiting the hypervisor overhead, or reducing the energy consumption for battery operated systems. We focus on a TDM/EDF scheduling that maximizes the remaining time in each slot in order to get a better robustness to WCET variations. We show that the TDM/EDF scheduling problem can be solved by a linear programming approach that we characterize.

2 Related work

Architectures based on a lightweight hypervisor permit concurrent executions of multiple operating systems or software modules with a small and measurable overhead at each virtual machine scheduling. Such architecture problematics are closed to hierarchical scheduling.
Today, some hypervisors supporting real-time scheduling implement ARINC fixed-time partitioning, in order to meet avionics requirements. The ARINC (Aeronautical Radio, Incorporated is a company, which was created in 1929) has defined standards for safety critical systems. One of them, the ARINC 653 norm, defines requirements that should be met in order to support highly critical levels in software architecture, like strict temporal

*philippe.thierry@fr.thalesgroup.com. Thales Communications France, DSC/OPS/SAT/NSS, 160 ave de Valmy, 92704 Colombes, France.
†lgeorge@ieee.org. LISSI, UPEC, 120 rue Paul Armangot, 94400 Vitry sur Seine, France.
‡hermant@ece.fr. LACSC, ECE, 37 quai de Grenelle, 75015 Paris, France.

Slot level variables	*Task level variables*
Considering m slots, we denote: U_i : the load associated to VM_i $SC = \{SC_1, \ldots, SC_m\}$ the Slot duration $\tau = \{\tau_1, \ldots, \tau_m\}$ the Set of task sets in each slot ST_i the ARINC period associated to VM_i SC_i the slot duration given to VM_i	Considering n tasks, we denote: τ_i the task set of the virtual machine VM_i, $\tau_i = \{\tau_i^1, \ldots, \tau_i^n\}$ d_j the deadline of a task τ_i^j of VM_i c_j the WCET of a task τ_i^j of VM_i t_j the period of a task τ_i^j of VM_i

Table 1: Definition of slot level and task level variables

scheduling using TDM slots in order to separate functionalities. Such architectures are based on a specific hierarchical scheduling based on TDM time slots. Thus, the integration of virtual machines makes time slots contents opaque and then unaccessible to the global scheduler.
A lot of work has been done on hierarchical scheduling. In [1], the authors define task groups depending on task priority classes, with a global knowledge of the local tasks to schedule from the global scheduler. This permits using multiple scheduling schemes depending on the priority class, also defining inter-classes priorities. Such global scheduling scheme is based on dynamic behaviour defining inter-classes priorities, as shown in [2]. In this last paper, the authors describe the impact of the VM executions on each others, describing how the choice of the global scheduling policy impacts the response time of virtualized tasks. In such hierarchical scheduling, time slots are dynamically defined, depending on the VMs current needs and on their relative priority. In these two articles, the time slots duration is not a part of the system configuration, but a consequence of the system execution. On the other hand, in [3], the authors explain how communicating real-time functions located in various modules through an AFDX network can be configured in order to reduce the communication delay. This paper contributes, in the case of IMA (Integrated Modular Avionics) systems, to a correlated definition of time slots allocated to the various modules functions, independently of hierarchical scheduling consideration.
The various works done in IMA TDM-based systems and hierarchical scheduling based systems have not considered at the same time virtualization problematics and schedulability of TDM-based architectures like those defined in avionics. The aim of this article is to take into account both TDM and virtualization problematics.

3 Formal constraints of an ARINC-based system schedulability

Some variables are introduced in order to formalize the schedulability of ARINC-based virtualized systems in Table 1.

In order to guarantee the schedulability of an ARINC-based virtualized system architecture, some constraints need to be satisfied at the hypervisor (slot) level. They define restrictions on the ARINC period and the virtual machines scheduling: Equation 1 guarantees a processor utilization less than or equal to 100%. Equation 2 is a necessary condition in order to have a periodic time slot for each VM. Equation 3 is a sufficient

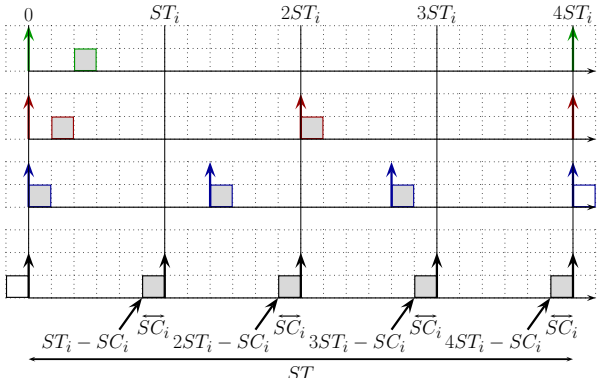


Figure 1: Sample impact on the scheduling scheme

condition to have non-overlapping time slots, as defined in [4]. Figure 1 describes a sample time slots definition.
Some constraints should also be satisfied at the virtual machine (task) level: Equation 4 is a necessary condition which guarantees that the deadline of a task should be greater than the inactivity period of the associated VM. Equation 5 is a necessary and sufficient condition for the feasibility of the task set with EDF. To show that, let us consider Slot i and the tasks scheduled with EDF in that slot.

Linear Programming Problem: LP 1.

Considered Table 1, Given the problem: *Maximize* $min_{i=1...,n} SC_i(1 - U_i)$ **under the constraints:**

Slot level constraints

$$\sum_{i=1}^{m} U_i \leq 1 \quad (1)$$

$$\forall i \in \{1, \ldots, m\}, \frac{ST}{ST_i} \in \mathbb{N}^* \quad (2)$$

$$\sum_{i=1}^{m} SC_i \leq gcd(ST_1, \ldots, ST_m) \quad (3)$$

Tasks level constraints

$$\forall i \in \{1, \ldots, m\}, \forall \tau_i^j \in S_i, d_j > ST_i - SC_i \quad (4)$$

$$\forall k \in \mathbb{N}^*, \forall t \in [kST_i - SC_i, kST_i],$$
$$h(t) \leq t - k(ST_i - SC_i) \quad (5)$$
$$with\ h(t) = \sum_{j=1}^{n} max(0, 1 + \lfloor \frac{t-d_j}{t_j} \rfloor) c_j,$$

Slot i is periodically processed by the Hypervisor with a period set to ST_i time units, and the tasks are scheduled with EDF during SC_i time units in that slot. The first thing to do is to determine the worst-case scenario in terms of feasibility for these tasks. This is achieved when (i) all the tasks are released simultaneously (at time 0) and (ii) Slot i has been previously processed (from time $-SC_i$ to time 0) so that the tasks are executed in the next slot, i.e. $ST_i - SC_i$ time units later. This scenario clearly maximizes both the cumulative workload and the delay experienced by the tasks before their execution. The second thing to do is to write down the feasibility condition for the tasks scheduled with EDF in Slot i when considering the worst-case scenario described previously. All the tasks released in time interval $[0, t]$ with an absolute deadline less than or equal to t are scheduled with EDF and meet their absolute deadlines if and only if they are processed in at most $t - k(ST_i - SC_i)$ time units. This translates into: For all integer k, with $k \geq 1$, for all t in time interval $[kST_i - SC_i, kST_i]$, the processor demand function at

time t, $h(t)$, is less than or equal to $t - k(ST_i - SC_i)$.

References

[1] Giuseppe Lipari, Enrico Bini (2003). *A methodology for designing hierarchical scheduling systems*. Scuola Superiore Sant'Anna, Pisa.

[2] Tommaso Cucinotta, Gaetano Anastasi, Luca Abeni (2009). *Respecting temporal constraints in virtualised services*. Scuola Superior Sant'Anna, Pisa, University of Trento, Italy.

[3] Ahmad Al Sheikh, Olivier Brun, Pierre-Emmanuel Hladik (2010). *Partition Scheduling on an IMA platform with Strict Periodicity and Communication Delay*. CNRS & UTC.

[4] ohamed Marouf, Yves Sorel (2010). *Schedulability conditions for non-preemptive hard real-time tasks with strict period*. INRIA, Université de Nice Sophia-Antipolis.

Probabilistic analysis of periodic real-time tasks with random execution times on identical processors

Liliana Cucu-Grosjean (Speaker) *

1 Introduction

Context of the study Requests in real-time environment are often of a recurring nature. When different instances of those activities are generated in a predictable manner, the scheduler deals with periodic activities. The real-time performances of periodic activities on uniprocessor, distributed or network systems have been extensively studied when worst-case execution times are considered. For these activities, the corresponding response times can be evaluated and such analyses have an increased degree of pessimism. Such pessimism is not affordable or/and necessary for all real-time applications. Different approaches can be considered to avoid this problem (probabilistic approaches [1], agent systems [2], game theory [3], etc) and this paper investigates the use of probabilistic approaches.

Problem definition In this paper we discuss the problem of evaluating the response times of real-time periodic tasks with variable execution times that are scheduled using preemptive fixed-priority global scheduling. By fixed-priority scheduling we mean that during the entire scheduling all jobs of a task have the same priority. By global scheduling we mean that a job might migrate from one processor to other processor. By identical processors we mean that all processors have the same computing power. The communication time between processors are considered negligible compared to the execution times of tasks and included in these latter parameters.

Our contribution We extend the probabilistic response time analysis given in [4] to the case of several processors by providing a new function calculating the response time in the case of several identical processors. To the best of our knowledge this is the first paper providing response time calculations for periodic tasks with execution times given by independent random variables that are scheduled on identical processors under a fixed-priority algorithm.

Model and associated notations We consider $\tau = \{\tau_1, \tau_2, \cdots, \tau_n\}$ a set of n periodic tasks. The set of tasks is ordered in the decreasing order of their priority. Thus, all jobs of τ_i have higher priority than any job of τ_{i+1}, $\forall 1 \leq i \leq n$.

Each task is characterized by an offset O_i, a period T_i and a relative deadline D_i. It means that the j^{th} job of τ_i is released at time instant $O_i + (j-1)T_i$ and must finish its execution by time instant $O_i + (j-1)T_i + D_i$. Each task τ_i has an associated execution time given by a discrete random variable. We denote[1] by $\mathcal{C}_i$ the execution time of a

*liliana.cucu@inria.fr. INRIA Nancy-Grand Est, 615 rue du Jardin Botanique, Villers les Nancy, 54600, France

[1] In this paper we utilise calligraphic letters to denote random variables

task τ_i (see Equation (1)). It is assumed that the random variables giving the execution times are independent.

$$\mathcal{C}_i = \begin{pmatrix} c_k \\ P(C = c_k) \end{pmatrix}_{k \in \{1, \cdots, k_i\}} \tag{1}$$

In Equation (1) $c_k \in [c_i^{min}, c_i^{max}]$ and $k_i \in \mathbb{N}^*$ is the number of values that random variable $\mathcal{C}_i$ has. Using the notations presented before, we define completely a task τ_i by $(O_i, \mathcal{C}_i, T_i, D_i)$.

2 Existing results for the case of one processor

In [4] the authors propose a probabilistic uniprocessor analysis of periodic tasks with execution times given by independent random variables. The response time of a task is the average of response times of all jobs of the task within a given time interval (see Equation (2)). The response time $\mathcal{R}_i^j$ of a job released at $t = O_i + (j-1)T_i$ is obtained by adding to the existing backlog $\mathcal{B}_i(t)$ its own execution time and the execution times of all higher priority jobs that might arrive after its arrival (see Equation (3)). $\mathcal{I}_i$ is the sum of execution times of all these higher priority tasks arrived after t.

$$\mathcal{R}_i = \frac{1}{m_i} \sum_{j=1}^{m_i} \mathcal{R}_i^j \tag{2}$$

$$\mathcal{R}_i^j = \mathcal{B}_i(t) \otimes \mathcal{C}_j \otimes \mathcal{I}_i \tag{3}$$

The backlog at time instant t is obtained as $\mathcal{B}_i(t) = SHRINK(\mathcal{X}, t)$, where $\mathcal{X}$ is the sum of execution times of jobs arrived from 0 to t. Thus the backlog is defined as what is left after scheduling during t time units. For a complete definition of $SHRINK$, see Equation (4).

$$SHRINK(\mathcal{X}, \delta) = \begin{pmatrix} shrink(x_k, \delta) \\ P(X = x_k) \end{pmatrix} \tag{4}$$

where $\mathcal{X} = \begin{pmatrix} x_k \\ P(X = x_k) \end{pmatrix}$ and $shrink(x, \delta) = \begin{cases} 0, & if\ x - \delta < 0 \\ x - \delta, & elsewhere. \end{cases}, \forall x, \delta.$

To prove the stability of their method, Diaz et al. prove that the sequence of backlogs at the end of each hyperperiod $\{\mathcal{B}_1, \mathcal{B}_2, \cdots, \mathcal{B}_k, \cdots\}$ is a Markov chain and a stationary distribution exists when $\overline{U} = \sum_{i=1}^{n} \frac{\overline{C}_i}{T_i}$.

3 Proposed results for the case of m identical processors

We underline by the mean of an example the impossibility of a direct application of the uniprocessor analysis given in [4] to the case of several processors. We consider two identical processors and τ a task system of three periodic tasks with $\tau_1 = (0, \begin{pmatrix} 1 \\ 1 \end{pmatrix}, 3, 3)$, $\tau_2 = (1, \begin{pmatrix} 3 \\ 1 \end{pmatrix}, 6, 6)$ and $\tau_3 = (2, \begin{pmatrix} 1 & 2 \\ 0.5 & 0.5 \end{pmatrix}, 6, 6)$.

If we apply directly the formulation of the backlog $\mathcal{B}_3(2)$ by considering that at each time instant 2 time units are executed, we obtain $\mathcal{B}_i(2) = SHRINK(\mathcal{C}_{1-3}(3), 4) = \begin{pmatrix} 4-4 \\ 1 \end{pmatrix}$. According to this latter result, at time instant 2 there is nothing left to schedule. In reality at time instant 2, the first job of τ_2 has one time unit left to schedule. Therefore an extension of the analysis given in [4] must keep a trace of the tasks that are taken into account. In order to obtain these traces, we build an associated matrix called *Parallel* for any random variable $\mathcal{C}$ defined as

$$Parallel(i,j)(\mathcal{C}) = \begin{cases} (0,0) & if\ j > x_i; \\ (1, P(C = c_i)), & elsewhere. \end{cases} \quad (5)$$

where the number of lines is equal to the number of possible values of $\mathcal{C}$ and the number of columns is equal to the largest possible value of $\mathcal{C}$. From any matrix A we extract the corresponding random variable $\mathcal{X}$ by an inversion process $extract(A) = \mathcal{X}$. We define two operations $\otimes_{Parallel}$ and $shrink_{Parallel}(.,t)$ that extend the operations of convolution of random variables, respectively, the *shrink* function defined in [4].

We modify Equation (3) and we obtain the response time of a job j of task τ_i released at time instant t

$$\mathcal{R}_i^j = extract(B(t) \otimes_{Parallel} Parallel(\mathcal{C}_j) \otimes_{Parallel} Parallel(\mathcal{I}_i)) \quad (6)$$

where $B(t) = shrink_{Parallel}(Parallel(\mathcal{C}_{1-i}), t)$.

Theorem 1. *A stationary distribution exists for the sequence of backlogs at the end of each hyperperiod* $[S_i + kP_i, S_i + (k+1)P_i), \forall i$ *and* k.

Proof. The idea of the proof is similar to the results in [5] based on the periodicity of a feasible schedule and on the fact that at the end of each hyperperiod there is no accumulation of backlog. □

References

[1] Burns, A. and Bernat, G. and Broster, I.(2003). *A Probabilistic Framework for Schedulability Analysis.* Third International Embedded Software Conference.

[2] Teredesai, T. and Ramesh, V.C.(1998). *A multi-agent mixed initiative system for real-time scheduling.* IEEE International Conference on Systems, Man and Cybernetics.

[3] Chryssolouris, G. and Dicke, K. and Lee, M.(1994). *An approach to real-time flexile scheduling.* International Journal of Flexible Manufacturing Systems.

[4] Díaz, J.L and Garcia, D.F. and Kim,K. and Lee, C.G. and Bello, L.L. and López J.M. and Mirabella, O.(2002). *Stochastic Analysis of Periodic Real-Time Systems.* 23rd of the IEEE Real-Time Systems Symposium.

[5] Cucu, L. and Goossens, J.(2007). *Feasibility Intervals for Fixed-Priority Real-Time Scheduling on Uniform Multiprocessors.* the 11th IEEE International Conference on Emerging Technologies and Factory Automation.

Learning in stochastic machine scheduling

Sebastián Marbán (Speaker) * Cyriel Rutten † Tjark Vredeveld ‡

1 Introduction

Over the last few decades a vast amount of research has focused on stochastic scheduling problems, e. g., [2, 4, 6]. A full range of articles is concerned with criteria that guarantee the optimality of simple policies for special scheduling problems or the quality of non-optimal policies. All these papers have in common that the processing times of the jobs are random variables for which the parameters of the underlying distributions, like expected value, are known for certain. In this work, we relax this assumption. That is, we study a stochastic scheduling problem in which also the parameters of the processing time distributions are uncertain. We can learn about the value of these parameters by processing jobs and observing their realized processing times. However, experimenting with different jobs to learn about the value of the corresponding parameters can be costly in terms of the waiting times of the still to be processed jobs. Hence, learning should be conducted carefully in order to optimize the objective.

2 Problem definition

Given is a set of jobs, each of which needs to be scheduled on a single machine. This machine can process at most one job at a time and once a job has been initiated it must remain on the machine until completion, i. e., preemption of jobs is not allowed. Moreover, the machine and all jobs are available for processing from the beginning. The processing time of a job is a random variable. The goal is to minimize the total completion time in expectation, $\sum_j \mathbb{E}[C_j]$.

The set of jobs is divided into two classes. Each class J_i consists of n_i jobs, with $i \in \{1, 2\}$. The processing time of a job in class J_i is a random variable, which is independently and exponentially distributed with parameter ϑ_i. We assume that ϑ_i is unknown but fixed for each i, thereby distinguishing from traditional stochastic scheduling. The scheduler however, may have certain beliefs about ϑ_i. By adopting a Bayesian framework, these beliefs can be represented by a gamma distribution. Depending on the confidence in his beliefs about ϑ_i, the scheduler can either choose this distribution to be very peaked or relatively flat. Furthermore, the scheduler can update his beliefs on ϑ_i every time a job of class J_i is processed. In this way, the scheduler gradually learns about ϑ_i, thereby enhancing his decision making in the future.

*s.marban@maastrichtuniversity.nl. Maastricht University, The Netherlands
†c.rutten@maastrichtuniversity.nl. Maastricht University, The Netherlands
‡t.vredeveld@maastrichtuniversity.nl. Maastricht University, The Netherlands

3 An approximative learning policy

In the scheduling problem under consideration, an optimal policy, OPT, minimizes total completion time in expectation, thereby taking into account the uncertainty about the parameters. That is, in its decision making, OPT will anticipate and act upon the additional information to be revealed when processing a job of a certain class. Burnetas and Katehakis [1] and Hamada and Glazebrook [3] present an optimal policy using an impractical dynamic programming approach. The impracticality arises from the necessity to recursively solve a vast amount of non-linear equations, causing the computational complexity to explode with the size of the instance. This calls for the need to develop policies of low computational effort which yield good qualitative performance. Based on the traditional Shortest Expected Processing Time policy (SEPT), we propose an example of such a policy, the learning policy LSEPT. Whenever the machine is idle, LSEPT starts processing a job of the class with shortest expected processing time. Here, the expected processing time is a weighted average of the observed realizations and the expected processing time prior to seeing any realization. Hence, LSEPT learns by updating the expected processing time of a class every time a job of this specific class has been completed.

In terms of decision making, LSEPT and OPT could be interpreted as having a short-term and long-term view, respectively. LSEPT processes a job from the class with minimal expected processing time. OPT, however, might choose to process a job from a class whose expected processing time is not minimal. As a trade-off, OPT benefits from the additional information which is acquired regarding the uncertain parameter ϑ_i. Since LSEPT does not anticipate on the future revelation of processing time realizations and how they might contribute to learning about the parameter ϑ_i, one might expect LSEPT to perform suboptimal. On the other hand, it is well-known that SEPT is optimal for stochastic single machine scheduling, see [5]. This raises the question how effective LSEPT is within a scheduling problem with unknown processing time distribution parameters.

4 Our results

We show that for two job classes, the performance guarantee, which is the worst-case ratio of the expected performance of LSEPT over the expected performance of OPT, is upper bounded by

$$\frac{n_1^2 + n_2^2 + 2n_1n_2 + n_1 + n_2}{n_1^2 + n_2^2 + n_1 + 3n_2} \leq 2.$$

We also provide an instance for which this bound is tight. To our knowledge, this exemplifies one of the first tight performance guarantees in stochastic scheduling, where the tightness follows from non-degenerate processing time distributions. Finally, we remark that LSEPT is asymptotically optimal whenever the number of jobs of one class remains fixed, while the number of jobs of the second class tends to infinity.

5 Future work

Preliminary results hint that the performance guarantee of LSEPT for m job classes is upper bounded by m and lower bounded by $1 + \sqrt{m-1}$. Proving these bounds still remains to be done. In general, learning policies in stochastic scheduling provide many new interesting open problems.

References

[1] A.N. Burnetas and M.N. Katehakis. On sequencing two types of tasks on a single processor under incomplete information. *Probability in the Engineering and Informational Sciences*, 7(1):85–119, 1993.

[2] B. C. Dean. *Approximation Algorithms for Stochastic Scheduling Problems.* PhD thesis, Massachusetts Institute of Technology, 2005.

[3] T. Hamada and K.D. Glazebrook. A Bayesian sequential single machine scheduling problem to minimize the expected weighted sum of flowtimes of jobs with exponential processing times. *Operations Research*, 41(5):924–934, 1993.

[4] M. Pinedo. *Off-line deterministic scheduling, stochastic scheduling, and online deterministic scheduling: A comparative overview.* In J. Y.-T. Leung, editor, *Handbook of scheduling: Algorithms, Models, and Performance analysis*, chapter 38. Chapman & Hall/CRC, 2004.

[5] M.H. Rothkopf. Scheduling with random service times. *Management Science*, 12(9):703–713, 1966.

[6] M. Uetz. *Algorithms for deterministic and stochastic scheduling.* PhD thesis, Cuvillier Verlag, Göttingen, Germany, 2002.

Energy aware scheduling: minimizing total energy cost and completion time by α-points and α-speeds

Rodrigo A. Carrasco (Speaker) * Garud Iyengar † Cliff Stein ‡

1 Introduction

Since it was first formulated in the early 50's, the minimum weighted completion time problem and its many generalizations have been studied extensively, and many algorithms currently exist that either efficiently solve the polynomially solvable instances or are able to compute an approximate solution with provable guarantees for the strongly NP-hard instances. See [6] for a survey of currently known solution approaches. One specific technique that uses *α-completion points* for computing schedules has been shown to result in very good approximation results on many different scheduling problems, both in theory and in practice [3, 8]. In this paper we use the α-point techniques to compute approximately optimal solutions for energy aware scheduling problems.

Yao, Demers, and Shenker's seminal paper in 1995 [9], introduced the problem of the trade-off between speed and power consumption for scheduling problems. The interest in this problem was first driven by the need of improving the life of battery powered gadgets and later partially by environmental issues but also by the need to control power consumption in the now massive data centres and server clusters some companies have to maintain (e.g., search engines, cloud-computing clusters, etc.). Since energy consumption is not linearly dependent on the job's processing speed, there is a possibility to gain much by reducing the speed without seriously compromising other scheduling constraints or the overall scheduling performance. In one of the most studied models the power consumption is of the form $P(s) = vs^{\beta}$, where s is the speed at which the machine runs each job, v a job dependent weight, and $\beta \geq 2$ is a constant. Hence, the total energy consumed by a job will be $E(s) = v\rho s^{\beta-1}$ where ρ is the processing cycles requirement of the job. Several dynamic speed scaling algorithms have been proposed in [5, 9]. Albers [1] surveys many of the currently known results on energy saving algorithms, including dynamic speed scaling and power down algorithms.

Recently, these two problems have been linked, i.e. one wants to compute both the schedule and the speeds for each job that minimizes some desired performance metric.

*rac2159@columbia.edu. Department of Industrial Engineering and Operations Research, Columbia University, Mudd 313, 500W 120th Street, New York, NY 10027. Research partially supported by NSF grants CCF-0728733 and CCF-0915681, and Fulbright/Conicyt Chile Scholarship.

†garud@ieor.columbia.edu. Department of Industrial Engineering and Operations Research, Columbia University, Mudd 314, 500W 120th Street, New York, NY 10027. Research partially supported by NSF grant DMS-1016571, ONR grant N000140310514, and DOE grant DE-FG02-08ER25856.

‡cliff@ieor.columbia.edu. Department of Industrial Engineering and Operations Research, Columbia University, Mudd 326,500W 120th Street, New York, NY 10027. Research partially supported by NSF grants CCF-0728733 and CCF-0915681.

Some recent results can be found in [2, 4]. In general, current results require some simplifications to obtain a good approximation, like unit sized jobs or polynomial energy cost functions.

The main contribution of this paper is that it presents several new algorithms which extend α-point algorithms to energy aware scheduling problems. The algorithms require only minor regularity conditions on the energy cost functions, and result in approximation ratios that are very close to the scheduling counterparts without any associated energy costs. Furthermore, with a small modification, these algorithms can be used for the weighted tardiness problem as well.

2 Energy Aware Scheduling

The problem setting is as follows. We are given n jobs. Job j, $j = 1, \ldots, n$, has a processing requirement of $\rho_j \in \mathbb{N}_+$ machine cycles, a release time r_j, and an associated positive finite weight w_j. Let s_j denote the speed at which job j runs on the machine and let C_j denote its completion time. Let $\Pi = \{\pi(1), \ldots, \pi(n)\}$ denote the order in which the jobs are processed, i.e. $\pi(k) = j$ implies that job j is the k-th job to be processed. Then the completion time of the j-th job to be processed is $C_{\pi(j)} = \max\{r_{\pi(j)}, C_{\pi(j-1)}\} + \frac{\rho_{\pi(j)}}{s_{\pi(j)}}$, with $C_{\pi(0)} = 0$, i.e., we implicitly assume that preemption is not allowed.

Let $E_j(s_j)$ denote the energy cost of running job j at speed s_j. Initially we consider $E_j(s_j) = v_j \rho_j s_j^{\beta-1}$, where $\beta \geq 2$ and v_j are known constants. Later we show that our algorithm works for more general energy functions $E_i(s_i)$ that satisfy the following three regularity conditions: (1) $E_j(s)$ is convex, (2) $\lim_{s\to 0} E_j(s) > -\infty$, and (3) $\exists \xi < \infty$ such that $E_j(s)$ is increasing $\forall s \geq \xi$.

The objective is to compute a feasible schedule Π, possibly subject to precedence and/or release date constraints, and the vector of speeds $\mathbf{s} = \{s_1, \ldots, s_n\} \in \mathbb{R}_+^n$ that minimizes the total cost,

$$f(\Pi, \mathbf{s}) = \sum_{j=1}^{n} \left[E_j(s_j) + w_{\pi(j)} C_{\pi(j)} \right]. \tag{1}$$

To compute an approximate solution we propose the Schedule by α-intervals and α-speeds (SAIAS) algorithm, with uses an interval-and-speed-indexed LP formulation of the problem, where time is divided into geometrically increasing intervals, with $(1+\epsilon)$ denoting the growth factor. The algorithm requires the computation of α-points similar to the algorithms in [3] to schedule the jobs, but additionally uses the fractional result to define a probability mass function (pmf) over a set of possible speeds. This pmf is later used to compute the speeds at which each job runs, which we call α-speeds.

The main results of this paper can be summarized in the following theorems.

Theorem 1. *The SAIAS algorithm with $\alpha = \frac{1}{2}$ is a $4(1+\epsilon)$-approximation algorithm for the $1||\sum E_j(s_j) + \sum w_j C_j$ and the $1|prec|\sum E_j(s_j) + \sum w_j C_j$ problems, and with $\alpha = \sqrt{2} - 1$ is a $(3 + 2\sqrt{2})(1+\epsilon)$-approximation algorithm for the $1|r_j, prec|\sum E_j(s_j) + \sum w_j C_j$ problem.*

Because the speed of each job is also a decision variable, we can extend the α-point based algorithms to a weighted tardiness objective. The approximation factors are worse,

since the tardiness problem is inherently harder, but these results demonstrate the power of the α-point based algorithms.

The tardiness T_j of job j with due date d_j is defined as $T_j = (C_j - d_j)^+$. Thus, the new objective function is

$$g(\Pi, \mathbf{s}) = \sum_{j=1}^{n} E_j(s_j) + \sum_{j=1}^{n} w_{\pi(j)} \left(C_{\pi(j)} - d_{\pi(j)}\right)^+ . \tag{2}$$

A variant of the SAIAS algorithm (SAIAS-T) is used to compute an approximate solution for this problem. In this algorithm the α-speeds are later increased by a factor γ to ensure that the jobs finish within specified intervals, which in turn guarantees that the tardiness cost in (2) is correctly bounded. We obtain the following result.

Theorem 2. *The SAIAS-T algorithm with* $\gamma = \frac{(1+\epsilon)}{\alpha(1-\alpha)}$ *and* $\alpha = \frac{1}{2}$ *is a* $4^{\beta}(1+\epsilon)^{\beta-1}$*-approximation algorithm for the* $1||\sum E_j(s_j) + \sum w_j T_j$ *and the* $1|prec|\sum E_j(s_j) + \sum w_j T_j$ *problems.*

Finally, the $1|r_j|\sum E_j(s_j) + \sum w_j T_j$ problem requires a new algorithm called Schedule by Rounding and Speeding (SRS), which uses the rounding procedure developed by Shmoys and Tardos [7] together with a speed up factor to achieve the following result,

Theorem 3. *The SRS algorithm is a* $\left(\frac{2}{\epsilon}\right)^{\beta-1}(1+\epsilon)^{\beta}$*-approximation algorithm for the* $1|r_j|\sum E_j(s_j) + \sum w_j T_j$ *problem.*

References

[1] S. Albers (2009). *Algorithms for Energy Saving.* Efficient Algorithms, 173–186.

[2] L.L. Andrew, A. Wierman, and A. Tang (2009). *Optimal speed scaling under arbitrary power functions.* ACM SIGMETRICS Performance v. 37, n. 2, 39.

[3] L.A. Hall, D. Shmoys, and J. Wein (1997). *Scheduling to Minimize Average Completion Time: Off-line and On-line Approximation Algorithms.* Industrial Engineering.

[4] S. Irani and K. Pruhs (2005). *Algorithmic problems in power management.* ACM SIGACT News v. 36, n. 2, 63.

[5] S. Irani, S. Shukla, and R. Gupta (2007). *Algorithms for power savings.* ACM Transactions on Algorithms (TALG) v. 3, n. 4, 41.

[6] K. Pruhs, J. Sgall, and E. Torng (2004). *Online scheduling.* Handbook of scheduling: algorithms, models, and performance analysis, 1–41.

[7] D. Shmoys and E. Tardos (1993). *An approximation algorithm for the generalized assignment problem.* Mathematical Programming, v. 62.

[8] M. Skutella (2006). *List Scheduling in Order of α-Points on a Single Machine.* Online.

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

Meeting deadlines: How much speed suffices?*

S. Anand† Naveen Garg† Nicole Megow (Speaker)‡

We consider the problem of scheduling jobs that arrive online over time at their release dates with hard deadlines on identical parallel machines. Jobs are allowed to be preempted and migrated. A scheduling instance is called *feasible*, if there exists a schedule for all jobs such that no job misses its deadline. The task is to design an online algorithm that finds a schedule for any feasible instance. We call such an algorithm *optimal*. An optimal offline solution can be found easily by solving a maximum flow problem [5]. In the online setting, several algorithms are known to be optimal on a single machine [4]. But on multiple machines, the problem is much more difficult than its offline counterpart. In fact, for $m \geq 2$, there does not exist any optimal online algorithm [4]. In view of this fact, Phillips, Stein, Torng, and Wein [9] proposed the use of *resource augmentation* [6]: Given an online algorithm A we determine the speed $s \geq 1$ such that A is optimal on m speed-s processors for any instance that is feasible for m processors of unit speed. We are interested in the smallest s for which there is an optimal online algorithm. It is known, that any such algorithm needs a speed of at least $6/5$ [9].

An algorithm is *deadline ordered* if the schedule it yields depends only on the relative ordering of the deadlines of the jobs and not on the actual deadlines. A well-known example of a deadline ordered algorithm is *Earliest Deadline First* (EDF) which at any time schedules the m jobs with the earliest deadline. EDF is optimal on a single machine [3], and on m machines, speed $s = 2 - 1/m$ is necessary and sufficient to guarantee its optimality [9]. Since its introduction more than a decade ago, this upper bound on the speed requirement for online algorithms has been improved only slightly. Lam and To [7] proposed a more complex deadline ordered algorithm with a speed requirement of $2 - 2/(m+1)$. They also showed that any deadline ordered online algorithm for m machines needs a speed of at least

$$\alpha_m := \frac{1}{1 - \left(1 - \frac{1}{m}\right)^m}.$$

For $m = 2$ this quantity equals $4/3$, matching the currently best known upper bound [7], and for arbitrary m it is at most $e/(e-1) \approx 1.58$.

Our main result is a new deadline ordered online algorithm which is optimal for speed α_m, and thus, matches the lower bound in [7]. The algorithm and its analysis build on a *yardstick schedule* (YS) which was proposed in [7]. This schedule is constructed online on m speed-1 machines and has the property that all jobs meet their deadlines. But, it is not a feasible schedule as it may process jobs simultaneously on multiple

*This work was done as part of the Indo-German Max Planck Center for Computer Science (IMPECS).

†anand.42@gmail.com, naveen@cse.iitd.ac.in. Indian Institute of Technology Delhi, India.

‡nmegow@mpi-inf-mpg.de. Max-Planck-Institut für Informatik, Saarbrücken, Germany.

machines. Roughly speaking, we consider jobs in EDF-order, let each of them run on a single machine until it is *underworked* (i.e., the total amount of processing done on it is less than the time period since it was released), and from that moment on we run the job simultaneously on as many machines as are available until it is not underworked anymore. Our new algorithm attempts to mimic YS. Every job finishes in our schedule at the same time as in YS. However, our algorithm does not process jobs simultaneously on multiple machines, and hence, it requires extra speed to keep up with YS. We will show how to distribute work load, depending on schedule YS, to the extra time available in each unit-length time slot in our schedule.

Our new algorithm leads to improved feasibility tests for periodic [1] and sporadic [2] multi-processor real-time task systems. To see that we prove a relationship between the optimality of the yardstick schedule and a lower bound on the total workload, which can be computed by a fully polynomial time approximation scheme [2].

We also consider two well-known non-deadline ordered algorithms and provide lower bounds on the speed necessary for them to schedule a feasible instance. Algorithm *Least Laxity First* (LLF) schedules at any point in time m jobs with minimum laxity (i.e., the difference between the time remaining until the deadline and the remaining processing requirement) among the available jobs. LLF is known to be optimal on a single machine [4], and on m machines when they have speed $2 - 1/m$ [9]. We provide a lower bound on the speedup by demonstrating a feasible scheduling instance for which LLF requires a speedup of at least $\frac{1+\sqrt{1+4x^2}}{2x}$ where $x = \frac{m}{m-1}$. This quantity is $\frac{1+\sqrt{17}}{4} \approx 1.281$ for $m = 2$ and tends to $\frac{1+\sqrt{5}}{2} \approx 1.618$ when m goes to infinity. Notice that for $m \geq 7$ the lower bound on the speed for LLF exceeds the upper bound on the speed sufficient for our new algorithm.

An algorithm that tries to combine features of EDF and LLF is the *Earliest Deadline until Least Laxity* (EDZL). At any point in time, EDZL gives highest priority to jobs which cannot be delayed further, i.e, have zero laxity, and other jobs are scheduled in EDF order. This algorithm dominates EDF [8], and simulations in real-time systems environments with recurrent jobs show that it performs very well. However, we show that EDZL is not optimal for speed less than $2 - 1/m$.

References

[1] V. Bonifaci, H.-L. Chan, A. Marchetti-Spaccamela, and N. Megow. Algorithms and complexity for periodic real-time scheduling. In *Proc. of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1350–1359, 2010.

[2] V. Bonifaci, A. Marchetti-Spaccamela, and S. Stiller. A constant-approximate feasibility test for multiprocessor real-time scheduling. To appear in *Algorithmica*, 2011. DOI: 10.1007/s00453-011-9497-2.

[3] M. L. Dertouzos. Control robotics: The procedural control of physical processes. In *IFIP Congress*, pages 807–813, 1974.

[4] M. L. Dertouzos and A. K. Mok. Multiprocessor on-line scheduling of hard-real-time tasks. *IEEE Transactions on Software Engineering*, 15(12):1497–1506, 1989.

[5] W. A. Horn. Some simple scheduling algorithms. *Naval Research Logistics Quarterly*, 21(1):177–185, 1974.

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

[7] T. W. Lam and K.-K. To. Trade-offs between speed and processor in hard-deadline scheduling. In *Proc. of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 623–632, 1999.

[8] M. Park, S. Han, H. Kim, S. Cho, and Y. Cho. Comparison of deadline-based scheduling algorithms for periodic real-time tasks on multiprocessor. *IEICE Transactions*, 88-D(3):658–661, 2005.

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

Mixed-Criticality Scheduling

S. K. Baruah [*] V. Bonifaci [†] G. D'Angelo [‡] H. Li [*]
A. Marchetti-Spaccamela [§] N. Megow [†] L. Stougie (Speaker) [¶]

There is an increasing trend in embedded systems towards implementing multiple functionalities upon a single shared computing platform. This can force tasks of different criticality to share a processor and interfere with each other. These *mixed-criticality* (MC) systems are the focus of our research. We consider the scheduling of finite collections of jobs to be executed on a single machine (processor), allowing preemption.

A job in an MC system with L criticality levels is characterized by a 4-tuple of parameters: $J_j = (r_j, d_j, \chi_j, c_j)$, where r_j is the release time, d_j is the deadline ($d_j \geq r_j$), $\chi_j \in \{1, \ldots, L\}$ is the *criticality level* of the job and c_j is an L-tuple $(c_j(1), \ldots, c_j(L))$ representing the *worst-case execution times* (WCET) of job J_j at level $1, \ldots, L$, respectively. Each job J_j in a collection $J_1, \ldots, J_n$ should receive execution time C_j within time window $[r_j, d_j]$. The value of C_j is not known but is discovered by executing job J_j until it signals completion. A collection of realized values $(C_1, C_2, \ldots, C_n)$ is called a scenario. The criticality level of a scenario $(C_1, \ldots, C_n)$ is defined as the smallest integer ℓ such that $C_j \leq c_j(\ell)$, $\ell = 1, \ldots, L$. (We only consider scenarios where such an ℓ exists.) A schedule for a scenario $(C_1, \ldots, C_n)$ of criticality ℓ is *feasible* if every job J_j with $\chi_j \geq \ell$ receives execution time C_j during its time window $[r_j, d_j]$. Notice the crucial aspect of this model that, in a scenario of level ℓ, it is necessary to guarantee only that jobs of criticality at least ℓ are completed before their deadlines. In other words, once a scenario is known to be of level ℓ, the jobs of criticality $1, \ldots, \ell - 1$ can safely be dropped. Throughout we will assume that $c_j(\ell) \geq c_j(k)$ if $\ell > k$ and that for all j, $c_j(\ell) = c_j(\chi_j)$ for all $\ell > \chi_j$.

A *clairvoyant* scheduling policy knows the scenario of I, i.e., $(C_1, \ldots, C_n)$, prior to determining a schedule for I. We call an instance I *clairvoyantly-schedulable* if for each scenario of I there exists a feasible schedule.

By contrast, an *on-line* scheduling policy discovers the value of C_j only by executing J_j until it signals completion. In particular, the criticality level of the scenario becomes known only by executing jobs. An on-line scheduling policy is *correct* for instance I if for any scenario of instance I the policy generates a feasible schedule.

An instance I is *MC-schedulable* if it admits a correct on-line scheduling policy.

The MC-SCHEDULABILITY problem is to determine whether a given instance I is MC-schedulable or not. It is easy to see that for deciding MC-schedulability one only

[*] baruah@cs.unc.edu, lihaohan@cs.unc.edu. University of North Carolina, USA.
[†] bonifaci@mpi-inf.mpg.de, nmegow@mpi-inf.mpg.de. Max-Planck-Institut für Informatik, Saarbrücken, Germany.
[‡] gianlorenzo.dangelo@univaq.it. University of L'Aquila, Italy.
[§] alberto@dis.uniroma1.it. Sapienza University of Rome, Italy.
[¶] lstougie@feweb.vu.nl. Vrije Universiteit Amsterdam & CWI, the Netherlands.

needs to consider scenarios in which for each i, $C_i = c_i(\ell)$ for some ℓ.

Example. Consider an instance I of a *dual-criticality* system: $L = 2$. I has 2 jobs:

$J_1 = (0, 2, 1, (1, 1))$, $J_2 = (0, 3, 2, (1, 3))$

Here, any scenario in which C_1 and C_2 are no larger than 1, has criticality 1; all other scenarios we consider have criticality 2. It is easy to verify that I is clairvoyantly-schedulable. The following describes an on-line scheduling policy for instance I:

S_0: Execute J_2 over [0,1]. If J_2 has no remaining execution (i.e., C_2 is revealed to be no greater than 1), then continue with scheduling J_1 over $(1, 2]$; else continue by completing scheduling J_2.

It is easy to see that policy S_0 is correct for instance I. However, S_0 is not correct if we modify the deadline of J_1 obtaining the following instance I':

$J_1 = (0, 1, 1, (1, 1))$, $J_2 = (0, 3, 2, (1, 3))$

It is easy to see that I' is clairvoyantly schedulable but not MC-schedulable.

With respect to complexity we prove that MC-SCHEDULABILITY is strongly NP-hard even if $L = 2$. We do not know if the problem belongs to NP. It does belong to PSPACE. For L constant the problem is in NP, hence NP-complete. Certain subcases are polynomial time solvable, for instance the case that all jobs have equal deadlines.

Since MC-SCHEDULABILITY is intractable we concentrate here on sufficient (rather than exact) MC-schedulability conditions that can be verified in polynomial time. We study two widely-used scheduling policies that yield such sufficient conditions and compare their capabilities under the *resource augmentation metric*: the minimum speed of the processor needed for the algorithm to schedule all instances that are MC-schedulable on a unit-speed processor. We show that the second policy we present outperforms the first one in terms of the resource augmentation metric.

The first, straightforward, approach is to map each MC job J_j into a "traditional" job with the same arrival time r_j and deadline d_j and processing time $c_j = c_j(\chi_j) = \max_\ell c_j(\ell)$ (by monotonicity), and determine whether the resulting collection of traditional jobs is schedulable using some preemptive single machine scheduling algorithm such as the *Earliest Deadline First* (EDF) rule. This test can clearly be done in polynomial time. We will refer to mixed-criticality instances that are MC-schedulable by this test as *worst-case reservations schedulable* (WCR-schedulable) instances.

Theorem 1. *If an instance is WCR-schedulable on a processor, then it is MC-schedulable on the same processor. Conversely, if an instance I with L criticality levels is MC-schedulable on a given processor, then I is WCR-schedulable on a processor that is L times as fast, and this factor is tight.*

The second approach is a *fixed priority policy*: Off-line, before the actual execution times are known, a priority list of the jobs is determined and at each moment in time the available job with the highest priority is scheduled. The priority list is constructed recursively using the approach commonly referred to in the real-time scheduling literature as the "Audsley approach" [1,2]; it is also related to a technique introduced by Lawler [6]. First determine the lowest priority job: Job J_i has lowest priority if there is at least $c_i(\chi_i)$ time between r_j and d_j its release time and its deadline available when every other job J_j is executed before J_i for $c_j(\chi_i)$ time units (the WCET of job J_j according to the criticality level of job i). The procedure is repeatedly applied to the set of jobs excluding the lowest

priority job, until all jobs are ordered, or at some iteration a lowest priority job does not exist.

Because the priority of a job is based only on its own criticality level, the instance I is called *Own Criticality Based Priority (OCBP)-schedulable* if we find a complete ordering of the jobs. If at some recursion in the algorithm no lowest priority job exists, we say the instance is not OCBP-schedulable. Clearly, if a priority list exists, it can be determined in polynomial time.

Theorem 2. *If an instance is OCBP-schedulable on a processor, then it is MC-schedulable on the same processor. Conversely, if instance I with L criticality levels is MC-schedulable on a given processor, then I is OCBP-schedulable on a processor that is s_L times as fast, with s_L equal to the root of the equation $x^L = (1+x)^{L-1}$, and this factor is tight. Furthermore, it holds that $s_L = \Theta(L/\ln L)$.*

We note that for $L = 2$ in the above theorem, $s_2 = (1+\sqrt{5})/2$ is equal to the golden ratio ϕ. We show that under fixed priority policies OCBP is in a sense best possible, by proving that instances with L criticality levels exist, that are clairvoyantly schedulable, but not Π-schedulable for any fixed priority policy Π on a machine that is less that s_L times as fast, with s_L being the root of the equation $x^L = (1+x)^{L-1}$.

Related work. The mixed-criticality model presented here has first been proposed and analyzed by Baruah, Li and Stougie [4]. Most of the results presented appear in Baruah et al. [3]. The mixed-criticality model has been extended to task systems by Li and Baruah [7] and by Bonifaci et al. [5].

References

[1] N. C. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical report, The University of York, England, 1991.

[2] N. C. Audsley. *Flexible Scheduling in Hard-Real-Time Systems*. PhD thesis, Department of Computer Science, University of York, 1993.

[3] S. K. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow, and L. Stougie. Scheduling real-time mixed-criticality jobs. In P. Hliněný and A. Kučera, editors, *Proc. 35th Symp. on Mathematical Foundations of Computer Science*, volume 6281 of *Lecture Notes in Computer Science*, pages 90–101. Springer, 2010.

[4] S. K. Baruah, H. Li, and L. Stougie. Towards the design of certifiable mixed-criticality systems. In *Proc. 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 13–22. IEEE, 2010.

[5] V. Bonifaci, G. D'Angelo, A. Marchetti-Spaccamela, S. van der Ster, and L. Stougie. Mixed-criticality scheduling of sporadic task systems. Submitted to *10th Workshop on Models and Algorithms for Planning and Scheduling Problems*, 2011.

[6] E. L. Lawler. Optimal sequencing of a single machine subject to precedence constraints. *Management Science*, 19(5):544–546, 1973.

[7] H. Li and S. K. Baruah. An algorithm for scheduling certifiable mixed-criticality sporadic task systems. In *Proc. 16th IEEE Real-Time Systems Symposium*, pages 183–192. IEEE, 2010.

Using oracles for the design of efficient approximation algorithms

Marin Bougeret (Speaker) [*] Pierre-Francois Dutot [†] Denis Trystram [‡]

We are interested here in oracle techniques for the design of approximation algorithms. Following the classical definition, an oracle is a black box capable of answering correctly and instantaneously any question. Several classical $PTAS$ design techniques can be expressed using oracle formalism (by allowing the algorithm to "guess" some values during the computation).

Our objective in this work is to point out the interest of oracle techniques, beyond the design of $PTAS$. Indeed, questions to the oracle (*i.e.* guessed values) leading to non polynomial algorithms must also be considered, as the complexity may be exponential, but in a parameter that is supposed to be "small". Moreover, we aim at showing how it is possible to "degenerate" questions asked to the oracle to derive fast implementations of these interactive algorithms. These ideas will be illustrated on the classical makespan minimization on uniform machines problem ($Q||C_{max}$).

Context : oracle algorithms

Given an instance I of an optimization problem, an oracle algorithm A_{or} asks the oracle for a *guess*, in the form of a string $r_I^* \in R_I$, that generally provides some information on the structure of an optimal solution. Then, the algorithm constructs a solution $A_{or}(I, r_I^*)$ for the initial problem. From such an algorithm, it is possible to derive a "classical" algorithm A (without oracle), by either re-executing $A_{or}(I, r)$ for any $r \in R_I$, or constructing separately r_I^* (using another algorithm). Taking the example of scheduling problems, a very classical question r_I^* is an "optimal configuration" of a well-chosen small subset of k tasks (among the n of the instance), where k is constant. Such information may allow arbitrarily good approximation ratios (like $1 + \frac{1}{k}$) at the price of subset enumeration, when simulating the oracle.

Hence, it is clear that there exist deep connections between oracle algorithms and techniques for designing approximation schemes. As shown in [1], an oracle formulation allows natural alternative definitions of several classical techniques (as those presented in [9]). Most of such techniques are based on information obtained by exhaustive enumeration or by binary search. Replacing them by oracle answers separates difficulties due to the information determination from the ones due to its utilization.

[*] marin.bougeret@ens-lyon.fr. Ecole Normale Superieure de Lyon, France
[†] pierre-francois.dutot@imag.fr. Grenoble University, France
[‡] trystram@imag.fr. Grenoble University and Institut Universitaire de France

Application on the classical $Q||C_{max}$ problem

Let us consider the problem of minimizing the makespan when scheduling independent tasks on uniform machines as a case study. It is shortly denoted by $Q||C_{max}$.

Several approximation algorithms have been proposed for this problem. The 2 ratio (achieved by the classical Longest Processing Time algorithm [4]) has been improved to $\frac{3}{2}$ in [5] (using the dual approximation technique), and to 1.382-approximation in [2]. Among all existing approximation schemes, the most relevant here are the following (we list below the time complexity to achieve a ratio of $(1+\epsilon)$):

- $\mathcal{O}(mn^{\frac{10}{\epsilon^2}+3})$ in [5]
- $\mathcal{O}((\frac{1}{\epsilon}n^2)^{m-1})$ (also applies to $R||C_{max}$) in [6]
- $\mathcal{O}((n+1)^{\frac{m}{\epsilon}}poly(n,m))$ (also applies to $R||C_{max}$) in [8]
- $\mathcal{O}(n)+(\frac{log(m)}{\epsilon})^{\mathcal{O}(m^2)}$ (also applies to $R|c_{ij}|C_{max}$) in [3]
- $\mathcal{O}(2^{\mathcal{O}(1/\epsilon^2 log(1/\epsilon)^3)}poly(n,m))$ in [7]

We propose an oracle algorithm based on [5]. For any $a \in \mathbb{N}^*$, our algorithm guarantees an $1+\frac{1}{a}$ ratio by asking *some information* on the "big" tasks scheduled on each machine (*i.e.* whose computation requires more than a fraction $\frac{1}{a}$ of the total computation time on this machine).

Firstly, notice that the classical guess (*i.e.* asking the index of the big tasks scheduled on each machine) would lead to an approximation scheme with the same complexity as the one in [8]. Thus, we show how to reduce the amount of information asked, and thus the size of R_I, for small values of a (typically $a=3$ or 4). We get for instance a $\frac{4}{3}$ (resp. a $\frac{5}{4}$)-approximation by only asking the number of big tasks for each machine, leading to an algorithm in $\mathcal{O}(2^m poly(n,m))$ (resp.$\mathcal{O}(3^m poly(n,m))$). Thus, these approximation algorithms can be faster than the better approximation schemes applied for ϵ equal to $\frac{1}{3}$ (resp. $\frac{1}{4}$), as there is no constant hidden in the exponent.

Secondly, we discuss efficient implementations where the algorithms avoid asking some sub-parts of the question. The key idea is to check if some additional *a priori* unexpected conditions become true during the execution on the particular current instance, allowing then to make a local optimal decision (without oracle query).

The approach presented in this work leads to the following natural questions for $Q||C_{max}$.

- Using only one bit of information for each machine, what information should be asked to obtain a ratio better than $\frac{4}{3}$?
- How to reduce the amount of information used for larger values of a ?

References

[1] M. Bougeret. Systèmes interactifs our la résolution de problèmes complexes. PhD Thesis, 2010. http://moais.imag.fr/membres/marin.bougeret/publis/these.pdf.

[2] B. Chen. Tighter bound for MULTIFIT scheduling on uniform processors. *Discrete Applied Mathematics*, 31(3):227–260, 1991.

[3] A.V. Fishkin, K. Jansen, and M. Mastrolilli. Grouping techniques for scheduling problems: simpler and faster. *Algorithmica*, 51(2):183–199, 2008.

[4] Teofilo Gonzalez, Oscar H. Ibarra, and Sartaj Sahni. Bounds for lpt schedules on uniform processors. *SIAM Journal on Computing*, 6(1):155–166, 1977.

[5] D. Hochbaum and D. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. *SIAM Journal on Computing*, 17(3):539–551, 1988.

[6] Ellis Horowitz and Sartaj Sahni. Exact and approximate algorithms for scheduling nonidentical processors. *Journal of the ACM (JACM)*, 23(2):317–327, 1976.

[7] K. Jansen. An EPTAS for scheduling jobs on uniform processors: using an MILP relaxation with a constant number of integral variables. *Automata, Languages and Programming*, pages 562–573, 2009.

[8] J.K. Lenstra, D.B. Shmoys, and E. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46(1):259–271, 1990.

[9] P. Schuurman and G. Woeginger. Approximation schemes - a tutorial. In *Lectures on Scheduling*, 2000.

On the Configuration-LP for Scheduling on Unrelated Machines

José Verschae Andreas Wiese (Speaker) *

1 Introduction

One of the most prominent open problems in machine scheduling is scheduling jobs on unrelated machines to minimize the makespan, denoted by $R||C_{\max}$ in the three-field notation. We are given n jobs, m machines, and processing times $p_{i,j}$ for each job j on each machine i. The goal is to assign the jobs to the machines to minimize the overall makespan, i. e., the time when the last machine finishes.

In a seminal work, Lenstra, Shmoys, and Tardos [7] present a 2-approximation algorithm for the problem. The algorithm is based on a canonical linear program formulation. In the same paper, they proved that it is NP-hard to approximate the problem with a factor better than 3/2. The gap between 3/2 and 2 has persisted for more than 20 years, even though the problem is considered to be very important in the scheduling community.

The best known approximation algorithms for this problem and its special cases are derived by linear programming techniques [2,6,7]. A special role plays the *configuration-LP*. It is the strongest linear program for the problem considered in the literature and it implicitly contains a vast class of inequalities. In fact, for the most relevant cases of $R||C_{\max}$ the best known approximation factors match the best known upper bounds on the integrality gap of the configuration-LP. For the restricted assignment case, the LP is even the only known linear program which yields the respective bound [2].

There are two interesting special cases of the problem: the restricted assignment case and the unrelated graph balancing case. In the *restricted assignment* case, for each job j there is a value p_j such that for all machines i we have that $p_{i,j} \in \{p_j, \infty\}$. In the *unrelated graph balancing* case each job can be assigned to at most two machines (but with possibly different processing times). These two cases are sort of perpendicular to each other. One of our main results is the analysis of the configuration-LP for the general case of $R||C_{\max}$ and for the unrelated graph balancing case. For both cases, we show that the configuration-LP has an integrality gap of 2 and hence it cannot help to obtain a better approximation factor than 2.

A related problem which has drawn a lot of attention recently is the *MaxMin-allocation* problem. In that problem we are also given a set of jobs, a set of unrelated machines and processing times $p_{i,j}$ as before. The load of a machine i, denoted by ℓ_i, is the sum of the processing times assigned to machine i. The objective is to maximize

*both authors are from TU Berlin, Institut für Mathematik, Straße des 17. Juni 136, 10623 Berlin, Germany. {verschae,wiese}@math.tu-berlin.de.

the minimum load of the machines, i.e., to maximize $\min_i \ell_i$. The idea behind this objective function is a fairness property: Consider that jobs represent resources that must be assigned to machines. Each machine i has a personal valuation of job (resource) j, namely $p_{i,j}$. The objective of maximizing the minimum machine load is equivalent to maximizing the total valuation of the machine that receives the least total valuation.

In contrast to $R||C_{\max}$, here the configuration-LP has a super-constant integrality gap of $\Omega(\sqrt{m})$ [8]. This is tight up to logarithmic factors due to a result by Asadpour and Saberi [9] who constructively show an upper bound of $O\left(\sqrt{m}\log^3 m\right)$ on the integrality gap. The problem can be subdivided into the same cases as $R||C_{\max}$. For the restricted assignment case, the configuration-LP performs much better. The best known upper bound on its integrality gap is due to Asadpour, Feige, and Saberi [4] who prove an upper bound of 4. Unfortunately, this does not yield a polynomial time approximation algorithm. The best known approximation algorithm achieves a performance ratio of $O(1)$, see [10, 11].

For the special case that every job can be assigned to at most two machines (but still with possibly different execution times on them) Chakrabarty et al. [5] show that the configuration-LP has an integrality gap of 2, yielding a $(2+\varepsilon)$-approximation algorithm. Moreover, it is NP-hard to approximate even this special case with a better ratio than 2 [5]. In fact, the proof uses only jobs which have the same processing time on their two respective machines.

2 Our Contribution

As mentioned before, our main result for the minimum makespan problem is that the configuration-LP has an integrality gap of 2, even in the case of unrelated graph balancing. This implies that any set of cuts that involves only one machine per inequality cannot help to improve the integrality gap of the LP-relaxation of Lenstra et al. [7]. Recall that for the restricted assignment case the configuration-LP has an integrality gap of $33/17 < 2$ [2]. Hence, our result gives an indication that the core complexity of $R||C_{\max}$ lies in the unrelated graph balancing case rather than in the restricted assignment case. In particular, our instances use processing times from the set $\{\varepsilon, 1, \infty\}$. For this case, Svennson [2] proves even an upper bound of $5/3+\varepsilon$ for the integrality gap of the configuration-LP for the restricted assignment problem.

Additionally, we study special cases for which we obtain better approximation factors than 2. In particular, we obtain a $1+5/6$ approximation guarantee for the special case of $R||C_{\max}$ where the processing times belong to the set $[\gamma, 3\gamma] \cup \{\infty\}$ for some $\gamma > 0$. Note that the strongest known NP-hardness reductions create instances with this property. Moreover, we show that there exists a $(2 - g/p_{\max})$-approximation algorithm, where g denotes the greatest common divisor of the processing times, and $p_{\max}$ the largest finite processing time.

We also consider restricted cases of the MaxMin-allocation problem. Our main result for this problem is in the unrelated graph balancing setting, for which we present a simple purely combinatorial algorithm with quadratic running time which has a performance guarantee of 2. This improves on the LP-based $(2+\varepsilon)$-approximation algorithm by Chakrabarty et al. [5]. Their algorithm resorts to the ellipsoid method to approximately solve a linear program with exponentially many variables where the separation problem of the dual is the KNAPSACK problem which is solved only approximately. Our algorithm

is significantly simpler to implement and moreover best possible, unless $P = NP$.

We refer to our technical report [3] for all details of our results.

References

[1] D. B. Shmoys and É. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, 62:461–474, 1993.

[2] O. Svensson. Santa Claus Schedules Jobs on Unrelated Machines. *ArXiv e-prints*, November 2010.

[3] J. Verschae and A. Wiese. On the configuration-LP for scheduling on unrelated machines. Technical Report 025-2010, Technische Universität Berlin, November 2010.

[4] A. Asadpour, U. Feige, and A. Saberi. Santa claus meets hypergraph matchings. In *Proceedings of the 11th International Workshop and 12th International Workshop on Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX-RANDOM 2008)*, volume 5171 of *Lecture Notes in Computer Science*, pages 10–20. Springer, 2008. Approximation factor 4 shown in the technical report, available for download at `http://www.stanford.edu/~asadpour/publication.htm`.

[5] D. Chakrabarty, J. Chuzhoy, and S. Khanna. On allocating goods to maximize fairness. In *Proceedings of the 50th Annual Symposium on Foundations of Computer Science (FOCS 2009)*, pages 107–116, 2009.

[6] T. Ebenlendr, M. Krčál, and J. Sgall. Graph balancing: a special case of scheduling unrelated parallel machines. In *Proceedings of the 19th annual ACM-SIAM symposium on Discrete algorithms (SODA 2008)*, pages 483–490, 2008.

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

[8] N. Bansal and M. Sviridenko. The santa claus problem. In *Proceedings of the 38th annual ACM symposium on Theory of computing (STOC 2006)*, pages 31–40, 2006.

[9] A. Asadpour and A. Saberi. An approximation algorithm for max-min fair allocation of indivisible goods. In *Proceedings of the 39th annual ACM symposium on Theory of computing (STOC 2007)*, pages 114–121, 2007.

[10] U. Feige. On allocations that maximize fairness. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2008)*, pages 287–293, 2008.

[11] B. Haeupler, B. Saha and A. Srinivasan. New Constructive Aspects of the Lovasz Local Lemma. http://adsabs.harvard.edu/abs/2010arXiv1001.1231H.

Scheduling Jobs in Parallel for Energy Savings

Jessica Chang* Harold Gabow † Samir Khuller (Speaker)‡

1 Introduction

We study the following pre-emptive scheduling problem. We are given a collection of n jobs, each job J_i has an integer length ℓ_i and a set T_i of time intervals with integer boundaries in which it can be feasibly scheduled. (When T_i is one interval, this is equivalent to having a release time and deadline.) It is assumed that the sum of the lengths of intervals in T_i is at least ℓ_i. For ease of notation, we may sometimes refer to job J_i as job i. Additionally, time is slotted and for a given parallelism parameter P, the system (or machine) can schedule up to P jobs at a time slot. One can think of P as the number of processors. If the machine satisfies any jobs at time slot t, we say that the machine is "active at time t". The goal is to pre-emptively schedule all jobs, i.e. satisfy them completely within their feasible regions, while minimizing the number of slots during which the machine is active. The machine consumes a fixed amount of energy per active slot. In other words, subject to each job J_i being scheduled within its feasible region T_i, and subject to at most P jobs being scheduled at any time, we would like to minimize the total time spent satisfying the jobs. Note that there may be instances when there is no feasible schedule for all the jobs (however this case is easy to check).

Motivation: Power management strategies have been widely studied in the scheduling literature. Many of the models are motivated by the energy consumption of the processor. Consider, alternatively, the energy consumed by the operation of large storage systems. Data is stored in memory which may turned on and off, and each task or job needs to access a subset of data items to run. At each time step, the scheduler can satisfy at most one unit of work from each of at most P jobs. The only requirement is that the data required by the satisfied jobs should be contained in the memory banks that are on. Note that for a time slot significantly large (e.g. on the order of hours), the overhead cost for starting a memory bank is negligible compared to the energy spent being "on" for that unit of time. The problem studied in this paper is the special case where all the data is in one memory bank.

More broadly, consider the following operational problem. Suppose that a ship can carry up to P cargo containers from one port to another. Jobs have delivery requirements leading to release times and deadlines. Finding an optimal schedule corresponds to the minimum number of times we need to send the ship to deliver all the packages on time.

*Dept. of Computer Science and Engineering, University of Washington, Seattle WA 98195, jschang@cs.washington.edu. Research supported by an NSF Graduate Research Fellowship.

†University of Colorado, Boulder CO 80309, Harold.Gabow@colorado.edu.

‡Dept. of Computer Science, University of Maryland, College Park MD 20742, samir@cs.umd.edu. Research supported by NSF CCF-0728839, NSF CCF-0937865 and a Google Research Award.

The motivating assumption is that it costs roughly the same to send the ship, regardless of load and that there is an upper bound on the load capacity.

In the scheduling literature, often problems with unit processing times are trivial since they can be solved using matching techniques when time is slotted. When time is not slotted, but jobs have to be scheduled non-preemptively, then several papers have considered the feasibility question [3, 5]. However, in different models which allow for overlap in job satisfaction, e.g. broadcast scheduling, the problems often turn out to be NP-complete; in fact, several variants of broadcast scheduling have been shown to be NP-complete. The problem considered in this paper also contains an element of "overlap" since we can schedule up to P jobs in a slot at unit cost and wish to minimize the number of activated slots.

A related problem one might consider is the bipartite matching problem in which each node on the left needs to be matched with a node on the right. Each node on the right has a capacity of P, and we are interested in minimizing the number of nodes on the right that have at least one node from the left assigned to it. This problem can easily be shown to be NP-hard.

When we have unit length jobs and each T_i has a single interval, then the problem can be viewed as as the problem of stabbing the intervals with the least number of vertical lines. When the stabbers have a capacity, then dynamic programming can be used to develop a polynomial time algorithm [1], albeit with high complexity. Hence it is slightly surprising that for unit length jobs and single interval T_i, we can develop a fast (greedy) algorithm to obtain an optimal solution to the scheduling problem defined above. Our algorithm is a greedy scheme, which intuitively abides by a lazy activation principle: schedule jobs in batches of size up to P delaying the batch as long as possible. At each step, we select "filler" jobs (with later deadlines) to fill slots which otherwise would have at least one and less than P jobs, based on an Earliest Deadline First (EDF) strategy. The algorithm as described does not quite work, since we may schedule some jobs using the lazy activation principle and later discover that these jobs should have been scheduled earlier to make space for other jobs with later deadlines. One way to address this problem is to dynamically re-assign jobs to time slots. Our first attempt was based on this idea, but it resulted in a slower algorithm with a more complicated analysis. However, we are able to address this issue by pre-processing the jobs to create a new instance with "adjusted" deadlines, so that at most P jobs have the same deadline. Then, no re-assignment of jobs is required.

This work proposes a simple model for measuring energy usage on a parallel machine. We could also consider this as a basic form of "batch" processing. Two papers consider the closely related problem of minimizing "busy time" [2, 4].

It would be interesting to consider the online model where along with a given set of (offline) jobs, new jobs arrive over time.

Main Results: For the case where jobs are unit length and T_i forms a single interval, we develop an optimal algorithm for this problem whose running time is $O(n \log n)$. For notational convenience, we specify T_i by the pair (r_i, d_i), where r_i and d_j are J_i's integer release time and deadline, respectively. Our algorithm takes n jobs as input with release times and deadlines and outputs a schedule with the smallest number of active slots.

In addition, we consider the generalization to arbitrary T_i. In this case, the complex-

ity of the problem depends on the value of P, since for any fixed $P \geq 3$, the problem is NP-hard. When $P = 2$ this problem can be solved optimally in $O(m\sqrt{n})$ time where m is the total number of time slots which are feasible for some job.

Our algorithm can be extended to other versions of the power minimization problem. For example the following result models a situation where power is limited and we wish to schedule the maximum number of jobs that can be done in α active slots. For any given integer α, a schedule for the greatest possible number of jobs using at most α active slots can be found in time $O(\sqrt{n}m)$.

In addition we consider preemptive scheduling for $P = 2$. Now each job j has an arbitrary integral length $\ell(j)$, and a set T_j of $\geq \ell(j)$ time slots in which one unit of its length can be executed. We wish to assign each job j to exactly $\ell(j)$ of these time slots. again minimizing the number of active time slots. We develop several results for this model.

If a schedule executing every job to completion exists, such a schedule minimizing the number of active time slots can be found in time $O(\sqrt{L}m)$ for $L = \sum_j \ell(j)$.

Now suppose we cannot schedule all the jobs to completion. Then it is NP-complete to even schedule the greatest possible number of jobs (for any fixed number of processors). We show the proof for $P = 1$; the extension for higher P is trivial. The reduction is is as follows: create a time slot for each element in X. We create a collection of m jobs (one corresponding to each set) of length 3 each. Each job can be scheduled in the time slots that correspond to the elements in the corresponding subset. We can schedule $\frac{n}{3}$ jobs if and only if there is a solution to the 3 Exact Cover problem.

We also note that the problem with unit jobs and $P = 2$, where each job has an arbitrary collection of time slots that are feasible for it, is at least as hard as computing a perfect matching. We create a job for each vertex and a common time slot for adjacent vertices – a schedule in which all jobs are paired up corresponds to a perfect matching.

References

[1] G. Even, R. Levi, D. Rawitz, B. Schieber, S. Shahar, M. Sviridenko. Algorithms for capacitated rectangle stabbing and lot sizing with joint set-up costs. *ACM Trans. on Algorithms*, Vol. 4(3) (2008).

[2] M. Flammini, G. Monaco, L. Moscardelli, H. Shachnai, M. Shalom, T. Tamir, S. Zaks. Minimizing total busy time in parallel scheduling with application to optical networks. *IPDPS Conference*, 2009.

[3] M. Garey, D. Johnson, B. Simons and R. Tarjan. Scheduling unit-time jobs with arbitrary release times and deadlines. *SIAM J. on Computing*, 10(2):256–269, 1981.

[4] R. Khandekar, B. Schieber, H. Shachnai and T. Tamir. Minimizing busy time in multiple machine real-time scheduling. *Foundations of Software Technology and Theoretical Computer Science Conference*, pp. 169–180, 2010.

[5] B. Simons and M. Warmuth. A fast algorithm for multiprocessor scheduling of unit length jobs. *SIAM J. on Computing*, 18(4):690–710, 1989.

Vehicle refueling with limited resources*

Murat Fırat †‡ Cor Hurkens (Speaker) ‡ Gerhard J. Woeginger ‡

1 Introduction

In our vehicle refueling problem, there is a fixed route on which a series of fuel stations are located. Various amounts of fuel are needed between successive stations. At the stations, limited amounts of fuel can be bought with varying prices. The capacity of the fuel tank may also vary between adjacent fuel stations. The main goal is to reach the final destination with cheapest cost of fuel. The vehicle refueling problem corresponds to an inventory-capacitated lot-sizing problem with zero setup costs, zero inventory holding costs, linear cost functions for production, and non-stationery inventory capacity. The movements of the vehicle between adjacent stations correspond to stages in time and the fuel is the commodity in demand for each stage. The fuel available at stations corresponds to production capacities and the fuel carried in the tank corresponds to keeping some commodity stock for later demands.

In this study, we show that vehicle refueling problem can be solved in $O(n^2 \log D \log nP)$ for some constants D, P due to its equivalence to minimum cost transportation problem. In the last section we propose an $O(n \log n)$ time algorithm.

2 Problem Description and Notation

In our problem, a vehicle makes a route visiting several cities in a fixed order, from city 1 to city n. In each city, there is a fuel station. We are given the set $S = \{1, \ldots, n\}$ including the fuel stations of all cities. It is assumed that the vehicle starts its travel with an initial fuel u_0 in city 1. The distances between cities, or stations, are specified in terms of the necessary fuel amounts. We are given d_i and T_i denoting respectively the distance and the vehicle tank capacity between stations i and $i+1$ for $i = 1, \ldots, n-1$. At the station i, the fuel price is $p_i \geq 0$ and the upper bound for fuel amount is U_i. In our analysis, without loss of generality, all ties in fuel prices are broken by station indices in the way that the earlier station has the lower price. Let us call this convention *tie-free fuel prices.*

To avoid dealing with initial fuel, we add city 0 to vehicle's route with any distance to city 1, without loss of generality assume $d_0 = d_1$, fuel price $p_0 = 0$, $U_0 = d_0 + u_0$, and $T_0 = U_0$. Moreover, no fuel is purchased at station n, hence the set of stations changes to $S = \{0, \ldots, n-1\}$. Finally, without loss of generality, we let $T_{n-1} = d_{n-1}$.

Objective. Let x_i denote the refueling amount at station i. The value of a solution is defined as $\sum_{i=0}^{n-1} p_i x_i$. The objective of the vehicle refueling problem is to find a solution with smallest solution value.

*This research is supported by France Telecom/TUE Research agreement No. 46145963.

†Corresponding author: m.firat@tue.nl.

‡Department of Mathematics and Computer Science, TU Eindhoven, P.O. Box 513, 5600 MB Eindhoven, Netherlands.

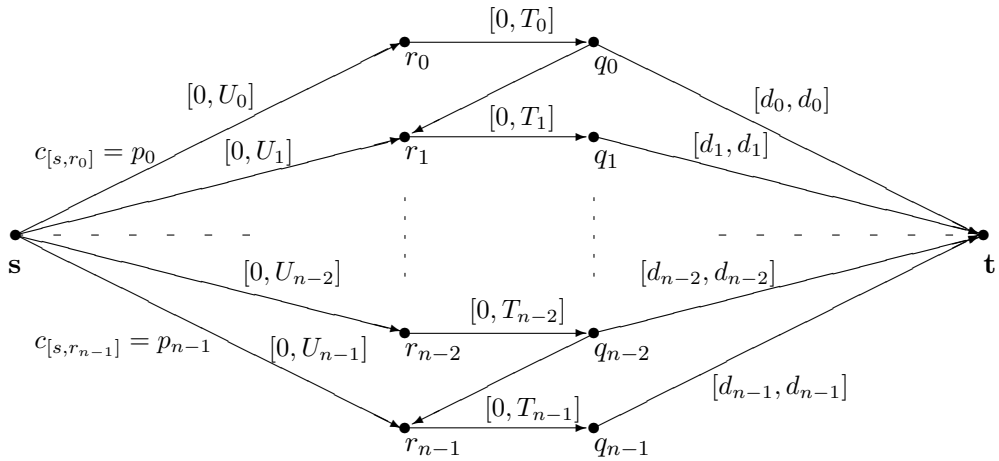


Figure 1: The equivalent minimum cost transportation network

3 Optimal vehicle refueling

3.1 A special case

Theorem 1. *([4]) The vehicle refueling problem with fixed tank capacity and unlimited fuel availability such that $U_i \geq T_i = T, \forall i \in S$, can be solved in linear time.*

In the proposed algorithm in [4], the tank is filled at a station if the fuel price is cheapest in its neighborhood, otherwise refueling is done in necessary amounts to reach cheaper stations. The authors of [2] propose $O(n \log n)$ time algorithm for this special case and they studied some extensions of the vehicle refueling problem.

3.2 An equivalent minimum cost transportation problem

We define a specific network, shown in Figure 1, on which any solution to the minimum-cost transportation problem can be converted into the solution of the vehicle refueling problem.

Proposition 2. *The minimum cost transportation problem on the network shown in Figure 1 is equivalent to the vehicle refueling problem.*

Theorem 3. *The vehicle refueling problem can be solved in $O(n^2 \log D \log(nP))$ with minimum cost transportation algorithm. Here $D = \sum_{i \in S} d_i$ and $P = \max_{i \in S}\{p_i\}$.*

The above theorem is proven using the equivalence of vehicle refueling problem and minimum cost transportation network in in Figure 1. Double scaling algorithm solves the minimum cost transportation problem optimally in mentioned time [3].

3.3 Vehicle refueling algorithm

We propose a greedy algorithm in which the fuel amounts from different stations are handled separately and can be used partially whenever needed. At a station, if the total amount of fuel exceeds tank capacity, then the excess fuel is removed starting from the most expensive fuel. Moreover, at every station, the cheapest fuel is purchased in an amount just enough to go to the next station. Hence the algorithm makes decisions of removals and purchases at every station. Note that fuel from a certain station may be partially removed and/or partially purchased. At station $n-1$, the algorithm allows d_{n-1} amount of fuel in tank, to arrive at the destination with an empty tank. The refueling amount from a station is determined by dropping the total fuel removal from the available amount at that station.

Algorithm 1. Vehicle refueling algorithm

```
1: i ← 0;    // initialize i..
2: L ← ∅;    // initialize L..
3: for i < n do
4:         L ← L ∪ {i};
5:         RemoveExcess(L, T_i); // remove the most expensive fuel..
6:         Purchase(L, d_i); // purchase the cheapest fuel..
7:         i ← i + 1;
8: end
```

In Algorithm 1, if the subset of stations in list L is stored in binary heaps, then inserting a station by respecting price order can be done in $O(\log n)$. The running time of the algorithm is $O(n \log n)$, since the fuel amount insertion is done in all stations. The following is our main result in this study.

Theorem 4. *Algorithm 1 solves the vehicle refueling problem optimally.*

In the proof of Theorem 4, we consider a solution of Algorithm 1 with refueling amounts x_i for $i = 0, \ldots, n-1$ and its corresponding flow on the network in Figure 1. By Proposition 2, we know that this flow is feasible. Next, we show that there is no negative cost cycle in the residual network of this flow. This will imply the optimality of the flow, see [1], and hence optimality of the vehicle refueling solution.

References

[1] A.V. Goldberg and R.E. Tarjan (1989) "Finding minimum-cost circulations by canceling negative cycles", *Journal of the ACM*, Vol. 36, No.4, pp. 873-886.

[2] S. Khuller, A. Malekian, and J. Mestre (2007) " To fill or not to fill: The gas station problem", *in Proceedings ESA*, pp. 534-545.

[3] Ravindra K. Ahuja, Thomas L. Magnanti, and James Orlin (1993) *Network Flows: Theory, Algorithms, and Applications*, Prentice Hall.

[4] S.H. Lin, R. Gertsch and J.R. Russell (2007) "A linear-time algorithm for finding optimal vehicle refueling policies", *Operations Research Letters*, Vol. 35, pp. 290-296.

A Column Generation Approach for the Job-Shop Scheduling Problem with Availability Constraints

Sadia Azem [*] Riad Aggoune [†] Stéphane Dauzère-Pérès (Speaker) [‡]

1 Introduction

We propose a mathematical model to solve the job-shop scheduling problem with resource availability constraints, in which each variable corresponds to a possible time-indexed schedule of a given job. This schedule specifies, in each period of the scheduling horizon, whether an operation of the job is processed or not on its machine. Because the number of variables is huge, a column generation approach is developed to select the optimal set of schedules. The model and the approach can naturally consider resource unavailability periods, but also release dates and due dates of jobs. It is also easy to extend this work to the flexible job-shop scheduling problem.

We assume that the unavailability periods of the machines are fixed and known in advance. Although the approach can be generalized to any criterion that depends on the jobs, we consider in this abstract the minimization of the sum of the job completion times. We assume that preemption is not allowed (an operation cannot be interrupted by another operation or an unavailability period). However, we will explain in the workshop how the approach can be extended to consider the preemption of an operation by an unavailability period and the flexibility of the starting dates of the unavailability periods.

The job-shop scheduling problem with resource availability constraints can be defined as a set of n jobs $J = \{J_1, J_2, \ldots, J_n\}$ to be processed on a set of m machines $M = \{M_1, M_2, \ldots, M_m\}$. Each job J_i is composed of a fixed linear sequence (routing) of n_i operations $\{O_{i1}, O_{i2}, \ldots, O_{ij}, \ldots, O_{in_i}\}$. Each machine can process only one operation at a time, and each operation O_{ij} needs only one machine during p_{ij} time units. There are m_r unavailability periods $\{h_{r1}, h_{r2}, \ldots, h_{rk}, \ldots, h_{rm_r}\}$ on each machine M_r. The starting date S_{rk} of unavailability period h_{rk} of duration p'_{rk} is known in advance. The machine on which operation O_{ij} is processed is denoted mr_{ij}. The goal is to determine the starting date t_{ij} and the completion time C_{ij} of each operation O_{ij} in order to minimize the sum of job completion times $\sum_{i=1}^{n} C_{in_i}$. As mentioned above, the column generation approach is well-suited for any additive criterion. The job-shop scheduling problem with resource availability constraints is NP-hard since the problem without unavailability periods is already NP-hard.

[*] azem@emse.fr. Department of Manufacturing Sciences and Logistics, CMP, École des Mines de Saint-Étienne, 880 Avenue de Mimet, F-13541 Gardanne, France.

[†] riad.aggoune@tudor.lu. Centre de Recherche Public Henri Tudor, 6 rue de Luxembourg, L-4002 Esch-sur-Alzette, Luxembourg.

[‡] dauzere-peres@emse.fr. Department of Manufacturing Sciences and Logistics, CMP, École des Mines de Saint-Étienne, 880 Avenue de Mimet, F-13541 Gardanne, France.

2 A Column Generation Approach

Previous research works on the resolution of time-indexed formulations by column generation can be found in [5] [3], [6] and [2]. However, these papers study single-machine or parallel machine scheduling problems. To our knowledge, only the paper of Lancia *et al.* [4] aims at solving the classical job-shop scheduling problem. However, no indications are given on how the column generation approach is developed and implemented and what are the difficulties and properties. Our approach is similar to the one of Lancia *et al.* [4], although we consider unavailability periods.

Let us denote by T the schedule length. A solution of the problem can be described by the n schedules of the jobs on the machines. $S(J_i)$ denotes the set of possible schedules for job J_i, i.e. $S(J_i)$ includes all the *feasible* schedules of J_i. A schedule is feasible if it satisfies the routing constraints for job J_i and the unavailability periods for the machines. Note that the larger T, the larger $|S(J_i)|$, the (exponential) number of schedules for J_i. To determine a globally feasible solution, it is necessary to select one schedule in $S(J_i)$ for each job J_i in J, so that the set of selected schedules satisfies the machine constraints (two operations cannot overlap on the same machine).

In order to introduce our formulation, we need the additional parameters below:

S: Set of all schedules, i.e. $S = \bigcup_{i=1}^{n} S(J_i)$,

$J_i(s)$: Job associated to a given schedule s, i.e. $s \in S(J_i(s))$,

$a_{tj}^s = 1$ if the j^{th} operation of job $J_i(s)$ is processed at period t on its associated machine and 0 otherwise,

C_s: Cost associated to schedule s (for instance the completion time of job $J_i(s)$).

The following binary variable is used: $x_s = 1$ if the s^{th} feasible schedule is selected for job $J_i(s)$ and 0 otherwise. The integer programming formulation (MP) is:

$$f^* = \min f = \sum_{s \in S} C_s x_s \tag{1}$$

$$\sum_{s \in S(J_i)} x_s = 1 \qquad i = 1, \ldots, n \tag{2}$$

$$\sum_{s \in S} \sum_{j=1; mr_{J_i(s)j}=r}^{n_{J_i(s)}} a_{tj}^s x_s \leq 1 \qquad t = 1, \ldots, T;\ r = 1, \ldots, m \tag{3}$$

$$x_s \in \{0, 1\} \qquad \forall s \in S \tag{4}$$

Constraint (2) ensures that one and only one schedule is selected for each job. Constraint (3) guarantees that there is never more than one operation on each machine and at each period. The number of variables is $|S|$, which can quickly become very large, even for small values of n and m. Note that the scheduling criterion, considered in the objective function (1), can be very general as long as it only depends on each job: Sum of the (weighted) completion times $\sum_s C_s x_s$ ($\sum_s w_{J_i(s)} C_s x_s$), sum of the (weighted) tardiness $\sum_s T_s x_s$ ($\sum_s w_{J_i(s)} T_s x_s$) and/or earliness $\sum_s E_s x_s$ ($\sum_i w_{J_i(s)} E_s x_s$), etc. Where $w_{J_i(s)}$ is the weight of job $J_i(s)$, $d_{J_i(s)}$ is the due date of $J_i(s)$, $T_s = \max(C_s - d_{J_i(s)}, 0)$ and $E_s = \max(d_{J_i(s)} - C_s, 0)$. The criterion for each job J_i could actually be non regular or non linear.

The column generation approach (see for example [1]) that we developed, together with numerical experiments, will be presented in the workshop. The pricing problem is solved using a dynamic programming algorithm that aims at minimizing, for each job

J_i, the total cost of the schedule where a cost w_{ij}^t (computed with dual variables of the linear relaxation of (MP)) is associated to the completion of operation O_{ij} at period t. We will show that the model and the approach can rather easily be adapted to consider the preemption and the flexibility of unavailability periods. The numerical experiments illustrate that, as expected, the time horizon T has a strong impact on the ability to solve instances efficiently. Moreover, a Branch-and-Price approach still needs to be developed to obtain optimal solutions.

References

[1] C. Barnhart, E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh and P.H. Vance (1998). *Branch-and-Price: Column Generation for Solving Huge Integer Programs*, Operations Research, 46, 3, 316-329.

[2] L.-P. Bigras, M. Gamache and G. Savard (2008). *Time-Indexed Formulations and the Weighted Tardiness Problem*, INFORMS Journal of Computing, 20, 1, 133-142.

[3] Z.L. Chen and W.B. Powell (1999). *Solving Parallel Machine Scheduling Problems by Column Generation*, INFORMS Journal of Computing, 11, 1, 78-94.

[4] G. Lancia, F. Rinaldi and P. Serafini (2007). *A Compact Optimization Approach for Job-Shop Problems*, MISTA 2007 (3rd Multidisciplinary International Conference on Scheduling: Theory and Applications), Paris, France, 293-300.

[5] J.M. van den Akker, J.A. Hoogeveen and S.L. van de Velde (1999). *Parallel Machine Scheduling by Column Generation*, Operations Research, 47, 6, 862-872.

[6] J.M. van den Akker, C.A.J. Hurkens and M.W.P. Savelsbergh (2000). *Time-Indexed Formulations for Machine Scheduling Problems: Column Generation*, INFORMS Journal of Computing, 12, 2, 111-124.

A two-machine flow shop problem consisting of a discrete processor and a batch processor under uncertainty

Bastian Bludau (Speaker) * Karl-Heinz Küfer †

1 Introduction

Manufacturing systems often involve a combination of both discrete processors and batch processors. A *discrete processor* is a machine which can process only one job at a time, whereas a *batch processor* can process a batch of jobs simultaneously.

We consider a two-machine flow shop problem consisting of a discrete processor δ followed by a batch processor β. A finite number of n jobs

$$J = \{J_i = (p_i, v_i, d_i) \mid i = 1, \ldots, n\}$$

has to be processed on these two machines, where p_i denotes the processing time of job J_i on δ, $v_i > 0$ denotes the volume (capacity) required for processing this job on β, and d_i denotes its due date. The batch processor β can simultaneously process a subset $J' \subseteq J$ of the n jobs as long as the constraint $\sum_{J_i \in J'} v_i \leq V$ is fulfilled. Here, $V > 0$ represents the maximum possible volume of a batch on β (We assume that $v_i \leq V$ for all $i = 1, \ldots, n$.). The processing time of a batch on β is assumed to be a constant $T > 0$, independent of the jobs contained in it. The completion time of a job in a batch is defined as the completion time of the batch containing it. Let C_i be the completion time of job J_i. The tardiness T_i of job J_i is then defined as $T_i = \max(C_i - d_i, 0)$, and the corresponding unit penalty U_i is 1 if $C_i > d_i$, and 0 otherwise. We focus on three well-known due date based performance measures, namely maximum tardiness $T_{\max}$, total tardiness $\sum_{i=1}^{n} T_i$, and the number of tardy jobs $\sum_{i=1}^{n} U_i$.

Each job first has to be processed on the discrete processor δ and afterwards on the batch processor β. This flow shop configuration will be denoted by $\delta \to \beta$. Combining this notation with the well-known triplet notation from standard scheduling theory, our problem can be written as $\delta \to \beta \,|\, v_i \,|\, T_{\max}$, $\delta \to \beta \,|\, v_i \,|\, \sum T_i$, and $\delta \to \beta \,|\, v_i \,|\, \sum U_i$, respectively, where v_i represents the fact that the job sizes do not have to be identical.

A solution for this problem requires three distinct decisions to be made about

- job scheduling (that is, the sequencing of jobs on δ),
- batch composition (that is, the assignment of jobs to batches), and
- batch scheduling (that is, the sequencing of batches on β).

The complexity of the problem stems from the interaction of these three subproblems.

*bastian.bludau@itwm.fraunhofer.de. ITWM Fraunhofer Institut für Techno- und Wirtschaftsmathematik, Fraunhofer-Platz 1, 67663 Kaiserslautern, Germany.

†karl-heinz.kuefer@itwm.fraunhofer.de. ITWM Fraunhofer Institut für Techno- und Wirtschaftsmathematik, Fraunhofer-Platz 1, 67663 Kaiserslautern, Germany.

2 NP-hardness

The deterministic problem described in Section 1 has been introduced in 1992 by Ahmadi et al. [1] for the special case of identically sized jobs, that is $v_i = 1$, $i = 1, \ldots, n$. In particular, Ahmadi et al. showed that both problems $\delta \rightarrow \beta \,|\, v_i = 1 \,|\, C_{max}$ and $\delta \rightarrow \beta \,|\, v_i = 1 \,|\, \sum C_i$ can be solved in polynomial time. These results are due to the fact that in this case the optimal job sequence on the discrete processor δ can be determined without considering the subsequent batch composition and batch sequencing on β. This transforms the two-machine problem into a single batching machine problem with dynamic job arrivals. Unfortunately, in our case of non-identically sized jobs under due date based objective functions, this property does not hold any longer. Concerning the complexity class of our problems, Uzsoy [3] showed that already for the makespan objective, the single batching machine problem with non-identical job sizes is equivalent to the standard BIN PACKING problem. Since BIN PACKING is strongly NP-hard (Garey and Johnson [2]), this particularly shows that all our three problems $\delta \rightarrow \beta \,|\, v_i \,|\, T_{\max}$, $\delta \rightarrow \beta \,|\, v_i \,|\, \sum T_i$, and $\delta \rightarrow \beta \,|\, v_i \,|\, \sum U_i$ are NP-hard in the strong sense as well.

3 Heuristics

As a consequence of the NP-hardness, efficient algorithms that obtain optimal solutions are unlikely to exist. We will therefore concentrate on different classes of heuristics to get reasonably good solutions in a short time. To this end, we use a simple pairwise interchange argument to show that permutation schedules dominate other schedules. It follows that the knowledge about the ordered batch composition is sufficient to compute the whole corresponding permutation schedule. Alternatively, we can also tackle the problem from another point of view: We develop a dynamic program which computes the best possible permutation schedule for a given input job sequence in $\mathcal{O}(n^4)$ operations. This algorithm generalizes the dynamic program originally obtained for the special case of identically sized jobs without due dates by Ahmadi et al. [1].

Both approaches can be used to create different classes of heuristics as follows:

- Heuristics of Class 1: We first sort the n jobs according to some given sorting criterion. We then use standard heuristics from bin packing theory to compute the ordered batch composition, which is in turn sufficient to compute the corresponding schedule.
- Heuristics of Class 2: As before, we first sort the n jobs according to some given sorting criterion. Afterwards, the optimal permutation schedule for the corresponding job sequence is determined with the help of dynamic programming.

Heuristics of Class 1 can be computed in $\mathcal{O}(n \log n)$, while heuristics of Class 2 need $\mathcal{O}(n^4)$ operations. Since bin packing heuristics usually rearrange the job sequence, both classes of heuristics can also be combined to a third class, which then dominates Class 1 (at the expense of computational speed).

In the talk, we will present a variety of different heuristics of all three classes, where we use both standard and adapted dispatching rules as initial sorting criterion. We will present detailed numerical experiments to evaluate the efficiency of these heuristics. For a better comparison, we will furthermore give numerical results obtained by CPLEX for the same problem expressed as a mixed integer program (MIP).

4 Uncertainty of data

The two-machine flow shop problem considered in the talk comes from a practical application in the manufacturing process of ceramic substrates. (In this regard, the discrete processor δ stands for the process of forming raw materials into the desired shape, while the batch processor β represents the oven for the subsequent firing process.) In view of practical applicability, however, our model is too simple. Following the vast majority of scheduling research, we have so far assumed complete information about the problem to be solved and a static, deterministic environment. Under this assumption, the computed schedule can be executed without any disruption. In real-life applications, however, machines break down, activities happen to take longer than expected, and new jobs have to be inserted while other jobs will be canceled. All these unforeseen events may force a once computed schedule to be adjusted over and over again, which in turn leads to the unwanted effect of *schedule nervousness*.

To be protected against such possible schedule disruptions, we need a *proactive* scheduling approach in order to create *robust* schedules. In this respect, we first have to model the idea of "robustness" mathematically. While the definition of robustness as well as corresponding notation and terminology differs widely in the literature, the basic idea remains the same: We want to find a schedule that is insensitive to uncertainty within a certain range. Altogether, we are faced with the multi-criteria problem of finding a schedule that is, on the one hand, acceptable with respect to our performance measure(s), and, on the other hand, robust in the sense mentioned above.

We will address this problem within a stochastic version of the two-machine flow shop problem described in Section 1, where the processing times p_i, $i = 1, \ldots, n$, of the discrete processor δ are not known deterministically any longer. A simple but nevertheless effective technique to obtain robust schedules is the usage of buffer times. Let $B_j \subseteq J$ be the set of all jobs that are scheduled to be processed in the j-th batch. Moreover, let b_j denote the corresponding starting point of batch B_j. We call a schedule α-robust if $b_j \geq \max_{J_i \in B_j} C_i^\delta + \alpha$ for all $j = 1, 2, \ldots$, where C_i^δ denotes the completion time of Job J_i at the discrete processor δ. Focussing on this idea, we transform the heuristics developed for the deterministic problem into α-robust heuristics to be able to further investigate and evaluate the impact of α-robustness on the overall schedule quality. Particular attention will be given to the already mentioned trade-off between the level of robustness and the price we have to pay for it.

References

[1] J. H. Ahmadi, R. H. Ahmadi, S. Dasu and C. S. Tang (1992). *Batching and Scheduling Jobs on Batch and Discrete Processors*. Operations Research, Vol. 39, No. 4, 750-763.

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

[3] R. Uzsoy (1994). *Scheduling a single batch processing machine with non-identical job sizes*. International Journal of Production Research, Vol. 32, No. 7, 1615-1635.

Accelerating a Flow Shop Scheduling Algorithm on the GPU

Tomáš Zajíček * Přemysl Šůcha (Speaker) †

GPGPU (General Purpose computing on Graphics Processing Units) or GPU computing are terms denoting general purpose scientific and engineering computing on GPU (graphics processing unit). Existing GPU applications are mostly oriented on fields such as medical imaging and natural resource exploration, and creating breakthrough applications in areas such as image recognition and real-time HD video playback and encoding. On the other hand, in the recent time there is a growing attempt to use this specialized hardware to solve NP-hard combinatorial problems.

Nowadays, majority of scientific GPGPU use CUDA (Compute Unified Device Architecture) [6]. It is a parallel computing architecture allowing to write programs for the CUDA GPU, in the C language, independently on the number of available GPU computational cores.

Basically, there are two GPGPU computing models on CUDA. In a *homogeneous computing model* all computations are performed on GPU. On the contrary, in a *heterogeneous computing model* the sequential part of the application runs on the CPU and the computationally-intensive part is accelerated by the GPU. The weak point of the heterogeneous computing model is the communication bandwidth between CPU and GPU which often constitute a bottleneck in many applications.

In recent years there are several works proposing solutions of NP–hard combinatorial problems on GPUs. Almost all works use heterogeneous computing model. One of the earliest works is [4] which proposes a parallel tabu search algorithm for the traveling salesman problem and for the flow shop problem. This approach is implemented using shader programming which is less comfortable than using CUDA framework. GPGPU approach to quadratic 3-dimensional assignment problem is suggested in [5]. Identically as in the previous paper they use tabu search algorithm which iteratively use GPU for the neighbour local search. A GPU based parallelization of a metaheuristic algorithm for the flexible job shop scheduling problem is presented in [1]. The authors propose two double–level parallel meta-heuristic algorithms based on their method of the neighbourhood determination. Another method using heterogeneous computing model for the knapsack problem is presented in [2]. The proposed method is based on the dense dynamic programming. On the other hand, the homogeneous computing model is used in a genetic algorithm presented in [7]. It is based on island models and the solution is examined using Griewank's, Michalewicz's and Rosenbrock's artificial benchmark functions. Up to our knowledge, there are not many works using homogeneous computing model for NP–hard combinatorial problems.

*`zajictomfel.cvut.cz`. Department of Control Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague.

†`suchap@fel.cvut.cz`. Department of Control Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague.

At the conference, we will present a genetic algorithm (GA) implemented on a GPU for permutation flow shop scheduling problem [3]. This approach is based on the homogeneous computing model in order to maximize the algorithm efficiency. The genetic algorithm is based on work presented in [7]. In addition, we suggest several improvements (e.g. another mechanism for individuals migration, different data storing in global memory) allowing to apply this algorithm on a wider group of NP-hard combinatorial problems. According to the preliminary results, we achieved speedup from 60 to 120 with respect to the equivalent sequential CPU version.

References

[1] Wojciech Boejko, Mariusz Uchroński, and Mieczyslaw Wodecki. Parallel hybrid metaheuristics for the flexible job shop problem. *Computers and Industrial Engineering*, 59:323–333, September 2010.

[2] Vincent Boyer, Didier El Baz, and Moussa Elkihel. Dense dynamic programming on multi gpu. Technical Report LASS-CNRS no. 10509, Septembre 2010.

[3] Peter Brucker. *Scheduling Algorithms*. SpringerVerlag, 5th edition, 2007.

[4] A. Janiak, W. Janiak, and M. Lichtenstein. Tabu search on gpu. *Journal of Universal Computer Science*, 14(14):2416–2427, 2008.

[5] The Van Luong, Lakhdar Loukil, Nouredine Melab, and El-Ghazali Talbi. A gpu-based iterated tabu search for solving the quadratic 3-dimensional assignment problem. *Computer Systems and Applications, ACS/IEEE International Conference on*, 0:1–8, 2010.

[6] NVIDIA. *NVIDIA CUDA C Programming Guide 3.1*. NVIDIA, 2010.

[7] Petr Pospichal, Jiri Jaros, and Josef Schwarz. Parallel genetic algorithm on the cuda architecture. In *Applications of Evolutionary Computation*, volume 6024 of *Lecture Notes in Computer Science*, pages 442–451. Springer Berlin / Heidelberg, 2010.

Scheduling malleable tasks with arbitrary processing speed functions

M.S. Barketau* M. Y. Kovalyov * M. Machowiak (Speaker)†
J. Węglarz †

1 Introduction

A computing task is called *malleable* if it can be processed on several processors at the same time, its processing speed depends on the number of assigned processors, and the set of processors assigned to the same task can change over time. The following problem will be studied.

There are n malleable tasks to be scheduled for processing on m identical parallel processors. Each task j is associated with its amount of work, p_j. Let $r_j(t)$ denote the number of processors allotted to task j at time moment t, and let $f_j(r)$ denote the processing speed of task j if it is allotted r processors. All $f_j(r)$ are assumed to be strictly increasing continuous integrable functions with $f_j(0) = 0$. A schedule specifies an allocation of processors to the tasks over time. We limit ourselves to schedules in which there is a finite number of time intervals where the same number of processors is assigned to the same task for all tasks in the same interval. A schedule can be completely characterized by the number of these time intervals, L, the interval lengths, $\Delta_1, \ldots, \Delta_L$, where the corresponding intervals are $[0, \Delta_1]$, $[\Delta_1, \Delta_1 + \Delta_2]$, ..., and the number of processors $r_j^{(l)}$, $r_j^{(l)} \in \{0, 1, \ldots, m\}$, assigned to each task j in each interval l, $j = 1, \ldots, n$, $l = 1, \ldots, L$. The makespan of the corresponding schedule is $C_{\max} = \sum_{l=1}^{L} \Delta_l$. A schedule is called feasible if $\sum_{i=1}^{n} r_i^{(l)} \leq m$, $l = 1, \ldots, L$, and $\int_0^{C_{\max}} f_j(r_j(t))dt = \sum_{l=1}^{L} \Delta_l f_j(r_j^{(l)}) = p_j$, $j = 1, \ldots, n$. The problem is to find a schedule with the minimum makespan. We denote this problem as P-DSCR and the relaxed problem in which the numbers of assigned processors are not required to be integer as P-CNTN. For problem P-CNTN, processors represent a continuously divisible renewable resource, whose amount is upper bounded by m at each time moment.

Problems P-DSCR and P-CNTN were studied by Węglarz et al. [3, 4] for the case $n \leq m$. It was shown that both problems are solvable in $O(n)$ time if all processing speed functions are convex, and in $O(n \max\{m, n \log^2 m\})$ time if they are all concave. Justification of these results heavily relies on the statements proved for the case $n \leq m$ in the earlier works of Węglarz [1, 2].

*United Institute of Informatics Problems, Surganova 6, 220012, Minsk, Belarus
†Poznan University of Technology, Institute of Computing Science, ul.Piotrowo 3a, Poznan, Poland

2 Arbitrary strictly increasing functions

The following theorem characterizes the case of strictly increasing piecewise linear continuous functions $f_j(r)$ such that $f_j(r) = a_j^{(k)} r + b_j^{(k)}$, $r \in [k, k+1]$, $k = 0, \ldots, m-1$, $j = 1, \ldots, m$. Set U is a set of feasible transformed resource allocations and set conv$U$ is a convex hull of set U.

Theorem 1. *If $f_j(r)$ are strictly increasing piecewise linear continuous functions, then* conv$U$ *is a bounded polyhedron with vertices from the set U.*

Proof. The set D of feasible resource allocations can be represented as a union of bounded *elementary* polyhedrons each of which is defined by $2n+1$ half-spaces $r_j \geq k_j$, $r_j \leq k_j + 1$, $j = 1, \ldots, n$, and $r_1 + r_2 + \ldots + r_n \leq m$, where $k_j \in \{0, 1, \ldots, m-1\}$. There are m^n elementary polyhedrons because k_i and k_j can take different values for different i and j. Linear continuous functions $a_j^{(k_j)} r + b_j^{(k_j)}$, $j = 1, \ldots, n$, $k_j \in \{0, 1, \ldots, m-1\}$, transform corresponding bounded elementary polyhedron into the bounded polyhedron defined by $2n$ half-spaces $u_j \geq a_j^{(k_j)} k_j + b_j^{(k_j)}$, $u_j \leq a_j^{(k_j)}(k_j + 1) + b_j^{(k_j)}$ and half-space $\sum_{j=1}^{n} \frac{u_j - b_j^{(k_j)}}{a_j^{(k_j)}} \leq m$. Thus, U is a union of a finite number of bounded polyhedrons and conv$U$ is a bounded polyhedron with all vertices belonging to U. ∎

Here we assume that $f_j(r)$ are piecewise linear strictly increasing continuous functions with $f_j(0) = 0$. In this case we propose an enumerative algorithm to find an optimal schedule for problems P-CNTN and P-DSCR.

It is shown in Theorem 1 that D is a union of elementary polyhedrons each being an intersection of an n-dimensional *cube* and the half-space $r_1 + r_2 + \ldots + r_n \leq m$. Let vector $(k_1, k_2, \ldots, k_n)$, $k_j \in Z$, $1 \leq k_j \leq m$, $1 \leq j \leq n$, define the cube given by $k_j - 1 \leq r_j \leq k_j$, $j = 1, \ldots, n$.

Theorem 2. *If cube $(k_1, k_2, \ldots, k_n)$ intersects with hyperplane $r_1 + r_2 + \ldots + r_n = m$ and the intersection contains more than one point, then $k_1 + k_2 + \ldots + k_n \leq m + n - 1$ and $k_1 + k_2 + \ldots + k_n \geq m + 1$.*

Proof. Let $k_1 + k_2 + \ldots + k_n > m + n - 1$. Then $k_1 + k_2 + \ldots + k_n \geq m + n$. Thus, for the coordinates of the points inside this cube relation $r_1 + r_2 + \ldots + r_n \geq m$ is satisfied. Furthermore, there is at most one such point where this relation is a strict equality. Similarly, if $k_1 + k_2 + \ldots + k_n < m + 1$, then $k_1 + k_2 + \ldots + k_n \leq m$. Then for the coordinates of the points inside the cube $r_1 + r_2 + \ldots + r_n \leq m$ is satisfied and there is at most one point inside the cube where this relation is a strict equality. ∎

Theorem 3. *If the intersection of the cube $(k_1, k_2, \ldots, k_n)$ and the hyperplane $r_1 + r_2 + \ldots + r_n = m$, $m \in Z$, contains more than one point, then the intersection of the cube $(k_1, k_2, \ldots, k_n)$ and the halfspace $r_1 + r_2 + \ldots + r_n \leq m$ is a polyhedron. Furthermore, all vertices of this polyhedron belong to the set of vertices of this cube.*

Proof. Each vertex of the resulting polyhedron is defined by the intersection of n hyperplanes among which are facets of the cube and the hyperplane $r_1 + r_2 + \ldots + r_n = m$. If a vertex is defined only by the facets of the cube, then it is clearly a vertices of the cube. If it is defined by n hyperplanes including the hyperplane $r_1 + r_2 + \ldots + r_n = m$,

then, since m is an integer, it follows that all its coordinates are integer. Therefore, it is also a vertex of the cube. ∎

The number of cubes that have an intersection with the hyperplane $r_1+r_2+\ldots+r_n = m$ in more than one point is $\sum_{r=\max\{m+1,n\}}^{m+n-1} C_{r-1}^{n-1}$, where C_n^k is the number of possible combinations from n to k. Since $C_{n-1}^{k-1} \le C_n^k$ for $k \le n$, we have $\sum_{r=\max\{m+1,n\}}^{m+n-1} C_{r-1}^{n-1} \le \sum_{r=0}^{n+m-2} C_{n+m-2}^r = 2^{n+m-2}$. Then the total number of vertices of the considered cubes is $O(2^{2n+m-2})$.

The algorithm we propose consists of two stages. At stage 1 it finds all the vertices of the unit cubes that intersect with the hyperplane and their transformations by functions $f_j(r)$.

Vertices of the polyhedron conv$U$ belong to the set U. By Theorems 2 and 3 and the fact that functions $f_j(r)$ are continuous and piecewise linear, we deduce that vertices of the polyhedron conv$U$ are among the above mentioned $O(2^{2n+m-1})$ transformed vertices.

At stage 2 the algorithm enumerates all hyperplanes each of which is defined by n points among those found at stage 1 and such that all the remaining points are the same side of this hyperplane in the space R^n as the point $(0, 0, \ldots, 0)$. It is clear that all the hyperplanes that define facets of the polyhedron conv$U$ are among these hyperplanes. Calculate the intersection point of the line p/C with each such hyperplane and find the hyperplane with the maximum value of C which is equal to C_{max}^0. Denote this intersection point as u^0. The complexity of stages 1 and 2 is $O(2^{(2n+m-2)n})$.

Let $v_1, \ldots, v_n$ be the base points for the hyperplane found on stage 2. Then we can solve the following set of $n+1$ linear equalities: $\sum_{l=1}^n \lambda_l v_l = u^0$, $\sum_{l=1}^n \lambda_l = 1$, $l = 1, \ldots, n$, for $\lambda_l \ge 0$, $l = 1, \ldots, n$. Let the solution contain $L^+ \le n$ positive values which are $\lambda_1, \ldots, \lambda_{L^+}$ without loss of generality. Then an optimal schedule contains L^+ intervals. Interval l, $1 \le l \le L^+$, has length $\lambda_l C_{max}^0$ and resource allocation vector $r^{(l)}$ corresponding to the point v_l. Recall that the resource allocations $r^{(l)}$, $l = 1, \ldots, L$, have integer components according to Theorem 3. Therefore, the corresponding solution is optimal for both problems P-CNTN and P-DSCR and its complexity $O(2^{(2n+m-2)n})$.

References

[1] Węglarz J. (1981) Project scheduling with continuously-divisible, doubly constrained resources, *Management Science*, 27, 1040-1052.

[2] Węglarz J. (1982) Modelling and control of dynamic resource allocation project scheduling systems, In S.G.Tzafestas(ed.), *Optimization and control of Dynamic Operational Research Models*, Amsterdam: North-Holand.

[3] Blazewicz J., Kovalyov M.Y., Machowiak M., Trystram D., Węglarz J. (2004) Malleable Tasks Scheduling to Minimize the Makespan, *Annals of Operations Research*, vol. 129, 65-80.

[4] Blazewicz J., Kovalyov M.Y., Machowiak M., Trystram D., Węglarz J. (2006) Preemtable Malleable Task Scheduling Problem, *IEEE Transactions on Computers*, vol. 55, no 4, 485-490.

On the quality and complexity of Pareto equilibria in the Job Scheduling Game

Leah Epstein* Elena Kleiman (Speaker)†

1 Introduction

The rise of the Internet as a global platform for communication, computation, and commerce brought up the necessity to reconsider the prevalent paradigm in system design which assumes a central authority which constructs and manages the network and its participants, with a purpose of optimizing a global social objective. Designing a protocol intended for use in a global network such as the Internet, we have to take into account that it consists of multiple independent and self-interested users which strive to optimize their private objective functions. In networks of such scale and complexity and in presence of raw economic competition between the parties, it is impossible to introduce a single regulatory establishment enforcing binding commitments on the players.

In light of the above, there is an increased need to design efficient protocols that motivate self-interested agents to cooperate. Here "cooperation" may be defined as any enforceable commitment that makes it rational for the self interested players to choose a given strategic profile. In the settings in discussion, any meaningful agreement between the players must be self-enforcing. When deciding which particular strategy profile to offer for the users, the first and most basic requirement one has to consider is its stability, in a sense that no player would have an interest to unilaterally defect from this profile, given that the other players stick to it. This is consistent with the notion of Nash equilibrium (NE), which is an accepted concept of stability in non-cooperative game theory. The second requirement is that the profile must be efficient. A fundamental concept of efficiency considered in economics is the Pareto efficiency, or Pareto optimality. This efficiency criterion assures that it is not possible for a group of players to change their strategies so that every player is better off (or no worse off) than before. One may justifully argue that Nash stability and Pareto optimality should be minimal requirements for any equilibrium concept intended to induce self-enforceability in presence of selfishness. There are even stronger criteria for self-enforceability, requiring fairness in terms of fair competition without coalitions, demanding from the profile to be resilient to groups of players willing to coordinate their decisions, in order to achieve mutual beneficial outcomes. This is compatible with the definition of Strong Nash equilibrium (SNE). However, this requirement is sometimes too strong that it excludes many reasonable profiles.

We thus restrict ourselves to profiles that satisfy the requirements of Nash stability and Pareto efficiency. In a sense, Pareto optimal Nash equilibria can be considered as

*Department of Mathematics, University of Haifa, 31905 Haifa, Israel. lea@math.haifa.ac.il.
†Computer Science Department, Technion, Haifa 32000, Israel. elena.kleiman@gmail.com.

intermediate concepts between Nash and Strong Nash equilibria; Pareto optimal equilibria are stable under moves by single players or the grand coalition of all players, but not necessarily under arbitrary coalitions. We distinguish between two types of Pareto efficiency. In a *weakly* Pareto optimal Nash equilibrium ($WPO\text{-}NE$) there is no alternative strategy profile beneficial for all players. A *strictly* Pareto optimal Nash equilibrium ($SPO\text{-}NE$) is also stable against deviations in which some players do not benefit but are also not worse off and at least one player improves his personal cost. Obviously, any *strictly* Pareto optimal equilibrium is also *weakly* Pareto optimal, but not wise-versa.

We consider strict and weak Pareto optimal pure Nash equilibria for scheduling games on the most common three machine models, identical machines, uniformly related machines and unrelated machines with the social goal of minimization of the makespan, that is, the maximum load of any machine. This class of games is particularly important to our discussion as it models a great variety of problems in modern networks. We investigate the quality of these solution concepts in the job scheduling game by comparison to an optimal solution, adopting a worst-case approach. As both papers that originally suggested to compare the NE and SNE to an optimal solution to study their quality [1, 10] demonstrated this approach in scheduling games, this gives an additional incentive to consider these solution concepts for this particular game class.

The quality measures which consider Nash equilibria are the Price of Anarchy [10] and the more optimistic Price of Stability [2], which are defined as the worst-case ratio between the social cost of the worst/best Nash equilibrium to the social cost of an optimal solution. This concept is applied analogously to Strong Nash equilibria as well as to weakly/strictly Pareto optimal Nash equilibria yielding the Strong Price of Anarchy and the Strong Price of Stability as well as the weak and strict Pareto Prices of Anarchy and Stability. Some natural questions in this context are whether the Pareto Prices of Anarchy are significantly smaller than the standard Price of Anarchy, whether the weak Pareto Price of Anarchy is much larger than the Strong Price of Anarchy, and finally, whether there is any relation between the Strong Price of Anarchy and the strict Pareto Price of Anarchy. In other words, does the requirement that the equilibrium must be Pareto optimal leads to greater efficiency, and does the further demand that the equilibrium must be stable against arbitrary coalitions helpful.

2 Related work and our contribution

Pareto efficiency of resource assignments is an important issue in economics and welfare economics. However Nash equilibria may generally be Pareto inefficient. The analysis of job scheduling in the algorithmic game theory (AGT) context was initiated in [10]. The standard and strong Prices of Anarchy and Stability in the three scheduling models were completely established in [1, 4, 5, 7, 9, 10]. The previous work on Pareto efficiency of Nash equilibria in AGT was mainly concerned with weak Pareto equilibria, probably since a solution which is not weakly Pareto optimal is clearly unstable. However, the strict Pareto is a stronger and more meaningful efficiency notion, as it captures an important aspect of human social behavior; The weak Pareto suggests that some assignment is socially preferable over another by everyone. In reality, such unanimity of preferences among all persons is very rare. We focus on both these important concepts.

Existence of strict Pareto optimal Nash equilibria in scheduling games (among others) was proved in [8]. Weak Pareto Nash equilibria in routing and job scheduling games

were considered recently in [3]. They do not consider the quality of Pareto optimal Nash equilibria with respect to the social goal. Among other results, it is shown in [3] that any Nash equilibrium assignment is necessarily weakly Pareto optimal for both identical and related machines. Moreover, for any machine model, any assignment which achieves the social optimum must be weakly Pareto optimal. We consider these issues for $SPO\text{-}NE$ assignments. We fully characterize the weak and strict Pareto Prices of Anarchy of the job scheduling game in cases of identical, related and unrelated machines.

Next, we consider the complexity of recognition of weak and strict Pareto optimality of NE. We show that recognition of $WPO\text{-}NE$ or $SPO\text{-}NE$ can be done in polynomial time for identical machines and related machines. For unrelated machines, we show that the recognition of $WPO\text{-}NE$ is NP-hard in the strong sense and the recognition of $SPO\text{-}NE$ is NP-hard. We reflect upon the differences between the results for weak and strict Pareto equilibria also compared to strong equilibria, and make conclusions regarding the relations between the quality measures in this game.

References

[1] N. Andelman, M. Feldman, and Y. Mansour. Strong price of anarchy. *Games and Economic Behavior*, 65(2):289–317, 2009.

[2] E. Anshelevich, A. Dasgupta, J. M. Kleinberg, É.Tardos, T. Wexler, and T. Roughgarden. The price of stability for network design with fair cost allocation. *SIAM Journal on Computing*, 38(4):1602–1623, 2008.

[3] Y. Aumann and Y. Dombb. Pareto efficiency and approximate pareto efficiency in routing and load balancing games. In *Proc. of the 3rd International Symposium on Algorithmic Game Theory (SAGT'10)*, 2010.

[4] B. Awerbuch, Y. Azar, Y. Richter, and D. Tsur. Tradeoffs in worst-case equilibria. *Thoeretical Computer Science*, 361(2-3):200–209, 2006.

[5] A. Czumaj and B. Vöcking. Tight bounds for worst-case equilibria. *ACM Transactions on Algorithms*, 3(1), 2007.

[6] E. Even-Dar, A. Kesselman, and Y. Mansour. Convergence time to Nash equilibrium in load balancing. *ACM Transactions on Algorithms*, 3(3):32, 2007.

[7] A. Fiat, H. Kaplan, M. Levy, and S. Olonetsky. Strong price of anarchy for machine load balancing. In *Proc. of the 34th International Colloquium on Automata, Languages and Programming (ICALP'07)*, pages 583–594, 2007.

[8] T. Harks, M. Klimm, and R. H. Möhring. Strong Nash equilibria in games with the lexicographical improvement property. In *Proc. of the 5th International Workshop on Internet and Network Economics (WINE'09)*, pages 463–470, 2009.

[9] E. Koutsoupias, M. Mavronicolas, and P. G. Spirakis. Approximate equilibria and ball fusion. *Theory of Computing Systems*, 36(6):683–693, 2003.

[10] E. Koutsoupias and C. H. Papadimitriou. Worst-case equilibria. In *Proc. of the 16th Annual Symposium on Theoretical Aspects of Computer Science (STACS'99)*, pages 404–413, 1999.

The Price of Anarchy for Related Machine Scheduling

Ruben Hoeksma (Speaker)* Marc Uetz*

1 Introduction

Consider the problem of scheduling n nonpreemptive jobs on m machines, where each job j has a processing requirement p_j and each machine i has speed s_i. The processing time of job j on machine i equals p_j/s_i. The objective is to minimize the sum of completion times $\sum_j C_j$, where C_j denotes the completion time of job j. This classical model is referred to as *uniformly related machine scheduling*, or $Q\,||\sum C_j$.

We analyze a simple *coordination mechanism* for this problem, where each machine i sequences its jobs shortest processing time first (SPT), and each job j chooses a machine in order to minimize its own completion time C_j. The outcome of the coordination mechanism is a (pure) *Nash equilibrium*, a solution where no job can unilaterally change to another machine and thereby improve its own completion time. Minimizing $\sum_j C_j$ then corresponds to the *utilitarian* social choice function, as it maximizes the total utility of the jobs. Yet, in contrast to the *egalitarian* social choice function $C_{\max}$, this model has not received much attention in the literature. Moreover, as SPT minimizes $\sum C_j$ on each single machine, this coordination mechanism is most natural and almost suggests itself.

For the classical problem $Q\,||\sum C_j$, the optimal solution can be computed efficiently by the well known MFT algorithm of Horowitz and Sahni [3]. Also Nash equilibria can be computed efficiently by scheduling the jobs in SPT order, assigning each job to the machine that minimizes the jobs' completion time, also known as the Ibarra Kim algorithm [4]; see [5]. Nash equilibria are in general not optimal for minimizing $\sum_j C_j$.

The ratio between the worst Nash equilibrium and the optimal solution is the *price of anarchy* (PoA) [6]; it measures the cost of decentralization. What is intriguing about the model is that both optimum solution and Nash equilibrium can be computed efficiently by simple algorithms. Moreover, Nash equilibria correspond to outcomes of the most natural heuristic for the problem, namely SPT. Surprisingly, the quality of this heuristic for $Q\,||\sum C_j$ has not been analyzed. We (almost) close this gap and show the following.

Theorem 1. *The price of anarchy for the coordination mechanism for $Q\,||\sum C_j$ with SPT on each machine is bounded from below by $\frac{e}{e-1} \approx 1.58$ and from above by 2.*

This result complements very resent results by Cole et al. and Correa and Queyranne [1, 2], which show that the price of anarchy for coordination mechanisms for unrelated machine scheduling and restricted related machine scheduling games with WSPT on each machine is exactly 4. To prove the upper bound they in fact show that $R||\sum w_jC_j$ games

*r.p.hoeksma@utwente.nl, m.uetz@utwente.nl. Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente, PO Box 217, 7500 AE Enschede, The Netherlands

are $(2, 1/2)$-smooth, as defined by Roughgarden [7]. Hence their results are bounds for the *robust price of anarchy*, generalizing also beyond pure Nash equilibria. The same holds for our result, as it is based on a smoothness argument, too. But in the analysis we need to explicitly exploit properties of the optimal solution. We conjecture the PoA to be $\frac{e}{e-1}$ rather than 2, as the analysis of the upper bound is not tight.

2 Sketch of Proofs

For the lower bound, consider an instance with n jobs and one fast machine with speed s and $n-s$ slow machines with speed 1. Furthermore, let the smallest s jobs have length $p_j = 1$ and let all other jobs have length $p_j = \left(\frac{s}{s-1}\right)^{n-s}$. This construction ensures that the schedule with all jobs on the fastest machine is a Nash equilibrium, while the optimal solution distributes the jobs over all the machines. For such an instance the price of anarchy is equal to

$$\text{PoA} = \frac{\left(\frac{s}{s-1}\right)^{s} - \frac{1}{2} \cdot \left(\frac{s}{s-1}\right)^{-n+2s-1}}{\left(\frac{s}{s-1}\right)^{s} - 1},$$

which approaches $\frac{e}{e-1}$ as n and s go to infinity. For the upper bound, we show the following, which can be seen as a $(2, 0)$-smoothness argument according to [7]. If σ is an arbitrary choice of the jobs, and σ^* the choice in an optimal solution, we show

$$\sum\nolimits_{j=1}^{n} C_j(\sigma_j^*, \sigma_{-j}) \leq 2 \sum\nolimits_{j=1}^{n} C_j(\sigma^*),$$

where $(\sigma_j^*, \sigma_{-j})$ is the outcome where all jobs adhere to σ except for job j, who chooses the machine according to σ^*. Then it follows, when σ is a Nash equilibrium, that

$$\sum\nolimits_{j=1}^{n} C_j(\sigma) \leq \sum\nolimits_{j=1}^{n} C_j(\sigma_j^*, \sigma_{-j}) \leq 2 \sum\nolimits_{j=1}^{n} C_j(\sigma^*),$$

which proves the upper bound of 2 on the price of anarchy. This upper bound also extends to mixed Nash or correlated equilibra, or no-regret sequences.

References

[1] R. Cole, J.R. Correa, V. Gkatzelis, V. Mirrokni, and N. Olver (2011). *Inner Product Spaces for MinSum Coordination Mechanism*. Proceedings 43rd STOC, to appear.

[2] J.R. Correa and M. Queyranne (2010). *Efficiency of Equilibria in Restricted Uniform Machine Scheduling with MINSUM Social Cost*. Manuscript, 2010.

[3] E. Horowitz and S. Sahni (1976). *Exact and approximate algorithms for scheduling nonidentical processors*. Journal of the ACM, 23(2):317-327.

[4] O. Ibarra and C. Kim (1977). *Heuristic algorithms for scheduling independent tasks on nonidentical processors*. Journal of the ACM, 24(2):280-289, 1977.

[5] N. Immorlica, L. (Erran) Li, V.S. Mirrokni, and A.S. Schulz (2009). *Coordination mechanisms for selfish scheduling*, Theor. Comput. Sci., 410(17):1589-1598, 2009.

[6] E. Koutsoupias and C. Papadimitriou (1999). *Worst-case equilibria*. In: Proceedings STACS 1999, volume 1563 of LNCS, pages 404-413, Springer, Berlin.

[7] T. Roughgarden (2009). *Intrinsic robustness of the price of anarchy*. Proceedings of the 41st STOC, pages 513- 522, ACM.

Performance of Distributed Game Theoretic Algorithms for Single Slot Scheduling in Wireless Networks

Eyjólfur Ingi Ásgeirsson (Speaker) * Pradipta Mitra †

1 Introduction

We consider the single slot scheduling problem in wireless networks, also called the capacity problem. Our goal is to maximize the number of successful connections in arbitrary wireless networks where a transmission is successful only if the signal-to-interference-plus-noise ratio at the receiver is greater than some threshold.

A central question in the context of wireless network communications is how to model interference between various attempted transmissions in the network. The models used in the literature can essentially be divided into two types. First, there is the *protocol model* where interference is modeled by an *interference graph*, and a transmission is successful if and only if none of the neighbors of the transmission in this graph also choose to transmit at the same time. Thus maximizing the number of scheduled links in a single slot becomes equivalent to the maximum independent set problem.

It is well-known though, that graph-based protocols are not very good in capturing reality and this has been demonstrated both theoretically and experimentally [6, 8]. As a result, a lot of recent algorithmic work has focused on the so-called *physical model* or the SINR model.

The capacity of random networks in the SINR model was studied by Gupta and Kumar [5], and a large number of papers have pursued the same theme. On worst case instances the problem is NP-hard [1]. Approximation algorithms for worst case instances have recently garnered attention, starting with the work of Moscibroda and Wattenhofer [7]. Since then, a large body of work has been produced for this problem [4].

We study a game theoretic approach towards single slot scheduling introduced by Andrews and Dinitz [1] and Dinitz [3]. We prove vastly improved bounds for the game theoretic algorithm. In doing so, we achieve the first *distributed* constant factor approximation algorithm for the single slot scheduling problem for the uniform power assignment. When compared to the optimum where links may use an arbitrary power assignment, we prove a $O(\log \Delta)$ approximation, where Δ is the ratio between the largest and the smallest link in the network. This is an exponential improvement of the approximation factor compared to existing results for distributed algorithms. All our results work for links located in *any* metric space. In addition, we provide simulation studies clarifying the picture on distributed algorithms for single slot scheduling.

*eyjo@ru.is. School of Science and Engineering, Reykjavík University, Menntavegur 1, 101 Reykjavík, Iceland.

†ppmitra@gmail.com. School of Computer Science, Reykjavík University, Menntavegur 1, 101 Reykjavík, Iceland.

2 Models and Algorithms

We assume that there is a set L of links, where each link $v \in L$ represents a potential transmission from a sender s_v to a receiver r_v, each points in a metric space. The distance between two points x and y is denoted $d(x, y)$.

The distance from v's sender to w's receiver is denoted $d_{vw} = d(s_v, r_w)$. The length of link v is the distance between the sender and the receiver of the link, denoted by $\ell_v = d(s_v, r_v)$.

The set may be associated with a *power assignment*, which is an assignment of a transmission power P_v to be used by each link $v \in L$. We assume $0 \leq P_v \leq P_{\max}$ for some fixed $P_{\max}$, for all v. The setting $P_v = 0$ means the sender is not transmitting. For simplicity we will assume $P_{\max} = 1$ without loss of generality. The signal received at point y from a sender at point x with power P is $P/d(x, y)^\alpha$ where the constant $\alpha > 0$ is the *path-loss exponent*.

In the *physical* or SINR-model of interference, a receiver r_v successfully receives a message from the sender s_v if and only if the following condition holds:

$$\frac{P_v/\ell_v^\alpha}{\sum_{\ell_w \in S \setminus \{\ell_v\}} P_w/d_{wv}^\alpha + N} \geq \beta, \tag{1}$$

where N is the environmental noise, the constant $\beta > 0$ denotes the minimum SINR (signal-to-interference-noise-ratio) required for a message to be successfully received, and S is the set of concurrently scheduled links in the same *slot*.

We say that S is *SINR-feasible* (or simply *feasible*) if (1) is satisfied for each link in S. Let $\Delta = \frac{l_{\max}}{l_{\min}}$ where $l_{\max}$ and $l_{\min}$ are respectively, the maximum and minimum lengths in L.

We will use the notation OPT to denote the largest set that can be scheduled using uniform power, and OPT_P to denote the largest set that is feasible using some arbitrary power assignment. Thus the maximum capacity of the network is $|OPT|$ or $|OPT_P|$, depending the flexibility one allows on the power assignments.

The games we are interested in have n players and every player has exactly two possible actions. Let $\mathcal{A} = \{0, 1\}^n$ be the space of all possible actions for the game, i.e. given a point $A \in \mathcal{A}$, the i^{th} coordinate a_i represents the action used by player i in profile A. For each player i there is a utility function $\alpha_i : \mathcal{A} \to \mathbb{R}$ denoting how good certain actions for that player are. We will want to consider modifications of strategy profiles: given $A \in \mathcal{A}$, let $A \oplus a_i'$ be the strategy set obtained by player i changing its action from a_i to a_i'. We will use superscripts to denote time, so A^t will be the action set at time t and a_i^t will be the action taken by player i at time t.

Definition 1. *The regret of player i at time T given strategy profiles $A^1, A^2, \ldots A^T$ is*

$$\max_{a_i \in \{0,1\}} \frac{1}{T} \sum_{t=1}^{T} \alpha_i(A^t \oplus a_i) - \frac{1}{T} \sum_{t=1}^{T} \alpha_i(A^t)$$

Having low regret essentially means that the player has done almost as well on average as the best single action, i.e. always staying silent or always transmitting, would have.

Theorem 2 ([2]). *There is an algorithm that has regret at most $O(\sqrt{\frac{\log(T/\delta)}{T}})$ with probability at least $1 - \delta$ for any $\delta > 0$, for any game with a constant number of possible actions per player.*

3 Results

We will prove the following:

Theorem 3. *If every sender uses a low-regret algorithm, then after* $O((\frac{n}{|OPT|})^2 \log n)$ *rounds the average number of successful connections is* $\Omega(|OPT|) = \Omega(|OPT_P|/\log \Delta)$ *with probability at least* $1 - \frac{1}{n}$*. Also, a property of this class of algorithms is that all algorithms in the class will use uniform power, that is, each sender* s_v *either transmits at full power* $P_v = 1$*, or does not transmit at all.*

Hence, there exists a randomized distributed $O(1)$-approximation algorithm to determine, with high probability, the capacity of a wireless network under uniform power. For arbitrary power assignments, the same algorithm achieves a $O(\log \Delta)$ approximation, with high probability. What is remarkable here is that a such results for general metrics have not been known even for centralized algorithms until recently.

References

[1] Matthew Andrews and Michael Dinitz (2009). *Maximizing Capacity in Arbitrary Wireless Networks in the SINR Model: Complexity and Game Theory.* INFOCOM'09 Proceedings of the 28th conference on Information communications, pp 1332–1340.

[2] Peter Auer, Nicolò Cesa-Bianchi, Yoav Freund and Robert E. Schapire (2002). *The Nonstochastic Multiarmed Bandit Problem*, SIAM Journal of Computing, 32(1), pp 48–77.

[3] Michael Dinitz (2010) *Distributed Algorithms for Approximating Wireless Network Capacity.* INFOCOM'10 Proceedings of the 29th conference on Information communications.

[4] Olga Goussevskaia, Yvonne-Anne Pignolet and Roger Wattenhofer (2010). *Efficiency of Wireless Networks: Approximation Algorithms for the Physical Interference Model*, Foundations and Trendsĺ in Networking: Vol. 4: No 3, pp 313-420.

[5] P. Gupta and P. R. Kumar (2000). *The Capacity of Wireless Networks*, IEEE Transactions on Information Theory, 46(2), pp 388–404.

[6] Ritesh Maheshwari, Shweta Jain and Samir R. Das (2008). *A measurement study of interference modeling and scheduling in low-power wireless networks*, Proc. of the 2^{nd} Conference on Embedded Networked Sensor Systems (SENSYS), pp 141–154.

[7] Thomas Moscibroda and Roger Wattenhofer (2006). *The Complexity of Connectivity in Wireless Networks*, INFOCOM'06 Proceedings of the 25th conference on Information communications.

[8] Thomas Moscibroda, Roger Wattenhofer and Yves Weber (2006). *Protocol Design Beyond Graph-Based Models*, Hotnets.

Fast minimum float computation in activity networks under interval uncertainty

Thierry Garaix (Speaker) * Christian Artigues Cyril Briand †

1 Introduction, problem definition and related work

An important part of recent scheduling research aims at tackling uncertainty, which occurs, under various forms, in practical applications. Dealing with activity scheduling, a basic form of uncertainty involves uncertain durations. We consider an activity-on-node network $G(V, A)$ where $V = \{0 \ldots, n+1\}$ represents a set of activities and A represents simple precedence relations among the activities (PERT scheduling). Let d_i denote the duration of an activity $i \in V$. We assume d_i is an uncertain parameter belonging to a given interval $[d_i^{\min}, d_i^{\max}]$. As 0 and $n+1$ correspond to dummy start and end activities, respectively, we consider that $d_i^{\min} = d_i^{\max} = 0$ for $i = 0, n+1$. Let $\mathcal{D}$ denote the set of possible realizations of duration vector d (scenario set). We have $\mathcal{D} = \{d \in \mathbb{R}^{n+2} | d^{\min} \leq d \leq d^{\max}\}$. Given a scenario $d \in \mathcal{D}$, let $l^*_{i,j}(d)$ denote the length of the longest path from i to j in G. Given an activity $i \in V$ and a duration scenario $d \in \mathcal{D}$, consider the earliest start time of i, denoted by $est_i(d) = l^*_{0,i}(d)$, the latest start time of i, denoted by $lst_i(d) = l^*_{0,n+1}(d) - l^*_{i,n+1}(d)$ and the float of i denoted by $f_i(d) = lst_i(d) - est_i(d) = l^*_{0,n+1}(d) - l^*_{i,n+1}(d) - l^*_{0,i}(d)$. We consider the problem (MinF) of determining the minimum float of each activity over the scenario set: $\min_{d \in \mathcal{D}} f_i(d), \forall i \in V$.

Chanas and Zielínski [1] have proved that this problem is strongly NP-hard. Previous studies established two properties allowing to restrict the solution space. First, Dubois *et al.* [3] showed that for each activity there always exists an extreme scenario d (that yields the minimal float) such that $d_j = d_j^{\min}$ or $d_j = d_j^{\max}$ for each $j \in V$. This allows to restrict the search to the set of extreme scenarios ($\hat{\mathcal{D}}$) with $|\hat{\mathcal{D}}| \leq 2^n$. Second, Dubois *et al.* [4] showed that the search can be further restricted to the set of path-induced extreme scenarios. If p denotes a $(0-n+1)$ path in G, d is an extreme scenario induced by p if $d_i = d_i^{\max}$ for $i \in p$ and $d_i = d_i^{\min}$ for $i \in V \setminus p$. Dubois *et al.* [4] derived from this property the so-called *path* algorithm to solve MinF. It consists in enumerating all $(0-n+1)$ paths, and, for each of them, to compute the float of each activity under the induced extreme scenario. If $\mathcal{P}$ denotes the set of $(0-n+1)$ paths, the algorithm runs in $O((n+m)|\mathcal{P}|)$ where $m = |A|$. A recent experimental evaluation by Fortin *et al.* [5] reports that the path algorithm fails in producing solutions in reasonable time for high-density networks.

*garaix@emse.fr. Ecole des Mines de Saint-Etienne, CMP Georges Charpak, F-13541 Gardanne, France.

†{artigues,briand}@laas.fr.

[1] CNRS ; LAAS ; 7 avenue du colonel Roche, F-31077 Toulouse Cedex 4, France.

[2] Université de Toulouse ; UPS, INSA, INP, ISAE ; UT1, UTM, LAAS ; F-31077 Toulouse Cedex 4, France.

2 Integer programming Formulations

We propose two new mixed integer programming (MIP) formulations based on the path-induced extreme scenarios: an activity independent formulation (a) and an activity dependent formulation (b) given below. For each activity, a binary flow variable x_i indicates if i is located on the optimal path ($x_i = 1$) or not ($x_i = 0$). The formulations also involve earliest start time continuous variables S_i, whose value is the length of the longest path from 0 to i in the optimal scenario. Γ_i^{-1} (Γ_i) denotes the set of predecessors (successors) of activity i, respectively.

(a) $\min z(k) = S_{n+1} - \sum_i d_i^{max} x_i$
$z(k) \geq z(k-1)$
$S_j \geq S_i + d_i^{min}(1 - x_i) + d_i^{max} x_i \quad \forall (i,j) \in A$
$x_i \leq \sum_{j \in \Gamma_i^{-1}} x_j \leq 1 \quad \forall i \neq 0, n+1$
$x_i \leq \sum_{j \in \Gamma_i} x_j \leq 1 \quad \forall i \neq 0, n+1$
$x_0 = x_{n+1} = 1$
$\sum_{i \in U_k} x_i \geq 1$
$x_i \in \{0,1\} \quad \forall i \in V$

(b) $\min f_k = S_{n+1} - \sum_i d_i^{max} x_i$
$S_j \geq S_i + d_i^{min}(1 - x_i) + d_i^{max} x_i \quad \forall (i,j) \in A$
$x_i \leq \sum_{j \in \Gamma_i^{-1}} x_j \leq 1 \quad \forall i \neq 0, n+1$
$x_i \leq \sum_{j \in \Gamma_i} x_j \leq 1 \quad \forall i \neq 0, n+1$
$x_0 = x_{n+1} = 1$
$x_k = 1$
$x_i \in \{0,1\} \quad \forall i \in V$

The activity-independent formulation (a) is solved iteratively. At iteration k, a set U_k of activities with unknown floats is considered (at iteration 0, $U_0 = V \setminus \{0, n+1\}$). At the next iteration, U_{k+1} is set to $U_k \setminus F_k$ where F_k is the set of activities located on the optimal path for which the minimum float is obtained. The activity-dependent MIP (b) is just a particular case of (a), where at each iteration k, the set U_k is reduced to the single activity k.

3 Branch and bound

We establish a new dominance property allowing further restrictions on the set of paths to consider. If p denotes a path, let $p[a,b]$ denote its sub-path from a to b and let $l_p(d)$ denote the length of p for scenario d.

Theorem 1. *For each activity $i \in V$, there exists a $(0-n+1)$ path p inducing scenario $d(p)$ verifying $l^*_{0,i}(d(p)) = l_{p[0,i]}(d(p))$, $l^*_{i,n+1}(d(p)) = l_{p[i,n+1]}(d(p))$ and*
$f_i(d(p)) = \max_{a,b|i \in p[a,b]}(l^*_{a,b}(d^{\min}) + d_a^{\max} - d_a^{\min} - l_{p[a,b]}(d^{\max})) = \min_{d \in \mathcal{D}} f_i(d)$

The theorem states that the search can be restricted to scenarios $d(p)$ for which p is the longest $(0-n+1)$ path traversing i. Also, the longest path from 0 to $n+1$ in G and p diverge at unique activity a and converge at a unique activity b, where i is located between a and b on p. A corollary allows to compute a lower bound for a partial path.

Corollary 2. *Let p be a path from $x \in V$ to $y \in V$ with $i \in p$. The float of the scenario induced by any path p' extending path p towards $(0, n+1)$ verifies*
$f_i(d(p')) \geq l^*_{a,b}(d^{\min}) + d_a^{\max} - d_a^{\min} - l_{p[a,b]}(d^{\max})$, $\forall i \in V$, $\forall a, b,\ i \in p[a,b]$

We propose a branch and bound algorithm to solve MinF for a given activity i by building the optimal path via partial path extension. A stack $\mathcal{Q}$ contains initially a node representing the single-activity path $\{i\}$. A typical iteration deletes the node on top of the stack and extends it in a direction that is changed at each iteration. For the left extension, the predecessors of the leftmost activity of the partial path p are scanned. According to Theorem 1, each candidate path p' is discarded if p' is not a

longest $(j - i)$ path for scenario $d(p')$, j being the scanned predecessor or if its lower bound (see Corollary 2) exceeds the minimum so-far obtained float. Right extension is symmetrical. Each non-discarded node is inserted on top of the stack for depth-first search. For each node, the above mentioned lower-bound computations and dominance checks can be done in linear time by precomputing the all-pairs longest path at minimal durations.

4 Experimental results and concluding remarks

Experiments were conducted to compare the MIP formulations (a), (b) that were directly solved by Cplex, the Dubois *et al.* [4, 5] Path algorithm, an improved variant of the Path algorithm[1] and the branch and bound algorithm. We generated random network instances using RanGen [2]. We display below a comparison of algorithm average CPU time in seconds for increasing order strengths OS (OS measures the network density [2]).

	MIP (a)	MIP (b)	Path	Improved Path	Branch and bound
OS=0.5	44.62	4.99	0.21	0.12	0.01
OS=0.8	197.49	23.12	52.71	33.96	0.02
OS=0.9	99.52	17.43	(stopped)	2843	0.02

We conclude that the proposed branch and bound algorithm outperforms all other methods thanks to the proposed dominance rule and lower bound. MIP formulation (b), which, requires exactly n iterations, performs better than the MIP (a) formulation that requires less iterations at the expense of a larger search space. Both perform better than the Path algorithm and its improved variant for high-density networks.

References

[1] S. Chanas and P. Zielínski (2002). *The computational complexity of the criticality problems in a network with interval activity times.* European Journal of Operational Research 136:541–550.

[2] E. Demeulemeester, M. Vanhoucke and W. Herroelen (2003). *RanGen: a random network generator for activity-on-the-node networks*, Journal of Scheduling 6(1):17–38.

[3] D. Dubois, H. Fargier, and V. Galvagnon (2003). *On latest starting times and floats in activity networks with ill-known durations.* European Journal of Operational Research 147:266–280.

[4] D. Dubois, H. Fargier, and J. Fortin (2005). *Computational methods for determining the latest starting times and floats of tasks in interval-valued activity networks.* Journal of Intelligent Manufacturing 16:407–422.

[5] J. Fortin, P. Zielínski, D. Dubois and H. Fargier (2010). *Criticality analysis of activity networks under interval uncertainty.* Journal of Scheduling 13:609–627.

[1] In this variant, once a path p is obtained, the float is computed in $m + |p|$ operations instead of $2m + n$ in the original algorithm

Allocating Resources in Clouds of Game Servers

Tao Li [*] Joel Wein (Speaker) [†]

1 Introduction

Moving applications to a distributed cloud of servers and delivering them remotely is an important trend in all of computing; therefore it is not surprising that it is being investigated for digital games as well, given the great popularity of digital games as a mode of entertainment. Traditionally, a digital game is played at a local computer or game console. In a distributed or cloud-based model the computing element that produces the game content ("game server") is remote, and the user sends instructions to the remote server, which processes them, calculates the next set of game state and graphics, and delivers them over the network to the player, often encoded as a video stream. Currently there are a handful of companies in their infancy that offer such services, but with significant limitations.

We introduce resource allocation problems encountered while designing a system to serve collections of interactive digital games from a cloud of remote servers. We discuss experiments that inform our modeling of the problem domain, define two classes of relevant resource allocation problems, and briefly point to initial simulation results for enhanced algorithms. It is our sense that this research introduces interesting and practical algorithmic questions that will be of interest to the MAPSP community.

The problem of optimizing resource allocation for game servers to deliver digital experiences has significant differences from traditional distributed systems scheduling. First, most digital games are generated in a loop, each iteration of which produces a *frame* that is ready for delivery over a network. Each iteration requires both CPU and GPU (Graphics Processing Unit) resources so the scheduling considerations are more closely related to those of shop scheduling than to those of single or multiprocessor scheduling. Second, the latency constraints are quite strict for games to be playable. If a game freezes at the wrong moment and as a result a character is eliminated after hours of careful building ... for those who play digital games, no further elaboration is necessary.

We are working towards building a two-tier system that serves multiple games from a cloud of game servers in a resource-efficient way. The higher tier will make decisions about which games to assign to which servers, while the lower tier will make good scheduling decisions on each server to effectively serve the assigned games. We know of no prior work that defines or studies the algorithmic issues in such a system; the closest we have found is one project that defines a general architecture for the delivery of graphics applications from a cloud [3] and another that discusses how to steal compute cycles from a collection of servers, each of which is serving one graphics application [2].

[*] `tli01@students.poly.edu`. Polytechnic Institute of NYU
[†] `wein@poly.edu`. Polytechnic Institute of NYU

2 Problem 1: Scheduling One Server

We have done substantial experimentation and instrumentation of popular digital games which has enabled us to devise a first simple model of one underlying scheduling problem. We have a number of games to schedule on one server. The server has one CPU and one GPU. Each game $g(i)$ runs in a loop, first requiring $p(i)$ time on the CPU and then (not necessarily immediately thereafter) $q(i)$ time on the GPU. Each game also has a frame rate $f(i)$. At most one game can be utilizing the CPU at any time and at most one game can be utilizing the GPU at any time; the goal is to construct a schedule that respects these constraints while optimizing criteria to be discussed momentarily.

Obviously this initial model is much simplified. It ignores the "multiple threads on multiple cores" nature of the CPU processing, the smaller intervening spikes of CPU and GPU utilization, and the fact that the game likely does not utilize the entire CPU while processing. However, the model captures key elements of the scheduling challenge while maintaining a simplicity that can lead to implementable solutions of value. We have developed a set of test workloads based on our afore-mentioned detailed instrumentations and done simulations to develop appropriate scheduling policies for these workloads.

The goal of a scheduling policy is to schedule as many games on one server as possible while providing a sufficient quality of service to each player. More specifically, if the problem is within capacity - namely it is possible to schedule all the games at their desired frame rates – we'd like to schedule all these games so as to allow each to achieve its' appropriate frame rate. If the problem is over capacity the goal is to schedule all these games as best as possible while bounding the degradation of the player's experiences. We could have several goals in bounding this degradation: define a minimal acceptable frame rate for each game (below its usual rate) and keep each game as close as possible to it, or degrade all games by the same percentage, or reduce the frame rates of the games with higher frame rates first preferentially.

Additionally, a key element in the successful delivery of a game over a network is making the game experience sufficiently responsive so that the end user is unconcerned that the computer that is generating the experience is remote. Claypool and coauthors broadly categorize digital games into three categories: first person shooter, sports/role playing, and real-time strategy, and argue that in order to achieve playability, the games respectively need response times not exceeding 80 ms, 150 ms and 400 ms [1]. While internet latencies are variable and dependent on many factors, as a general rule of thumb 20-30 ms should usually be sufficient for a one-way trip of 1000 miles in the US. So, crudely estimated, if client and server are within 20-30ms of each other, and processing can happen in 20 or so ms, it should be possible to achieve playability.

Thus there are two important elements in achieving playability in a cloud of game servers: (1) efficient processing of user input and production of game state at the server, and (2) delivery of the produced graphical content to the end user. Our focus in this project is on optimizing (1) and exploring how such optimization can lead to relaxation of the constraints for (2). There is, in fact a multidimensional tradeoff between the load on the game server, the service requirements of the games being served, and the set of game servers that are feasible for a particular client. In addition, the production of certain frames must be treated as higher priority, as they must be serviced to achieve the latency requirements in order to indicate responsiveness and achieve playability. If the end user is not doing anything, the rate at which frames are produced can be lower.

If the user enters input, a response must be generated within latency bounds. We call such frames that must be generated within these latency bounds *vital frames*.

We note that the scheduling domain we have presented can be characterized as a two-machine flowshop. Simple variants of that problem, such as optimizing the makespan, can be solved in polynomial time. The variant we consider here however, with the multiple repetitive operations of loop iterations, and the imposed deadlines for vital frames, is likely NP-hard; a simple heuristic with a provable performance bound would be of great value.

We have conducted experiments on workloads generated from our instrumented data, and developed an algorithm called VFDL ("Vital Frame Deadline") that, when compared to naive scheduling approaches, such as FirstComeFirstServed(FCFS), in high-load settings, can achieve better rates of frames delivered per second while enabling all games to be delivered with sufficient responsiveness to insure playability, while FCFS misses playability latency constraints by 10-15% for the majority of served games.

3 Problem 2: Resource Management Across Multiple Servers

As each game to be scheduled on a server iteratively uses first the CPU and then the GPU, the choice of games assigned to a server will substantially impact the resource utilization of that server. For example, if all games assigned to a server had CPU requirements of duration far in excess of their GPU requirements, the GPU would sit idle much of the time. Therefore, there is a need for a game assignment strategy, preferably online, as game players arrive over time, that assigns games to servers in a fashion that balances and optimizes CPU and GPU utilization. We have developed new assignment strategies for this problem but omit discussion in this abstract.

References

[1] Mark Claypool. The effect of latency on user performance in real-time strategy games. *Elsevier Computer Networks, special issue on Networking Issues in Entertainment Computing*, 2005.

[2] Fumihiko Ino, Akihiro Ogita, Kentaro Oita, and Kenichi Hagihara. Cooperative multitasking for gpu-accelerated grid systems. In *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, 2010.

[3] Weidong Shi, Yang Lu, Zhu Li, and J. Engelsma. Scalable support for 3d graphics applications in cloud. In *2010 IEEE 3rd International Conference on Cloud Computing (CLOUD)*, 2010.

Scheduling and performance of divisible MapReduce applications

Joanna Berlińska * Maciej Drozdowski (Speaker) †

1 Introduction

We consider scheduling MapReduce parallel application [2, 5] as two divisible computations working in tandem, and then study its performance determinants.

MapReduce [2, 5] is a distributed processing paradigm which consists in dividing computations on big data sets into two steps of mapping and reducing. In the mapping step a file (the dataset) is distributed between m mapper machines, each of which produces a set of $(key_1, value_1)$ pairs using a *Map* function. The set of $(key_1, value_1)$ pairs produced on a mapper is divided into r parts. Each of such parts is dedicated to one reducer machine. The reducers process the m input data parts using a *Reduce* function and produce yet another set of $(key_2, value_2)$ pairs. For example, in the inverted index computation all documents comprising certain *word*s must be identified. The *Map* function emits pairs $(word, docID)$ for each *word* encountered in the input data, where $docID$ is a document identifier (e.g. a URL). In the *Reduce* function $(word, docID)$ pairs are sorted, and finally pairs $(word, list\_docIDs)$ are emitted, where $list\_docIDs$ is a sorted list of $docID$s of the files comprising the *word*.

Divisible load model is a scheduling model of parallel computations assuming that computations are arbitrarily divisible, and can be performed independently in parallel. The divisible work medium is generally termed load, and in most cases refers to big datasets. The assumption on the divisibility and independence of computation grains is fulfilled with good accuracy by many practical applications: search for patterns in text and database files, processing measurement data, image and video processing, linear algebra. Surveys of Divisible Load Theory (DLT) can be found, e.g., in [1, 3, 4].

The problem consists in dividing the input file of size V into parts $\alpha_1, \ldots, \alpha_m$ for m mapper computers and scheduling communications in a network with limited bandwidth, so that schedule length is minimum.

2 A scheduling model

The two-step MapReduce, from the scheduling point of view, involves five stages of communication and computation. The first stage is computation startup, which includes, e.g, code deployment. We assume that activation of each processor takes S units of time,

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

†Maciej.Drozdowski@cs.put.poznan.pl. Institute of Computing Science, Poznan University of Technology, Piotrowo 2, 60-965, Poznań, Poland.

and the processors are activated consecutively. Thus, all processors are activated in time mS. The second step is mapping. We assume that reading remote data, processing it and storing the results for further reducing is performed with rate A_i on mapper i. Thus, load α_i is processed on the ith mapper in time $A_i\alpha_i$. A constant fraction γ of the results is produced by all mappers. For example, mapper i produces $\gamma\alpha_i$ bytes of output in r files of roughly equal size $\gamma\alpha_i/r$. In the third stage the files are transferred to the reducers over a network with communication rate C, and bandwidth limited to l/C. The limit l means that at most l concurrent communication channels can be opened in the whole network without reduction of the speed. We will call it bisection width limit. Hence, all the results of mapper i could be read by the reducer machines in time $C\gamma\alpha_i$ provided that there were no bisection width limit. Hence, the communications have to be scheduled to avoid congestion. The fourth step is reducing which lasts $t_R = s^{red} + a^{red}x\log x$ units of time for x bytes of input load. Thus, the reducing time includes startup s^{red}, and nonlinear component $a^{red}x\log x$ representing sorting time. The fifth step is storing the reducing results for further processing. However, we assume that the resulting files remain on the reducers, and the duration of this step is included in the reducing time t_R. Since reducers receive roughly equal load $\gamma V/r$ and work independently, the scheduling problem boils to minimizing the length of the partial schedule including the first three stages: startup, mapping, and mapper to reducer communication.

Two algorithms for load partitioning have been proposed. Below we outline the simpler one, which ignores the bisection width limit l. It is assumed in this method that mapper computers are activated one by one, and the time of computation on mapper i and exporting its results to the first reducer is equal to the time of starting mapper $i+1$ and computing on it. This leads to a system of linear equations

$$(A_i + \gamma C/r)\alpha_i = S + A_{i+1}\alpha_{i+1} \qquad \text{for } i = 1, \ldots, m-1 \tag{1}$$

$$\sum_{i=1}^{m} \alpha_i = V \tag{2}$$

which can be solved in $O(m)$ time for α_i. Then, the schedule length is

$$\max_{i=1}^{m}\{iS + \alpha_i(A_i + \gamma C) + \frac{\gamma C}{r}\sum_{j=i+1}^{m}\alpha_j\} + t_R \tag{3}$$

A different algorithm based on linear programming was designed to account for the bisection width limit l.

3 Performance implications

The performance simulations lead us to the following conclusions:

- Since the order of complexity of reducing is higher than for the other stages, MapReduce scales well with the number of reducers r.
- It seems that sorting in the reducing stage is a bottleneck for the performance of MapReduce.
- The amount of results γV produced by the mappers is a key parameter controlling performance of MapReduce. The value of γ determines the balance of the work between mapping and reducing.

- Increasing bandwidth between mappers and reducers, be it by increasing the number of channels l or the communication speed $1/C$, has similar effect.

4 Conclusions and future work

To our best knowledge, it is one of the first attempts of analyzing MapReduce, as well as two divisible applications working in tandem, i.e. when the first one produces load for the second one. Communications between the computations are explicitly scheduled.

Further study may include, for example, unequal load partitioning between the reducers, more advanced scheduling of communications, longer chains of divisible computations.

References

[1] V.Bharadwaj, D.Ghose, V.Mani, T.Robertazzi (1996). *Scheduling divisible loads in parallel and distributed systems*. IEEE Computer Society Press, Los Alamitos, CA.

[2] J.Dean, S.Ghemawat (2004). MapReduce: Simplified Data Processing on Large Clusters, in: Proc. OSDI'04: Sixth Symposium on Operating System Design and Implementation, 137–150. `http://labs.google.com/papers/mapreduce.html`.

[3] M.Drozdowski (2009). *Scheduling for Parallel Processing*. Springer, London.

[4] T.Robertazzi (2003). Ten reasons to use divisible load theory, *IEEE Computer* 36, 63-68.

[5] Wikipedia (2011). MapReduce, `http://en.wikipedia.org/wiki/MapReduce`.

Cyclic Scheduling - New Application and Concept of Core Precedences

Zdeněk Hanzálek (Speaker)[†] Claire Hanen [*] Přemysl Šůcha [†]

1 Introduction

Cyclic resource constrained project scheduling problem with temporal constraints (cyclic RCPSP/TC) is a useful way to model scheduling problems occurring either in code generation for VLIW architectures as well as in production systems. We studied the scheduling problems [1] induced by the IEEE 802.15.4/ZigBee [2], Wireless Sensor Network (WSN), and we observed that the problem has a cyclic scheduling nature.

In this paper, we introduce a new kind of constraints, called core precedence constraints, in a cyclic RCPSP/TC. We use them to model energy saving in a ZigBee scheduling problem. Then we study the impact of these new constraints on an efficient approach for cyclic RCPSP/TC, the decomposed software pipelining. Using 3-SAT we prove that our problem is $\mathcal{NP}$-complete even if there are no resource constraints.

1.1 Motivating problem

The IEEE 802.15.4/ZigBee standards are leading technologies for low-cost, low-power and low-rate WSNs. In the beacon-enabled variant, the paths of the data flows are determined by the tree topology.

We consider that all communication nodes may have sensing or/and actuating capabilities, therefore they can be sources or/and sinks of periodic data flows. The data flows traverse different clusters on their routing paths from the source nodes to the sink nodes. Particular task represents communication of given data flow in the given cluster. Each data flow is represented by an acyclic oriented graph consisting of tasks and their precedence relations. The clusters may have collisions when they are in the neighborhood. Thus, the key problem is to find a periodic schedule which specifies when the clusters are active while avoiding possible inter-cluster collisions and meeting all data flows' end-to-end deadlines.

The cyclic nature of the problem is given by the three aspects: (1) In order to ensure minimal energy consumption, the cluster is active only once during the period. Therefore, all the flows in the cluster are joined together and the cluster is inactive in the rest of the period. (2) The flows can have opposite directions, and therefore the

[*] claire.hanen@lip6.fr, C. Hanen, LIP6, UPMC, Paris, and university of Paris-Ouest-Nanterre-La-Défense, France

[†] hanzalek@fel.cvut.cz, suchap@fel.cvut.cz, Z. Hanzálek, P. Šůcha, Department of Control Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, Technická 2, 166 27, Prague 6, Czech Republic

best phasing (i.e. complete occurrence of the flow is executed in one period) of cluster activity in one direction is the worst phasing in the opposite direction (i.e. complete occurrence of the flow is executed in k periods, where k is the longest chain of clusters traversed by the two flows) . (3) End-to-end delay, given as a time between the instant when a source cluster i starts to communicate the message and the instant when the sink cluster j receives this message, is bounded by deadline $\widetilde{d}_{ij}$.

2 Definition of the problem with core precedences

In the cyclic RCPSP/TC model, a set $\mathcal{T}$ of n tasks with processing time $\{p_i\}_{1\leq i\leq n}$ is supposed to be executed infinitely many times. For each task T_i, and each integer q, we denote by T_i^q the q^{th} occurrence of task T_i. An infinite schedule s assigns to each occurrence T_i^q of task T_i a starting time s_i^q.

We focus on periodic schedules, i.e. schedules such that each task is repeated every λ time unit: $\forall T_i \in \mathcal{T}, \quad s_i^q = s_i + (q-1)\lambda$, where s_i is the starting time of the first occurrence of T_i and λ is the period of the schedule.

The resource constraints are defined as follows: There are m resource types. Availability of resource type $k \in \{1, \ldots, m\}$ is denoted by R_k. To each task T_i and each type of resource k is associated an integer request r_{ik}, such that each occurrence of T_i uses r_{ik} units of type k resource during its execution. If s is a schedule then for any time unit t, and any resource k, the number of resource requirements made by the tasks T_i^q such that $s_i^q \leq t < s_i^q + p_i$ is not greater than R_k.

The usual model for cyclic precedence constraints is called "uniform precedences". Such constraints are defined by a bivalued graph $G_U = (\mathcal{T}, \mathcal{E}_\mathcal{U})$ with integer valuations l and h called respectively length and height of its arcs. For an arc $a = (T_i, T_j)$ we denote by l_{ij} its length and by h_{ij} is height. For a periodic schedule, a uniform constraint induces a simple linear inequality: $s_j - s_i \geq l_{ij} - \lambda h_{ij}$.

This model is convenient to represent the precedence relations within each data flow of our motivating problem. Indeed, if two tasks of data-flow communicate (from a cluster to a neighbor cluster), this communication may induce a precedence constraint with delay between occurrences of these two tasks belonging to the same iteration, and thus induce a uniform precedence constraint with height equal to 0.

The end-to-end delay $\tilde{d}_{ij}$ can be modeled by a uniform constraint from T_j to T_i with length $p_j - \tilde{d}_{ij}$ and height 0.

Let s be a periodic schedule of period λ. It is useful to consider another representation of s: For a task T_i, we decompose s_i, the starting time of its first occurrence modulo period λ. We call retiming of T_i, and denote by α_i the greatest integer such that there exist $\sigma_i \in [0, \lambda)$ with $s_i = \sigma_i + \alpha_i\lambda$. Where $(\sigma_i)_{i\in\{\mathcal{T}\}}$ is called the core of the periodic schedule. Thus, σ_i is an acyclic schedule of tasks that fulfills the resource constraints. The retiming α_i models the span of the schedule of an iteration over different periods.

A uniform precedence constraint (T_i, T_j) induces the following inequality:

$$\sigma_j - \sigma_i \geq l_{ij} - \lambda(h_{ij} + \alpha_j - \alpha_i) \tag{1}$$

2.1 Core precedence constraints

Now, in our example, the constraints that bind the tasks in a cluster in order to insure minimal energy consumption cannot be modeled by uniform precedence constraints.

Thus we introduce a new type of constraints, called *core precedences*, aimed at modeling these constraints. A core precedence constraint links the core execution times of two tasks T_i and T_j with a time lag denoted by $c_{ij} \in \mathbb{Z}$

$$\sigma_j - \sigma_i \geq c_{ij} \tag{2}$$

We consider a graph $G_C = (\mathcal{T}, E_C)$ of such core constraints in addition with an instance of the cyclic RCPSP/TC problem.

Notice that a core precedence links the execution times of tasks occurrences in any period interval of the steady state of a periodic schedule, regardless of the iteration number of the considered occurrences.

So, it shares this indifference with usual resource constraints, whereas it shares the notion of precedence with uniform constraints. Such constraints might also result from partial choices with respect to usual resource constraints.

2.2 Modeling energy efficiency in ZigBee by core precedences

In our motivating example, all tasks of the different data-flows that use a given cluster are supposed to be grouped during each period, so as to minimize the energy consumption, regardless of the iteration number of the concerned data-flow.

This can be modeled by a cluster dummy task T_i representing the starting of the cluster activity with processing time 0. For each task T_j of a data flow passing through the cluster, we can define two core precedences : (1) (T_i, T_j) with core value 0 that model the fact that the data-flow task always starts after the dummy task in any period. (2) (T_j, T_i) with core value $-(x - p_j)$ where x is the maximum allowed activation interval of the cluster in a period.

In this way we ensure that all tasks performed by the cluster are executed within x time units and that the cluster is inactive in the rest of the period.

3 Reported work

At the workshop we will present: (1) ILP model for a cyclic RCPSP/TC with uniform and core constraints assuming fixed period λ. (2) We recall the main features of the decomposed software pipelining (DSP) [3], and we discuss its extension to handle core constraints. (3) Using 3-SAT we prove that our problem is $\mathcal{NP}$-complete even if there are no resource constraints. (4) We present a heuristic algorithm being a priority-rule based method with unscheduling step.

References

[1] Hanzalek, Z., Jurcik, P., Energy efficient scheduling for cluster-tree Wireless Sensor Networks with time-bounded data flows: application to IEEE 802.15.4/ZigBee. IEEE Trans. on Industrial Informatics. Vol. 6/3, pp. 438 - 450, August 2010.

[2] ZigBee Specification, San Ramon, CA,053474r13, 2006.

[3] A. Benabid and C. Hanen, Decomposed Software Pipelining for cyclic unitary RCPSP with precedence delays, Journal of Scheduling, 2011.

Flow Shop Scheduling for Sustainable Manufacturing

Kan Fang* Nelson A. Uhan (Speaker) † Fu Zhao‡
John W. Sutherland §

Under the pressures of global climate change, rising energy costs and the importance of energy security, manufacturing enterprises are paying more and more attention to reducing their energy consumption and carbon footprint. Until recently, most of the focus in this regard has been on improving the energy efficiency of manufacturing equipment. However, using existing machinery more effectively—for example, through better scheduling strategies—may also lead to significant reductions in energy and power consumption and carbon footprint. In this work, we examine scheduling as a means to address the environmental concerns arising with manufacturing enterprises. In particular, we study a shop scheduling problem with energy- and environment-related criteria, in addition to the classic time-related objectives.

The scheduling problem we study is as follows. We are given a set of jobs $\mathcal{J} = \{1, \ldots, n\}$ and a set of machines $\mathcal{M} = \{1, \ldots, m\}$ in a flow shop environment. In particular, each job $j \in \mathcal{J}$ must be processed nonpreemptively on each of the machines, and the machines are ordered so that a job cannot start on machine i until it is completed on machine $i-1$, for $i = 2, \ldots, m$. Each job on each machine can be run at a different speed. We consider two different assumptions on the available machine speeds:

(i) *Discrete speeds.* In this case, the processing time of job $j \in \mathcal{J}$ on machine $i \in \mathcal{M}$ at speed $s \in \mathcal{S} = \{1, \ldots, d\}$ is p_{ijs}. The power consumption of machine $i \in \mathcal{M}$ when processing job $j \in \mathcal{J}$ at speed $s \in \mathcal{S}$ is q_{ijs}. Note that this case allows for an arbitrary relationship between speed and power.

(ii) *Continuous speeds.* In this case, job $j \in \mathcal{J}$ on machine $i \in \mathcal{M}$ requires p_{ij} units of work. A job that requires p units of work and is processed at speed s has a processing time of p/s. In addition, the power consumption of a machine running at speed s is Ks^α for some constants $\alpha \geq 1$ and $K \geq 0$.

We also consider two different assumptions on the intermediate storage between machines: (i) *unlimited intermediate storage*, and (ii) *zero intermediate storage* (sometimes referred to as "blocking"), in which a completed job cannot proceed to the next machine until all jobs already being processed on the next machine are finished. The *peak power consumption* of a schedule is the maximum total power consumption of all machines over all time instants. Peak power consumption is a common basis of energy costs in the

*fang19@purdue.edu. School of Industrial Engineering, Purdue University, West Lafayette, IN, USA.
†nuhan@purdue.edu. School of Industrial Engineering, Purdue University, West Lafayette, IN, USA.
‡fzhao@purdue.edu. Division of Environmental and Ecological Engineering and School of Mechanical Engineering, Purdue University, West Lafayette, IN, USA.
§jwsuther@purdue.edu. Division of Environmental and Ecological Engineering, Purdue University, West Lafayette, IN, USA.

manufacturing industry. The objective is to find a schedule that minimizes the makespan while ensuring that the peak power consumption is at most $Q_{\max}$.

There is a considerable body of related work on speed scaling scheduling in the computer science community. For an older survey of the work in this area, we refer the reader to [5]. Some examples of more recent work includes [1–4, 8, 9] (note that this list is by no means complete). In this body of work, each job typically needs to be processed on a single machine (either in single or multiple machine environments). This differs from typical manufacturing environments, in which each job needs to be routed through multiple machines in a shop environment. In contrast to the speed scaling literature, the literature on reducing environmental impacts in manufacturing scheduling is rather sparse. Mouzon et al. [6] considered the problem of scheduling jobs on a single CNC machine to minimize the total energy consumption and total completion time, and showed that better scheduling strategies significantly reduce total energy consumption. Mouzon and Yildrim [7] looked at the same single machine problem, except minimizing total energy consumption and total tardiness. To the best of our knowledge, our work is the first to consider shop scheduling with energy- and environment-related constraints.

Contributions of this work. We investigate both mathematical programming and combinatorial approaches to the different variations of the scheduling problem described above. In particular:

- For the case of discrete speeds and unlimited intermediate storage, we propose two mixed integer linear programs, one using linear ordering variables with disjunctive constraints, and the other using positional and assignment variables. We investigate the relative strength of these formulations. In addition, using CPLEX and Gurobi, we study the computational performance of these two formulations on a set of hypothetical instances arising from the manufacturing of cast iron plates with slots. Preliminary results indicate that even relatively small instances ($m = 2, n = 36, d = 2$) are computationally challenging. These mixed integer programs can be extended to include constraints on the *carbon footprint* of a schedule. The carbon footprint of a schedule is linearly proportional to the schedule's total energy consumption, where energy is power integrated over time.

- For the case of continuous speeds, zero intermediate storage, and two machines ($m = 2$), we show that there exists an optimal schedule in which the total power consumption at any time instant is exactly $Q_{\max}$. In addition, we show that this problem can be seen as a special case of the asymmetric traveling salesman problem. Furthermore, if the work of jobs is *consistent* across machines (i.e. $p_{1j} \leq p_{1k}$ implies $p_{2j} \leq p_{2k}$ for all $j, k \in \mathcal{J}$), then an optimal schedule can be found in $O(n^2)$ time. In addition, if $p_{1j} = p_{2j} \equiv p_j$ for each job $j \in \mathcal{J}$, then an optimal schedule can be found in $O(n \log n)$ time; in particular, if the jobs are sorted so that $p_1 \leq p_2 \leq \cdots \leq p_n$, then the schedule $(1, 3, 5, \ldots, n, \ldots, 6, 4, 2)$ is optimal.

- For the case of discrete speeds, zero intermediate storage, and two machines, we show again that the scheduling problem can be seen as a special case of the asymmetric traveling salesman problem.

- For the case of continuous speeds, unlimited intermediate storage, and two machines, we show that when the total power consumption at any time instant is ex-

actly $Q_{\max}$, given a fixed permutation of the jobs, an optimal schedule (i.e. machine speeds for each job) can be found in polynomial time by dynamic programming.

References

[1] S. Albers and H. Fujiwara. Energy-efficient algorithms for flow time minimization. *ACM Transactions on Algorithms*, 3:49:1–49:17, 2007.

[2] N. Bansal, T. Kimbrel, and K. Pruhs. Speed scaling to manage energy and temperature. *Journal of the ACM*, 54:3:1–3:39, 2007.

[3] N. Bansal, K. Pruhs, and C. Stein. Speed scaling for weighted flow time. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 805–813, 2007.

[4] D. P. Bunde. Power-aware scheduling for makespan and flow. *Journal of Scheduling*, 12:489–500, 2009.

[5] S. Irani and K. R. Pruhs. Algorithmic problems in power management. *ACM SIGACT News*, 36:63–76, 2005.

[6] G. Mouzon, M. B. Yildirim, and J. Twomey. Operational methods for minimization of energy consumption of manufacturing equipment. *International Journal of Production Research*, 45:4247–4271, 2007.

[7] G. Mouzon and M. B. Yildrim. A framework to minimise total energy consumption and total tardiness on a single machine. *International Journal of Sustainable Engineering*, 1:105–116, 2008.

[8] K. Pruhs, P. Uthaisombut, and G. Woeginger. Getting the best response for your erg. *ACM Transactions on Algorithms*, 4:38:1–38:17, 2008.

[9] K. Pruhs, R. van Stee, and P. Uthaisombut. Speed scaling of tasks with precedence constraints. *Theory of Computing Systems*, 43:67–80, 2008.

Mathematical modelling of a real flexible job shop in aero engine component manufacturing

Karin Thörnblad (Speaker) [*] Michael Patriksson [†]
Ann-Brith Strömberg [‡] Torgny Almgren [§]

1 Introduction

We formulate two mixed integer programming models stemming from a real flexible job shop problem with a total of ten resources and five main processing multipurpose machines. The models are compared w.r.t. memory usage, computation times, and accuracy.

The job-shop, called the multitask cell, is supposed to carry out a large variety of jobs since five of the cell's resources are multi-purpose machines that are able to process three types of operations: milling, turning, and drilling. Typically, each product visits the multitask cell multiple times on its way to completion. Inside the multitask cell each part follows a specific routing consisting of three to five operations, starting and ending respectively with the mounting and removing of fixtures at one of the three set-up stations. The second operation is always processed in one of the multitask machines. Some parts need manual and/or robot deburring. Figure 1 shows a part's possible path in the cell drawn with dashed lines.

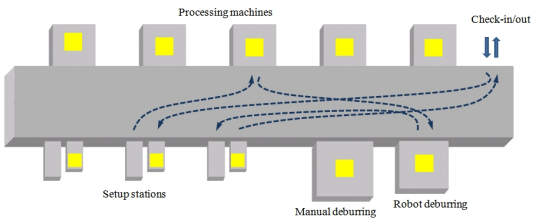


Figure 1: A schematic overview of the production cell.

[*] karint@chalmers.se. Department of Mathematical Sciences, Chalmers University of Technology and University of Gothenburg, 412 96 Göteborg, Sweden.
[†] mipat@chalmers.se. Department of Mathematical Sciences, Chalmers University of Technology and University of Gothenburg, 412 96 Göteborg, Sweden.
[‡] anstr@chalmers.se. Department of Mathematical Sciences, Chalmers University of Technology and University of Gothenburg, 412 96 Göteborg, Sweden.
[§] torgny.almgren@volvo.com. Department of Logistics Development, Volvo Aero Corporation, 461 81 Trollhättan, Sweden.

2 The time-indexed formulation

Due to initial problems with high computation times, the model has been decomposed into two to be solved in sequence. The first model, called the *machining model*, finds an optimal sequence of operations for each of the five processing machines. The second model, henceworth called the *feasibility model*, generates a feasible schedule for all ten resources, with the optimal sequence for the five processing machines as input data. The loss of accuracy from this decomposition is very small since the workload of the processing resources are much higher than for the other resources. In [1] we present the first developed machining model, here called the *engineer's model*. The feasibility model is based on the same logic. These two models work well, but the computation times still are far too long for practical usage.

Therefore, we have developed a new model using discrete time steps, inspired by [2] and [3], in order to find the optimal sequences of operations for the processing resources. The discretization approximates all data in multiples of the length of the discrete time interval chosen, and hence the final schedule may alter from the result of the engineer's model.

The main decision variables are defined as

$$x_{jku} = \begin{cases} 1, & \text{if job } j \text{ starts at resource } k \text{ at the beginning of time interval } u, \\ 0, & \text{otherwise.} \end{cases}$$

The discrete time machining model minimizes the sum of the total tardiness (h_j) and completion time (s_j) for all jobs $j \in \mathcal{J}$ and is formulated as to

$$\text{minimize} \quad \sum_{j\in\mathcal{J}} (s_j + h_j), \tag{1a}$$

$$\text{subject to} \quad \sum_{k\in\mathcal{K}} \sum_{u\in\mathcal{T}} x_{jku} = 1, \quad j \in \mathcal{J}, \tag{1b}$$

$$\sum_{u\in\mathcal{T}} x_{jku} \le \lambda_{jk}, \quad j \in \mathcal{J}, k \in \mathcal{K} \tag{1c}$$

$$\sum_{\nu=(u-p_j+1)_+}^{u} x_{jk\nu} + \sum_{\mu=(u-p_q+1)_+}^{u} x_{qk\mu} \le 1, \quad j,q \in \mathcal{J}, j \ne q, k \in \mathcal{K}, u = 0,\dots,T, \tag{1d}$$

$$\sum_{k\in\mathcal{K}} \left(x_{jku} + \sum_{\nu=0}^{u+v_{jq}^{\mathtt{pm}}-1} x_{qk\nu} \right) \le 1, \quad (j,q) \in \mathcal{Q}, u = 0,\dots,T - v_{jq}^{\mathtt{pm}} + 1, \tag{1e}$$

$$x_{jku} = 0, \quad (j,q) \in \mathcal{Q},\ k \in \mathcal{K},\ u = T - v_{jq}^{\mathtt{pm}},\dots,T, \tag{1f}$$

$$\sum_{k\in\mathcal{K}} \sum_{u\in\mathcal{T}} u x_{jku} + p_j + p_j^{\mathtt{pm}} = s_j, \quad j \in \mathcal{J}, \tag{1g}$$

$$s_j - h_j \le d_j, \quad j \in \mathcal{J}, \tag{1h}$$

$$h_j \ge 0, \quad j \in \mathcal{J}, \tag{1i}$$

$$x_{jku} = 0, \quad j \in \mathcal{J},\ k \in \mathcal{K},\ u = 0,1,\dots,r_j, \tag{1j}$$

$$x_{jku} = 0, \quad j \in \mathcal{J},\ k \in \mathcal{K},\ u = 0,1,\dots,a_k, \tag{1k}$$

$$x_{jku} \in \{0,1\}, j \in \mathcal{J},\ k \in \mathcal{K},\ u \in \mathcal{T}, \tag{1l}$$

where $(u)_+ := \max\{0, u\}$ for all $u \in \mathcal{T}$, r_j and d_j denotes realease and due dates respectively, $v_{jq}^{\mathtt{pm}}$ is the planned lead time between preceding machining operations, p_j is the processing time, and λ_{jk} is 1 if a job is allowed to be processed on a resource, and 0 otherwise.

3 Computational results

Presently, the multitask cell is processing about 30 different machining operations on eight different products. Results from the engineer's model and the discrete time model with varying discrete time intervals were compared for six real data scenarios collected from the cell during the autumn of 2010. For each scenario, we created up to ten different instances, comprising the first $5m$ jobs in the current queue to be scheduled, where $m \in \{1, \ldots, 10\}$.

The starting and completion times obtained from the optimal solution of the discrete machining model are given in terms of multiples of the length of the discrete time interval ℓ, and the optimal value of the objective function differs from that of the engineer's model. Therefore, the completion times s_j are recalculated using the original non-discrete data while retaining the ordering of operations on each processing machine received from the discrete time model. The optimal value after this postprocessing is then compared with the optimal value from the engineer's model.

Although the smallest processing time for the multipurpose machines was 0.6h (the largest was 22.4h), the optimal value of the discrete time maching model with a discrete interval length of at most 1h were identical with that of the engineer's model for all instances except one with 20 jobs, that we were able to solve using this model.

The computations were carried out using AMPL-CPLEX12 as optimization software on a computer with two 2.66GHz Intel Xeon 5650, each with six cores. The total memory was 48 Gbyte. Optimum was reached for all problem sizes with at most 15 jobs for the engineer's model, and with at most 45 jobs for the discrete time model. No results were obtained for four of the instances with 20 jobs and more for the engineer's model, since CPLEX ran out of memory, and the same happened for instances with 50 jobs and more, for the discrete time model with time interval length of 1h. This should however, not cause any problem for the real application, since the maximum amount of storage in the multitask cell is around 35 parts.

References

[1] K. Thörnblad and A-B. Strömberg and T. Almgren and M. Patriksson (2010). *Optimization of schedules for a multitask production cell.* 22nd Nofoma conference proceedings, Kolding, Denmark.

[2] J.M van den Akker and C.A.J. Hurkens and M.W.P Savelsberg (2000). *Time-Indexed Formulations for Machine Scheduling Problems: Column Generation.* INFORMS Journal on Computing 12, 111–124.

[3] L.A. Wolsey (1997). *MIP modelling of changeovers in production planning and scheduling problems.* European Journal of Operational Research 99, 154–165.

An exact polynomial time algorithm for the preemptive mixed shop problem with two unit operations per job

Aldar Dugarzhapov * Alexander Kononov (Speaker) †

1 Introduction

Shop scheduling problems appear ubiquitously in modeling of many real-life phenomena such as manufacturing production lines, packet exchange in communication networks, timetabling, etc. These problems received much attention from researchers in different communities. Recently, new exact polynomial-time algorithms were obtained for job shop and mixed shop scheduling problems with at most two unit operations per job. We consider a preemptive counterparts of these shop scheduling problems and present new exact polynomial time algorithms for the mixed shop problems.

Mixed Shop Scheduling Problem. In shop scheduling problems, the processing of each job is split into several operations. We are given a set of n *jobs* $\mathcal{J} = \{J_1, \ldots, J_n\}$ and a set of m *machines* $\mathcal{M} = \{M_1, \ldots, M_m\}$. Each job J_j consists of ν_j *operations* $O_{1j}, \ldots, O_{\nu_j j}$, where each operation O_{ij} has to be completed on the specific machine $M_k \in \mathcal{M}$ during p_{ij} time units. The value p_{ij} is called the *processing time* or *length* of the operation O_{ij}. The schedule is called *feasible* if at any moment in time every job can be processed by at most one machine and every machine can execute at most one operation. Every operation can be preempted allowing the execution of other operations. The execution of the interrupted operation can be resumed later on the same machine. The goal is minimizing the makespan.

The two basic models of shop scheduling problems or their sub-problems are considered in scheduling literature.

Job Shop: The operations $O_{1j}, \ldots, O_{\nu_j j}$ of the job J_j define a sequence in which they must be processed. In other words, the operation $O_{i-1,j}$ must be completed before the operation $O_{i,j}$ starts.

Open Shop: The operations $O_{1j}, \ldots, O_{\nu_j j}$ of the job J_j can be executed in arbitrary order.

A mixed shop is a combination of the job shop and the open shop and an instance of the mixed shop problem can include the job shop jobs as well as the open shop jobs. In this paper we will use the standard three-field scheduling notations. Job shop, open shop and mixed shop problems with the goal to minimize the length of the schedule will

*aldar.dugarzhapov@polytechnique.org. Ecole Polytechnique, Palaiseau, France.
†alvenko@math.nsc.ru. Sobolev Institute of Mathematics, Koptyuga 4, 630090, Novosibirsk, Russia.

be denoted respectively as $\langle J| * |C_{max}\rangle$, $\langle O| * |C_{max}\rangle$ and $\langle JO| * |C_{max}\rangle$, where in the field $*$ the different restrictions on the problem parameters are listed. In our paper we use the following restrictions: $p_{ij} = 1$ means that all processing time of operations are equal to 1, $op \leq 2$ means that each job has at most two operations. In addition, symbol $pmtn$ means that operations of jobs may be preempted.

2 Related results

To the best of our knowledge, Masuda et al. [3] considered the mixed shop problem for the first time. They considered the non-preemptive case with two machines, at most two operations per job and where all job shop jobs must be executed in the same order. In that paper, an $O(n \log n)$ algorithm was proposed to construct a schedule with minimum makespan. This result was improved by Strusevich in [7], [8], a polynomial time algorithm was developed for the two-machine mixed shop problem with an arbitrary order of executions of job shop jobs. We note that the different cases of the preemptive and non-preemptive two-machine problems became NP-hard even for two jobs if the number of operations is not fixed [2], [5]. For the comprehensive survey of the complexity of other mixed shop problems see paper by Shakhlevich et al. [6].

In our paper we consider the special case of the preemptive problem with an arbitrary number of machines. We suppose that each job has at most two unit operations, i.e. the processing time of each operation is equal to one. We denote this problem as $JOm|pmtn, p_{ij} = 1, op \leq 2|C_{\max}$ if the number of machines is fixed and equal to m and $JO|pmtn, p_{ij} = 1, op \leq 2|C_{\max}$ if the number of machines is the part of the input.

The restriction on the number of operations per job plays the crucial role to design exact polynomial algorithms for the job shop and the mixed shop scheduling problems. As shown in [1], the multigraph edge coloring theorem due to Melnikov and Vizing [4] can be reformulated for the non-preemptive job shop scheduling problem with unit processing times and at most two operations per job. This result implies polynomial time algorithm for the job shop problem. Next in [1], this algorithm was extended to the non-preemptive mixed shop problem.

3 Our results

Let us consider preemptive counterparts of shop scheduling problems with at most two unit operations per job. We note that for any problem instance of the job shop problem with integral data there exists an optimal preemptive schedule where all interruptions and all starting and completion times occur at integral dates. Thus, the preemptive job shop scheduling problem with at most two unit operations per job is redundant to its non-preemptive version and can be solved in polynomial time.

For the mixed shop problem the situation is different. Let us consider the following instance with four machines, four job shop jobs J_2, J_3, J_4, J_5 and two open shop jobs J_1 and J_6. Jobs J_2 and J_5 have first operations on machine 3. Jobs J_3 and J_4 have first operations on machine 4. Jobs J_2 and J_4 have second operations on machine 2. Jobs J_3 and J_5 have second operations on machine 1. Job J_1 has operations on machine 1 and 2. Job J_6 has operations on machine 3 and 4. An optimal schedule has a length $\frac{7}{2}$ and it is shown on Fig. 1.

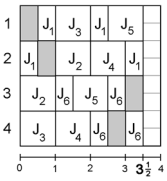


Figure 1: An instance with a fractional optimal schedule

In our paper we present an exact polynomial time algorithm for the preemptive mixed shop scheduling problem with at most two operations per jobs and obtain the following results.

Theorem 1. *Each instance of* $JO3|pmtn, p_{ij} \leq 1, op \leq 2|C_{max}$ *allows an optimal schedule without preemptions.*

Theorem 2. *Problem* $JO|pmtn, p_{ij} = 1, op \leq 2|C_{\max}$ *is solvable in polynomial time.*

Acknowledgements

The second author was supported by the Russian Foundation for Basic Research (grant no. 09-01-00059).

References

[1] A. Kononov, S. Sevastyanov, M. Sviridenko, Complete Complexity Classification of Short Shop Scheduling, In: A. Frid et al. (ed.) CSR 2009. LNCS, vol. 5675, pp. 227-236. Springler, Heidelberg (2009)

[2] J.K. Lenstra, A.H.G. Rinnooy Kan, Computational complexity of discrete optimization problems, *Annals of Discrete Mathematics* 4 (1979), 121-140.

[3] T. Masuda, H. Ishii, T. Nishida, The mixed shop scheduling problem, *Discrete Applied Mathematics* 11 (2), (1985), 175-186.

[4] L. Melnikov, V. Vizing, The edge chromatic number of a directed/mixed multigraph, *Journal Graph Theory* 31 (4) (1999), 267-273.

[5] N.V. Shakhlevich, Yu.N. Sotskov, F. Werner, Shop-scheduling problems with fixed and non-fixed orders of the jobs, *Preprint N 14, Otto-von-Guericke-Universitat, Magdeburg*, (1997).

[6] N.V. Shakhlevich, Yu.N. Sotskov, F. Werner, Complexity of mixed shop scheduling problems: A servey, *European Journal of Operational Research* 120, (2000), 387-397.

[7] V.A. Strusevich, On nonhomogeneous two-stage deterministic processing systems, *Kibernetica* 3, (1989), 88-94 (in Russian).

[8] V.A. Strusevich, Two machine super-shop scheduling problem, *Journal of the Operational Research Society* 42 (6), (1991), 479-492.

Efficient enumeration of optimal and approximate solutions of the two-machine flow-shop problem

Sergey Sevastyanov (Speaker) * Bertrand M.T. Lin †

We consider the "most classical" scheduling problem stated by Johnson in his seminal paper [6], where he presented an exact efficient algorithm for the case of two machines. (As is known now due to Garey, Johnson and Sethi [4], already three-machine problem is strongly NP-hard.) According to the common three-field classification [5], the problem is normally denoted as $F2\,||\,C_{\max}$. It can be formulated as follows.

We are given n jobs $\{J_1, \ldots, J_n\} = \mathcal{J}$ that should be processed on two machines: M_1 and M_2. Each job J_j has exactly two operations (O_{j1} and O_{j2}) that must be executed in the predefined order: first O_{j1} on machine M_1, and next O_{j2} on machine M_2, with processing times a_j and b_j respectively. Each machine may process its operations also consecutively, in a certain order that should be chosen in the course of problem solution. One needs to find a schedule S with the minimum length, or the minimum *makespan* denoted as $C_{\max}(S)$.

As follows from the general project scheduling theory, to minimize the makespan for a given flow shop, it suffices to restrict our consideration to *active schedules* only, each uniquely defined by sequences of operations on machines M_1 and M_2. Moreover, as shown by Johnson [6], we need not enumerate all combinations of two sequences of operations on the machines. It suffices to consider only pairs of identical job sequences on both machines (so called *permutational schedules* S_π specified by a unique permutation π of job indices), assuming that all n jobs (including those having zero length operations on the corresponding machine) are elements of each sequence.

In his analysis of optimality of a given sequence, Johnson defined a relation "$\preceq$" on the set of jobs:

$$J_i \preceq J_j, \text{ if } \min\{a_i, b_j\} \leq \min\{a_j, b_i\}.$$

He found a sufficient condition for a sequence of jobs to be optimal (so called, *Johnson's rule*): a permutation of jobs (job indices) $\pi = (\pi_1, \ldots, \pi_n)$ is optimal, if

$$J_{\pi_i} \preceq J_{\pi_j}, \text{ for all } i < j. \tag{1}$$

And though the relation "$\preceq$" does not define a linear order (for it is not transitive: for three jobs J_1, J_2, J_3 with vectors of processing times $(5,4), (2,2), (3,4)$ respectively, we have $J_1 \preceq J_2$, $J_2 \preceq J_3$, which does not imply $J_1 \preceq J_3$), Johnson showed that for any

*seva@math.nsc.ru. Sobolev Institute of Mathematics, Novosibirsk State University, Novosibirsk, 630090, Russia. Supported by the Russian Foundation for Basic Research (grants nos. 08-01-00370 and 08-06-92000-HHC-a) and by the Federal Target Grant "Scientific and educational personnel of innovation Russia" for 2009-2013 (government contract no. 02.740.11.0429).

†bmtlin@mail.nctu.edu.tw. Institute of Information Management, Department of Information and Finance Management, National Chiao Tung University, Hsinchu, Taiwan.

problem instance there always exist job sequences (being further referred to as *Johnson's sequences*) that satisfy (1). (As is well known, there may be many such sequences.)

A part of sequences implementing Johnson's rule can be found with the help of two ideas (so called, *special Johnson's rule*). Let $\delta_j \doteq b_j - a_j$ be called a *balance* of job J_j, and let the whole set of jobs be partitioned into two subsets: $\mathcal{J}_L$ and $\mathcal{J}_R$. All jobs in $\mathcal{J}_L$ (called *left jobs*) have non-negative balance, and all jobs in $\mathcal{J}_R$ (*right jobs*) have non-positive balance. The distribution of zero-balanced jobs among these two subsets may be taken arbitrarily (which does not affect the ultimate result). The first idea is that the left jobs should go first, and the right jobs should go next. The second idea prescribes the left jobs follow in nondecreasing order of a_j, and right jobs — in nonincreasing order of b_j. Since such a permutation meets (1), it defines an optimal active schedule. (As became known later, any such permutation can be found in $O(n \log n)$ time.)

It can be seen that in the case when all operation processing times are different, Johnson's rule defines a unique job sequence, because relation (1) becomes a transitive anti-symmetric relation. Yet in the case when some left jobs have equal-length operations on machine M_1, or some right jobs have equal-length operations on machine M_2, or there are jobs with zero balance, Johnson's sequence becomes not uniquely defined. In general, Johnson's rule provides a wide variety of optimal job sequences, giving a possibility for choosing the desired solution from among a wide family of optimal solutions and enabling one to optimize some additional criteria.

Among the papers addressing the issue of enumeration of all job sequences that meet Johnson's rule, we can mention the book by Bellman, Esogbue and Nabeshima [1], where a so called *General Working Rule* (GWR) was derived, enabling one to generate any Johnson's sequence of jobs. This rule can be easily transformed into an efficient procedure of enumerating all Johnson's sequences without repetition.

However, it is clear that in general case the set of optimal sequences is not limited to Johnson's sequences only, because rule (1), while being sufficient, is not necessary for a sequence of jobs to be optimal. So, the problem of enumerating all optimal permutations of jobs for $F2\,||\,C_{\max}$ problem is more general, and is of its own interest. A number of authors addressed this issue, trying to design various enumeration procedures that would be more effective than the direct enumeration of all $n!$ permutations of n jobs (see, e.g., [8], [9], [3], [2], [7]). Yet they could not guarantee efficiency of their procedures. Moreover, in none of the above mentioned papers a definition of "efficiency" for an enumerative procedure was given. In this paper we adhere the following definitions.

Definition 1. We say that a listing algorithm has *polynomial delay* if the time required to generate each new output element (as well as the time after the last output before halting) is polynomial in the input size.

Definition 2. We say that a listing algorithm is *polynomial space* if at any step it consumes memory space polynomial in the input size.

In this paper we present the first (up to our knowledge) algorithm that enumerates all optimal job permutations for the two-machine flow-shop problem with polynomial delay and polynomial space. In fact, our result is somewhat more general.

Definition 3. A sequence π of job indices is Δ-*optimal*, if $C_{\max}(S_\pi) \leq B + \Delta$, where B is the total load of machine M_2.

Our main result consists in designing an algorithm $\mathcal{A}(\Delta)$ and proving the following

Theorem 4. *For any given $\Delta \geq 0$ and a given instance of problem $F2\,||\,C_{\max}$, algorithm $\mathcal{A}(\Delta)$ enumerates all Δ-optimal permutations of n jobs without repetition. The algorithm is polynomial-delay, spending $O(n \log n)$ time for generating each new permutation and for halting, when all proper permutations are enumerated. The algorithm is polynomial-space, consuming $O(n)$ memory space at every step of the algorithm.*

The result of this type may be useful for purposes of multi-criteria optimization.

Our next result concerns the structure of the set of optimal solutions. Given an instance of our problem, let its Δ-optimal permutations of jobs be considered as vertices of a graph G_Δ. Two permutations π' and π'' are connected by an edge, iff one permutation can be obtained from the other by a transposition of two neighboring elements. We proved the following

Theorem 5. *For any $\Delta \geq 0$ graph G_Δ is connected.*

Corollary 6. *Any optimal permutation of jobs can be obtained from any Johnson's (optimal) permutation by consecutive transpositions of neighboring elements, so that all intermediate permutations are also optimal.*

References

[1] R. Bellman, A.O. Esogbue, and I. Nabeshima (1982). *Mathematical aspects of scheduling and applications*, Pergamon Press, Oxford.

[2] J.C. Billaut and P. Lopez (1997). New results for the enumeration of optimal job sequences, Unpublished manuscript.

[3] T.C.E. Cheng (1992). Efficient implementation of Johnson's rule for the $n/2/F/F_{\max}$ scheduling problem, *Computers ind. Engng.*, **22**, 495–499.

[4] M.R. Garey, D.S. Johnson and R. Sethi (1976). The Complexity of Flowshop and Jobshop Scheduling, *Mathematics of Operations Research*, **1**, 117Ű-129.

[5] R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnoy Kan (1979). Optimization and approximation in determenistic sequencing and scheduling: A survey. *Annals of Discrete Mathematics* **5**, 287–326.

[6] S.M. Johnson (1954). Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, **1**, 61–67.

[7] Y. Lin and J. Deng (1999). On the structure of all optimal solutions of the two-machine flowshop scheduling problem, *OR Transactions*, **3**(2), 10–20.

[8] S.N.N. Pandit and Y.V. Subrahmanyam (1975). Enumeration of all sequences, *Opsearch*, **12**, 35–39.

[9] W. Szwarc (1981). Extreme solutions of the two machine flow shop problem, *Naval Research Logistics Quarterly*, **28**(1), 103–114.

Efficient enumeration of optimal and approximate solutions of the two-machine flow-shop problem

Roman Čapek (Speaker) * Přemysl Šůcha Zdeněk Hanzálek

1 Introduction

This research is motivated by the production processes that involve several ways how to complete a product (e.g. wire harnesses production in Styl Plzeň). In other words, more process plans can be defined for one type of a product. Since only one of these process plans has to be chosen, we call them *alternative process plans*. Traditional scheduling approach (see [1]) assumes an exactly given set of activities and therefore there is only one process plan. We use an approach extended by a formulation of alternative process plans where so called *alternative branchings* can be defined to model the possibility of choice. In our approach, both the traditional scheduling and the decision which activities will be present in the schedule are integrated into one problem.

In this paper, we focus on the scheduling on dedicated machines considering sequence dependent setup times and alternative process plans while the goal is to minimize the total weighted tardiness (TWT).

2 Related Works

Abdul-Razaq et al. [2] and Madureira [3] published a survey of scheduling techniques for the single machine total weighted tardiness (STWT) problem. Recently, many authors employed evolutionary algorithms to solve the total weighted tardiness problem. Chou [4] presented an experienced learning genetic algorithm, Anghinolfi [5] used a discrete particle swarm optimization and Tasgetiren [6] published a discrete differential evolution method for solving the STWT problem. On the other hand, up to our best knowledge there is no work dealing with the total weighted tardiness and alternative process plans on dedicated machines and therefore we focus on such a case. In our previous work ([7]), we have focused on the resource constrained project scheduling problem with positive and negative time-lags while minimizing the schedule length. In the present paper, we also use the alternative process plans but the criterion is adapted to take into account due dates. Moreover, problem constraints are also modified such that simple precedences are assumed instead of generalized temporal relations.

*`capekrom@fel.cvut.cz`. Department of Control Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, Czech Republic.

3 Problem Statement and Solution Method

We consider a problem of scheduling a set of n activities $\mathcal{A} = \{1 \dots n\}$ on a set of m dedicated machines $\mathcal{M} = \{M_1 \dots M_m\}$. Processing time p_j, due date d_j and the machine dedication θ_j are given for each activity. The problem can be formulated as $PD\,|\text{res}\ m11, o_{ij}, alts|\sum w_j \cdot T_j$ using the well-know $\alpha|\beta|\gamma$ notation [1, 8]. More precisely, we deal with the non-preemptive scheduling where activities are constrained by the precedence relations (included in alternative process plans). The term 'res $m11$' represents the fact that each activity requires at most one unit of a resource and each resource has a capacity equal to one. Finally, o_{ij} denotes the presence of the sequence dependent setup times and $alts$ stands for the alternative process plans. The goal of the scheduling is to minimize the objective function $\sum w_j \cdot T_j$, i.e. the total weighted tardiness where $T_j = \max(0, s_j + p_j - d_j)$ stands for the tardiness of activity j, s_j is the start time and w_j is the weight of activity j.

To solve the proposed model, we use the discrete differential evolution (DDE) algorithm proposed by Tasgetiren [6]. For our $PD\,|\text{res}\ m11, o_{ij}, alts|\sum w_j \cdot T_j$ problem, we have to extend the selected method to handle alternative process plans at first. The algorithm has to return not only the sequence of activities on each machine (and consequently also start times of activities) but also the presence of activities in the schedule. Therefore, we have to extend the population based method by a functionality of selecting and rejecting of activities according to the defined alternative process plans. Furthermore, the algorithm has to be also extended to handle different machine environment - dedicated machines instead of single machine. Precedence relations between activities are also considered.

The advantage of the population based method is the utilization of an intensification phase for individuals. In this phase, the algorithm can profit from the specific features of the problem and improve individuals using local search or more complex method in each iteration of an evolution. In this work, we would like to investigate the properties of our problem and focus on estimation of the lower and upper bounds of the objective function. The DDE method uses so called *Destruct and Construct* (DC_d) procedure where a subset of d activities is removed from the schedule and then the activities are reinserted into different positions in the schedule. Since we have to evaluate the objective function repeatedly for each activity insertion into a partial schedule, a fast evaluation and a good estimation of the bounds for the objective function is a great benefit for the algorithm effectiveness. Due to the presence of the precedence relations, it is also necessary to estimate the earliest start times of activities in a schedule with respect to the definition of alternative process plans.

4 Contribution and Future Work

The main contribution of our work is in the interconnection of the scheduling with alternative process plans and the objective function taking into account the total weighted tardiness of activities. We propose a model allowing to define different ways how to complete a product while meeting of due dates is considered. For the given problem, we propose a population based algorithm able to solve instances. The key point of successful implementation of the evolutionary algorithm is an addition of alternative process plans into the representation of individuals and also into the evolution scheme.

In the future work, we will focus on the extension of an objective function by the total processing costs. In this case, a processing cost is assigned to each activity and the goal is to create a schedule such that both the total weighted tardiness and the total processing costs are minimized. An extended approach can be used, for example, to model problems with discretely controllable processing times (or other activity attributes).

References

[1] J.Blazewicz, K.H.Ecker, E.Pesch, G.Schmidt and J.Weglarz: *Scheduling Computer and Manufacturing Processes*. Springer-Verlag New York, 1996.

[2] T.S.Abdul-Razaq, C.N.Potts and L.N.Van Wassenhove: *A survey of algorithms for the single machine total weighted tardiness scheduling problem*. Discrete Applied Mathematics, Volume 26, pp. 235Ű253, 1990.

[3] A.M.Madureira: *Meta-heuristics for the single-machine scheduling total weighted tardiness problem*. Proceedings of the IEEE International Symposium on Assembly and Task Planning, pp.405-410, 1999.

[4] F.D.Chou: *An experienced learning genetic algorithm to solve the single machine total weighted tardiness scheduling problem*. Expert Systems with Applications, Volume 36, pp. 3857-3865, 2009.

[5] D.Anghinolfi and M.Paolucci: *A new discrete particle swarm optimization approach for the single-machine total weighted tardiness scheduling problem with sequence-dependent setup times*. European Journal of Operational Research, Volume 193, pp. 73-85, 2009.

[6] M.F.Tasgetiren, Q.K.Pan and Y.Ch.Liang: A discrete differential evolution algorithm for the single machine total weighted tardiness problem with sequence dependent setup times. European Journal of Operational Research, Volume 36, pp. 1900-1915, 2009.

[7] R.Čapek, P.Šůcha and Z.Hanzálek: *Production Scheduling with Alternative Process Plans*. Under review after major revision in European Journal of Operational Research.

[8] H.Kellerer and V.A.Strusevich: *Scheduling problems for parallel dedicated machines under multiple resource constraints*. Discrete Applied Mathematics, Volume 133, pp. 45-68, 2003.

An AFPTAS for due date scheduling with related machines of general costs

Leah Epstein * Asaf Levin (Speaker) †

1 Introduction

We study the following problem. We are given an unlimited supply of machines of r different types. All machines are available for one time unit. A machine of type i has a speed b_i associated with it and its cost is c_i. We assume $1 = b_1 > b_2 > \cdots > b_r$. We normalize the costs so that $c_1 = 1$. A set of n jobs I is given, where job j has a processing time requirement (also called size) $0 < s_j \leq 1$. The goal is to assign the jobs to machines, so that the total cost is minimized. Note that it is allowed to use multiple instances of any of the machine types, but this counts towards the total cost.

Specifically, the goal is to partition the jobs into subsets $J_1, J_2, \ldots, J_k$, where each subset ℓ has a type associated with it, $t(\ell)$. The partition must satisfy $\sum_{j \in J_i} s_j \leq b_{t(\ell)}$, and the cost of J_ℓ is $c_{t(\ell)}$. That is, the total cost is $\sum_{\ell=1}^{k} c_{t(\ell)}$.

As our main result, we design an asymptotic fully polynomial time approximation scheme (AFPTAS) for the problem. Note that the number of machine types r is seen as a part of the input (it is not assumed to be a constant), and there are no assumptions on the relation between the costs c_i of the different machine types (e.g., we do not assume that the machine cost function is a concave function of the speed).

This scheduling problem can be presented as a bin packing problem with variable sized bins, and was previously studied as such [3], where an APTAS was designed for it. Related problems, where the cost of a machine is a concave function of the speed which it is forced to use were studied in [9, 10]. Variable sized bin packing is the special case where $b_i = c_i$ for all i, and an AFPTAS for this problem was given by Murgolo [11], generalizing the AFPTAS (and the APTAS) known for standard bin packing [2, 8].

The problem has applications in production due date scheduling, packing of goods for mixed truckload and less-than-truckload services, and in data storage on non-uniform disks [3, 9, 10]. In bin packing problems there is a common assumption that data storage devices have the property that the cost per unit of storage area is constant. It was argued in [3] that the following common assumption is not necessarily true. For small disks the cost is usually larger than half the cost of a disk which is twice as large. On the other hand, a large disks which uses a new technology can be much more expensive than twice the price of a disk which is half its size.

*lea@math.haifa.ac.il. Department of Mathematics, University of Haifa, 31905 Haifa, Israel.

†levinas@ie.technion.ac.il. Faculty of Industrial Engineering and Management, The Technion, 32000 Haifa, Israel.

2 The approach

The difficulties in solving the problem arise in several aspects. Since the number of machine types is non-constant, several tricky reductions are required in order to reduce the number of machine types. Still, the remaining machine types can be very different in terms of the ratio of cost to speed, also called density. While the sequence of costs can be assumed to be monotone, this is not the case with the sequence of densities. It is not difficult to see that even if the density of the machines type 1 is large, still some jobs cannot be assigned into slower machines due to their sizes. Once instances of this machine type are used, an algorithm is faced with the decision of which smaller jobs can use the resulting empty space. Moreover, in some cases a slower machine can be preferred to a slightly faster one, even if it has a larger density, since its cost is still smaller.

One natural attempt to solve the problem would be to apply a dynamic programming algorithm on the different machine types, starting with the slower ones, as in [7], where a PTAS is given for scheduling on uniformly related machines so as to minimize the makespan. The difference between the two problems is that in our case the number of machines is to be determined by the algorithm, but on the other hand, all jobs need to be completed by the deadline.

Another natural attempt would be to apply techniques coming from the bin packing problem. The approximation schemes of [2, 8] strongly rely on the fact that the cost of a bin is uniform, and in [11] the methods rely on the fact that the cost of a bin is proportional to its size. Note that it is fairly straightforward to extend the results of [11] for the case that the cost of a bin is a concave function of its size.

In order to solve the problem, we develop of a novel set of reductions and methods. In addition, we combine a number of methods, including those mentioned above, rounding [6, 7], shifting [4, 5] and the usage of *medium* sets of elements [1, 12], which is applied in our case not to jobs or machine speeds, but to machine densities.

In our reductions we partition the machine types into subsequences, and deal with each subsequence almost independently. The input jobs are partitioned into subsequences too, but it is unfortunately impossible to partition the input into completely independent parts, since it is necessary to allow the assignment of small jobs to very fast machines, as well as their assignment to much slower machines. We develop an AFPTAS for an auxiliary problem on (long but constant sized) subsequences of machine types. The solutions for the partial inputs are combined via dynamic programming, taking into account the remaining smaller jobs that still need to be packed.

3 Additional results

The problem which is studied in [9, 10] is defined as follows. An unlimited supply of machines, available for one time slot is given. The jobs are to be assigned to machines, so that the total size of jobs assigned to a machine does not exceed 1. A machine which received a total size of S of jobs can run in a speed of $\frac{1}{S}$. A concave function $f : (0, 1] \to (0, 1]$ is given, where $f(0) = 0$ and $f(1) = 1$. The cost of a machine which uses a speed s is $f(s)$. The goal is to assign all jobs to machines so as to minimize the cost. An APTAS was given in [9].

We show a reduction from a generalized version of this problem with an arbitrary

monotone non-decreasing non-negative function f, to the problem which we study, and thus we obtain an AFPTAS for this generalized version as well.

References

[1] N. Bansal, J. R. Correa, C. Kenyon, and M. Sviridenko. Bin packing in multiple dimensions: Inapproximability results and approximation schemes. *Mathematics of Operations Research*, 31:31–49, 2006.

[2] W. Fernandez de la Vega and G. S. Lueker. Bin packing can be solved within $1 + \varepsilon$ in linear time. *Combinatorica*, 1:349–355, 1981.

[3] L. Epstein and A. Levin. An APTAS for generalized cost variable-sized bin packing. *SIAM Journal on Computing*, 38(1):411–428, 2008.

[4] D. S. Hochbaum. Various notions of approximations: Good, better, best and more. In D. S. Hochbaum, editor, *Approximation algorithms*. PWS Publishing Company, 1997.

[5] D. S. Hochbaum and W. Maass. Approximation schemes for covering and packing problems in image processing and vlsi. *Journal of the ACM*, 32(1):130–136, 1985.

[6] D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: theoretical and practical results. *Journal of the ACM*, 34(1):144–162, 1987.

[7] D. S. Hochbaum and D. B. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. *SIAM Journal on Computing*, 17:539–551, 1988.

[8] N. Karmarkar and R. M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (FOCS'82)*, pages 312–320, 1982.

[9] J. Y.-T. Leung and C.-L. Li. An asymptotic approximation scheme for the concave cost bin packing problem. *European Journal of Operational Research*, 191(2):582–586, 2008.

[10] C.-L. Li and Z.-L. Chen. Bin-packing problem with concave costs of bin utilization. *Naval Research Logistics*, 53(4):298–308, 2006.

[11] F. D. Murgolo. An efficient approximation scheme for variable-sized bin packing. *SIAM Journal on Computing*, 16(1):149–161, 1987.

[12] H. Shachnai, T. Tamir, and G. J. Woeginger. Minimizing makespan and preemption costs on a system of uniform machines. *Algorithmica*, 42(3-4):309–334, 2005.

Faster approximation algorithms for scheduling with fixed jobs

Klaus Jansen (Speaker) [*] Lars Prädel [†] Ulrich M. Schwarz [‡]
Ola Svensson [§]

1 Introduction

In parallel machine scheduling, an important issue is the scenario where either some jobs are already fixed in the system [2, 11] or intervals of non-availability of some machines must be taken into account [1,2,6]. The first problem occurs when high-priority jobs are already scheduled in the system while the latter problem is due to regular maintenance of machines. Both models are relevant for turnaround scheduling [10] and distributed computing [3] where machines are donated on a volunteer basis.

Formally, the problem can be defined as follows: an instance consists of m machines and n jobs with non-negative processing times $p_1, \ldots, p_n \in \mathbb{N}$. The first k jobs are fixed via a list $(m_1, s_1), \ldots, (m_k, s_k)$ giving a machine index and starting time for the respective job. We assume that these fixed jobs do not overlap. A schedule is a non-preemptive assignment of the jobs to machines and starting times such that the first k jobs are assigned as encoded in the instance and that the jobs do not intersect.

For the problem with *fixed jobs*, the objective is to minimize the makespan of all jobs, including the fixed ones. In the setting with *non-availability*, the fixed jobs are not included when finding the makespan. Without loss of generality, we may assume $m < n$. Both problems generalize the well-known problem $P||C_{\max}$(scheduling jobs on parallel identical machines to minimize the makespan) [4] and hence are strongly NP-hard.

2 Related Work

Scheduling with fixed jobs was studied by Scharbrodt, Steger, and Weisser [11, 12]. They mainly studied the problem for constant m; for this strongly NP-hard formulation (which consequently does not admit an FPTAS) they presented a PTAS. They also found approximation algorithms for general m with ratios 3 [12] and $(2 + \epsilon)$ [11]; since the finishing time of the last fixed job is a lower bound for the optimal makespan $C^*_{\max}$, they can simply use a PTAS for the well-known problem $P||C_{\max}$ [5] to schedule the

[*] kj@informatik.uni-kiel.de Institut für Informatik, Christian-Albrechts-Universität zu Kiel, 24098 Kiel, Germany.
[†] lap@informatik.uni-kiel.de Institut für Informatik, Christian-Albrechts-Universität zu Kiel, 24098 Kiel, Germany.
[‡] ums@informatik.uni-kiel.de Institut für Informatik, Christian-Albrechts-Universität zu Kiel, 24098 Kiel, Germany.
[§] osven@kth.se KTH Royal Institute of Technology, SE-100 44 Stockholm, Sweden.

remaining $n-k$ jobs after the fixed job which finishes last. Scharbrodt, Steger, and Weisser [11] also proved that for scheduling with fixed jobs there is no approximation algorithm with ratio $(3/2-\epsilon)$, unless $\mathsf{P}=\mathsf{NP}$, for any $\epsilon \in (0, 1/2]$.

For the variant with non-availability, the scheduling problem is harder to approximate. First, there does not exist a polynomial time algorithm with a constant approximation ratio, unless $\mathsf{P}=\mathsf{NP}$, even if for each time step there is at least one available machine [1]. Furthermore, scheduling with non-availability does not admit a polynomial time approximation algorithm with ratio $(3/2-\epsilon)$ [2], unless $\mathsf{P}=\mathsf{NP}$, even if a constant fraction ρ of machines is always available. On the other hand, Diedrich and Jansen [2] presented a $(3/2+\epsilon)$-approximation for arbitrary $\epsilon > 0$ for both settings. However both methods rely on large enumeration steps and involves up to $m^{1/\epsilon^{1/\epsilon^2}}$ calls to a subroutine to approximately solve a difficult maximization subproblem, the Multiple Subset Sum Problem (MSSP) with accuracy ϵ (i.e. selecting a subset of items with total maximum size that can be packed into a set of bins without exceeding the capacities). We denote by $T_{MSSP}(n, \epsilon)$ the time complexity of this subroutine for n items. The best currently known algorithm for MSSP (and for the more general Multiple Knapsack Problem (MKP) with additional item profits) is an *efficient PTAS* due to Jansen [7] with a running time of $T_{MSSP}(n, \epsilon) = 2^{\mathcal{O}(1/\epsilon^5 \log(1/\epsilon))} + poly(n)$ for n items and at most n target bins, which was subsequently improved to $T_{MSSP}(n, \epsilon) = 2^{\mathcal{O}(1/\epsilon \log^4(1/\epsilon))} + poly(n)$ [8]. If the integrality gap for bin packing with different bin sizes is bounded by a constant (similar to the *Modified Integer Roundup Conjecture* (MIRUP) of Scheithauer and Terno [13] for the classical bin packing problem), the running time can be further reduced to $T_{MSSP}(n, \epsilon) = 2^{\mathcal{O}(1/\epsilon \log^2(1/\epsilon))} + poly(n)$ [8].

3 New Results

We present improved algorithms for scheduling with fixed jobs and scheduling with non-availability (see also [9]). These algorithms achieve exactly the bound of 3/2 and, are both faster and conceptually simpler than the previous algorithms in [2]. Formally stated, our results are the following:

Theorem 1. *Scheduling with fixed jobs admits an approximation algorithm with ratio* $3/2$ *and running time* $\mathcal{O}(n \log n + \log(n \max_{j=1,\dots,n} p_j)(n + T_{MSSP}(n, 1/8)))$.

Theorem 2. *Scheduling with non-availability, as long as a constant fraction* $\rho \geq 1/m$ *of machines is always available, admits a* $3/2$*-approximation with running time* $\mathcal{O}(n \log n + \log(n \max_{j=1,\dots,n} p_j)(n + T_{MSSP}(n, \rho/8)))$.

The underlying ideas are to guess via binary search the makespan T, to discard too-small values of T, to generate the gaps $g(T)$ for the non-fixed jobs and to use an approximation algorithm for the multiple subset sum problem as a subroutine to pack almost all non-fixed jobs into the gaps. The main difficulty now are the large jobs J_j with $p_j > T/2$, since an unpacked large job could cause an approximation ratio close to 2. But using an interesting exchange argument we are able to insert the non-packed large jobs into the schedule without increasing the total load of the non-packed jobs too much.

References

[1] F. Diedrich, K. Jansen, F. Pascual, and D. Trystram: Approximation algorithms for scheduling with reservations, Algorithmica 58 (2010), 391-404.

[2] F. Diedrich and K. Jansen: Improved approximation algorithms for scheduling with fixed jobs, Proceedings of Symposium on Discrete Algorithms, SODA 2009, 675-684.

[3] L. Eyraud-Dubois, G. Mounie, and D. Trystram: Analysis of Scheduling Algorithms with Reservations, Proceedings of International Parallel and Distributed Processing Symposium, IPDPS 2007, 1-8.

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

[5] D.S. Hochbaum and D.B. Shmoys: A polynomial approximation scheme for scheduling on uniform processors: using the dual approximation approach, SIAM Journal on Computing, 17 (1988), 539-551.

[6] H.-C. Hwang, K. Lee, and S.Y. Chang: The effect of machine availability on the worst-case performance of LPT, Discrete Applied Mathematics, 148 (2005), 49-61.

[7] K. Jansen: Parameterized approximation scheme for the multiple knapsack problem, SIAM Journal on Computing, 39 (2009), 1392-1412.

[8] K. Jansen: A fast approximation scheme for the multiple knapsack problem, Institut für Informatik der Christian-Albrechts-Universität zu Kiel, Tech. Report 0916, 2009.

[9] K. Jansen, L. Prädel, U.M. Schwarz, and O. Svensson: Faster approximation algorithms for scheduling with fixed jobs, Proceedings of Computing: The Australasian Theory Symposium, CATS 2011, 3-9.

[10] N. Megow, R.H. Möhring, and J. Schulz: Decision support and optimization in shutdown and turnaround scheduling, Informs Journal on Computing, to appear.

[11] M. Scharbrodt, A. Steger, and H. Weisser: Approximability of scheduling with fixed jobs, Proceedings of Symposium on Discrete Algorithms, SODA 1999, 961-962 and Journal of Scheduling 2 (1999), 267-284.

[12] M. Scharbrodt: Produktionsplanung in der Prozessindustrie: Modelle, effiziente Algorithmen und Umsetzung, Dissertation, Fakultät für Informatik, Technische Universität München, 2000.

[13] G. Scheithauer and J. Terno: The modified integer round-up property of the one-dimensional cutting stock problem, European Journal of Operational Research, 84 (1995), 562-571.

Logarithmic-approximations for the relocation problem

Alexander Grigoriev * Alexander Kononov (Speaker) †
Maxim Sviridenko ‡

1 Introduction

The relocation problem is a resource constraint scheduling problem typically rising in house redevelopment projects. For example, in XX Century a large-scale public housing project was initiated in East Boston Area [1,2] to rehabilitate a set of old buildings. Before demolishment of an old house, all its residents must be relocated to some temporary houses. Then, the old house is destroyed, a new house is built, and the residents are relocated again to their new accommodations. Given the capacities of the houses (old, new and temporary), the authority has to schedule house renovation activities such that at any moment in time all residents are (at least temporarily) housed. Here, the typical objective is to minimize the makespan, i.e., the project duration.

Formally, the problem can be described as follows. We are given a set of activities $J = \{1, 2, \ldots, n\}$ with processing requirement p_j for each activity $j \in J$. There is a single resource needed for processing of the activities. Let Ω_0 be the initial amount of the resource. For each activity $j \in J$, we are given the start time S_j. At time S_j activity j consumes α_j units of the resource. Clearly, this amount of the resource should be available at S_j, and it should be devoted exclusively for activity j. At its completion time $C_j = S_j + p_j$, activity $j \in J$ releases β_j units of the resource, and this amount becomes freely available for any other activities. No activity preemption is allowed. Given the resource constraint, the task is to find a schedule $\sigma = \{S_j : \ j \in J\}$ minimizing the makespan $C_{\max} = \max_{j \in J} C_j$. Notice, we have no limits on the amount of activities being processed at the same time. We also assume that all numbers in the input are non-negative integers.

Regarding the computational complexity, the relocation problem is hard to approximate within a factor better than $3/2$, unless $P = NP$, even if all activities have unit processing times; see [3]. Kononov and Lin in [3] discuss approximability of the more restricted versions of the problem. There, the crucial additional assumption is that the processing times are the same for all jobs. In this setting, polynomial-time constant-approximation algorithms are developed. In contrast, in this paper we develop the first polynomial-time $O(\log n)$-approximation algorithm for the relocation problem with arbitrary processing times.

*a.grigoriev@maastrichtuniversity.nl. Department of Quantitative Economics, Maastricht University SBE, P.O.Box 616, 6200MD Maastricht, The Netherlands.

†alvenko@math.nsc.ru. Sobolev Institute of Mathematics, Akademik Koptyug pr. 4, 630090 Novosibirsk, Russia.

‡sviri@us.ibm.com. IBM T.J. Watson Research Center, P.O. Box 218, Yorktown Heigths, NY 10598, USA.

2 Approximation algorithms with performance guarantees $O(\log p_{\max})$ and $O(\log n)$

Notice, without loss of generality we may assume $\alpha_j \leq \beta_j$ for all $j \in J$, for otherwise we make use of the schedule mirroring argument as in [3]. Moreover, let the activities be arranged in non-decreasing order of the release-to-consumption ratios, i.e., $\beta_1/\alpha_1 \leq \beta_2/\alpha_2 \leq \ldots \leq \beta_n/\alpha_n$. Here, if $\alpha_j = 0$ for some $j \in J$, we naturally assume that the ratio is $+\infty$.

Now, we are ready to present the ideas for the approximation algorithms. We start with a polynomial-time $O(\log p_{\max})$-approximation algorithm, where $p_{\max} = \max_{j \in J} p_j$.

1. We slightly modify the instance rounding up the processing times of the activities to the nearest power-of-two numbers. Then, let the set of activities having processing time 2^i, $i \in \{0, 1, \ldots, \lceil \log_2 p_{\max} \rceil\}$, be denoted by J_i.

2. At each step of the algorithm we select several activities from a single set J_i, $i \in \{0, 1, \ldots, \lceil \log_2 p_{\max} \rceil\}$, such that the aggregated resource requirement of selected activities does not exceed the available resource. Then, selected activities are started and completed simultaneously. Let the completion time of the activities scheduled in step $k \geq 1$ of the algorithm be denoted by t_k. Finally, we compute the amount Ω_k of the resource available at time t_k.

   If activities of set J_i are scheduled at step k, for simplicity, we say: *set J_i is scheduled at step k*. Further we specify the choice of a set to be scheduled at step $k \geq 1$ of the algorithm, and the choice of activities to be processed at step k.

   At step $k = 1$ we consider activities from set J_0. We pick an activity $j \in J_0$ that has the lowest index (equivalently, the highest value of the release-to-consumption ratio). If the amount of available resource is at least α_j, we start processing j at time 0 and we decrease the amount of available resource by α_j. Otherwise, we disregard j and recurse on the (not-disregarded) rest of set J_0. Since all activities from set J_0 have unit processing times, the completion time t_1 of the scheduled activities is 1. Let Ω_1 be the amount of resource becoming available at time $t_1 = 1$.

   Now, consider step $k \geq 2$ of the algorithm. First, we specify which set should be scheduled at step k. This is done according to the following rule: for all $0 \leq i \leq \lceil \log_2 p_{\max} \rceil - 1$, set J_i must be scheduled exactly four times between any two consecutive scheduling of set J_{i+1} (for completeness we assume that prior to time 0 all sets are scheduled). Given the initial set J_0, this rule uniquely defines the sequence of scheduled sets. For instance, if there are only two sets, J_0 and J_1, the sequence is formed by repetition of subsequence $S_2 = [J_0, J_0, J_0, J_0, J_1]$. If the number of sets is 3, the sequence is generated by repetition of $S_3 = [S_2, S_2, S_2, S_2, J_2]$. Recursively, given subsequence S_ℓ determined for ℓ sets, the sequence for $\ell + 1$ sets is formed by repetition of $S_{\ell+1} = [S_\ell, S_\ell, S_\ell, S_\ell, J_\ell]$.

   Given a set to be scheduled at step $k \geq 2$, similarly to step 1, we select the activities from the set according to non-decreasing order of the release-to-consumption ratios such that the total amount of consumed resources is up to Ω_{k-1}. We start processing the selected activities at time t_{k-1}.

3. We stop the algorithm when all activities are scheduled.

Clearly, the algorithm runs in time $O(n \log n)$. To provide an intuition for the correctness of the claimed performance guarantee, we have the following propositions.

Proposition 1. *When rounding up the processing times of the activities to the nearest power-of-two numbers, we increase the makespan by at most factor of* 2.

Further we assume that the processing times of all activities are power-of-two numbers.

Proposition 2. *There exists a nested (laminar) schedule of the activities such that its makespan is at most a constant times the optimal makespan.*

Proposition 3. *The makespan of the schedule returned by the algorithm is at most* $O(\log p_{\max})$ *times the makespan of an optimal nested schedule.*

Intuitively, the last proposition follows from the following facts. First, any nested schedule can be *stretched*, i.e., rescheduled in such a way that at any moment in time only activities with the same processing requirements are executed, and the precedence order of the activities is preserved, i.e., if one activity starts after completion of another in the nested schedule, the same order takes place in the stretched schedule. One can show that *stretching* affects the makespan by at most factor of K, where K is the number of distinct processing times. Second, the algorithm above represents one such stretching. Third, the sufficiency of the resources is guaranteed by non-decreasing order of the release-to-consumption ratios. Therefore, the algorithm outputs a feasible schedule with makespan at most $O(\log p_{\max})$ times the optimum.

Finally, using standard scaling techniques we turn the $O(\log p_{\max})$- into $O(\log n)$-approximation: at small additional cost to the makespan, we transform the processing times of the activities to numbers polynomial in n. This brings us to the main theorem:

Theorem 4. *The relocation problem admits* $O(\log p_{\max})$- *and* $O(\log n)$-*approximation algorithms, both running in* $O(n \log n)$-*time.*

References

[1] E.H. Kaplan (1986). *Relocation models for public housing redevelopment programs.* Planning and Design 13: 5-Ű19.

[2] E.H. Kaplan and O. Berman (1988). *Orient heights housing projects.* Interfaces 18: 14-Ű22.

[3] A. Kononov, B.M.-T. Lin (2006). *Relocation Problems with Multiple Working Crews.* Discrete Optimization 3: 366–381.

Approximating the Joint Replenishment Problem

Neil Dobbs Tomasz Nowicki Maxim Sviridenko (Speaker)
Grzegorz Swirszcz*

1 Introduction

The Joint Replenishment Problem is one of the fundamental problems in Inventory and Supply Chain Management. We are given a warehouse and the set of retailers $\{1, \ldots, n\} = [n]$. We are given the discrete time horizon $\{1, \ldots, T\} = [T]$ and the set of demands $\mathcal{D}$ consisting of pairs (i, t) for retailer of type $i \in [n]$ and time $t \in [T]$. Let d_{it} be the amount of demand for retailer i that arrived at time t. The demand (i, t) must be satisfied by an order placed in the time interval $[0, t]$. If the demand for retailer i at time t is satisfied by an order placed at time $s \leq t$ then it incurs per unit holding costs of h_{ist}. A retailer i can place an order at warehouse at any time τ and incur the retailer ordering cost K_i, at the same time the warehouse places an order and incurs the warehouse ordering cost of K_0. Therefore, it makes sense to combine many retailer orders together. For a example if at time τ we place orders for retailers from the set $S \subseteq [n]$ then we incur ordering cost of $K_0 + \sum_{i \in S} K_i$. The goal is to define the set of warehouse and retailer orders to satisfy all the demand and minimize the total ordering and holding costs.

Literature Overview. The Joint Replenishment Problem (JRP) and related One Warehouse Multiple Retailer Problem (OWMR) are well known abstract models of the practical inventory management. It is usually assumed that the holding costs are linear, i.e. $h_{ist} = (t - s)\Delta^i$ but in this work we will work with more general holding costs first introduced by Levi et al. [3] that allow us to model various types of perishable goods. It is well-known that JRP is NP-hard, Arkin et al. [2]. Moreover, Souza and Nonner [5] showed that the general model is APX-hard, i.e. there exists a constant $c > 1$ (it could be very close to one) such that there is no c-approximation algorithm for JRP with general holding costs unless $P = NP$.

A polynomial time algorithm is called c-approximation algorithm if it always finds a solution of value at most c times the optimal value. Levi et al. [4] designed a linear programming based 1.8-approximation algorithm even in a more general setting of OWMR problem. Nonner and Souza [5] suggested a different type of randomized rounding for the same linear programming relaxation. They proved that their algorithm has performance guarantee of 5/3 for the special case of the holding costs $h_{ist} \in \{0, +\infty\}$, i.e. every demand must be served in its corresponding time window.

The literature on various heuristics, enumerative approaches and generalizations of the JRP is quite extensive we refer an interested reader to the recent survey [1] on the topic.

*IBM Watson Research Center, Math. Sciences, PO Box 218, Yorktown Heights, NY 10598, USA

Our results. In this paper we consider the special case when $h_{ist} = 0$ for all $s \in [r_{it}, t]$ and $h_{ist} = +\infty$ otherwise. This special case models the situation when we have perishable goods with negligible holding costs within expiration time window. We consider the randomized rounding algorithm suggested by Nonner and Souza [5] and show that the performance guarantee of that algorithm can be improved to approximately 1.574 if we use a different probability measure for random variables in the algorithm. We also conjecture that our probability measure is optimal for this class of algorithms. Unfortunately, it can be shown that our probability measure does not generate an algorithm with good performance guarantee in the case of general holding costs but we strongly believe that this type of algorithms works well even for general holding costs. The main challenge and an open problem is to find a good probability measure for the general case.

2 Linear Programming Formulation and Randomized Rounding Algorithm

We will use the notation $H_{ist} = d_{it}h_{ist}$ for the cumulative holding cost. We consider the linear programming relaxation of the JRP.

$$\min \quad \sum_{s=1}^{T} K_0 x_{0s} + \sum_{i=1}^{n}\sum_{s=1}^{T} K_i x_{is} + \sum_{i=1}^{n}\sum_{s=1}^{T}\sum_{t=1}^{T} H_{ist} y_{ist}, \tag{1}$$

$$\sum_{s=1}^{t} y_{ist} = 1, \qquad (i,t) \in \mathcal{D}, \tag{2}$$

$$y_{ist} \le x_{is}, \qquad (i,t) \in \mathcal{D}, i \in [n], s \in [T], \tag{3}$$

$$x_{is} \le x_{0s} \qquad i \in [n], s \in [T], \tag{4}$$

$$x_{0s}, x_{is}, y_{ist} \ge 0, \qquad i \in [n], s, t \in [T]. \tag{5}$$

The objective function (1) minimizes the total order opening and holding costs. The constraint (2) requires every demand pair (i,t) to be served by some order. The constraint (3) allows demand to be served only by an opened order of the same retailer. Finally, the constraint (4) requires to open a warehouse order at any time with opened retailer order.

The standard assumption on holding costs is that H_{ist} are non-increasing in s (see [3]). This property basically implies that any integral or fractional solution x_{it} completely defines variables y_{ist}. Each demand (i,t) is fractionally served by the closest to it fractionally opened retailer i orders.

Let (x^*, y^*) be an optimal solution for the linear programming relaxation (1)-(5) (we assume polynomial running time of our LP solver). For each retailer $i \in [n]$ define a piecewise linear increasing continuous function

$$\omega_i : \left[0, \sum_{\tau=1}^{T} x^*_{0\tau}\right] \to \left[0, \sum_{\tau=1}^{T} x^*_{i\tau}\right]$$

such that $\omega_i(0) = 0$ and $\omega_i(\sum_{\tau=1}^{t} x^*_{0\tau}) = \sum_{\tau=1}^{t} x^*_{i\tau}$ for each $t \in [T]$. The inequalities (4) and the definition of ω_i imply the simple property that $\omega_i(z) - \omega_i(z') \le z - z'$ for any

$0 \le z' \le z \le \sum_{\tau=1}^{T} x^*_{0\tau}$ (basically all linear functions comprising $\omega_i(x)$ have derivatives ≤ 1), i.e. $\omega_i(x)$ are contracting (or 1-Lipshitz) functions.

Consider the following rounding algorithm that finds an integral solution for our optimization problem. The input to the algorithm is the fractional solution (x^*, y^*) and a probability distribution defined on the interval $[0, 1]$ defined by the probability measure $\mu(x)$.

1. Define intervals $I_t = (\sum_{\tau=1}^{t-1} x^*_{0\tau}, \sum_{\tau=1}^{t} x^*_{0\tau}]$ for time $t \in [T]$ and intervals $I_{it} = (\sum_{\tau=1}^{t-1} x^*_{i\tau}, \sum_{\tau=1}^{t} x^*_{i\tau}]$ for retailer $i \in [n]$ and time $t \in [T]$.

2. Let $D_0 = 0$, $\Lambda = \emptyset$. Consecutively draw random variable d_i from the interval $[0, 1]$ according to the probability measure $\mu(x)$. Define $D_i = D_{i-1} + d_i$. If $D_i > \sum_{\tau=1}^{T} x^*_{0\tau}$ then stop the process and go to the step 3. Otherwise, $\Lambda = \Lambda \cup \{D_i\}$, we set $i = i + 1$ and repeat.

3. For each retailer $i \in [n]$, independently apply the following process. Initialize $y = 0$ and $\Lambda_i = \emptyset$. Define time $y' = \omega_i^{-1}(\omega_i(y) + 1))$ (note that $y' \ge y + 1$ since all ω_i are contracting functions). Let $z' = \max\{z \in \Lambda : z \le y'\}$ and $\Lambda_i = \Lambda_i \cup \{\omega_i(z')\}$. After that we set $y = z'$ and repeat the process again until $\omega_i(y) + 1 > \sum_{\tau=1}^{T} x^*_{i\tau}$.

4. Open retailer i order at the times $t \in T$ such that there exists $z \in \Lambda_i$ such that $z \in I_{it}$. Open warehouse order at the times $t \in T$ such that there exists $D \in \Lambda$ such that $D \in I_{0t}$.

It remains to define the probability measure $\mu(x)$ to completely describe the approximation algorithm. Fix $\theta \approx 0.36455$. We define $\mu[0, \theta) = 0$ and

- $\mu[0, x) = \log(x/\theta)$ for $x \in [\theta, 2\theta)$;
- $\mu[0, x) = \log(x/\theta) - \int_{[2\theta, x]} \frac{\log((x-\theta)/\theta)}{x} dx$ for $x \in [2\theta, 1)$;
- finally, $\mu[1] := 1 - \mu[0, 1)$.

References

[1] Y. Aksoy and S. Erenguc, Multi-item inventory models with co-ordinated replenishments: A survey. International Journal of Operations Production Management, 8(1):63Ű73, 1988.

[2] E. Arkin, D. Joneja, and R. Roundy. Computational complexity of uncapacitated multi-echelon production planning problems. *Operations Research Letters*, 8:61–66, 1989.

[3] R. Levi, R. O. Roundy, and D. B. Shmoys. Primal-dual algorithms for deterministic inventory problems. Math. Oper. Res. 31 (2006), no. 2, 267Ű284.

[4] R. Levi, R. Roundy, D. Shmoys and M. Sviridenko, A Constant Factor Approximation Algorithm for the One-Warehouse Multi-Retailer Problem, Management Science v. 54 (2008), pp. 763-776.

[5] T. Nonner and A. Souza, A 5/3-Approximation Algorithm for Joint Replenishment with Deadlines, Discrete Mathematics, Algorithms and Applications (DMAA), V. 1(2) 2009.

Quantifying the Sub-optimality of Uniprocessor Fixed Priority Scheduling

Robert I. Davis (Speaker) *

1 Introduction

This presentation examines the relative effectiveness of fixed priority pre-emptive scheduling in a uniprocessor system, compared to an optimal algorithm such as Earliest Deadline First (EDF). The quantitative metric used in this comparison is the processor speedup factor, equivalent to the factor by which processor speed needs to increase to ensure that any taskset that is schedulable according to an optimal scheduling algorithm can be scheduled using fixed priority pre-emptive scheduling, assuming an optimal priority assignment policy. Three types of taskset are considered, with implicit, constrained, and arbitrary deadlines. For constrained-deadline tasksets where all task deadlines are less than or equal to their periods, the maximum value for the processor speedup factor is shown to be $1/\Omega \approx 1.76322$, (where Ω is the mathematical constant defined by the transcendental equation $\ln(1/\Omega) = \Omega$). The presentation also covers speedup factors for fixed priority non-preemptive scheduling.

The research that will be presented has been previously published in [3], [4], and [5]. It examines the relative effectiveness of fixed priority (FP) scheduling of sporadic tasks in a uniprocessor system, compared to an optimal algorithm, such as Earliest Deadline First (EDF). The quantitative metric used in this comparison is the processor speedup factor [6], defined as the factor by which processor speed needs to increase to ensure that any sporadic taskset that is schedulable according to EDF can be scheduled using FP scheduling. Two classes of scheduling algorithm are considered:

1. *Pre-emptive.*
2. *Non-pre-emptive.*

Further, three categories of sporadic taskset are considered, based on the relationship between the period and deadline of each task.

1. *Implicit-deadline*: All tasks have deadlines equal to their periods.
2. *Constrained-deadline*: All tasks have deadlines less than or equal to their periods.
3. *Arbitrary-deadline*: Task deadlines may be less than, equal to, or greater than their periods.

2 Tasking Model

The research presented considers the scheduling of a set of sporadic tasks (or *taskset*) on a uniprocessor. Each taskset comprises a static set of n tasks ($\tau_1 \ldots \tau_n$), where n

*rob.davis@cs.york.ac.uk. Real-Time Systems Research Group, Department of Computer Science,University of York, York, UK.

is a positive integer. Each task τ_i is characterised by its bounded worst-case execution time C_i, minimum inter-arrival time or period T_i, and relative deadline D_i. Each task τ_i therefore gives rise to a potentially infinite sequence of invocations (or jobs), each of which has an execution time upper bounded by C_i an arrival time at least T_i after the arrival of its previous invocation, and an absolute deadline D_i time units after its arrival. The *utilisation* U_i of a task is given by its execution time divided by its period ($U_i = C_i/T_i$). The total utilisation U, of a taskset is the sum of the utilisations of all of its tasks.

The following assumptions are made about the behaviour of the tasks: The arrival times of the tasks are independent and unknown a priori, hence the tasks may share a common release time. The tasks are independent and so cannot block each other from executing by accessing mutually exclusive shared resources, with the exception of the processor in the case of non-preemptive scheduling. The tasks do not voluntarily suspend themselves.

A taskset is said to be *schedulable* with respect to some scheduling algorithm, if all valid sequences of jobs that may be generated by the taskset can be scheduled by the scheduling algorithm without any deadlines being missed.

3 Sub-optimality and speedup factors

The seminal work of Liu and Layland [7] characterises the maximum performance penalty incurred when an implicit-deadline taskset is scheduled using Rate-Monotonic, FP-P scheduling instead of an optimal algorithm such as EDF-P. Combining the utilisation bounds of Liu and Layland [7] for EDF-P and FP-P, shows that the processor speedup factor required to guarantee that FP-P scheduling can schedule any implicit-deadline taskset schedulable by EDF-P is $1/\ln(2) \approx 1.44270$.

The research that will be presented provides an analogous characterisation of the maximum performance penalty incurred when constrained-deadline tasksets are scheduled using Deadline-Monotonic, FP-P scheduling instead of an optimal algorithm such as EDF. Here, the speedup factor is equal to $1/\Omega \approx 1.76322$ (where Ω is the mathematical constant defined by the transcendental equation $\ln(1/\Omega) = \Omega$, hence, $\Omega \approx 0.567143$), see Davis et al. [3].

The presentation will also summarise research giving lower and upper bounds on the speedup factors for:

1. The pre-emptive case with arbitrary deadlines (Davis et al. [4]).
2. The non-pre-emptive case (comparing FP-NP and EDF-NP) for tasksets with implicit, constrained, and arbitrary deadlines (Davis et al. [5]).

These results are summarised in the table below.

	Pre-emptive		Non-Pre-emptive	
Task deadlines	Lower bound	Upper bound	Lower bound	Upper bound
Implicit	$1/\ln(2) \approx 1.44269$		$1/\Omega \approx 1.76322$	2
Constrained	$1/\Omega \approx 1.76322$		$1/\Omega \approx 1.76322$	2
Arbitrary	$1/\Omega \approx 1.76322$	2	$1/\Omega \approx 1.76322$	2

We note that the two cases where a tight bound is known correspond to the only cases where optimal priority assignment can be achieved independently of schedulability

testing. In the arbitrary deadline case for FP-P scheduling and all cases of FP-NP scheduling, Audsley's Optimal Priority Assignment algorithm [1], [2] is required to find the optimal priority ordering. This dependence of priority ordering on schedulability testing makes it more difficult to reason about the properties of a theoretical speedup-optimal taskset that requires the exact speedup factor to be schedulable. In these cases, the exact sub-optimality of fixed priority scheduling remains an open question.

4 Acknowledgements

This work was funded in part by the EPSRC project TEMPO (EP/G055548/1) and the EU funded ArtistDesign Network of Excellence.

References

[1] N.C. Audsley (1991). *Optimal priority assignment and feasibility of static priority tasks with arbitrary start times.* Technical Report YCS 164, Dept. Computer Science, University of York, UK.

[2] N.C. Audsley (2001). *On priority assignment in fixed priority scheduling.* Information Processing Letters, 79(1) pages 39-44.

[3] R.I. Davis, T. Rothvoss, S.K. Baruah, A. Burns (2009). *Quantification of the Sub-optimality of Uniprocessor Fixed Priority Pre-emptive Scheduling.* Real-Time Systems. Vol. 43, No. 3, pages 211-258.

[4] R.I. Davis, T. Rothvoss, S.K. Baruah, A. Burns (2009). *Quantifying the Sub-optimality of Uniprocessor Fixed Priority Pre-emptive Scheduling for Sporadic Tasksets with Arbitrary Deadlines.* In proceedings of Real-Time and Network Systems (RTNS'09), pages 23-31.

[5] R.I. Davis, L. George, P. Courbin (2010). *Quantifying the Suboptimality of Uniprocessor Fixed Priority Non-Pre-emptive Scheduling.* In proceedings of Real-Time and Network Systems (RTNS'10), pages 1-10.

[6] B. Kalyanasundaram, K. Pruhs (1995). *Speed is as powerful as clairvoyance.* In Proceedings of the 36th Symposium on Foundations of Computer Science, pages 214-221.

[7] C.L. Liu, J.W. Layland (1973). *Scheduling algorithms for multiprogramming in a hard-real-time environment.* Journal of the ACM, 20(1) pages 46-61.

Intra-Type Migrative Scheduling of Implicit-Deadline Sporadic Tasks on Two-Type Heterogeneous Multiprocessor

Gurulingesh Raravi (Speaker) * Björn Andersson†*
Konstantinos Bletsas *

1 Introduction

Consider the problem of scheduling a set of implicit-deadline sporadic tasks to meet all deadlines on a two-type heterogeneous multiprocessor platform. Each processor is either of type-1 or type-2 with each task having different execution time on each processor type. Jobs can migrate between processors of same type (referred to as *intra-type migration*) but cannot migrate between processors of different types. We present a new scheduling algorithm namely, LP-Relax(THR) which offers a guarantee that if a task set can be scheduled to meet deadlines by an optimal task assignment scheme that allows intra-type migration then LP-Relax(THR) meets deadlines as well with intra-type migration if given processors $\frac{1}{THR}$ as fast (referred to as *speed competitive ratio*) where $THR \leq \frac{2}{3}$.

2 LP-Relax Algorithm

The system considered in this paper is as follows: (i) **Tasks (denoted as τ):** n implicit-deadline sporadic tasks, i.e., for each task, its deadline = its minimum inter-arrival time (denoted as T_i) (ii) **Utilization:** The utilization of a task $\tau_i \in \tau$ on a processor of type-t (where $t \in \{1, 2\}$) is given by U_i^t where $U_i^t = \frac{C_i^t}{T_i}$ (C_i^t denotes the worst-case execution time of a task τ_i on a processor of type-t) and (iii) **Processors:** The platform consists of m processors of which mP^1 processors are of type-1 and mP^2 processors are of type-2. The following assumptions are made: (i) **Intra-Type Migrative:** The jobs released by tasks can only migrate between processors of the same type but not between processors of different types, (ii) **No job parallelism:** A job can be executing on at most one processor at any given point in time and (iii) **Independent tasks:** The execution of jobs are independent, i.e., they neither share any resources nor have data dependency.

Let THR denote a real number number in the range $(0, 2/3]$, selectable by the algorithm designer. Based on THR, let us define the following disjoint sets:

$$H12 = \{\tau_i \in \tau : U_i^1 > THR \quad \wedge \quad U_i^2 > THR\} \tag{1}$$

$$H1 = \{\tau_i \in \tau : U_i^1 \leq THR \quad \wedge \quad U_i^2 > THR\} \tag{2}$$

$$H2 = \{\tau_i \in \tau : U_i^1 > THR \quad \wedge \quad U_i^2 \leq THR\} \tag{3}$$

$$L = \{\tau_i \in \tau : U_i^1 \leq THR \quad \wedge \quad U_i^2 \leq THR\} \tag{4}$$

*{ghri, baa, ksbs}@isep.ipp.pt. CISTER-ISEP Research Center, Polytechnic Institute of Porto, Rua Dr. António Bernardino de Almeida 431, 4200-072 Porto, Portugal.

†baandersson@sei.cmu.edu. Software Engineering Institute, Carnegie Mellon University, 4500 Fifth Avenue Pittsburgh, PA 15213-2612, USA.

```
function: LP-Relax(THR)
1  Form sets H12, H1, H2, L as defined by Eq. 1-4
2  if (H12 ≠ φ) then declare FAILURE end if
3  UP^1 := UP^2 := 0 //used capacities of two processor types
4  τp^1 := assign(H1, UP^1, 1)
5  if (τp^1 ≠ H1) then declare FAILURE end if
6  τp^2 := assign(H2, UP^2, 2)
7  if (τp^2 ≠ H2) then declare FAILURE end if
8  LPR := formulate_LP(L, UP^1, UP^2, mP^1, mP^2)
9  (Z, X) = LP-Solver(LPR) //Solve LPR using an LP-Solver
10 if (Z > 1) then declare FAILURE
11 for each task τ_i ∈ L do   //x_{i,1}, x_{i,2} ∈ X
12   if (x_{i,1} = 1 ∧ x_{i,2} = 0) then
13     τp^1 := τp^1 ∪ {τ_i} //assign τ_i to processor type-1
14     UP^1 := UP^1 + U_i^1
15   end if
16   if (x_{i,1} = 0 ∧ x_{i,2} = 1) then
17     τp^2 := τp^2 ∪ {τ_i} //assign τ_i to processor type-2
18     UP^2 := UP^2 + U_i^2
19   end if
20 end for
21 if (∃i such that 0 < x_{i,1} < 1 and 0 < x_{i,2} < 1) then
22   Let τ_f denote that task and t1 denote its favorite
     and t2 its non-favorite type, i.e., U_i^{t1} ≤ U_i^{t2}
     (where t1 = 1, t2 = 2 or t1 = 2, t2 = 1)
23   UP^{t1} := UP^{t1} + x_{f,t1} · U_f^{t1}
24   UP^{t2} := UP^{t2} + x_{f,t2} · U_f^{t2}
25   if (UP^{t1} − x_{f,t1} · U_f^{t1} + U_f^{t1} ≤ mP^{t1}) then
26     UP^{t1} := (UP^{t1} − x_{f,t1} · U_f^{t1} + U_f^{t1})
27     τp^{t1} := τp^{t1} ∪ {τ_f}
28     declare SUCCESS
29   end if
30   if (UP^{t2} − x_{f,t2} · U_f^{t2} + U_f^{t2} ≤ mP^{t2}) then
31     UP^{t2} := (UP^{t2} − x_{f,t2} · U_f^{t2} + U_f^{t2})
32     τp^{t2} := τp^{t2} ∪ {τ_f}
33     declare SUCCESS
34   end if
35   declare FAILURE
36 else
37   declare SUCCESS
38 end if
```

Figure 1: The LP-Relax(THR) algorithm for assigning tasks on to a two-type heterogeneous multiprocessor platform.

```
function assign(ts: set of tasks; U: out current utilization
                of processors; k: favorite processor type of ts)
                return set of tasks
1  assigned_tasks := ∅
2  Use any order for tasks ts, but maintain it
   during the execution of the function assign.
3  τ_i := first task in ts
4  if (U + U_i^k ≤ mP^k) then
5    U := U + U_i^k
6    assigned_tasks := assigned_tasks ∪ {τ_i}
7    if (remaining tasks exist in ts) then
8      τ_i := next task in ts
9      go to line 4.
10   end if
11 end if
12 return assigned_tasks
```

(a) Assigning heavy tasks to type-k processors

```
function formulate_LP(L, UP^1, UP^2)
        return the formulation
1  Let myLP denote the following linear program
2  Minimize Z subject to:
3     ∀τ_i ∈ L : x_{i,1} + x_{i,2} = 1
4     (Σ_{τ_i∈L} x_{i,1} · U_i^1) + UP^1 ≤ mP^1 · Z
5     (Σ_{τ_i∈L} x_{i,2} · U_i^2) + UP^2 ≤ mP^2 · Z
6     ∀τ_i ∈ L : x_{i,1} and x_{i,2} are non-negative real
      numbers
7  return myLP
```

(b) LP formulation for assigning light tasks to processors

Figure 2: Sub-routines used by LP-Relax(THR) algorithm while assigning the task set.

Note that $H12 \cup H1 \cup H2 \cup L = \tau$.

A task is termed *heavy* on a processor type if its utilization on that processor type is greater than THR. The set $H12$ represents those tasks that are heavy on both types of processors and hence these tasks cannot be assigned to any processor type to meet deadlines if the processor speed would be scaled by THR. $H1$ and $H2$ represent those tasks that are heavy on processors of type-2 and type-1 respectively and hence they must be assigned to processors of type-1 and type-2 respectively. L represents those tasks that are not heavy (termed *light*) on both the processor types and hence they can be assigned to processors of any type.

The new algorithm is shown in Figure 1 which uses sub-routines *assign()* and *formulate_LP()* shown in Figure 2(a) and Figure 2(b) respectively.

The algorithm first assigns tasks that are heavy on a certain processor type say, type-1 to processors of type-2 and vice versa (if the schedulability test permits). We can formulate the problem of assigning *light* tasks to processors as an Integer Linear Program (ILP) and then relax it to Linear Program (LP). For assigning light tasks, LP-Relax solves this relaxed LP formulation using an LP-solver which gives the optimal solution at the *vertex* of the *convex* region. This vertex solution gives the assignment (if the schedulability test permits) of all the light tasks to unique processor type except at most

one task say, τ_{fract}, which may be assigned to both the processor types – see Lemma 6 in [5]. Finally, if τ_{fract} was assigned to two processor types then LP-Relax algorithm (re-)assigns τ_{fract} to a single processor type (if the schedulability test permits). Once the task assignment is done, an optimal scheduling algorithm for identical multiprocessors, such as Pfair [1] can be used on each processor type to schedule the tasks.

We now list some important properties about the performance guarantee offered by LP-Relax algorithm. The proofs of these properties are omitted here due to space constraint and can be found in Section 4 in [5].

Theorem 1. *The speed competitive ratio of LP-Relax(2/3) is at most 1.5.*

The following theorem states the speed competitive ratio of LP-Relax algorithm when the maximum utilization of any task on any type of processor (type-1 or type-2) is upper bounded by α, i.e., $\forall \tau_i \in \tau : U_i^1 \leq \alpha \wedge U_i^2 \leq \alpha$ where $0 < \alpha \leq \frac{2}{3}$.

Theorem 2. *The speed competitive ratio of LP-Relax($\frac{m-\alpha}{m}$) is at most ($\frac{m}{m-\alpha}$).*

Corollary 3. *If a task set τ is schedulable by any algorithm that allows intra-type migration on a computing platform Π of m processors (having mP^1 processors of type-1 and mP^2 processors of type-2) then LP-Relax($\frac{2}{3}$) also schedules τ on a computing platform Π' of at most $m+1$ processors (having either mP^1+1 processors of type-1 and mP^2 processors of type-2 or mP^1 processors of type-1 and mP^2+1 processors of type-2).*

Corollary 4. *If migration of tasks is allowed between processors of different types (i.e., fractional assignment of tasks is allowed – for a task τ_i, $0 < x_{i,1} < 1$ and $0 < x_{i,2} < 1$) then LP-Relax (with THR=1) is optimal.*

Note: Though Corollary 4 is an important result, we would like to mention that it is not the first of its kind. A feasibility condition exists [3] for checking the schedulability of a task set on a heterogeneous multiprocessor platform (having more than two types of processors) that allows task migration between processors of any type.

Acknowledgments

This work was partially funded by the Portuguese Science and Technology Foundation (Fundação para a Ciência e a Tecnologia - FCT), REHEAT and REJOIN projects.

References

[1] J. Anderson and A. Srinivasan, *Pfair Scheduling: Beyond Periodic Task Systems*, 7th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (2000).

[2] S. Baruah, *Task partitioning upon heterogeneous multiprocessor platforms*, 10th IEEE International Real-Time and Embedded Technology and Applications Symposium (2004).

[3] S. Baruah, *Feasibility Analysis of Preemptive Real-Time Systems upon Heterogeneous Multiprocessor Platforms*, 25th IEEE International Real-Time Systems Symposium (2004).

[4] C. N. Potts, *Analysis of a linear programming heuristic for scheduling unrelated parallel machines*, Discrete Applied Mathematics (1985).

[5] G. Raravi, B. Andersson and K. Bletsas, *Scheduling Implicit-Deadline Sporadic Tasks on Two-Type Heterogeneous Multiprocessor Platform with Restricted Migration*, Tech. Report, CISTER/ISEP, Polytechnic Institute of Porto, HURRAY-TR-110109, (2011).

Production Optimization and Scheduling in a Steel Plant: Hot Rolling Mill

Matteo Biondi * Iiro Harjunkoski † Sleman Saliba (Speaker) ‡

1 Introduction

Production scheduling in the steel industry has been recognized as one of the most difficult industrial scheduling problems. Many different and often contradicting constraints must be taken into account while defining a feasible and, possibly, optimal schedule for the production.

In one of the most typical production configuration, the steel-making process can be roughly subdivided into three parts: the melt shop, where melt steel is produced and cast into semi-finished products (slabs), see e.g. Harjunkoski & Grossmann [1]; the hot rolling, in which slabs are transformed by means of a mechanical and thermal process in the final product (coils, billets, wires,Ě); cold rolling and finishing line operations can achieve customers' specifications for final dimensions, surface quality and mechanical properties. These production steps are highly interconnected; the ideal situation would be to comprise all of them into one optimization model. In our present work, we will focus on the problem of production scheduling on the hot rolling mill.

2 Hot Rolling

The hot strip mill typically consists of several processing stages (reheating furnace, roughing mill, finishing mill, down coiler), on which the slabs need to be processed sequentially in order to be rolled to coils. Strict production rules determine the sequence of the slabs on the hot rolling mill. These production rules are based on physical and metallurgical facts, as well as local experience and quality requirements. Some rules exist in order to avoid wearing or too big temperature changes. Many rules arise from ensuring the product (e.g. surface) quality, for instance through the fact that roll width and thickness changes are limited.

Additionally, orders that are produced in the hot rolling mill need to meet customer due dates, if the product is sold right after the rolling mill, or internal due dates, if the coils are to be further processed in other sections of the steel plant (e.g. cold rolling mill). In this case, the product mix in the hot rolling mill has to also be balanced in order to "feed" different parallel down-stream processes.

Due to the complexity and the variety of plant designs in metals hot rolling only few mathematical programming approaches with applications to real world steel plants have

*matteo.biondi@it.abb.com. ABB S.p.A, 16153 Genova, Italy
†iiro.harjunkoski@de.abb.com. ABB AG, Corporate Research, 68526 Ladenburg, Germany
‡sleman.saliba@de.abb.com. ABB AG, Corporate Research, 68526 Ladenburg, Germany

been published. Lopez et al. [2] suggested a heuristic based on Tabu Search, which was successfully applied to Dofasco, a Canadian steel producer, but failed to be applied to other steel plants. Most recently, Zhao et al. [4] applied a two-stage scheduling method to the hot rolling area of Baosteel, China.

3 The Scheduling Algorithm

A production schedule for the hot rolling mill consists of a set of production campaigns (rolling programs), which are composed of a finite number of slabs/coils. The hot rolling scheduling problem consists of creating feasible rolling programs and sequencing them on the plant.

A pure single-step mathematical programming approach can neither capture all the relevant problem aspects nor meet the performance criteria. Therefore, a two-step approach combining heuristic-mathematical programming methods has been developed.

1. Build hot rolling programs,
2. Sequence the built rolling programs.

The programs built in step 1 should be as long as possible and contain as many orders as possible meeting production and quality rules. The procedure to build rolling programs takes into account all rules for allowed width and thickness changes, as well as metallurgical and physical constraints related to subsequent coil compatibilities. The procedure first applies a construction heuristics to form program parts belonging to a certain width class: From a set of orders of a given width class and steel family, a "skeleton" of the program section is built. This "skeleton" contains only the minimal number of coils to fulfill the hard constraints to ensure that the program part is feasible from the production point of view. After this, each part is filled up with other compatible orders to maximize their length. The built program sections (also called program bodies) are thereafter combined into rolling programs by assigning them a cost/profit and by solving a min-cost-flow problem.

The vertices of the graph are the program bodies $b_i, i = 1, \ldots, n$, a source s and a sink t. For each body b_i that can be followed by body b_j in a feasible program, we introduce an arc. Moreover, we connect the source s to each body that can be the first body in a program and each body that can be the last body in a program with the sink t.

Each arc connecting two bodies has an associated cost that corresponds to the negated profit of the bodies b_i and b_j and the profit of combining the two bodies to the same program. The arcs connecting the source and the sink have no additional cost. All arcs have capacity of one unit of flow.

The objective is now to minimize the cost of the flow from the source to the sink. In addition to the traditional capacity and flow conservation constraints, we introduce a uniqueness constraint. This constraint ensures that the incoming flow for each body vertex is less than or equal to 1, such that each program body is used at most once. The flow value is determined by solving upfront a max-flow-problem on the same graph.

The built programs are then sequenced, which is a traditional scheduling-type of problem. An MILP formulation of the problem is proposed taking into account due date and production mix constraints. The formulation is an extension of the slot-based approach by Pinto and Grossmann [3].

4 Benefits

Using the described approach for building hot rolling programs, we can ensure that all production requests that were not included in a program can neither form a valid program by themselves nor be added to already built programs.

The concept of building skeletons and filling the skeletons to form program parts ensures that we always consider the most valuable production requests first. Valuable production requests are e.g. coils with early due dates or coils with minimal finishing thicknesses.

Moreover, the utilization of the minimum cost flow problem for composing program parts to full programs ensures that the most valuable program parts are selected and that the combination of program parts is optimal in terms of similar due dates and common steel properties. Valuable program parts are e.g. bodies that contain a high number of valuable coils and that form a long sequence of production requests in kilometers. Therefore, our approach results in a feasible rolling program meeting the quality requirement, while maximizing the number of rolled production requests and the value of the production requests, as well as minimizing the work roll changes and the usage of waste material.

Considering production requests of one month (about $2000-5000$ coils), the program building procedure takes less than 30 seconds of computational time. The second step, sequencing the programs, requires more computational effort. Restricting the computational time of the sequencing MILP to 120 seconds yields sufficiently good sequences. Therefore, we can ensure a total computational time of strictly less than three minutes.

Finally, since we consider orders for up to several weeks (e.g., production requests in next month's order book), the scheduling department gains a better insight into the order book and the additional material needed to fill the gaps in the order portfolio. The visualization of the production plan for the next weeks and the highlighting of key performance indicators enable the schedulers to plan and react more accurately to the business plan and, therefore, improve the productivity of the plant.

References

[1] I. Harjunkoski and I.E. Grossmann (2001). *A Decomposition approach for the scheduling of a steel plant production.* Computers Chem. Eng, 25, pp. 1647-1660.

[2] I. Lopez, M.W. Carter and M. Gendreau (1998). *The hot strip mill production scheduling problem: A tabu search approach.* European Journal of Operations Research, 106(2-3), pp. 317-335.

[3] J.M. Pinto and I.E. Grossmann (1995). *A continuous time mixed integer linear programming model for short-term scheduling of multistage batch plants.* Industrial and Engineering Chemistry Research, 34, 3037-3051.

[4] J. Zhao, W. Wang, Q. Lui, Z. Wang and P. Shi (2009). *A two-stage scheduling method for hot rolling and its application.* Control Engineering Practice, Volume 17, Issue 6, June 2009, Pages 629-641.

Mixed-Criticality Scheduling of Sporadic Task Systems

Vincenzo Bonifaci (Speaker) * Gianlorenzo D'Angelo †
Alberto Marchetti-Spaccamela ‡ Suzanne van der Ster § Leen Stougie §

1 Introduction

There is an increasing trend in embedded systems towards implementing multiple functionalities upon a single shared computing platform. This can force tasks of different criticality to share a processor and interfere with each other. These *mixed-criticality* (MC) systems are the focus of our research.

We consider the scheduling of finite collections of jobs as well as the scheduling of so-called sporadic task systems; see [2] for an introduction to task systems. Although a mixed-criticality system could in principle have many criticality levels, in this paper we limit ourselves to two criticality levels.

In the remainder of this introduction we describe the model and give some notation. In Section 2 we consider scheduling mixed-criticality task systems on a single machine. In Section 3 we consider the case of multiple identical machines, both in the setting of finite collections of jobs and of task systems.

MC jobs. A job in an MC system is characterized by a 4-tuple of parameters: $J_j = (r_j, d_j, \chi_j, c_j)$, where r_j is the release time, d_j is the deadline ($d_j \geq r_j$), $\chi_j \in \{1, 2\}$ is the *criticality level* of the job and c_j is a pair $(c_j(1), c_j(2))$ representing the *worst-case execution times* (WCET) of job J_j at level 1 and level 2 respectively; it is assumed that $c_j(1) \leq c_j(2)$. Each job J_j in a collection $J_1, \ldots, J_n$ should receive execution time C_j within time window $[r_j, d_j]$. The value of C_j is not known but is discovered by executing job J_j until it signals completion. A collection of realized values $(C_1, C_2, \ldots, C_n)$ is called a scenario. The criticality level of a scenario $(C_1, \ldots, C_n)$ is defined as the smallest integer ℓ such that $C_j \leq c_j(\ell)$, $\ell = 1, 2$. (We only consider scenarios where such an ℓ exists.) The crucial aspect of the model is that, in a scenario of level ℓ, it is necessary to guarantee only that jobs of criticality at least ℓ are completed before their deadlines. In other words, once a scenario is known to be of level 2, the jobs of criticality 1 can be safely dropped.

MC task systems. Let $T = (\tau_1, \ldots, \tau_n)$ be a system of n tasks, each task $\tau_j = (c_j, p_j, \chi_j)$ having an *execution time vector* $c_j = (c_j(1), c_j(2))$, a *period* p_j, and a *criticality level* $\chi_j \in \{1, 2\}$. Again we assume that $c_j(1) \leq c_j(2)$. Task τ_j generates a potentially infinite sequence of jobs, with successive jobs being released at least p_j

*bonifaci@mpi-inf.mpg.de. Max-Planck-Institut für Informatik, Saarbrücken, Germany.
†gianlorenzo.dangelo@univaq.it. University of L'Aquila, Italy.
‡alberto@dis.uniroma1.it. Sapienza University of Rome, Italy.
§sster@feweb.vu.nl, lstougie@feweb.vu.nl. Vrije Universiteit Amsterdam, the Netherlands.

units apart. Each such job has a deadline that is p_j time units after its release (*implicit deadlines*). The criticality of such job is χ_j, and its WCET vector is given by c_j.

MC-schedulability. An (online) algorithm schedules a sporadic task system T correctly if it is able to schedule *every* job sequence generated by T such that if the criticality level of the corresponding scenario is ℓ, then all jobs of level at least ℓ are completed between their release time and deadline. A system is called *MC-schedulable* if it admits some correct scheduling algorithm.

Utilization. Let $L_k = \{j \in \{1, \ldots, n\} : \chi_j = k\}$. Define

$$u_j(k) = \frac{c_j(k)}{p_j}, \qquad k = 1, 2,\ j = 1, \ldots, n;$$
$$U_i(k) = \sum_{j \in L_i} u_j(k), \qquad i, k = 1, 2.$$

Vector u_j is called the *utilization* of task j. It is well-known that in the case of a single criticality level, an (implicit-deadline) task system is feasible on m processors if and only if $U_1(1) \leq m$ and $u_j(1) \leq 1$ for all j.

Related work. The mixed-criticality model that we follow has first been proposed and analyzed, for independent collections of jobs, by Baruah, Li and Stougie [3]. Baruah et al. [1] gave further results for scheduling independent jobs on a single machine. The mixed-criticality model has been extended to task systems by Li and Baruah [5].

2 Single machine

The scheduling of a collection of independent mixed-criticality jobs on a single processor has already been treated in reference [1], where an algorithm with a speedup bound of 1.619 has been provided. Thus, in this section we focus on the case of a task system.

We consider a variant of the Earliest Deadline First algorithm, EDF-VD (EDF with virtual deadlines). When $U_1(1) + U_2(2) \leq 1$, EDF-VD is nothing but the usual EDF algorithm. Otherwise, it consists in applying EDF to the modified task system in which the period of each task in L_2 is scaled down by a factor $1 - U_1(1)$.

We give the following sufficient condition for schedulability by EDF-VD.

Theorem 1. *Assume T satisfies*

$$U_1(1) + \min\left(U_2(2), \frac{U_2(1)}{1 - U_2(2)}\right) \leq 1.$$

Then T is schedulable by EDF-VD on a unit-speed processor.

The above schedulability condition is then used to obtain a speedup guarantee.

Theorem 2. *Let $\phi = (\sqrt{5} + 1)/2 < 1.619$. If T satisfies $\max(U_1(1) + U_2(1), U_2(2)) \leq 1$ then it is schedulable by EDF-VD on a speed ϕ processor. In particular, if T is MC-schedulable on a unit-speed processor, it is schedulable by EDF-VD on a speed ϕ processor.*

3 Multiple identical machines

Scheduling a finite collection of independent jobs

For a single machine, Baruah et al. [1] analyzed the *Own Criticality Based Priority* (OCBP) rule and showed that it guarantees a speedup bound of ϕ on a collection of independent jobs. We show that this approach can be extended to multiple identical machines at the cost of a slightly increased bound.

Theorem 3. *Let J be a collection of jobs that is schedulable on m unit-speed processors. Then J is schedulable using OCBP on m processors of speed $\phi + 1 - 1/m$.*

Scheduling a sporadic task system without migration

In this case we consider scheduling algorithms that partition the tasks on the machines and then schedule them independently (no migration). Using Theorem 2, the partitioning problem becomes a two-dimensional *vector scheduling* problem [4] where the vectors to be packed are the utilization vectors (u_j) of the tasks. The two-dimensional vector scheduling problem can be approximated in polynomial time within a factor $1 + \epsilon$ for any $\epsilon > 0$ [4], and we are able to derive the following.

Theorem 4. *For any $\epsilon > 0$ there is a polynomial-time partitioning algorithm P such that any task system that is MC-schedulable by some non-migratory algorithm on m unit-speed processors can be scheduled by P and EDF-VD on m speed $\phi + \epsilon$ processors.*

Acknowledgement. We thank Sanjoy Baruah for showing us the interpretation of the speedup result as a sufficient schedulability condition and pointing reference [4] to us.

References

[1] S. K. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow, and L. Stougie. Scheduling real-time mixed-criticality jobs. In P. Hliněný and A. Kučera, editors, *Proc. 35th Symp. on Mathematical Foundations of Computer Science*, volume 6281 of *Lecture Notes in Computer Science*, pages 90–101. Springer, 2010.

[2] S. K. Baruah and J. Goossens. Scheduling real-time tasks: Algorithms and complexity. In J. Y.-T. Leung, editor, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, chapter 28. CRC Press, 2003.

[3] S. K. Baruah, H. Li, and L. Stougie. Towards the design of certifiable mixed-criticality systems. In *Proc. 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 13–22. IEEE, 2010.

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

[5] H. Li and S. K. Baruah. An algorithm for scheduling certifiable mixed-criticality sporadic task systems. In *Proc. 16th IEEE Real-Time Systems Symposium*, pages 183–192. IEEE, 2010.

Scheduling serial batching machine with two competitive agents

Ammar Oulamara (Speaker) * Ameur Soukhal †

1 Introduction

We consider batch scheduling problems on a single serial batching machine with two competitive agents. The problem that we want to analyze may be formulated as follows. There are two agents A and B, each agent has it own set of jobs to be processed on a common serial batching machine. Agent A has to schedule the set $J^A = \{J_1^A, \ldots, J_{n_A}^A\}$ of n_A jobs whereas agent B has to process the set $J^B = \{J_1^B, \ldots, J_{n_B}^B\}$ of n_B jobs. All jobs are available at time zero and all jobs should be processed without preemption. The processing time of job J_j^A (J_j^B) is denoted by p_j^A (p_j^B), its due date by d_j^A (d_j^B) and its weight by w_j^A (w_j^B). The serial batching machine can process several jobs sequentially as a batch, i.e., a job only becomes available when the complete batch to which it belongs has been processed. The length of a batch equals the sum of processing times of its jobs and when a new batch starts, a constant setup time s occurs. We assume that jobs of both agents cannot be batched together in the same batch. The objective is to find a schedule such that jobs of agents A and B are to be processed on a common serial batching machine, where the objective function of agent A is minimized under the condition that the value of the objective function of agent B will not exceed a fixed value Q.

The concept of scheduling with competitive agents has been treated in Agnetis et al. [1], Agnetis et al. [2] which addressed the scheduling models on single machine in which two- or multi-agent compete to scheduling their jobs and each agent has his own objective function. Leung et al. [5] generalizes the work of Agnetis et al. [1] by allowing the preemption and the dynamic arrival of jobs. Problems related to multi-agent scheduling are also considered in [3], [6], [7].

Extending the standard classification scheme for scheduling problem (Graham et al. [4]) and following the notation of multi-agent scheduling problem of Agnetis et. al. [2], noted by $\alpha|\beta|\gamma^A : \gamma^B$, where γ^A is the objective function of agent A and γ^B the objective function of agent B. The problems to be considered here may be classified as

- $(L_{max}^A : f_{max}^B)$: $1|s-batch|L_{max}^A : f_{max}^B$
- $(f_{max}^A : f_{max}^B)$: $1|s-batch|f_{max}^A : f_{max}^B$

*oulamara@loria.fr. LORIA -UMR 7503 laboratory, Nancy University - INPL, Campus Scientifique - BP 239 - 54506, Vandoeuvre-les-Nancy Cedex, France.

†ameur.soukhal@univ-tours.fr. Département Informatique, Ecole Polytechnique de l'Université de Tours, 64 av. Jean Portalis 37200 Tours, France.

- $(\sum w_i^A U_i^A : \sum w_i^B U_i^B)$: $\quad 1|s-batch|\sum w_i^A U_i^A : \sum w_i^B U_i^B$

Where for agent G, $G \in \{A, B\}$, we have

- $f_{max}^G = \max_{1 \le j \le n_G}\{f_j^G(C_j^G)\}$, maximum of regular fonctions
- $\sum(w_j^G)U_j^G = \sum_{j=1}^{n_G}(w_j^G)U_j^G$, number of late jobs

Since the jobs of agent B are scheduled such that value of the objective function of agent B is less than or equal to given fixed value Q (i.e. agent B will accept a schedule of cost up to Q), and since the function $f_j^B(.)$ are known, let us define a deadline $\overline{d}_j^B$ for each job J_j^B which corresponds to the latest completion time C_j^B for which $f_j^B(C_j^B) = Q$, i.e., the completion time at which the cost of job J_j^B is equal to Q. It is easy to see that if the inverse function $f_j^{-(B)}(.)$ is available the deadline $\overline{d}_j^B$ can be computed in constant time. In the rest we assume that all $\overline{d}_j^B$ can be easily computed.

2 Dynamic programming for $(L_{max}^A : f_{max}^B)$

The following lemma is usefull to derive a dynamic programming algorithm for the problem $(L_{max}^A : f_{max}^B)$.

Lemma 1. *For the problem $(L_{max}^A : f_{max}^B)$ there is an optimal schedule such that the jobs within agent A are processed in the EDD order.*

To derive a dynamic programming algorithm for the problem $(L_{max}^A : f_{max}^B)$, we define a partial schedule S for jobs $J_1^A, \ldots, J_{j_A}^A, J_1^B, \ldots, J_{j_B}^B$ to be in state (j_A, j_B, t) if the latest batch of S is completed at time t and all jobs of agent B are ontime. Let $F(j_A, j_B, t)$ the objective value of the state (j_A, j_B, t). The state (j_A, j_B, t) is obtained by adding a batch consisting of jobs $j_{k+1}^A, \ldots, J_{j_A}^A$ to partial schedule composed of jobs $J_1^A, \ldots, J_k^A, J_1^B, \ldots, J_{j_B}^B$ or by adding batch consisting of jobs $j_{k+1}^B, \ldots, J_{j_B}^B$ to partial schedule composed of jobs $J_1^A, \ldots, J_{j_A}^A, J_1^B, \ldots, J_k^B$.

The optimal value is given by $\min\{F(n_A, n_B, t) : 2s + P \le t \le n.s + P\}$ where $P = \sum_{j \in A \cup B} p_j$ and the corresponding schedule is obtained by backtracking. The optimal value of the objective function is obtained in $O(n_A^2.n_B^2(n.s + P))$. This dynamic programming algorithm can be easily adapted to solve the problem $(f_{max}^A : f_{max}^B)$.

3 Dynamic programming for $(\sum w_i^A U_i^A : \sum w_i^B U_i^B)$

We can see that all late jobs of agent A (B) can be scheduled into one batch after all other *early* batches. In the following, we assume that the jobs of agent A $(B)$are indexed in the nondecreasing order of their due dates, i.e;, $d_1^A \le \ldots \le d_{n_a}^A$ $(d_1^B \le \ldots \le d_{n_b}^B)$.

Lemma 2. *For the problem $(\sum w_i^A U_i^A : \sum w_i^B U_i^B)$, there exists an optimal schedule in which the early jobs of agents A(B) are scheduled in nondecreasing order of their indices.*

To derive a dynamic programming algorithm for the problem $(\sum w_i^A U_i^A : \sum w_i^B U_i^B)$, we define $C_G(j_A, j_B, w_A, w_B, d)$ the minimal completion time of last early job in a partial schedule S of jobs $J_1^A, \ldots, J_{j_A}^A$, $J_1^B, \ldots, J_{j_B}^B$ in which $\sum w_i^A U_i^A$ $(\sum w_i^A U_i^A)$ is w_A (w_B),

the last processed batch contains jobs of agent G, $G \in \{A, B\}$ and the earliest due date in the last batch of S is equal to d.
A schedule in state $(j_A, j_B, w_A, w_B, d, A)$ with value $C_A(j_A, j_B, w_A, w_B, d)$ is obtained by one of the following cases : (i) In a schedule defining $C_A(j_A, j_B, w_A, w_B, d)$, job j_A is late, then $C_A(j_A, j_B, w_A, w_B, d) = C_A(j_A - 1, j_B, w_A - w_{j_A}, w_B, d)$ (ii) In a schedule defining $C_A(j_A, j_B, w_A, w_B, d)$, job j_A is scheduled early jointly with at least one other job in the last early batch, in this case we have $C_A(j_A - 1, j_B, w_A, w_B, d) + p_{j_A} \leq d$ and $C_A(j_A, j_B, w_A, w_B, d) = C_A(j_A - 1, j_B, w_A, w_B, d) + p_{j_A}$ (iii) In a schedule defining $C_A(j_A, j_B, w_A, w_B, d)$, job j_A is the only job scheduled in the last early batch, then we must have $d = d_{j_A}$ and,$C_A(j_A, j_B, w_A, w_B, d) = \min_{1 \leq k \leq j_A - 1} C_A(j_A - 1, j_B, w_A, w_B, d_k) + s + p_{j_A}$ or $C_A(j_A, j_B, w_A, w_B, d) = \min_{1 \leq k \leq j_B} C_B(j_A - 1, j_B, w_A, w_B, d_k) + s + p_{j_A}$.
A schedule in state $(j_A, j_B, w_A, w_B, d, B)$ with value $C_B(j_A, j_B, w_A, w_B, d)$ is symmetrically established according to the above analysis.
Finally the optimal value of the weighted number of tardy jobs of agent A is $w_A^* = \min_{G \in \{A,B\}} \{w_A : C_G(n_A, n_B, w_A, Q, d) < \infty, 0 \leq w_A \leq \sum_{j=1}^{n_A} w_j^A,\ d \in \{d_1^G, \ldots, d_{n_G}^G\}\}$ and it can be computed in $O(n_A^2.n_B^2.W_A.W_B)$ where $W_A = \sum_{j=1}^{n_A} w_j^A$ and $W_B = \sum_{j=1}^{n_B} w_j^B$.

References

[1] A. Agnetis, P.B. Mirchandani, D. Pacciarelli and A. Pacifici. (2004). *Scheduling problems with two competing agents.* Oper. Res. 52, 229-242.

[2] A. Agnetis, D. Pacciarelli and A. Pacifici. (2007). *Multi-agent single machine scheduling.* Ann. Oper. Res. 150, 3-15.

[3] T.C.E Cheng, C.T. Ng and J.J. Yuan. (2006). *Multi-agent scheduling on a single machine to minimize total weighted number of tardy jobs.* Theoretical Computer Science, 362, 273-281.

[4] M.R. Graham, E.L. Lawler, J.K. Lenstra and A.H.G. Rinnooy Kan. (1979). *Optimization and approximation in deterministic machine scheduling: A survey.* Ann. Discrete Math. 5, 287-326.

[5] J.Y.-T. Leung, M. Pinedo and G. Wan. (2010). *Competitive two-agent scheduling and its applications.* Oper. Res. 58, 458-469.

[6] B. Mor and G. Mosheiov. (2010). *Scheduling problems with two competingagents to minimize minmax and minsum earliness measures.* Eur. J. Oper. Res. 206, 540-546.

[7] G. Wan, S.R. Vakati, J.Y-T. Leung and M. Pinedo. (2010). *Scheduling two agents with controllable processing times.* Eur. J. Oper. Res. 205, 528-539.

M machine scheduling under uncertainties on machine availabilities

Frédéric Guinand * Amine Mahjoub † Eric Sanlaville (Speaker) ‡

1 Model and Notations

A set of n independent tasks is to be scheduled on m machines. Task durations are denoted by $p_1, \ldots, p_n$. Machine i is unavailable during the time interval $[s_i, e_i]$, and the machines are numbered according to increasing order of the s_i's. Another hypothesis states that at any time, at least one machine is available. The objective is the minimization of the makespan.

This problem has been much studied during the recent years (see first survey in [3]), as it has immediate applications in workshop management (unavailability is due to maintenance periods) or in parallel computing (occurence of top priority tasks). However, even if the unavailability period remains of fixed duration, it may occur at another time than the forecasted one s_i. What happens now for the makespan? In other words, what is the performance decrease of the chosen schedule? This problem is tackled here.

The approach considered is *proactive-reactive* [4] : a first schedule, hereafter called *reference schedule*, is computed with the estimated values s_i (proactive phase). If there are disturbances on the unavailability periods during the execution, the schedule is modified to couple with these disturbances (reactive phase). This modification, called stabilization, is explain in the next section.

The uncertainty is modeled as follows : for one machine i, the *real* unavailability starting time is supposed to remain inside an interval $[s_i - \epsilon, s_i]$. The case where unavailability occurs later than expected leads to similar results. Several indicators may be used to measure the *robustness* of a method (in our proactiv-reactive approach, the method is defined by the computation of the reference schedule and by the stabilization process). A measure of its stability is given by the difference between the makespan of the real schedule and the makespan of the reference schedule. Another measure is the difference between the optimal makespan for the real unavailability periods and the makespan of the real schedule (the absolute deviation). Both proposed measures can be computed as upper bounds on all problem instances and all disturbances with a fixed ϵ.

In the case of the m machine scheduling of independent tasks, the deterministic problem is NP-Hard. However, it has been established that *LPT* (Longest Processing

*frederic.guinand@univ-lehavre.fr. Laboratoire LITIS, Université du Havre, rue Philippe Lebon, BP 540, 76058 Le Havre Cedex, France.

†mahjoub.amin@gmail.com. UTIC, Ecole Supérieure des Sciences et Techniques de Tunis, 5 avenue Taha Hussein, BP 56, Bab Menara, 1008 Tunis, Tunisie

‡eric.sanlaville@univ-lehavre.fr. Laboratoire LITIS, Université du Havre, rue Philippe Lebon, BP 540, 76058 Le Havre Cedex, France.

Time first) list schedules behave quite good, even when machines are not continuously available [1]. The objective of our study is to show that LPT is still a good choice under uncertainties on unavailability periods.

2 Computing the impact of disturbance on any schedule

The stabilization process is now presented. Note first that in our model, the tasks are not preemptive nor resumable. That is, if a task is interrupted because the machine becomes unavailable, then it must be entirely re-executed later. Let us consider a disturbance on machine m_i. The tasks which are no more executable before the unavailability on the machine m_i are scheduled after all the others, as soon as possible, on any machine and according to the same order as in the reference schedule.

The resulting schedule is called the *final schedule* or the *stabilized schedule*, see [1]. In figure 1, due to the disturbance on machine m_2 unavailabilty, the tasks T_1 and T_2 are rescheduled. T_1 is executed first on machine m_2, however T_2 is executed on machine m_1 because it is available first. The overall makespan is changed from C_2 to $\tilde{C}_1$.

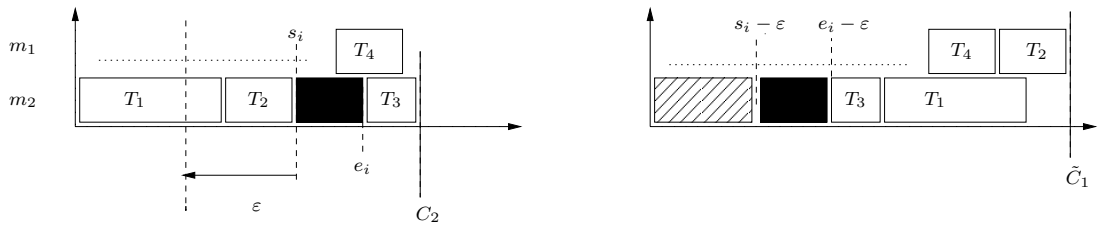


Figure 1: Effect of disturbance

The impact of disturbances is evaluated by the stability measure, for any reference schedule S with makespan C_S. Denote by $\tilde{S}$ the stabilized schedule, with $C_{\tilde{S}}$ its makespan. The goal is to bound the difference $C_{\tilde{S}} - C_S$. Obviously, this bound depends on ϵ and on $P = \max_{j\in\{1,\ldots,n\}} p_j$.

Theorem 1. *Let Q be the smallest multiple of P larger than ϵ : $Q = k \cdot P$ and $(k-1)P \leq \epsilon < k \cdot P$. Then, for any schedule S,*

$$C_{\tilde{S}} - C_S \leq P + Q = (k+1) \cdot P$$

Two cases must be distinguished for the proof, wether ϵ is smaller or larger than P.

The key point is that this bound is obtained without any hypothesis on S. Remark also that it is a step-function of ϵ. However, it is of course better when the reference schedule has a small makespan. Hence in the next section the special case of LPT schedules is considered.

3 Performances of LPT schedules

In this section, the performances of the schedules obtained by LPT rule are considered. It means that the schedule denoted by S is now obtained by LPT.
Since [2], a relative bound on the performance ratio of LPT for the deterministic problem is known (C^* denote the optimal makespan):

$$C_S/C^* \leq \frac{m+1}{2}$$

This is quite bad for a problem without precedence ! However, we suspected one might find more favourable results, especially the existence of an absolute bound (remember that in the case without unavailability, the relative bound is $4/3$, whereas there exists an absolute bound of P for any list schedule).

Theorem 2. *For any instance of the problem with unavailability,*

$$C_S - C^* \leq m \cdot P$$

and this bound is asymptotically tight.

In the previous section, we have established a general result about the stability of any schedule. Now, considering that the schedule denoted by S is obtained by LPT, we provide the following result, which is still a conjecture at the time this abstract is written.
Remember, that $C_{\tilde{S}}$ is the makepan of the stabilized LPT schedule. We denote by S' the schedule obtained by LPT rule for the real instance (a posteriori computation) :

Conjecture 3.

$$C_{\tilde{S}} - C_{S'} \leq P$$

Provided the conjecture is indeed verified, and due to theorem 2, we get a result on the robustness of LPT for the problem at hand:

$$C_{\tilde{S}} - \tilde{C}^* \leq (m+1) \cdot P$$

where $\tilde{C}^*$ denotes the optimal makespan of the real instance.

References

[1] Adel Esssafi, Amine Mahjoub, Grégory Mounié, Denis Trystram (2009). *LPT scheduling algorithms with unavailability constraints under uncertainties.* Proceedings of International Conference on Parallel Computing (ParCo), Lyon, France, sep 2009.

[2] C.Y. Lee (1996). *Machine scheduling with an availability constraint* j. of global optimization, 9:363-382.

[3] E. Sanlaville and G. Schmidt (1998). *Machine Scheduling with Availability Constraints*. Acta Inf. 35(9): 795-811.

[4] E. Sanlaville, A. Moukrim, J.C. Billaut (eds.) *Flexibility and Robustness in Scheduling*, ISTE-Wiley, London,2008

Media Streams Planning*

Hana Rudová (Speaker) † Pavel Troubil*

1 Introduction

Media streams planning introduces a problem where placement of latency-sensitive streams over network is processed. In contrast to other classical path placement problems [4], each stream represents a communication tree with the root at producer and leafs at consumers. This problem may seem to be suitable for the standard multicast transmissions but it cannot be used for two reasons. First, the planning is processed over heterogeneous networks where multicast is not generally available. Second, minimization of latency unavailable by multicast must be considered to allow transmission of high-bandwidth streams.

Our work introduces an alternative to the constraint programming approach [1] included in an application middleware CoUniverse [2] representing pioneering orchestration of component-based interactive collaborative environments. The CoUniverse (with the constraint programming solver) has been demonstrated in many deployment cases such as distributed classroom taught at the Louisiana State University and transmitted to a number of universities and institutes in the U.S., Argentina and the Czech Republic (regularly taught since 2007). An example of data transmission computed by the CoUniverse is demonstrated in Figure 1. Our aim is to replace actual constraint programming solver by a new integer linear programming solver, which appears to have a better performance.

2 Integer Linear Programming Formulations

Integer linear programming (ILP) solver implemented in Gurobi Optimizer[1] builds on top of the constraint programming (CP) solver [1]. Here binary variables $x_{s,l}$ called streamlinks are introduced to represent transmission of a stream s over a network link l. Basically, there are linear constraints handling network capacity limitations and constraints managing tree transmission of each stream from producer to his consumers. The aim of optimization is to minimize achieved latency over all streamlinks.

To implement the new ILP solver, it was necessary to redefine some of the constraints managing tree transmission to ensure their linearity. This results in the first ILP formulation which already has a better performance over the CP solver. The new solver improves the running time for two out of the three input network topologies. Hence, it

*This work has been supported by the Ministry of Education, Youth and Sports of the Czech Republic under the research intent No. 0021622419.

†hanka@fi.muni.cz, pavel@ics.muni.cz. Faculty of Informatics, Masaryk University, Botanická 68a, Brno, Czech Republic

[1]Gurobi Optimizer version 4.0.1 available from http://www.gurobi.com.

Figure 1: Data transmissions computed by the CoUniverse during the 7th Annual Global LambdaGrid Workshop (GLIF) 2007 (taken from [1]).

leads to an increase of input sizes (number of sites) which can be solved in real time, i. e., within a few seconds (media transfers are basically realized among several sites representing geographical structure of the network typically).

The initial formulation involves constraints avoiding cycles to ensure tree structure of each transmission. Performance analyses have shown that these constraints impact performance of the ILP solver severely. We have proposed several cycle avoidance methods, which are always applied upon a set of core constraints (initial formulation without the cycle avoidance).

In the CP solver, there are two alternatives based on the subtour formulation and the MTZ formulation [3] known from the traveling salesman problem. Due to relatively low number of cycles emerging in the solutions, we proposed an alternative methods, where application of the constraints is delayed. The problem is solved without any cycle avoidance constraints and they are added only to eliminate cycles which actually appear in the solution. Due to low number of the additional constraints (typically lower than number of nodes in the input network), running time is significantly shorter.

We have also proposed and analyzed network flows formulation for cycle avoidance. We introduced new integer linkflow variables $y_{s,l}$ counting the number of consumers receiving the stream s through the link l. While network flows formulation is not applicable to streamlinks due to violation of Kirchhoff's laws at data distribution nodes, it can be applied to linkflows. Measurements show that the network flows formulation of cycle avoidance performs the fastest of the presented formulations.

Our current work involves implementation in an open-source solver to allow its public distribution with the CoUniverse. Our future work includes consideration of heuristic algorithms for the current problem or further extensions of the problem such as inclusion

of partial knowledge of physical topology or availability of multicast in some subnetworks.

References

[1] Petr Holub, Hana Rudová, and Miloš Liška. Data transfer planning with tree placement for collaborative environments. *Constraints*, 2011. To appear.

[2] Miloš Liška, Petr Holub, Andrew Lake, and John Vollbrecht. CoUniverse orchestrated collaborative environment with dynamic circuit networks. In *ICN 2010: Ninth International Conference on Networks*, pages 300–305. IEEE Computer Society, 2010.

[3] Clair E. Miller, Albert W. Tucker, and Richard A. Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the ACM*, 7:326–329, 1960.

[4] Helmut Simonis. Constraint applications in networks. In Francesca Rossi, Peter van Beek, and Toby Walsh, editors, *Handbook of Constraint Programming*, pages 875–903. Elsevier, 2006.

Fast separation algorithms for three-index assignment problems

Trivikram Dokka (Speaker) [*] Ioannis Mourtos [†] Frits C.R. Spieksma [‡]

1 Introduction

Given a specific combinatorial optimization problem, an Integer Linear Program (ILP) formulation is of the form $\min\{c^T x : Ax = b, x \in Z_+^n\}$, where x denotes an n-dimensional column vector of variables, and where A, b, and c are given coefficients of appropriate dimensions. The convex hull of the feasible solutions is defined by the corresponding polyhedron $P_I = \text{conv}\{x \in Z_+^n : Ax = b\}$. The strongest possible inequalities valid for P_I are the so-called facet-defining inequalities. It is a standard technique in cutting plane algorithms to efficiently use these inequalities to strengthen the linear programming relaxation $P_L = \min\{c^T x : Ax = b, x \geq 0\}$.

In cutting plane methods, an inequality is added to the current linear program only when it is violated by a specific vector $x \in R^n$. The problem of determining whether such a vector violates an inequality of a specific family is called the *separation problem* for this family and an algorithm solving it is called a *separation algorithm*. Thus, an important task, once families of inequalities have been identified, is to design separation algorithms. These are often family-specific, and, in order to be useful, should be of low computational complexity.

It is customary to express the complexity of a separation algorithm in terms of n, the dimension of the vector x. This seems reasonable since, at the very least, one would need to inspect each entry of the vector x to decide whether a violated inequality exists. In fact, separation algorithms with a complexity of $O(n)$ have been called "best-possible" ([2], [4]). In this contribution, we point out that, due to the fact that the vector x can often be described (much) more compactly than listing all n entries, it is interesting to study the impact of this fact on the design of separation algorithms.

Indeed, as described above, separation algorithms are normally incorporated within a *Branch & Cut* scheme, i.e., a search scheme to find $z = \min\{c^T x : x \in P_I\}$ by strengthening the LP-relaxation at various nodes of the search tree. And indeed, within such a scheme, the separation algorithm receives as input not any $x \in R^n$, but instead a vector describing an optimal solution of the current LP-relaxation, i.e., an $x \in P_L \backslash P_I$. Hence it appears reasonable to ask whether this fact can be employed to speed-up separation algorithms, thus resulting in more efficient *Branch & Cut* schemes.

[*] trivikram.dokka@econ.kuleuven.be. Operations Research Group, Leuven University, Naamsestraat 69, B-3000 Leuven, Belgium.

[†] mourtos@aueb.gr. Department of Management Science and Technology, Athens University of Economics and Business, 76 Patission Ave. 1043 34 Athens, Greece.

[‡] frits.spieksma@econ.kuleuven.be. Operations Research Group, Leuven University, Naamsestraat 69, B-3000 Leuven, Belgium.

In this paper, we answer this question in the positive for specific classes of inequalities known to be valid for polytopes corresponding to three-index assignment problems. More specifically, we give separation algorithms for two classes of facet-defining inequalities of the 3-index assignment polytope that are faster than the known ones.

2 The three-index assignment polytope

The 3-index assignment problem, defined on three disjoint n-sets I, J, K and a weight function $w : I \times J \times K \longrightarrow \mathbb{R}$, asks for a collection of triples $M \subseteq I \times J \times K$ such that each element of any set appears in exactly one triple and function w is minimized (over all possible such collections). Its formulation as an ILP is

$$\begin{aligned}
\min \quad & \sum_{i\in I}\sum_{j\in J}\sum_{k\in K} w_{ijk}x_{ijk} & & \\
\text{s.t.} \quad & \sum_{j\in J}\sum_{k\in K} x_{ijk} = 1 & \forall i \in I, & (1)\\
& \sum_{i\in I}\sum_{k\in K} x_{ijk} = 1 & \forall j \in J, & (2)\\
& \sum_{i\in I}\sum_{j\in J} x_{ijk} = 1 & \forall k \in K, & (3)\\
& x_{ijk} \in \{0,1\} & \forall i \in I, j \in J, k \in K. & (4)
\end{aligned}$$

Let A^n denote the $(0,1)$ matrix corresponding to the constraints (1) - (3), which has n^3 columns and $3n$ rows. Notice that from this point in the text onwards, n denotes the cardinality of each set being 'assigned'; hence, the number of variables is n^3. Then, the 3-index assignment polytope is $P_I^n = \text{conv}\{x \in \{0,1\}^{n^3} : A^n x = e\}$, while its LP-relaxation is $P^n = \{x \in R^{n^3} : A^n x = e, x \geq 0\}$. A survey of Multi-index assignment problems can be found in [3, 5].

The *column intersection* graph of A^n, namely $G(V, E)$, has a node for each column of A^n, and an edge for every pair of columns that have a $+1$ entry in the same row. Notice a column contains three $+1$'s. We define the intersection of two columns c and d denoted by $|c \cap d|$, as the set of rows of A^n where both columns have $+1$ entry. It is easy to see that $V = I \times J \times K$ and $E = \{(c, d) : \{c, d\} \subseteq V, |c \cap d| \geq 1\}$, i.e., a node in V corresponds to a triple, and two nodes are connected if the corresponding triples share some index. A *clique* is a *maximal complete* subgraph.

In $G(V, E)$, there are two types of cliques that give rise to families of inequalities that are facet-defining for P_I and that are separable in polynomial time. To formally define the two types of clique inequalities, let:

- for each $c \in V : Q(c) = \{d \in V : |c \cap d| \geq 2\}$, and
- for each $c \in V : \text{co}Q(c) = \{d \in V : |c \cap d| = 1\}$, and
- for each $c, d \in V$ with $|c \cap d| = 0 : Q(c, d) = \{c\} \cup \{Q(d) \cap \text{co}Q(c)\}$,

Notice that $c \in Q(c)$.

As usual, $x(Q(c)) = \sum_{q\in Q(c)} x_q, x(Q(c, d)) = \sum_{q\in Q(c,d)} x_q$.

Definition 1. *For each $c \in V$, the facet-defining inequality $x(Q(c)) \leq 1$ is called a clique inequality of type I.*

Definition 2. *For each $c, d \in V$ with $|c \cap d| = 0$, the facet-defining inequality $x(Q(c,d)) \leq 1$ is called a clique inequality of type II.*

Separation of cliques of type I and II was first treated in Balas and Saltzman [1] through algorithms of $O(n^4)$ time complexity. Improved $O(n^3)$ algorithms (i.e., of complexity linear in the number of variables) were presented in Balas and Qi [2]. These separation algorithms were called 'best-possible'.

The following fact is from linear programming theory:

Proposition 3. *An extreme point of P^n has at most $3n$ non-zero variables.*

3 Fast Separation Algorithms: Our Results

We show how the complexities of current algorithms can be improved when we use as input-measure the number of positive components of the given vector x. Formally we denote this input measure as T: the number of positive components in given vector x. In a cutting plane context, T varies from $O(n)$ to $O(n^3)$. Given Proposition 3, it is relevant to use this input measure T to design faster separation algorithms.

Our main results are the following:

Theorem 4. *Given an arbitrary $x \in P^n$, a violated clique inequality of type I can be identified in $O(n^2 + T)$ time.*

Theorem 5. *Given an arbitrary $x \in P^n$, a violated clique inequality of type II can be identified in $O(nT)$ time.*

Theorem 6. *For each fixed $\epsilon > 0$, we can find out in $O(n^2 + T)$ time whether a clique inequality of type II with right-hand side $1 + \epsilon$ is violated.*

References

[1] E. Balas, M. J. Saltzman, Facets of the three-index assignment polytope, Discrete Appl. Math. 23 (1989) 201-229.

[2] E. Balas, L. Qi, Linear time separation algorithms for the three index assignment polytope, Discrete Appl. Math. 43 (1993) 1-12.

[3] Burkard, R., M. Dell'Amico, and S. Martello (2009), *Assignment Problems*, SIAM.

[4] L. Qi and D. Sun, Polyhedral methods for solving three index assignment problems, in Nonlinear Assignment Problems: Algorithms and Applications, P.M. Pardalos and L. Pitsoulis, eds. (Kluwer Academic Publisher, Nowell, MA. USA, 2000) 91-107.

[5] F. C. R. Spieksma, Multi-index assignment problems: complexity, approximation, applications, in Nonlinear Assignment Problems: Algorithms and Applications, P.M. Pardalos and L. Pitsoulis, eds. (Kluwer Academic Publisher, Nowell, MA. USA, 2000) pp. 1-12.

An Integrated Vehicle Routing and Duty Roster Planning of Toll Control Inspectors *

Ralf Borndörfer † Guillaume Sagnol ‡ Elmar Swarat (Speaker) §

1 Introduction

In 2005 a distance-based toll for all commercial trucks of twelve tonnes vehicle weight and above has been introduced on German motorways, in order to fund growing investments for maintenance and motorway extensions. These are caused by an increasing amount of freight transport. The toll enforcement is in responsibility of the Federal Office for Goods Transport (BAG). It is done by a combination of an automatic enforcement by stationary bridges with a mobile enforcement by random tours of 300 control vehicles through the whole network. We consider the problem of finding optimal control tours for these mobile control teams. A team is mostly composed of two inspectors or of only one. The goal is to guarantee a network-wide control which is proportional to spatial and time depending traffic distributions. An important restriction is that each team, and respectively each vehicle, may only control sections that are close to their home depot. The number of vehicles and its depots are given as fixed.

To the best knowledge of the authors there is no optimization approach to toll enforcement in the literature. There exists a lot of literature for other problems concerning evading, like tax evading or ticket evading in public transport. Those approaches mainly discuss the expected behaviour of evaders or payers, e.g. [2], or optimal levels of inspection, see [1]. A very recent approach in inspector scheduling in public transport was made at DSB S-tog in Denmark [3], but in contrast to our problem they only consider the temporal scheduling of the inspectors and not their routing through the network.

2 A graph model for an integrated staff and tour planning

The basic structure for our problem consists of a graph $G = (S, N)$, in which the nodes $s \in S$ are so-called "control sections", that are sub-parts of the network with a length of approximate 25-50 km. An edge $n \in N$ connects two sections, if they have at least one motorway junction or interchange in common. Furthermore, there is a given planning horizon T, e.g., four weeks, and a pre-chosen time discretization Δ. According to the

*This work was funded by the German Federal Office for Goods Transport (BAG).

†borndoerfer@zib.de. Zuse Institute Berlin, Department Optimization, Takustrasse 7, D-14195 Berlin, Germany.

‡sagnol@zib.de. Zuse Institute Berlin, Department Optimization, Takustrasse 7, D-14195 Berlin, Germany.

§swarat@zib.de. Zuse Institute Berlin, Department Optimization, Takustrasse 7, D-14195 Berlin, Germany.

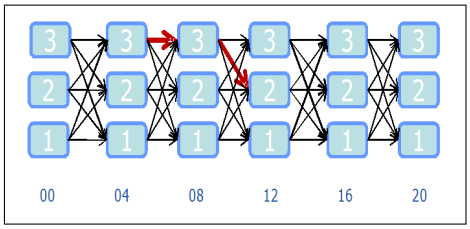


Figure 1: Example of a time-expanded control planning graph

requirement to define both the spatial routing and the temporal sequence of the tours we extend G to a time-expanded digraph $D = (V, A)$. There the nodes $v \in V$ are either defined as a pair of a section and a point in time, i.e. $v = (s, t) \in S \times [0, T]$ or they represent the artifical start and end nodes for the vehicle paths (depot nodes). The directed arcs connect either adjacent time intervals of a same section, $a = ((s, t_1), (s, t_2))$ with $t_2 = t_1 + \Delta$ starting at $t_1 = 0$ until $t_2 = T$, or they connect adjacent sections, i.e., if $(s_1, s_2) \in N$ it holds that $((s_1, t_i), (s_2, t_{i+1}) \in A \, \forall t_i \in \{0, \Delta, \ldots, T - \Delta\}$, see Figure 1 for an example. In addition to that, arcs are directed from the start depot to all other nodes (except for the end depot node) and from all nodes to the end depot node.

If we assign a time-dependent profit value to each section to control, our problem could be seen as a special vehicle routing problem with profits under some additional constraints. The vehicle routing problem is a well established research area, see [4] for an overview, and for the special case of dealing with profits Feillet et al. [5] give a survey on relevant literature.

We formulate our problem as a 0/1 multi-commodity-flow-problem in D. The vehicles represent the commodities and each feasible control tour relates to a length restricted path in D, starting and ending at the depot nodes. For each section s that must not be visited by a vehicle f we impose an outflow of zero for all $v = (s, t_i) \in V$ for the commodity representing f. Minimum control requirements for $s \in S$ lead to minimum outflow conditions for the set of nodes $v = (s, t_i)$. The profit value for each node $v \in V$ is set on all outflow arcs $a \in \delta^+(v)$, to collect it by each path visiting v. The resulting multi-commodity-flow-problem is formulated as an IP maximizing the overall profit.

The second task is the planning of the duties, the rosters and the staff assignments. In public transport this is usually done sequentially (see [6] Chapter 1) and partly anonymously. The use of mathematical methods is thereby already well established. Unfortunately, an analogues approach for the toll control is not possible according to the above mentioned spatial restrictions for the teams. Hence, we must add a person-based duty roster planning to the tour planning problem, to prevent infeasible staff assignments.

Similar to models for vehicle scheduling in public transport [7] we formulate the rostering problem by a multi-commodity flow problem in a graph, where the nodes represent duties and the arcs model a feasible sequence of two duties according to legal rules. Again there are two artifical start and end nodes that are connected with all other nodes. This problem is formulated by an IP minimizing the duty costs. Then the two planning problems are connected by coupling constraints to an integrated formulation. More precisely, those coupling constraints ensure that for each chosen tour path in D all inspectors in the corresponding team have a feasible duty roster path with a duty in the time horizon of the planned tour. The objective function is then a combination of collecting the profit and minimizing the cost. Our challenge is to compute the solution

instance	$\Delta(h)$	dc	lr	wtd	columns	rows	v(lp)	v^*	gap(%)	time(sec.)
T1	4	x	x	x	136264	17245	368997.56	350288.76	0.06	12283.0[†]
T2	4	x	x	-	136264	17237	449883.49	435343.00	-	181.62
T3	4	-	x	x	128808	17257	533597.42	499561.80	1.84	21600.00
T4	2	x	x	x	376328	22877	677133.58	644458.66	0.05	21600.00
T5	2	x	x	-	376328	22869	732316.22	709276.99	0.04	3554.2[†]
T6	2	-	x	x	368872	22889	846722.87	796495.47	1.52	21600.00

Table 1: IP-Solution analysis of some instances from a control region of Brandenburg with a time limit of 6h (21600 sec) and a tree memory limit of 10 GB. The term v(lp) denotes the root relaxation value and v^* the best integer value. The instances are characterised by the parameters dc, lr and twd. The dc stands for using duty costs, the lr that legal rules are fulfilled and the wtd that a desired working time window distribution across the day is part of the input.

of the integrated vehicle routing and duty roster planning. This approach is different to the routing and scheduling problems that are mostly discussed in the literature.

In Table 1 we present computational results from a control area of Brandenburg. We calculated four-week control plans using CPLEX 12.2 [8] as an IP solver. An important result is that all duty-cost instances could be solved near to optimality. And mainly for the four-hour discretization ($\Delta = 4$), the solution time reduces extremely if we omit the working time distribution constraints. However the problem is more difficult to solve if we replace the duty costs by coefficients that prefer an employee-friendly plan.

References

[1] C. Boyd and C. Martini and J. Rickard and A. Russell (1989). *Fare Evasion and Non-Compliance: A Simple Model.* Journal of Transport Economics and Policy. London School of Economics and Political Science and the University of Bath.

[2] M.G. Allingham and A. Sandmo (2006) *Income tax evasion: a theoretical analysis.* International Library of Critical Writings in Economics Vol 195 No. 3 2006. Edward Elgar Publishing Ltd.

[3] P. Thorlacius and J. Clausen and K. Brygge (2009). *Scheduling of inspectors for ticket spot checking in urban rail transportation.* DSB S-tog. Copenhagen.

[4] P. Toth and D. Vigo (2002) *The vehicle routing problem.* Society for Industrial and Applied Mathematics. Philadelphia.

[5] D. Feillet and P. Dejax and M. Gendreau (2005) *Traveling Salesman Problems with Profits* Transportation Science Vol. 39 No. 2 2005. Informs.

[6] S. Weider (2007) *Integration of Vehicle and Duty Scheduling in Public Transport* Technische Universität Berlin. Dissertation.

[7] A. Löbel (1997) *Optimal Vehicle Scheduling in Public Transit* Technische Universität Berlin. Dissertation.

[8] *User-Manual CPLEX 12.2* (2010) IBM ILOG CPLEX. IBM Software Group.

[†]Memory-Limit reached

Stability in multi-skill workforce assignments: complexity analysis and stable assignments polytope

Cor Hurkens (Speaker) * Murat Fırat †

1 Introduction

We analyze stability in multi-skill workforce schedules. In our stability analysis, we extend the notion of blocking pairs as stated in the Marriage model of Gale-Shapley. It is shown that finding stable schedules is NP-Hard. In some special cases stable schedules can be constructed in polynomial time. Finally, we define a set of inequalities that must be satisfied by all stable schedules.

Firstly, we define the preferences of the players: *technicians* and *jobs*. An assignemnt is said to be blocked if a technician-job pair not assigned can improve their individual preferences by being assigned. This is the same definition of a blocking pair in the milestone paper of [1]. The challenge in our assignment problem is to satisfy the multi-dimensional skill requirements of jobs. A job can be performed by a team of technicians provided that the collective capabilities of the team are above a certain threshold.

2 Problem Description and Notation

2.1 Skills

The degree of experience or expertise in a skill domain is interpreted by *hierarchical* levels. An expert possesses the highest level and a beginner qualifies as the lowest. Let $\mathbb{D}$ be the set of skill domains and $\mathbb{L}$ the set of hierarchical skill levels. The skill (l, d) is said to be at level l and belongs to domain d.

2.2 Technicians

We are given a set T of technicians and skills for technician for $t \in T$ are specified by $S_t \in \{0,1\}^{\mathbb{L}\times\mathbb{D}}$. If skill level of technician t in skill domain d is $l \in \mathbb{L}$, then $S_t^{(l',d)} = 1$ for $l' \leq l$ and $S_t^{(l',d)} = 0$ otherwise.

Skill value of technician t, denoted by ϵ_t, is found by aggregating the skills in all domains at all levels with corresponding weights. If we let $W \in \mathbb{R}^{\mathbb{L}\times\mathbb{D}}$ be the skill weight matrix, then the skill value of t is calculated by $\epsilon_t = \langle W, S_t \rangle$.

The skill of a team $T' \subseteq T$ is defined as the skill sum of individual technicians: $S_{T'} = \sum_{t\in T'} S_t$. Similarly, the skill value of T', denoted by $\epsilon_{T'}$, is found by $\epsilon_{T'} = \sum_{t\in T'} \epsilon_t = \langle W, S_{T'} \rangle$.

*c.a.j.hurkens@tue.nl. Department of Mathematics and Computer Science, TU Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands.

†m.firat@tue.nl. Department of Mathematics and Computer Science, TU Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands.

In an assignment, the job to which technician t is assigned is denoted by $J(t)$ and $T(j)$ denotes the team assigned to job j. Clearly, in a schedule $J(t) = j$ if and only if $t \in T(j)$.

2.3 Jobs

We are given a set J of jobs with skill requirements $RQ_j \in \mathbb{R}^{\mathbb{L}\times\mathbb{D}}, \forall j \in J$. All jobs have a work day length and are processed *in parallel* on a workday. Skill requirements are *cumulative* in the sense that any requirement in a higher level is carried to lower levels. For $j \in J$, we have $l' \leq l \Rightarrow RQ_j^{(l',d)} \geq RQ_j^{(l,d)}, \quad l, l' \in \mathbb{L}, d \in \mathbb{D}$. The *non-cumulative* or *explicit* skill requirement, denoted by RQ_j^*, is obtained from RQ_j as follows:

$$RQ_j^{*(l,d)} = \begin{cases} RQ_j^{(l,d)} & \text{if } l = |\mathbb{L}|, \\ RQ_j^{(l,d)} - RQ_j^{(l+1,d)} & \text{if } 0 < l < |\mathbb{L}|. \end{cases}$$

Contributions to Jobs: We make the assumption that technicians can use their skills in more than one skill domain simultaneously while performing a job. The *contribution level* of technician t to a job j in skill domain d, is defined as the maximum level that is both explicitly required j and reached by t. It is found by $CON_{(t,j)}^d = \max\left[\{0\} \cup \{l \in \mathbb{L} | RQ_j^{*(l,d)} > 0 \text{ and } S_t^{(l,d)} > 0\}\right]$.

In skill domain d, the highest contribution level that technician t can achieve is found by $CON_{(t,J)}^d = \max_{j\in J}\{CON_{(t,j)}^d\}$. Each technician orders skill domains *lexicographically* with respect to $CON_{(t,J)}$. Ties due to the same maximum contributions are broken by choosing the domain in which there is less competition and further ties are broken by choosing the domain with minimum index. The skill domain with highest ranking is called the *favorite* domain and it is denoted by $d_t^*(J) = Argmax\{CON_{(t,j)}^d, d \in \mathbb{D}\}$.

Definition 1. *$\Delta(j,t)$ is the set of skills that are required by job j for which t is not qualified. Skills in $\Delta(j,t)$ are called* exclusive. *Note that for an exclusive skill (l,d), we have $RQ_j^{*(l,d)} > 0$ and $S_t^{(l,d)} = 0$.*

Definition 2. *In an assignment, the skill (l,d) for which we have $RQ_j^{(l,d)} = S_{T(j)}^{(l,d)}$ and $RQ_j^{(l,d)} > 0$ is called* critical *and the contributions in this skill are also called* critical. *$C(j) \subset \mathbb{L} \times \mathbb{D}$ denotes the set of critical skills.*

Proposition 3. *In an assignment, let $t' \in T(j)$ and $t \notin T(j)$. Technician t* can *replace t' in $T(j)$ if and only if $S_{t'}^{(l,d)} \leq S_t^{(l,d)}$ for all $(l,d) \in C(j)$.*

3 Preference Structure

In this section we explain the preference criteria of technicians and jobs.

Preference criterion of technicians: Technician t *likes* job j more than j' if and only if $CON_{(t,j')}^{d_t^*(J)} < CON_{(t,j)}^{d_t^*(J)}$. Ties are broken by checking the contributions in t's lexicographic skill domain ordering. Further ties are broken by job indices.

Preference criterion of jobs: Job j *likes* technician t if and only if, j decreases the total skill value of its team by employing t and releasing one technician. Let the released

technician be t', then we know $S_{t'}^{(l,d)} \leq S_t^{(l,d)}$ for all $(l,d) \in C_\mu(j)$ by Proposition (3) and $\epsilon_t < \epsilon_{t'}$.

Blocking pairs: We adapt the idea of blocking pairs of [1] to our scheduling context. The pair (t,j) is said to block a schedule if and only if t and j *like* each other.

4 Complexity Analysis

The subset of problem instances in which there is one skill domain is denoted by 1D-nL-nT. In nD-1L-2T, jobs are performed two technicians and in a skill domain technicians are skilled or unskilled. We have the following results:

Theorem 4. *In feasible 1D-nL-nT instances,* stable *assignments can be constructed in polynomial time.*

Theorem 5. *Constructing a feasible assignment in nD-1L-2T is NP-Hard.*

5 Stable assignments polytope

We have $x_{tj} = 1$ if $j \in T(j)$, otherwise $x_{tj} = 0$. If $y_{tj} = 1$, then technician t likes job j, otherwise $y_{tj} = 0$. If $\beta_{(j,(l,d))} = 0$, then the requirement of j in skill (l,d) is critical, otherwise $\beta_{(j,(l,d))} = 1$. Lastly, if $\tau_{(t,j,t')} = 0$, then technician t can replace technician t' in the team $T(j)$, otherwise $\tau_{(t,j,t')} = 1$. Let $\beta'_{(j,l,d)}$ and y'_{tj} denote the negations of the corresponding terms. The set $J_<(j,t)$ of jobs denotes the jobs to which job j is preferred. Lastly, $\delta\epsilon_{t,t'} = 1$ if $\epsilon_t > \epsilon_{t'}$, 0 otherwise.

$$\sum_{j \in J} x_{tj} \leq 1, \qquad \forall t \in T \tag{1}$$

$$\sum_{t \in T} S_t x_{tj} \geq RQ_j, \qquad \forall j \in J \tag{2}$$

$$x_{tJ_<(j,t)} = y_{tj}, \qquad \forall (t,j) \in T \times J \tag{3}$$

$$\sum_{t \in T} S_t^{(l,d)} x_{tj} - RQ_j^{(l,d)} \geq \beta_{(j,(l,d))}, \qquad \forall j \in J, \forall (l,d) : RQ_j^{(l,d)} > 0, \tag{4}$$

$$\sum_{t \in T} S_t^{(l,d)} x_{tj} - RQ_j^{(l,d)} \leq |\mathbb{L}||\mathbb{D}|\beta_{(j,(l,d))}, \qquad \forall j \in J, \forall (l,d) : RQ_j^{(l,d)} > 0 \tag{5}$$

$$\sum_{(l,d) \in \Delta(j,t)} S_{t'}^{(l,d)} \beta'_{(j,(l,d))} \geq \tau_{(t,j,t')}, \qquad \forall (t,t'), \forall j \in J \tag{6}$$

$$\sum_{(l,d) \in \Delta(j,t)} S_{t'}^{(l,d)} \beta'_{(j,(l,d))} \leq |\mathbb{L}||\mathbb{D}|\tau_{(t,j,t')}, \qquad \forall (t,t'), \forall j \in J \tag{7}$$

$$x_{t'j} \leq y'_{tj} + \tau_{(t,j,t')} + \delta\epsilon_{t,t'}, \quad \forall (t,t'), \forall j \in J \tag{8}$$

(1) and (2) ensure the *feasibility.* In (3), technician t likes job j *iff* he is assigned to a job he likes less than j. (4) and (5) (6)*and*(7) determine the critical skills {technician replacements}. Finally, (8) is the key inequality to prevent instability.

Proposition 6. *An assignment satisfying the inequalities (1)-(8) is* stable.

References

[1] Gale, D. and Shapley, L.S. (1962) "College admissions and the stability of marriage", *The American Mathematical Monthly*, Vol. 69, pp. 9-15.

Scheduling of underground mining processes

Marco Schulze (Speaker) * Jürgen Zimmermann †

1 Introduction

During the last five decades, numerous publications (e.g. [1] and [5]) have appeared concerned with the application of optimization methods in the mining industry. Most of them focus on long-term production scheduling for underground mining, e.g. [7] as well as open pit mining, cf. [6]. In contrast, this paper addresses the short-term underground mine production scheduling problem that can be defined as specifying the sequence in which blocks should be removed from the mine. The aim is to minimize the makespan subject to a variety of constraints, because the management of the mining company is interested in an efficient utilization of the resources. The constraints relate to the mining extraction sequence, resource capacities and safety-related restrictions.

The extraction of the examined German potash mine is done by room-and-pillar mining. In this mining system the mined material is extracted across a horizontal plane while leaving pillars of untouched material to support the roof of the mine. Thus, open areas (rooms) emerge between the pillars. As mining advances, a grid-like pattern of rooms and pillars is formed. There are two types of room-and-pillar mining: conventional mining and continuous mining. Except for some special applications, the excavation of potash is based on the former type involving drilling and blasting. This kind of underground mining is characterized by the following eight consecutive sub-steps (operations), that can be defined as a production cycle (see Fig. 1): scaling the roof, bolting the roof with expansion-shell bolts, drilling large diameter bore holes, removing the drilled material, drilling blast holes, filling the blast holes with an explosive substance, blasting and transportation of the broken material to a crusher.

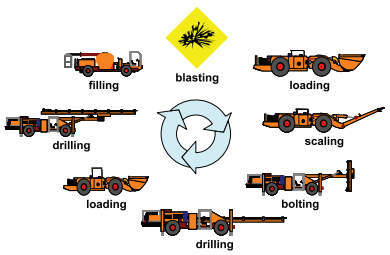


Figure 1: Sub-steps of the production cycle

*marco.schulze@tu-clausthal.de. Operations Research Group, Clausthal University of Technology, Julius-Albert-StraSSe 2, 38678 Clausthal-Zellerfeld, Germany.

†juergen.zimmermann@tu-clausthal.de. Operations Research Group, Clausthal University of Technology, Julius-Albert-StraSSe 2, 38678 Clausthal-Zellerfeld, Germany.

For each processing step except for the blasting step one special mobile machine is required. In order to excavate one block of a certain underground location (for example block 8 at location a2 in Fig. 2) it is necessary that all sub-steps of the preceding production cycle have been finished, e.g. the preceding block (block number 7 in Fig. 2).

After the completion of the first step of the production cycle, the remaining operations ought to be finished within a certain time limit τ. If an operation cannot start within the time limit, a security precaution is needed in which the roof is scaled once more.[1] Then, the next operation of the original cycle can be resumed (see Fig. 3, where τ has elapsed for a job after completion at stage 3).

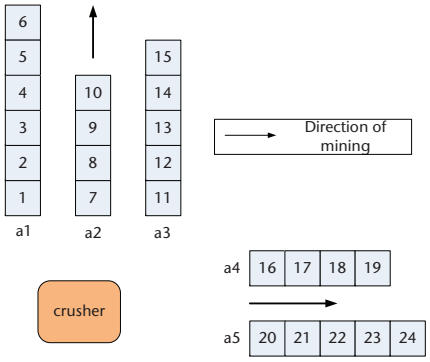


Figure 2: Schematical view of a mining region that consists of five locations (a1-a5), several blocks per location (4-6) and one crusher. For a better orientation the blocks are consecutively numbered.

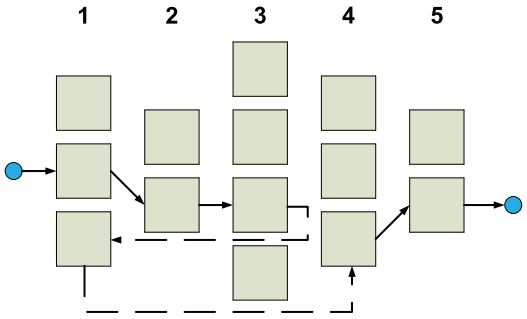


Figure 3: Hybrid flow shop design of the excavation process. The dashed line symbolizes that the time period τ is exceeded. Consequently the job has to visit stage 1 (scaling the roof) once more.

2 A mixed integer linear programming formulation

The described excavation process represents a common manufacturing environment that can be identified as a hybrid flow shop scheduling problem, cf. [2]. The hybrid flow shop problem is a generalization of the classical flow shop problem. There are K production stages, i.e. sub-steps of the production cycle, in series ($K = 7$, because we can ignore the step "blasting"), separated by unlimited intermediate buffers, and each stage k consists of $M^{(k)}$ unrelated parallel machines. The jobs (i.e. blocks) have to visit the stages in the same order starting from stage 1 through stage K. A machine can process at most one job at a time and a job can be processed by at most one machine at a time. Preemption of processing is not allowed. The scheduling problem consists of assigning jobs to machines at each stage and sequencing the jobs assigned to the same machine so that the makespan is minimized. In contrast to this "standard" form of the hybrid flow shop problem, cf. [8], we developed a mathematical model (MIP) for this problem including the specific restrictions associated with the underground mining production cycle.

[1] If jobs may visit each stage several times, it is called re-entry or recirculation, c.f. [4] or [3].

3 Computational Results

In order to test the introduced model we generated fifteen scenarios, each with ten instances, where the number of parallel machines at each stage and the number of underground locations as well as the total number of jobs were varied. When solving the small instances (15, 20 and 25 jobs) we set the time limit to 1,800 seconds and the maximum allowed gap to 0%. The results for these scenarios show something quite interesting: in case where we always have two parallel machines, the computation times are always lower compared to the cases with only one machine at each stage or the configuration with one to three parallel machines at each stage. After analyzing the results in more detail, we could see, that the LP bounds were always better when solving the scenarios with two parallel machines and that might be the reason for the faster results. All small instances could be solved to optimality within acceptable computation time, but to fulfill practical needs we have to consider larger instances with around 15-20 underground locations and approximately 120 jobs that have to be scheduled for an usual mining region. Consequently, we analyzed near practical instances with 60 and 100 jobs each. For these scenarios we set the time limit to 200,000 seconds and the maximum allowed gap to 5% in order to decrease the solution time. In case of 60 jobs solutions within the allowed gap were found in reasonable time for 28 out of 30 instances. Considering the instances with 100 jobs 19 out of 30 instances were solved in acceptable time while for the remaining instances the gap of 5% could not be reached within the time limit. When solving the configurations with only one machine and one to three parallel machines at each stage, we implemented lower bounds that considered the last stage (transportation of the broken material to a crusher) as a bottleneck stage. The results show that our lower bounds lead to significant faster results compared to the lower bounds that were generated by the solver itself in case of two parallel machines.

References

[1] A.M. Newman, E. Rubio, R. Caro, A. Weintraub and K. Eurek (2010). *A review of operations research in mine planning.* Interfaces, vol. 40, no. 3, pp. 222-245.

[2] R. Ruiz and J.A. Vázquez-Rodríguez (2010). *The hybrid flow shop scheduling problem.* European Journal of Operational Research, vol. 205, no. 1, pp. 1-18.

[3] H.-W. Kim and D.-H. Lee (2009). *Heuristic algorithms for re-entrant hybrid flow shop scheduling with unrelated parallel machines.* Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture, vol. 223, no. 4, pp. 433-442.

[4] M.L. Pinedo (2008). *Scheduling: Theory, Algorithms, and Systems.* 3rd edn., Springer, New York.

[5] A. Weintraub, C. Romero, T. Bjørndal and R. Epstein (2007). *Handbook of operations research in natural resources.* Springer, New York.

[6] S. Ramazan and R. Dimitrakopoulos (2004). *Recent applications of operations research and efficient MIP formulations in open pit mining.* Transactions of Society for Mining, Metallurgy, and Exploration, vol. 316, pp. 73-78.

[7] L.P. Trout (1995). *Underground Mine Production Scheduling Using Mixed Integer Programming.* Proceedings of the Application of computers and operations research in the minerals industries, Brisbane, Australia, pp. 395-400.

[8] S.A. Brah (1988). *Scheduling in a flow shop with multiple processors.* PhD Thesis, University of Houston.

Breaks, cuts, and patterns

Dries R. Goossens * Frits C.R. Spieksma (Speaker) †

1 Introduction

Consider a sports competition in which an even number of teams (say $2n$) compete. Each team has its own venue, and each pair of teams meets once in one of the team's venues. Clearly, this is a single round robin tournament which can be played in $2n-1$ rounds. A schedule is called compact when it uses the minimum number of rounds required to schedule all the games. In a compact schedule with an even number of teams, each team plays exactly one game in each round. If the league contains an odd number of teams, a dummy team may be added, reducing this situation to the case with an even number of teams. In each round, the team playing against the dummy team has a 'bye', i.e. does not play. In this work, we deal solely with compact schedules for an even number of teams.

Traditional terminology prescribes that the sequence of home matches ('H') and away matches ('A') played by a single team is called its *home-away pattern* (HAP). Given such a HAP, the occurrence of two consecutive home matches, or two consecutive away matches is called a *break*.

In this contribution, we will generalize the concept of a break. The idea is simple: instead of defining a break as two home games (or two away games) in a pair of consecutive rounds, we will view a break as two home (away) games in a given, *arbitrary* pair of rounds. More specific, each team i, $i = 1, \ldots, 2n$ specifies a set of pairs of rounds indicating that this team does not want to play either at home or away in both rounds of each pair (which need not be consecutive). The set of pairs is called the *break set* of team i, and is denoted by B_i, with $i = 1, \ldots, 2n$. It generalizes the traditional concept of a break. Indeed, the traditional setting arises when:

$$B_1 = B_2 = \ldots = B_{2n} = \{\{1,2\},\{2,3\},\{3,4\},\ldots,\{2n-2,2n-1\}\}.$$

We say that a home-away pattern (HAP) is *break-free* with respect to a break set B_i if no two home matches or two away matches are scheduled on a pair of rounds that is an element of B_i. We call a set of $2n$ home-away patterns a *pattern set* if they are pairwise distinct, and when each round has n H's (and hence n A's). Clearly, these conditions are necessary (but not sufficient!) for the existence of a feasible schedule.

*dries.goossens@econ.kuleuven.be. Operations Research Group, ORSTAT, Faculty of Business and Economics, KULeuven, Naamsestraat 69, B-3000 Leuven, Belgium.

†frits.spieksma@econ.kuleuven.be. Operations Research Group, ORSTAT, Faculty of Business and Economics, KULeuven, Naamsestraat 69, B-3000 Leuven, Belgium.

We call a pattern set *break-free* if the i-th HAP in the set is break-free with respect to B_i. The main problem can now be described as follows: assuming a compact schedule, and given $2n$ break sets B_i, with $i = 1, \ldots, 2n$, does there exist a break-free pattern set with respect to B_i?

In the next section, we motivate the concept of generalized breaks. We mention our results in the last section.

2 Motivation

In most major football leagues, the vast majority of the matches are scheduled on weekend days. However, as there are often more rounds than available weekends, some rounds need to be scheduled on Wednesdays. Since a home game on a Wednesday typically attracts less fans, teams generally do not appreciate a home game on a midweek round. Consequently, teams ask for a schedule where the assignment of home games on Wednesdays is balanced: if a team plays at home on one Wednesday round, they don't want to play at home on the next Wednesday round. Obviously, Wednesday rounds need not be consecutive, and hence generalized breaks arise.

In most sports competitions, the number of consecutive away (home) games is limited (see e.g. Goossens and Spieksma [2]). When at most two consecutive away (home) games are allowed for each team, we can express this condition with the following break set:

$$B_1 = B_2 = \ldots = B_{2n} = \{(1,3), (2,4), (3,5), \ldots, (2n-3, 2n-1)\}.$$

This type of constraint is also relevant for the traveling tournament problem. In this problem, the objective is to minimize the total distance traveled by the teams to complete a (double) round robin tournament. Obviously, combining a number of away games that are geographically close by in a single trip is useful to reduce the travel distance. However, the length of an away trip is usually limited to some value k, which can be expressed using generalized breaks. Indeed, a schedule for $2n$ teams will have at most k consecutive home (away) games if and only if it uses a break-free pattern set with respect to the following break set:

$$B_1 = B_2 = \ldots = B_{2n} = \{(1, k+1), (2, k+2), (3, k+3), \ldots, (2n-1, k+2n-1)\}.$$

3 Our results

We present the following results.

Theorem 1. *Given a break set B_i for each team $i = 1, \ldots, 2n$, deciding whether a break-free pattern set exists is NP-complete.*

Theorem 2. *Given a common break set B, the question whether or not a break-free pattern set exists, can be answered in polynomial time.*

Theorem 3. *Given a common break set B, finding a pattern set that minimizes the number of breaks is NP-hard.*

References

[1] D. de Werra (1981). *Scheduling in Sports*, in: Studies on Graphs and Discrete Programming, Annals of Discrete Mathematics **11**, 381-395.

[2] D.R. Goossens and F.C.R. Spieksma (2010). *Soccer schedules in Europe: an overview*, FBE Research Report KBI1011, K.U.Leuven.

The train positioning problem

Dirk Briskorn (Speaker) * Malte Fliedner †

1 Problem description

We consider a problem arising in the context of shunting yards. In a shunting yard trains carrying containers are positioned on a set of n tracks to be loaded or unloaded. The tracks are arranged such that they run in parallel over the entire yard's width. A set of m cranes positioned next to the tracks operate on all tracks within a specific section of the yard. Often, operations in such a yard are processed in two alternating phases. In the first phase trains that already have been served leave the yard and trains to be served next enter the yard. In the second phase trains are served by cranes loading containers onto trains or unloading containers from trains. The decisions to be made here can be described as follows.

- Trains have to be bundled into subsets of trains to enter the yard at the same time.
- The position of trains within the yard have to be determined.
- Crane operations have to be scheduled.

In this talk we focus on the second decision. The object is minimize the maximum workload among cranes in order to minimize the time needed to serve the current bundle of trains. We make the following simplifying assumptions.

- The crane's effort for serving a car does not depend on the corresponding train's position within the yard, that is the track and the position within the track.
- At most one train will be positioned on each track. Thus, considering the previous assumption we can assume that each train is preassigned to a certain track.
- Each section served by a crane has a width being a multiple of the length of a car. Therefore, we have a discrete set of positions for a car in each section.
- Each car of each train has to be served exactly once.

We can describe the resulting train positioning problem (TPP) more formally as follows. Given a set of trains where each train is specified by its length and an ordered set of crane sections where each crane section is specified by its width find a position for each train such that the maximum number of cars positioned in a crane's section is minimized.

*briskorn@wiso.uni-koeln.de. Seminar für ABWL, Supply Chain Management und Produktion, Universität zu Köln, Albertus Magnus Platz, D-50923 Köln, Germany.

†malte.fliedner@uni-jena.de. Lehrstuhl für Operations Management, Friedrich-Schiller-Universität Jena, Carl-Zeiß-Straße 3, D-07743 Jena, Germany.

In the following we restrict ourselves to the decision problem where we ask whether it is possible to find a positioning such that the maximum workload does not exceed a given number. Note that we can solve the optimization version employing binary seach on the maximum workload by iteratively solving instances of the decision version.

Fig. 1 provides an example with three cranes and four tracks. Each crane's section has width 4. The first car of trains 1, 2, 3, and 4 are positioned in slot 6, 1, 2, and 11, respectively.

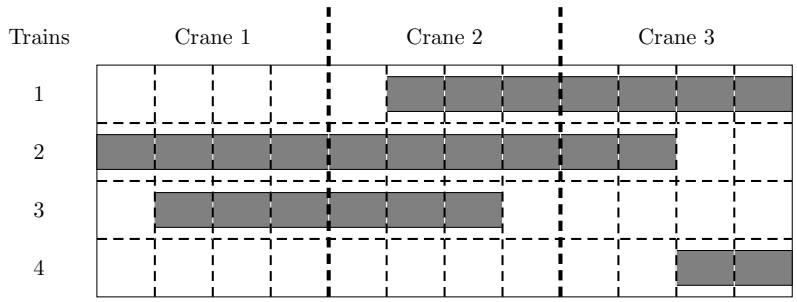


Figure 1: Yard with three cranes and four tracks

2 Results

Theorem 1. *TPP is strongly NP-hard even if all cranes' section have identical width.*

Note that the characteristics of cranes can be encoded by two integers, namely the number of cranes m and the sections' width. Thus, a trivial approach does not suffice to show that TPP with identical cranes' section widths is in NP. Accordingly, we first prove membership to NP. Then, the proof of the theorem is completed by a reduction from 3-PARTITION which is known to be strongly NP-hard, see [1].

Theorem 2. *TPP is ordinarily NP-hard for a fixed number (at least three) of cranes.*

We first show, that TPP is NP-hard even for three cranes having identical sections' widths by reduction from PARTITION which is known to be NP-hard, see [1]. Second, we provide an approach that solves TPP with a fixed number m of cranes in $O\left(mn^{m+1} \cdot l_{\max}\right)$ where $l_{\max}$ is the maximum length among trains.

Theorem 3. *TPP can be solved in* $O\left(m^n\right)$.

We provide an algorithm proving the theorem. Note that its run time complexity is polynomial in input size if the number of trains n is fixed and cranes' section widths are individual, that is we need $\Omega(m)$ integers to encode the problem instance. However, if cranes' section widths are identical, then this algorithm is pseudo-polynomial.

Last but not least we provide a simple 2-approximation algorithm for the optimization version of the problem.

References

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

Online Scheduling of Unit-Length Intervals on Parallel Machines

Stanley P.Y. Fung * Chung Keung Poon (Speaker) †
Duncan K.W. Yung ‡

1 Introduction

We study the online preemptive scheduling of weighted intervals on m identical machines. The input consists of a set of intervals that arrive online and each machine is capable of processing one interval at a time. Our goal is to maximize the total weight of completed intervals. More precisely, each interval I has an *arrival time* $a(I)$, a *deadline* $d(I)$ and a *weight* $w(I)$. To complete interval I, it must be processed on a machine continuously from time $a(I)$ to $d(I)$ without interruption. Before time $a(I)$, the scheduler does not know of I, not even its presence. Upon the arrival of I at time $a(I)$, all the parameters of I become known. Then the scheduler has to decide right away and without knowledge of the future intervals whether to start I and on which machine to process it, preempting the current interval there if any. We define the quantity, $d(I) - a(I)$, as the *processing time* (or *length*) of I. In this paper, we consider unit-length intervals and hence we have $d(I) = a(I) + 1$.

Previous results. When there is only one machine ($m = 1$), Woeginger [5] gave a deterministic 4-competitive algorithm and proved that it is optimal. If randomization is allowed, the best result is a 2-competitive algorithm by Fung et al. [3]. Epstein and Levin [1] gave a lower bound of $1 + \ln(2) \approx 1.693$ for randomized algorithms. For the 2-machine case, Fung et al. [4] presented an algorithm that has competitive ratio approximately 3.582 and gave a lower bound of 2. For the case of general m, Faigle and Nawijn [2] considered variable length but unweighted intervals and gave an optimal 1-competitive algorithm.

Our results. In this paper, we designed an online algorithm for the problem and proved that it is 2-competitive for all even m, and $(2 + \frac{2}{2m-1})$-competitive for all odd $m \geq 3$. Thus the competitive ratio is 2.4 when $m = 3$ and gradually approaches 2 as the number of machines m increases (and remains to be odd).

*pyfung@mcs.le.ac.uk. Department of Computer Science, University of Leicester, Leicester, United Kingdom.

†ckpoon@cs.cityu.edu.hk. Department of Computer Science, City University of Hong Kong, 83 Tat Chee Avenue, Kowloon Tong, Hong Kong.

‡Department of Computer Science, City University of Hong Kong, 83 Tat Chee Avenue, Kowloon Tong, Hong Kong.

2 Even Number of Machines

Suppose we have an even number of machines, i.e., $m = 2q$ for some integer $q \geq 1$. We partition the time axis into slots $s_1, s_2, \ldots$ of unit length. We also divide the machines into two groups, $\mathcal{A}$ and $\mathcal{B}$, each with q machines. For each odd slot s_i, (i.e., when i is odd), machines in group $\mathcal{A}$ will select the q heaviest intervals that arrive within the slot. To achieve this, machines in $\mathcal{A}$ start the first q intervals that arrive in s_i. After that, if a new interval I with heavier weight than any of the q intervals currently being processed arrives within s_i, the interval with the smallest weight will be preempted and I will be started. During the next slot s_{i+1}, the machines in $\mathcal{A}$ will continue the execution of those selected intervals that have not been finished by the end of s_i. After their completion, the machines wait quietly until the end of s_{i+1}, ignoring any new interval that arrives within s_{i+1}. At the same time, B will select the q heaviest intervals that arrives in s_{i+1} using the same method as A used in slot s_i.

It can be shown that the algorithm can always obtain the weight of the top q heaviest intervals arrived in each slot while OPT can obtain at most the top $2q$ heaviest ones. Hence we have the following theorem:

Theorem 1. *Our online algorithm is 2-competitive when m is even.*

3 The Case of Three Machines

If we allocate two machines for the odd slots and one machine for the even slots, then one can prove that the algorithm is 3-competitive using the same argument as in the previous section. A simple example shows that this bound is also tight. To get a smaller competitive ratio, one needs to have a way of sharing the 3 machines between two slots "evenly".

Let the three machines be A, B and C. In slot s_1, machines A and B will take care of intervals that arrive in the slot in a strictly greedy manner: They will start the first two intervals arrived in s_1. Whenever a new interval I arrives within slot s_1 and is of heavier weight than either of the two intervals currently being processed by A and B, the machine processing the lighter interval will preempt its interval and start I. Thus, at any moment A and B will be processing the heaviest and second heaviest intervals arrived in s_1 so far.

Let I_1 and J_1 be respectively the intervals being processed by A and B when slot s_1 ends. Without loss of generality, assume $w(I_1) \geq w(J_1)$. In slot s_2, machine A will complete I_1 and then wait until the end of s_2. Machine B and C will take care of new intervals that arrive in slot s_2. Roughly speaking, B will only abort J_1 when it can start an interval of large enough weight.

We can argue that the online algorithm can always gain the weight of the heaviest interval arrived in each slot. The more tricky (and crucial) part of the analysis is to show that the algorithm can also gain some of the second heaviest.

Theorem 2. *Our online algorithm is 2.4-competitive when $m = 3$.*

4 Extension to General Odd Number of Machines

Suppose we have $m = 2q + 1$ machines, $P_1, P_2, \ldots, P_{2q+1}$. We allocate machines $P_1, \ldots, P_{q+1}$ for slot s_1 to pick up the $q+1$ heaviest intervals arrived in slot s_1. Without loss of generality, assume P_{q+1} is processing the $(q+1)$-st heaviest interval, denoted J_1, arrived in s_1 when slot s_1 ends. Then $P_1, \ldots, P_q$ will run their intervals to completion in slot s_2. Machine P_{q+1} may preempt its interval J_1 when there are large enough intervals while $P_{q+2}, \ldots, P_{2q+1}$ act like machine C in the 3-machine case. We generalize the competitive analysis for the 3-machine case and obtain the followings:

Theorem 3. *Our online algorithm is* $(2 + \frac{2}{2m-1})$*-competitive when* m *is odd.*

References

[1] L. Epstein and A. Levin. Improved randomized results for the interval selection problem. In *16th Annual European Symposium on Algorithms*, volume 5193 of *Lecture Notes in Computer Science*, pages 381–392, September 2008.

[2] U. Faigle and W.M. Nawijn. Greedy k-coverings of interval orders. Technical report, University of Twente, 1991. Tech. Report 979.

[3] S.P.Y. Fung, C.K. Poon, and F. Zheng. Improved randomized online scheduling of unit length intervals and jobs. In *The 6th International Workshop on Approximation and Online Algorithms*, volume 5426 of *Lecture Notes in Computer Science*, pages 53–66, September 2008.

[4] S.P.Y. Fung, C.K. Poon, and F. Zheng. Online interval scheduling: randomized and multiprocessor cases. *Journal of Combinatorial Optimization*, 16(3):248–262, 2008.

[5] Gerhard J. Woeginger. On-line scheduling of jobs with fixed start and end times. *Theoretical Computer Science*, 130:5–16, 1994.

Interval Scheduling on Related Machines: Complexity and Online Algorithms

Clemens Thielen (Speaker) * Sven O. Krumke † Stephan Westphal ‡

1 Introduction

In classical scheduling problems, one is given a number of machines on which finitely many jobs must be scheduled. The task of the scheduler consists of assigning jobs to machines and choosing a starting time for each job on the machine it is assigned to in a way that optimizes some given objective function.

In *interval scheduling problems* (also known as *fixed job scheduling problems* or *k-track assignment problems*), the scheduler is *not* allowed to choose the starting times of the jobs, but each job (or *interval*) has a fixed time at which it must be started. If an interval is not started at the given starting time, it is lost and cannot be scheduled at all. The objective is related to the set of accepted intervals, e.g., maximizing the number of accepted intervals or the sum of their weights. The task of the scheduler consists of choosing which intervals to accept and determining which machines they should be assigned to. Such problems arise naturally in many situations such as the assignment of transports to loading/unloading terminals, bandwidth allocation for communication channels, or work planning for personnel.

We consider interval scheduling on *related machines*, where the machines have different speeds, so an interval finishes earlier when assigned to a faster machine and later when assigned to a slower machine. Problems of this nature arise whenever perishable products or raw materials (e.g., hot metal in a foundry or steel works) have to be processed immediately on arrival, but several facilities of different speed or performance are available to process them. Such differences in the performance of facilities processing the same materials are quite common in large factories (e.g., in steel works) where machines of different generations are often used contemporaneously. Another application is the assignment of emergency patients to doctors in a hospital. The time needed to treat an emergency patient varies depending on the type of injury or disease of the patient, but is also dependent on the level of skill and experience of the doctor the patient is assigned to.

Even though interval scheduling problems on a single machine and on identical machines are well studied in literature (cf., for example, [1–3]), interval scheduling on related machines has not been studied so far.

*thielen@mathematik.uni-kl.de. Department of Mathematics, University of Kaiserslautern, Paul-Ehrlich-Str. 14, D-67663 Kaiserslautern, Germany.

†krumke@mathematik.uni-kl.de. Department of Mathematics, University of Kaiserslautern, Paul-Ehrlich-Str. 14, D-67663 Kaiserslautern, Germany.

‡s.westphal@math.uni-goettingen.de. University of Goettingen, Institute for Numerical and Applied Mathematics, Lotzestr. 16-18, D-37083 Goettingen, Germany

2 Formal Problem Definition

We consider the problem of scheduling intervals online on m related machines. Each interval (or *job*) j arrives at its *release date* $r_j \geq 0$ and must be scheduled immediately on one of the machines to start at time r_j. Intervals that are not scheduled immediately at arrival are lost. Each machine i has a speed $s_i > 0$ at which it runs and which does not depend on the currently processed interval. The speeds of the machines are known in advance, i.e., before the first interval arrives. Each interval j has a *length* (or *processing requirement*) $p_j > 0$ which becomes known at its release date. Processing interval j on machine i needs time p_j/s_i, so the interval finishes at time $r_j + p_j/s_i$ if it is assigned to machine i. An interval that was started on a machine may be aborted before it completes in order to be able to start a new interval. In this case, the aborted interval is lost. Migration of an already started interval to another machine is not allowed. The objective is to maximize the number of accepted intervals (or, equivalently, to minimize the number of lost intervals).

3 Overview of Our Results

Before considering online algorithms for interval scheduling on related machines, we examine the complexity of the offline version of the problem. Our main result in this context is the following theorem:

Theorem 1. *The decision version of interval scheduling on related machines is strongly NP-complete. That is, given m machines with speeds $s_1, \ldots, s_m > 0$, n intervals with release dates $r_1, \ldots, r_n \geq 0$ and processing requirements $p_1, \ldots, p_n > 0$, and an integer $0 \leq l \leq n$, it is NP-complete to decide whether there exists a schedule in which at least l of the intervals are accepted.*

Our NP-hardness proof uses a reduction from 3-satisfiability (3-SAT). Note that NP-completeness of interval scheduling on related machines stands in sharp contrast to the complexity of the problem on identical machines, which is known to be efficiently solvable in polynomial time by using a min-cost flow formulation even when the intervals have weights [4, 5].

For the online version of interval scheduling on related machines, we show a lower bound of 5/3 on the competitive ratio of any deterministic online algorithm:

Theorem 2. *No deterministic online algorithm for interval scheduling on related machines can achieve a competitive ratio smaller than* 5/3.

The proof of this result uses instances with 5 intervals and 3 machines. We suspect that the construction can be generalized to yield a lower bound of $2 - \frac{1}{m}$ on m machines by using $2m - 1$ intervals. This more general bound would be interesting since, by our next result, it would imply that the natural greedy algorithms for interval scheduling on related machines are already best possible. These greedy (online) algorithms are defined by the following two rules:

The Accept Rule:
Always accept an arriving interval j as long as there are still machines available at the release date r_j.

The Finish Early Rule:
If all machines are already busy at the release date r_j of interval j, accept j if and only if there exists a machine i on which the interval $j(i)$ that is processed at time r_j will finish later than at time $r_j + p_j/s_i$. In this case, abort interval $j(i)$ and assign j to machine i.

Our main results on the greedy algorithms defined by the above rules are the following theorems:

Theorem 3. *Every online algorithm that uses the accept rule and the finish early rule is 2-competitive for interval scheduling on related machines.*

Theorem 4. *Every online algorithm for interval scheduling on related machines can be turned into an algorithm that uses the accept rule and the finish early rule and accepts at least the same number of intervals on every instance.*

Besides having a good competitive ratio, the greedy algorithms defined by the above rule are also computationally efficient: Using techniques from computational geometry, we show how one of these 2-competitive algorithms can be implemented to run in $\mathcal{O}(n \log m)$ time. Moreover, we provide experimental results which indicate that the algorithms perform very well in practice: Using up to 10 machines and up to 500 intervals, we see that the average competitivity on any instance size tested was no larger than 1.2 and the running time was less than a second.

Conceptually, our results show that, even though the problem is computationally hard already in the offline version, one can easily to obtain 2-competitive algorithms that also perform very well on average even for the online setting.

References

[1] G. J. Woeginger (1994). *Online Scheduling of Jobs with fixed Start and End Times.* Theoretical Computer Science 130 (1), pages 5-16.

[2] A. W. J. Kolen, J. K. Lenstra, C. H. Papadimitriou, and F. C. R. Spieksma (2007). *Interval Scheduling: A Survey.* Naval Research Logistics 54, pages 530-543.

[3] M. Y. Kovalyov, C.T. Ng, and T. C. E. Cheng (2007). *Fixed Interval Scheduling: Models, Applications, Computational Complexity and Algorithms.* European Journal of Operations Research 178 (2), pages 331-342.

[4] E. M. Arkin and E. B. Siverberg (1987). *Scheduling Jobs with Fixed Start and End Times.* Discrete Applied Mathematics 18 (1), pages 1-8.

[5] K. I. Bouzina and H. Emmons (1996). *Interval Scheduling on Identical Machines.* Journal of Global Optimization 9 (3-4), pages 379-393.

New Lower Bounds for Online Multi-threaded Paging Problem

Denis Trystram * Frédéric Wagner (Speaker) † Haifeng Xu ‡
Guochuan Zhang §

1 Introduction

The setup for classical paging problem is as follows: Given a task set $\mathcal{T}$ as well as a cache, which can hold at most K tasks, we have to serve a sequence of requests, each of which specifies some task, according to the arrival order. The objective function is to minimize the total number of faults. More precisely, when a requested task is not in the cache, load it into the cache, which incurs a fault, and remove some other tasks from the cache, if necessary, so that the total number of tasks stored in the cache doesn't exceed K; otherwise, the request could be served without incurring any cost.

We consider an extension of this problem, namely *multi-threaded paging problem*, which was introduced by Feuerstein and Loma [2]. Instead of one request chain, we need to serve Q (≥ 2) parallel request chains. Specifically, there is no precedence constraint between any two requests which are in different chains. For the online case of this problem, only the first unserved request, if any, is revealed in each chain.

2 New lower bounds for online algorithms

Feuerstein and Loma [2] presented a lower bound for online algorithms: $K+1-\frac{1}{Q}$, which tends to be $K+1$ if Q is large enough. Inspired by their approach, we can improve the lower bound a little bit.

Theorem 1. *The lower bound for any online algorithm is* $\frac{(K+2)Q-2}{Q+1}$.

Proof. We have Q request chains, each of which begins with a request sequence σ^i such that $|\sigma^i| = N$, and $\sigma^i \cap \sigma^j = \emptyset$ $(i \neq j)$. Moreover, we are able to make the processing cost of any online $\mathscr{A}$ is N, while the optimum processing cost is at most $\frac{N}{K}$ ([3]).

- **Stage 1:** For any online $\mathscr{A}$, without loss of generality, we can assume that $\mathscr{A}$ would finish the request chain σ^i before σ^j $(i < j)$.

*denis.trystram@imag.fr . Institut Universitaire de France & INP Grenoble, France.
†frederic.wagner@imag.fr . INP Grenoble, France.
‡xuhaifeng@zju.edu.cn . Department of Mathematics, Zhejiang University, China.
§zgc@zju.edu.cn . College of Computer Science, Zhejiang University, China.

As soon as $\mathscr{A}$ finishes some request sequence σ^i $(2 \le i \le Q)$ for the *first* time, the adversary generates σ^{i-1} again after σ^i. In particular, there is no request arriving for the first chain when $\mathscr{A}$ finishes σ^1.

If some request sequence σ^i in the second column is finished again, and there still exists some other request sequence to be served, then the adversary doesn't output any request for the corresponding chain.

- **Stage 2:** Let σ^{i-1} be the last sequence which is not finished. The adversary just outputs a request sequence, $\sigma = \sigma^Q \cdots \sigma^i \cdot \sigma^{i-2} \cdots \sigma^1$ (σ doesn't consist of σ^{i-1}).

It is easy to check, the optimum schedule just needs to serve the i_{th} chain. So,

$$C_{opt} = (Q+1) \times \frac{N}{K} .$$

No doubt, when $\mathscr{A}$ serves some request chain for the first time, it must incur a cost of N. However, it may incur a cost of $\frac{N}{K}$ when serving some request chain for the second time. Thus,

$$C_{\mathscr{A}} \ge Q \times N + (Q-1) \times \frac{N}{K} + (Q-1) \times \frac{N}{K} .$$

As a result,

$$\frac{C_{\mathscr{A}}}{C_{opt}} \ge \frac{Q(K+2)-1}{Q+1} \to K+2 \quad (Q \to +\infty).$$

□

Theorem 2. *When $K = 1$, the lower bound for any online algorithm is* 3.8.

Proof. When K=1 and Q=2, Alborzi *et al.* [1] presented a lower bound, $\alpha = 1.8$, for any online algorithm. Using the construction mentioned above, we derive a lower bound $\frac{(\alpha+2)Q-4}{Q+2} \to 3.8$ $(Q \to +\infty)$. □

Definition 3. *Given a request sequence σ, we say a sequence $\mathcal{S}(\sigma)$ is a* supersequence *of σ, if we can get $\mathcal{S}(\sigma)$ by inserting some requests into σ.*

Lemma 4. *For any request sequence σ generated from a set consisting of $K+1$ tasks, we can find a supersequence $\mathcal{S}(\sigma)$ of σ such that any online algorithm $\mathscr{A}$ would incur a fault for each request of $\mathcal{S}(\sigma)$, and the optimal cost of $\mathcal{S}(\sigma)$ is* $\min\left\{\frac{|\mathcal{S}(\sigma)|}{K}, |\sigma|\right\}$.

If we can't find such a supersequence, then we can conclude that $\mathscr{A}$ is not a competitive online algorithm.

Proof. We proceed by induction on the length of σ.

Let $\mathcal{S}_1(\sigma) := \sigma_1$. Assuming that we have found a supersequence $\mathcal{S}_i(\sigma)$ of $\boxed{\sigma_1 \cdots \sigma_i}$. Now we argue two cases depending on whether σ_{i+1} is stored in the cache of $\mathscr{A}$ or not.

Case I : σ_{i+1} is not stored in the cache of $\mathscr{A}$:

Thus we concatenate σ_{i+1} to the end of $\mathcal{S}_i(\sigma)$, namely $\mathcal{S}_{i+1}(\sigma) = \mathcal{S}_i(\sigma) \cdot \sigma_{i+1}$.

Case II : σ_{i+1} is stored in the cache of $\mathscr{A}$:

The adversary can always ask for the task which is not in the cache, until $\mathscr{A}$ evicts σ_{i+1} from the cache. Then the adversary generates σ_{i+1}.

If $\mathscr{A}$ never evicts σ_{i+1} for a sufficient long time, we can stop this procedure and claim that $\mathscr{A}$ is not a competitive online algorithm. Because the optimal cost is finite for a request sequence generated from a set consisting of K tasks, but the corresponding cost incurred by $\mathscr{A}$ is infinite.

Next, we estimate the optimal processing cost of $\mathcal{S}(\sigma)$ if $\mathscr{A}$ is competitive.

Since there are exactly $K+1$ different tasks, we get first upper bound directly: $C_{opt}(\mathcal{S}(\sigma)) \leq \frac{|\mathcal{S}(\sigma)|}{K}$.

On the other hand, during the procedure of the second case, all the requests before σ_{i+1} are generated from a set consisting of K tasks, since the cache capacity is K, one of which is occupied by σ_{i+1}. Thus, in order to concatenate σ_{i+1} to $\mathcal{S}_i(\sigma)$, the adversary generates one *phase* at most. Since the number of phases of $\mathcal{S}(\sigma)$ is bound by $|\sigma|$, and OPT pays unit cost for each phase, we have $C_{opt}(\mathcal{S}(\sigma)) \leq |\sigma|$.

As a result,

$$C_{opt}(\mathcal{S}(\sigma)) \leq \min\left\{\frac{|\mathcal{S}(\sigma)|}{K},\ |\sigma|\right\} \leq \frac{K+1}{K+2} \times \frac{|\mathcal{S}(\sigma)|}{K} + \frac{1}{K+2}|\sigma| \ .$$

□

Theorem 5. *When $Q = 2$, the lower bound for any online algorithm is $\frac{K(K+2)}{K+1}$.*

Fix a large integer $\mathcal{N}$ in advance. We have two request chains, σ^1 and σ^2, each of which is generated from two disjoint tasks sets consisting of $K+1$ tasks such that any deterministic online algorithm pay one for all the requests.

Without loss of generality, let σ^1 be the first one that generates $\mathcal{N}$ requests, then it is finished. After σ^2 generates $\mathcal{N}$ requests, we concatenate a supersequence $\mathcal{S}(\sigma^1)$ of σ^1 to the end of σ^2 according to Lemma 4.

Thus, we have:

$$C_{\mathscr{A}} = C_{\mathscr{A}}(\sigma^1) + C_{\mathscr{A}}(\sigma^2) + C_{\mathscr{A}}(\mathcal{S}(\sigma^1)) = 2\mathcal{N} + |\mathcal{S}(\sigma^1)| \ .$$

While the optimal solution just needs to process the second chain:

$$C_{opt} = C_{opt}(\sigma^2) + C_{opt}(\mathcal{S}(\sigma^1)) \leq \frac{K+1}{K(K+2)} \times (2\mathcal{N} + |\mathcal{S}(\sigma)|) \ .$$

So

$$\frac{C_{\mathscr{A}}}{C_{opt}} \geq \frac{K(K+2)}{K+1} \ .$$

References

[1] Alborzi, Torng, Uthaisombut and Wagner. *The k-client problem.* SODA, 1997.

[2] Feuerstein and Loma. *On-line multi-threaded paging.* Algorithmica, 2002.

[3] Sleator and Tarjan. *Amortized efficiency of list update and paging rules.* Communications of the ACM, 1985.

Minimizing predicted air travel delay*

Tanujit Dey † David Phillips ‡ Patrick Steele (Speaker) §

Air traffic delays are a current and growing problem with severe economic and environmental impacts. A recent congressional report has estimated that delays cost $41 billion in 2007 alone and caused an extra 740 million gallons of jet fuel to be burned [4]. Moreover, a 2009 Government Accounting Office report has estimated that the number of flights is going to increase from 50 million in 2008 to 80 million by the year 2025 [8]. We consider the problem of the air traveler determining how to best plan his travel minimizing either total travel time or delay.

Our specific contributions are as follows.

- We formulate the problem of finding a flight plan with minimum delay as a stochastic optimization problem we call the *shortest paths problem with correlated random lengths.* We also show how to extend our formulation to minimize more general functions such as overall flight time and flight costs.
- We develop a statistical model by using an ensemble technique to find significant variables for predicting delay for any given airline. The resulting prediction can then be used in cascade sampling.
- We give an algorithm to determine the relevant flight legs for the given origin and destination. Subsetting to the relevant flight legs allows us to tractably use Monte Carlo simulation in computing the flight plans with the highest probability of small delay.
- We implemented a visualization tool which displays results from our algorithms as well as other statistical analyses on the air transport graph. The visualization of both our algorithmic results and the statistical analyses makes it possible to discern meaningful trends in a large and complicated dataset. The data set used correponds to all U.S. domestic flights from October 1987 to April 2008.

Previous works focus on the scheduling decisions airlines can make to reduce overall delay. For example, [3] solve integer programs that help airline companies determine flight rerouting plans that minimize the delay to passengers when exogenous events (e.g., inclement weather) disrupt the flight schedule. [6] use stochastic optimization techniques to determine air traffic policies that reduce air space congestion. In related papers, [13]

*This work is based on the paper, A Graphical Tool to Visualize Predicted Minimum Delay, that will appear in the *Journal of Computational and Graphical Statistics.*

†`tdey@wm.edu`. Mathematics Department, The College of William & Mary. Supported in part by NSF grant DMS-0703532 and a William & Mary summer grant.

‡`djphil@wm.edu`. Mathematics Department, The College of William & Mary. Supported in part by NSF grant DMS-0703532 and a NASA/VSGC New Investigator grant.

§`prsteele@email.wm.edu`. Mathematics Department, The College of William & Mary. Supported in part by a Chappell fellowship from William & Mary.

and [12] present Markov decision process approaches to scheduling flights accounting for inclement weather.

In contrast, we focus on developing decision support tools that help individual travelers find flight plans that minimize expected delay. In formulating and solving the problem from the individual traveler's perspective our methods can also be used by airlines and air traffic policy makers to determine bottleneck airports with respect to delay.

An essential part of our model is the use of the *airport-flight graph*. We consider a *graph*, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ represents *vertices* and $\mathcal{E}$ represents *edges* between pairs of vertices. By considering the airports as vertices and the flight legs as edges, a graph is a natural way to represent the air travel system (all references cited that consider air travel use variations of this model). A flight plan in such a graph corresponds to a directed path in the network.

1 The stochastic optimization problem formulation

We reduce the problem of finding the flight time with the highest probability of minimum delay to a *shortest paths problem with uncertain edge costs*. For our graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, we assume a set of edge costs, $c_{ij}, \forall (i,j) \in \mathcal{E}$.

For the case of deterministic edge costs and graph structures, the shortest paths problem is well-studied, e.g., see the texts [1] and [5]. Early work on the case where the edge costs were randomized include [7] where the problem was reduced to the deterministic version by considering expected values of each edge costs. For the problem where edge costs were random, but independently generated, algorithms that find edges with probability guarantees include those by Loui [11] and adaptive algorithms by Bertsekas and Tsitsitklis [2] and Fan and Nie [10]. For the case where edge costs were random and correlated but the density functions were known, Fan, et al. [9] recently described an algorithm to determine the path with the highest probability of being less than a certain cost. Our problem differs from these previous results in that our probability density functions are unknown, although we are able to simulate the various edge costs. Moreover, since the delay of a given flight leg causes delays to other flight legs, we cannot assume independence in flight delays.

Recall that $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a graph where $\mathcal{V}$ are vertices representing airports and $\mathcal{E}$ are edges (i.e., pairs of nodes) that represent possible flight legs. We use C_{ij} to denote the random amount of delay on a given flight leg $(i,j) \in \mathcal{E}$. For a given origin, s, and destination, t, we can formulate our problem as follows:

$$\begin{array}{lll} \min & \sum_{(i,j)\in\mathcal{E}} C_{ij} x_{ij} & \\ \text{subject to} & 0 \le x_{ij} \le 1 & \forall (i,j) \in \mathcal{E} \\ \text{(a)} & \sum_{\{j:(i,j)\in\mathcal{E}\}} x_{ij} - \sum_{\{k:(k,i)\in\mathcal{E}\}} x_{ki} = 0 & \forall i \in \mathcal{V} \setminus \{\int, \sqcup\} \\ & \sum_{\{j:(s,j)\in\mathcal{E}\}} x_{sj} = 1 & \\ & \sum_{\{k:(k,t)\in\mathcal{E}\}} x_{kt} = 1 & \end{array} \tag{1}$$

To solve for x^*_{ij} in (1), we adopt a Monte Carlo simulation [14]. For each arc, we randomly generate a sample cost and then we solve a (deterministic) shortest paths problem. By repeating this several times, and counting up the number of times an arc appears on a shortest path, we estimate x_{ij} with a controllable sample.

The key bottleneck in our approach is the sampling required to efficiently simulate the flight delays. To solve this issue, we use a combination of network optimization

and statistical sampling techniques. In our talk, we discuss these key issues as well as computational results.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows : Theory, Algorithms, and Applications*, Prentice Hall, Englewood Cliffs, NJ, 1993.

[2] D. Bertsekas and J. Tsitsiklis, *An analysis of stochastic shortest path problems*, Mathematics of Operations Research, (1991), pp. 580–595.

[3] S. Bratu and C. Barnhart, *Flight operations recovery: New approaches considering passenger recovery*, Journal of Scheduling, 9 (2006), pp. 279–298.

[4] T. J. E. Committee, *Your flight has been delayed again, flight delays cost passengers, airlines and the u.s. economy billions*, congressional report, United States Congress, Washington, DC, 2008.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, The MIT Press and McGraw-Hill, second ed., 2001.

[6] D. Delahaye and A. Odoni, *Airspace congestion smoothing by stochastic optimization*, Lecture Notes in Computer Science, (1997), pp. 163–176.

[7] E. W. Dijkstra, *"A note on two problems in connection with graphs"*, Numerische Mathematik, 1 (1959), pp. 260–271.

[8] G. Dillingham, *Next generation air transportation system, status of transformation and issues associated with midterm implementation of capabilities*, Testimony before the Subcommittee on Aiviation, Committee on Transportation and Infrastructure, House of Representatives GAO-09-479T, United States Government Accountability Office, Washington, DC, 2009.

[9] Y. Fan, R. Kalaba, and J. Moore, *Shortest paths in stochastic networks with correlated link costs*, Computers and Mathematics with Applications, 49 (2005), pp. 1549–1564.

[10] Y. Fan and Y. Nie, *Optimal routing for maximizing the travel time reliability*, Networks and Spatial Economics, 6 (2006), pp. 333–344.

[11] R. Loui, *Optimal paths in graphs with stochastic or multidimensional weights*, Communications of the ACM, (1983).

[12] A. Nilim and L. El Ghaoui, *Robust solutions to markov decision problems with uncertain transition matrices*, Operations Research, 53 (2005), pp. 780–798.

[13] A. Nilim, L. El Ghaoui, and V. Duong, *Multi-Aircraft Routing and Traffic Flow Management under Uncertainty*, in 5th USA/Europe Air Traffic Management Research and Development Seminar, Budapest, Hungary, 2003, pp. 23–27.

[14] J. Spall, *Introduction to stochastic search and optimization: estimation, simulation, and control*, Wiley-Interscience, 2005.

Solving the one-machine scheduling problem with cumulative payoffs

Yasmina Seddik (Speaker) * Christophe Gonzales †
Safia Kedad-Sidhoum ‡

1 Introduction

We address a one machine non-preemptive scheduling problem. N jobs $J_1, \ldots, J_N$ have to be scheduled. Each job J_i has a release date $r_i \geq 0$ and a processing time $p_i > 0$. K delivery dates are given: $D_1, \ldots, D_K$, with $0 < D_1 < \cdots < D_K$.

Given a schedule S, variable V_k represents the number of jobs executed before D_k in S. The payoff of a given schedule S is defined by $v(S) = \sum_{k=1}^{K} V_k$. The payoffs related to the delivery dates are cumulative: jobs delivered at a given delivery date are also counted for each of the subsequent delivery dates. Extending the three-field notation of Graham et al. [2], the problem we address in this paper can be defined as $1|r_i| \sum_{k=1}^{K} V_k$.

We will refer to the payoff related to a job J_i as $v(J_i)$. For a given schedule S, $v(J_i)$ is the number of delivery dates before which J_i is scheduled in S. Let C_i be the completion time of job J_i in S; $v(J_i)$ can be represented by the following non-increasing stepwise function:

$$v(J_i) = \begin{cases} K & \text{if } 0 < C_i \leq D_1 \\ \vdots & \\ 2 & \text{if } D_{K-2} < C_i \leq D_{K-1} \\ 1 & \text{if } D_{K-1} < C_i \leq D_K \\ 0 & \text{if } D_K < C_i \end{cases}$$

In the example of Figure 1, jobs J_1 and J_2 are executed before D_1, the payoff of each of these jobs is thus 3. J_4's completion time occurs after D_1 but before D_2, so J_4's payoff is 2. Finally, J_3's completion time occurs after D_2 but not later than D_3: its payoff is 1. Consequently, the total payoff is $3 + 3 + 2 + 1 = 9$.

Figure 1: Example of a schedule

Problem $1|r_i| \sum_{k=1}^{K} V_k$ has a practical application in the domain of bibliographical digitization where a huge amount of books must be digitized, following a linear process. The books to be digitized become available at different times. In addition, the client sets

*yasmina.seddik@lip6.fr. LIP6 4 place Jussieu 75005 Paris, France.
†christophe.gonzales@lip6.fr. LIP6 4 place Jussieu 75005 Paris, France.
‡safia.kedad-sidhoum@lip6.fr. LIP6 4 place Jussieu 75005 Paris, France.

several delivery dates D_k, $k = 1, \ldots, K$, and target values Q_k, $k = 1, \ldots, K$, indicating the minimal number of books the client wants to be delivered at the corresponding delivery date D_k. Therefore, $Q_1 > 0$ and $Q_k = \alpha_k + \sum_{l=1}^{k-1} Q_l$ with $\alpha_k > 0$, for every $k = 2, \ldots, K$. In addition, the client naturally pays the digitizing company in function of the number of books actually digitized. Consequently, it is desirable to maximize the difference $V_k - Q_k$ for every $k = 1, \ldots, K$. Aggregating these K criteria by summation, we must maximize the objective function $\sum_{k=1}^{K} V_k - Q_k$ or, equivalently, $\max \sum_{k=1}^{K} V_k$ as $\sum_{k=1}^{K} Q_k$ is a constant.

If we do not take into account the cumulative payoffs, our problem is related to the problems with fixed delivery dates studied by Hall et al. [1]. They considered several classical scheduling criteria, including the following variant: the cost of a job J_i depends on the earliest delivery date occurring after the completion of J_i. In addition, they established complexity results for several problems, with different criteria and machine configurations.

As for the cumulative payoffs, Janiak and Krysiak [4] define a criterion which is a generalization of $\sum_{k=1}^{K} V_k$. They attempt to minimize a schedule's cost defined as the sum of the costs of the jobs in the schedule, and each job having its own distinct stepwise cost function. Janiak and Krysiak show that the problem without release date is NP-hard in the general case. They also provide a pseudopolynomial time algorithm for the special case in which the breakpoints are the same for all the job's stepwise functions.

Finally, there exists another class of problems closely related to $1|r_i| \sum_{k=1}^{K} V_k$: the generalized due date problem [3], where due dates are not related to the jobs. Instead, global due dates are defined, and before each of them, one job must be finished. Complexity results have been established for many problems with generalized due dates [3].

2 Complexity results

2.1 General problem

We show that $1|r_i| \sum_{k=1}^{K} V_k$ is unary NP-hard, by reducing the 3-Partition problem to the decision problem corresponding to $1|r_i| \sum_{k=1}^{K} V_k$.

2.2 Two delivery dates problem

The problem with two delivery dates $1|r_i|V_1 + V_2$ is proven to be NP-hard, by reducing to it the Partition problem. More precisely, $1|r_i|V_1 + V_2$ is weakly NP-hard (a pseudopolynomial time algorithm is provided, see Section 3.2).

2.3 Polynomial cases

Four polynomial cases have been identified, which result from relaxations of the general problem $1|r_i| \sum_{k=1}^{K} V_k$.

The problem with no release date $1|| \sum_{k=1}^{K} V_k$ can be optimally solved with the Shortest Processing Time rule (SPT) in time $O(NlogN)$.

The preemptive problem $1|r_i, pmtn| \sum_{k=1}^{K} V_k$ can be optimally solved with the Shortest Remaining Processing Time rule (SRPT) in time $O(NlogN)$.

The problem with identical processing times $1|r_i, p_i = p| \sum_{k=1}^{K} V_k$ can be optimally solved by ordering the jobs following their nondecreasing release dates ($O(NlogN)$).

The single delivery date problem $1|r_i|V$ is equivalent to the problem $1|r_i, d_i = d|\sum U_i$. We provide a polynomial time algorithm for this problem, based on the Moore-Hodgson algorithm ($O(NlogN)$) for problem $1||\sum U_i$ [5].

3 Exact methods

3.1 General problem

In order to solve $1|r_i|\sum_{k=1}^{K} V_k$, we implement a Branch and Bound method. We use the two following lower bounds. First, the problem can be partitioned in K subproblems, each corresponding to a delivery date. Then, the algorithm for $1|r_i|V$ can be iteratively applied to each interval between two consecutive delivery dates (here, and for the following, we set $D_0 = 0$), in order to obtain a feasible solution. Second, the same algorithm can be applied on the whole time-line: jobs can be processed from their release date to D_K, and D_K is the only delivery date taken into account.

The upper bounds derive from the solutions of the relaxed problems, such as no release dates, preemption, identical processing times (in this case, we must choose $p_i = p_{min}$, to be sure to obtain an upper bound).

Finally, the dominance rules are the following.

1. The jobs scheduled between two consecutive delivery dates are ordered following their nondecreasing release dates.

2. A dominance rule for problem $1|r_i|V_1 + V_2$: by applying the algorithm for $1|r_i|V$ on the first delivery date, we obtain the maximum number of jobs that can be processed before D_1, say N_1^M. We show that there exists an optimal solution for $1|r_i|V_1 + V_2$ such that N_1^M jobs are processed before D_1. This dominance rule can be useful in the Branch and Bound method, when the problem is solved until the $(K-2)$-th delivery date.

3.2 Two delivery dates problem

For $1|r_i|V_1 + V_2$ we provide a pseudopolynomial time algorithm, based on a dynamic programming approach, which runs in time $O(N(D_1(D_2)^2) + N \log N)$. The algorithm principally relies on the dominance rule 1. of Section 3.1. Then, the algorithm constructs schedules by adding iteratively jobs before D_1, between D_1 and D_2, or not adding them to the partial schedule.

References

[1] N. G. Hall, 'M. Lesaoana and C. N. Potts (2001). *Scheduling with Fixed Delivery Dates.* Operations Research, 49, 134-144.

[2] R. L. Graham, E. L. Lawler, J. K. Lenstra and A. H. G. Rinnooy Kann (1979). *Optimization and approximation in deterministic sequencing and scheduling: A survey.* Annals of Discrete Mathematics, 6, 287-326.

[3] N. G. Hall, S. P. Sethi and C. Sriskandarajah (1991). *On the complexity of generalized due date scheduling problems.* European Journal of Operational Research, 51, 100-109.

[4] A. Janiak and T. Krysiak (2007). *Single processor scheduling with job values depending on their completion times.* Journal of Scheduling, 10, 129-138.

[5] J. M. Moore (1968). *An n Job, One Machine Sequencing Algorithm for Minimizing the Number of Late Jobs.* Management Science, 15, 102-109.

To the conjecture on the minimum number of migrations in the optimal schedule for the $Pm \,|\, pmtn(delay = d) \,|\, C_{max}$ problem

Alexander Kozlov (Speaker) *

We consider the problem of scheduling independent jobs on parallel machines subject to migration delays so as to minimize the makespan. There are m identical machines $M_1, \ldots, M_m$ which are used to process n jobs, $J_1, \ldots, J_n$. Each job J_j $(j = 1, \ldots, n)$ has a processing time p_j. Each machine can process at most one job at a time, and each job can be processed by at most one machine at a time. Preemption with a migration delay d is allowed, that is, the processing of any job J_j on a machine M_i can be interrupted and resumed at any later time on M_i, and at least d time units later if J_j migrates to another machine M_k. The goal is to schedule the jobs so as to minimize the makespan. By extending the three-field notation [1, 2], this problem was denoted in [4] as $\langle \alpha \,|\, pmtn(delay = d) \,|\, C_{\max} \rangle$, where α is either Pm (for a fixed number m of machines), or P (for an arbitrary number of machines).

In classical scheduling models the migration of a job is done without a delay constraint. For instance, in the book by Tanaev a.o. [3], while considering problems with preemption the following assumption is made: "interruptions cause no time or other penalties". However, in production planning, for example, it is natural to reserve some time for the transition of a product from one machine to another. Simply speaking, the transportation of the product takes time. Besides, technical issues might make it necessary to wait for some time after completing the processing of a job (*e.g.*, a heated product might need to cool down first, or a product needs to be dried before its next operation can start). We refine the classical model of identical machines by adding a delay constraint d, which is independent of the job and the machines.

In [4] it was shown that the sub-problem X_λ of problem $X \doteq \langle P \,|\, pmtn(delay = d) \,|\, C_{\max} \rangle$ with m machines and delays $d \le \lambda(\Delta - p_{\max})$ (where $\Delta = \max\{p_{\max}, L\}$, $p_{\max} = \max_{j=1}^{n} p_j$, and $L = \frac{1}{m} \sum_{j=1}^{n} p_j$) can be solved in linear time for every $\lambda \in [0, 1]$ and is strongly NP-hard for each $\lambda > 1$. They also showed that for the two-machine problem, $\langle P2 \,|\, pmtn(delay = d) \,|\, C_{\max} \rangle$, there always exists an optimal schedule which yields at most one preemption, and for the three-machine problem, $\langle P3 \,|\, pmtn(delay = d) \,|\, C_{\max} \rangle$, there exists an optimal schedule which yields at most two migrations. They proposed the following conjecture.

Conjecture 1. *For any instance of $\langle Pm \,|\, pmtn(delay = d) \,|\, C_{\max} \rangle$ there exists an optimal schedule with at most $m - 1$ migrations.*

* alexandr_kozlov@ngs.ru. Sobolev Institute of Mathematics, 4 Acad. Koptyug avenue, 630090 Novosibirsk, Russia.

This conjecture, be it valid, implies the existence of a pseudo-polynomial time algorithm for the problem with a fixed number of machines. We can prove this conjecture for the case of four machines.

Theorem 2. *For any instance of* $\langle P4 \,|\, pmtn(delay = d) \,|\, C_{\max}\rangle$ *problem there exists an optimal schedule with at most three migrations.*

References

[1] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan (1979). Optimization and approximation in deterministic scheduling: A survey, *Annals of Discrete Mathematics*, **5**, 287–326.

[2] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys (1993). Sequencing and scheduling: Algorithms and complexity, In: Logistics of Production and Inventory. Volume 4 of Handbooks in Operation Research and Management Science. North-Holland, Amsterdam, 445–522.

[3] V.S. Tanaev, V.S. Gordon, and Ja.N. Shaphransky (1984). *Theory of scheduling. Single-stage systems*, M.: Nauka.

[4] Aleksei V. Fishkin, Klaus Jansen, Sergey V. Sevastyanov, and Rene Sitters (2005). Preemptive Scheduling of Independent Jobs on Identical Parallel Machines Subject to Migration Delays, In: "Algorithms — ESA 2005: 13th Annual European Symposium, Palma de Mallorca, Spain, October 3–6, 2005. Proceedings". G.S. Brodal and S. Leonardi (Eds.), *Lecture Notes in Computer Science*, **3669**, 580–591.

Particle swarm optimization algorithm for workforce job rotation scheduling

Hesham K. Alfares (Speaker) * Ahmad A. Zaid †

1 Introduction

In this paper, an efficient particle swarm optimization (PSO) methodology is presented for solving workforce job rotation scheduling problem. The workforce job rotation problem is concerned with assigning and sequencing a set of tasks of varying levels of occupational hazards or physical demands among several employees. Rotating tasks assures fairness and reduces risk, by avoiding the assignment of one or few workers continuously to the most exhausting or dangerous tasks. Workforce job rotation objectives include equal workload, minimum exposure to risky or demanding tasks, and maximum safety and productivity. The integer programming (IP) model of this problem is formulated, but its size is quite large for realistic problems, making optimum IP solutions impractical for real-life problems. Therefore, a PSO heuristic methodology is developed and applied to a number of test problems. The proposed PSO methodology shows promising results, producing near-optimal solution in short computation times.

2 Literature Review

Several approaches have been proposed in the literature for solving the job rotation scheduling problem. Due to the complexity of this problem, optimum solutions using IP (branch-and-bound) are difficult even for problems of moderate size. Therefore, recent research has been mostly focused on heuristic algorithms, especially meta-heuristics such as simulated annealing, ant colony optimization, and genetic algorithms. Seçkiner and Kurt [4] develop a simulated annealing (SA) heuristic for solving job rotation scheduling problems, whose objective is to ensure equal workloads among employees. Later on, Seçkiner and Kurt [5] present two ant colony optimization (ACO) algorithms for solving the same problem, and they perform comparisons that confirm that ACO is competitive with both IP and SA. Diego-Mas et al. [1] apply a multi-criteria genetic algorithm (GA) to construct job rotation schedules that balance multiple objectives. The objectives (criteria) considered in the model are: minimum risk of musculoskeletal disorders, maximum task diversification, worker disabilities, and worker preferences.

*alfares@kfupm.edu.sa. P.O. Box 5067, King Fahd University of Petroleum & Minerals, Dhahran 31261, Saudi Arabia.

†g200903610@kfupm.edu.sa. P.O. Box 5067, King Fahd University of Petroleum & Minerals, Dhahran 31261, Saudi Arabia

Particle swarm optimization (PSO) is a meta-heuristic approach originally proposed by Kennedy and Eberhart [3] for simulating social behavior. Each "particle" is an individual feasible solution that moves around the search-space guided by the fitness (objective) function and by the movements of its neighboring particles. PSO has been applied to several staff scheduling problems, but not to the workforce job rotation problems in particular. Staff scheduling can be defined as the assignment of employees to jobs/machines in each time period to achieve certain objectives while satisfying various constraints. There are several recent publications in the area of staff scheduling using PSO. For example Günther and Nissen [2] use PSO and evolutionary strategies (ES) for sub-daily staff scheduling for the logistics industry. They conclude that while both PSO and ES heuristics outperform manual staff scheduling, but PSO provides the best overall performance. Given this conclusion, the aim of this paper is to apply PSO to workforce job rotations problems.

3 The Integer Programming Model

The integer programming model represents a tour scheduling problem of full-time employees to satisfy varying work demands for 12 work hours per day and seven workdays per week. Binary decision variables X_{nmik} are used to indicate whether or not employee n is assigned to job m in shift k on day i. To achieve fairness, the IP model has a minimax objective, seeking to minimize the maximum employee cost among all employees. The constraints ensure satisfaction of work demands and other work conditions.

The combinatorial structure and binary integer restrictions make the problem is NP-complete and thus makes the optimum solution very difficult. Therefore, the only practical option is efficient solution using heuristic search methods such as particle swarm optimization. Because of this reason, the particle swarm algorithm is used to achieve workload balancing.

4 Particle swarm optimization (PSO) approach

A PSO metaheuristic is used to balance of the workload among workers, which is determined for each job based on the number of days and the type of job. The PSO formulation is based on the traveling salesman problem (TSP). Each order to be processed is represented as a "city" in the TSP network. In the single objective TSP, a matrix D shows the distance between each pair (ij) of cities. In the job rotation scheduling problem, each c_{mik} represents the cost, in terms of workload, of any worker performing job m in shift pattern k on day i. The job rotation scheduling problem model involves the $X_{nmik} = 0-1$ binary variables, while the standard PSO is essentially a real-coded algorithm. Therefore, some revisions are needed to enable it to deal with the binary-coded optimization problem.

5 Computational results

To facilitate comparison, 26 randomly generated test problems developed by Seçkiner and Kurt [4] were used for computational analysis. According to Seçkiner and Kurt [4], no optimal solutions were found by CPLEX solver 8.1 within the limit of 86,400 CPU

seconds, but near optimal solutions were obtained. These IP solutions were used as benchmarks for the proposed PSO algorithm. The results are compared according to objective function value (nearness to optimal solution) for each of the 26 test problems. Comparing PSO with integer programming, results show that the PSO is capable of producing near-optimal solutions in very short computation times. Comparing PSO with simulated annealing, preliminary results are very promising, indicating that PSO is competitive with SA. However, PSO computational times increase considerably as the number of worker increases. Currently, ongoing research is being performed to refine the proposed PSO algorithm in order to improve its efficiency (speed) and effectiveness (solution quality).

References

[1] Diego-Mas J.A., Asensio-Cuesta S., Sanchez-Romero M.A., Artacho-Ramirez M.A. (2009). *A multi-criteria genetic algorithm for the generation of job rotation schedules*. International Journal of Industrial Ergonomics, Volume 39, Issue 1, January 2009, Pages 23-33.

[2] Günther M., Nissen V. (2010). *Sub-daily Staff Scheduling for a Logistics Service Provider*. Künstliche Intelligenz, Volume 24, Issue 2, July 2010, Pages 105-113.

[3] Kennedy J., Eberhart R. (1995). *Particle Swarm Optimization*. Proceedings of IEEE International Conference on Neural Networks, Volume IV, 1995, pp. 1942-1948, IEEE: Piscataway.

[4] Seçkiner S.U., Kurt M. (2007). *A simulated annealing approach to the solution of job rotation scheduling problems*. Applied Mathematics and Computation, Volume 188, Issue 1, May 2007, Pages 31-45.

[5] Seçkiner S.U., Kurt M. (2008). *Ant colony optimization for the job rotation scheduling problem*. Applied Mathematics and Computation, Volume 201, Issues 1-2, July 2008, Pages 149-160.

Optimal robust algorithms for preemptive scheduling

Leah Epstein (Speaker) * Asaf Levin †

1 Introduction

We study preemptive scheduling on m identical machines. A set of n jobs is given, where p_j denotes the processing time of the j-th job. The goal is to assign the jobs to the machines, minimizing the maximum completion time of any machine. We next describe the set of feasible assignments. In preemptive scheduling, each machine can execute one job at each time, and every job can be run on at most one machine at each time. Idle time is allowed. A job is not necessarily assigned to a single machine, but its processing time can be split among machines. The intervals in which it is processed do not necessarily have to be consecutive. That is, a job is split under the constraint that the time intervals, during which it runs on different machines, are disjoint, and the length of their union is exactly the processing time of the job.

In the offline scenario, jobs are given as a set, while in the online problem, jobs arrive one by one to be assigned in this order. The exact time slots allocated to the job must be reserved during the process of assignment, and cannot be modified later. Idle time may be created during the process of assignment, and as a result, the final schedule may contain idle time as well. Note that in this variant, unlike non-preemptive scheduling, idle time can be beneficial, and this is why it is allowed to introduce it.

It is known for a while that the offline problem can be solved optimally, even for more general machine models, and in some of the cases, for other cost functions which are based on machine completion times [9,11,12,12–14]. A number of articles considered the online problem [1–4, 7, 8, 18], including many results where algorithms of optimal competitive ratio were designed. However, it is impossible to design an online algorithm which outputs an optimal solution [1, 16].

There is a recent interest in problems which possess features of both offline and online scenarios. These are not offline problems, in the sense that the input arrives gradually, but they are not purely online either, since some reassignment or reordering of the input is allowed. One such model is the model of robust algorithms or algorithms with bounded migration. Jobs arrive one by one. When a new job arrives, its processing time becomes known. The algorithm needs to maintain a schedule, but when a new job j arrives, it is allowed to change the assignment of its preceding jobs in a very restrictive way. More accurately, the total size of all parts of jobs which are moved to different time slots should be upper bounded by a constant factor, called *the migration factor*, times p_j, that is, at most a constant multiplicative factor away from the size of the new

*lea@math.haifa.ac.il. Department of Mathematics, University of Haifa, 31905 Haifa, Israel.

†levinas@ie.technion.ac.il. Faculty of Industrial Engineering and Management, The Technion, 32000 Haifa, Israel.

job. Algorithms which operate in this scenario are called *robust*. We expect a robust algorithm to be optimal not only for the entire input, but also for every prefix of the input, comparing the partial output to an optimal solution for this partial input. We would like to stress the fact that in the preemptive variant which we study, at each time the parts of a given job may be scheduled to use different time slots on possibly different machines. When the schedule is being modified, we allow to cut parts of jobs further, and only the total size of part of jobs which are moved to a different time slot, or to a different machine (or both), counts towards the migration factor.

Robust algorithms were studied in the past for scheduling and bin packing problems. This model was introduced by Sanders, Sivadasan and Skutella [15], where several simple strategies with a small migration factor, but an improved approximation ratio, compared to online algorithms [10], were presented, as well as a robust polynomial time approximation scheme (PTAS). The bin packing problem, was shown to admit an asymptotic robust PTAS [5] (and even the problem of packing d-dimensional cubes for any $d \geq 2$ into unit cubes of the same dimension admits an asymptotic robust PTAS [6]). On the other hand, not every problem which admits a PTAS admits a robust PTAS [6, 17].

Our main result is an algorithm of migration factor $1 - \frac{1}{m}$ which maintains a strongly optimal solution (a solution whose sorted vector of machine completion times is lexicographically minimal). We show that this result is tight in the sense that no optimal robust algorithm can have a smaller migration factor.

In addition to the basic case of identical machines, we study other, more general, machine models, such as uniformly related machines and identical machines with restricted assignment. We show that an optimal robust solution cannot be maintained in the last two machine models. Note that for identical machines, the study of scheduling with the goal function of the ℓ_p norm of machine completion times results in the same set of (strongly) optimal schedules. This is not necessarily the case for other machine models [9].

2 The approach

Next, we sketch the behavior of our algorithm. At the arrival time of a job, we use a simple algorithm, Loads which computes the sorted vector of machine loads in an optimal solution.

Algorithm Loads is based on the methods of McNaughton [14] and [12]. The algorithm creates a potential schedule of a simple form (which cannot be used by the algorithm, since it needs to modify its existing schedule rather than creating one from scratch). Loads is a recursive algorithm which assigns the largest remaining job to run non-preemptively on an empty machine, unless the remaining set of jobs can be scheduled in a balanced way. Thus, the sorted vector of completion times has a parameter k, where the k machines of smallest completion time have the exact same completion time.

After finding the output of Loads, our algorithm finds which machines have an augmented completion time, and the new job is assigned. There may be a time slot in which the job can be scheduled on one machine, but typically some parts of jobs need to be moved to make room for the parts of the new job. We carefully move parts of jobs. In the process it is necessary to make sure that no parts of a job are scheduled to run in parallel (not only parts of the new job), and that parts of jobs are not moved unnecessarily, to avoid an increase in the migration factor.

References

[1] B. Chen, A. van Vliet, and G. J. Woeginger. An optimal algorithm for preemptive on-line scheduling. *Operations Research Letters*, 18:127–131, 1995.

[2] T. Ebenlendr, W. Jawor, and J. Sgall. Preemptive online scheduling: optimal algorithms for all speeds. *Algorithmica*, 53(4):504–522, 2009.

[3] T. Ebenlendr and J. Sgall. Optimal and online preemptive scheduling on uniformly related machines. *Journal of Scheduling*, 12(5):517–527, 2009.

[4] L. Epstein. Optimal preemptive on-line scheduling on uniform processors with non-decreasing speed ratios. *Operations Research Letters*, 29(2):93–98, 2001.

[5] L. Epstein and A. Levin. A robust APTAS for the classical bin packing problem. *Mathemtical Programming*, 119(1):33–49, 2009.

[6] L. Epstein and A. Levin. Robust approximation schemes for cube packing. 2010.

[7] L. Epstein, J. Noga, S. S. Seiden, J. Sgall, and G. J. Woeginger. Randomized online scheduling on two uniform machines. *Journal of Scheduling*, 4(2):71–92, 2001.

[8] L. Epstein and J. Sgall. A lower bound for on-line scheduling on uniformly related machines. *Operations Research Letters*, 26(1):17–22, 2000.

[9] L. Epstein and T. Tassa. Optimal preemptive scheduling for general target functions. *Journal of Computer and System Sciences*, 72(1):132–162, 2006.

[10] R. Fleischer and M. Wahl. Online scheduling revisited. *Journal of Scheduling*, 3(5):343–353, 2000.

[11] T. F. Gonzales and S. Sahni. Preemptive scheduling of uniform processor systems. *Journal of the ACM*, 25:92–101, 1978.

[12] E. C. Horvath, S. Lam, and R. Sethi. A level algorithm for preemptive scheduling. *Journal of the ACM*, 24(1):32–43, 1977.

[13] E. L. Lawler and J. Labetoulle. On preemptive scheduling of unrelated parallel processors by linear programming. *Journal of the ACM*, 25(4):612Ű–619, 1978.

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

[15] P. Sanders, N. Sivadasan, and M. Skutella. Online scheduling with bounded migration. *Mathematics of Operations Research*, 34(2):481–498, 2009.

[16] J. Sgall. A lower bound for randomized on-line multiprocessor scheduling. *Information Processing Letters*, 63(1):51–55, 1997.

[17] M. Skutella and J. Verschae. A robust PTAS for machine covering and packing. In *Proc. 18th European Symp. on Algorithms (ESA)*, pages 36–47, 2010.

[18] J. Wen and D. Du. Preemptive on-line scheduling for two uniform processors. *Operations Research Letters*, 23:113–116, 1998.

Reassignment models in online scheduling on two related machines

György Dósa (Speaker) * Attila Benkő † Xin Han ‡

1 Rearrangement models

We consider several (semi) online non-preemptive scheduling problems on two related machines. The common thing among the investigated and compared models is that they all allow some kind of reassignment. But they differ from each other in the allowed way of reassignment.

The machines are related, i.e. the machines have speeds, we assume that the first (or slow) machine has speed 1 while the second (or fast) machine has speed $s \geq 1$. Jobs arrive one by one over list, and as soon as a new job arrives, it must be assigned to one of the machines. We are interested in minimizing the makespan, i.e. the maximum completion time of the machines, which equals to the biggest one of the load of the slow machine, and the load of the fast machine divided by it's speed s. Before coming the jobs, an integer K is fixed, it means that how many jobs can be moved, or can be stored temporarily in a buffer.

When $K = 0$ and $s = 1$, this problem degenerates into one of the most fundamental scheduling problems on two machines, online assigning jobs to identical parallel machines to minimize the makespan. It is known that this (offline) scheduling problem is NP-hard.

There are (at least) six different reassignment models what are dealt with in the last some years. These are the next:

1. *Bounded migration* [1]. In case of such problems when a new job comes some already scheduled jobs can be reassigned, here the bound of this reassignment is determined by the sizes, or reassignment cost of the jobs. So this model is quite different from the other reassignment models, where the possibility of some rearrangement is determined by a constant K: At most K jobs, having also some further conditions, can be moved at some point of the scheduling.

2. *Having a (reordering) buffer* [2–5]. In case of such a buffer problem we have a buffer of size K, i.e. at any time of the running at most K jobs can be temporarily stored in the buffer. When a new job comes it can be assigned to one of the machines (then this assignment can not be changed later), or put into the buffer, if there is room for it in the buffer. If the buffer is full (there are already K jobs in the buffer), then the

*dosagy@almos.vein.hu. Department of Mathematics, University of Pannonia, 8200 Veszprém, P.O. Box 158, Hungary.

†bekno.attila@almos.vein.hu. Department of Mathematics, University of Pannonia, 8200 Veszprém, P.O. Box 158, Hungary.

‡hanxin.mail@gmail.com. School of Software, Dalian University of Technology, Road 8, Ecomony and Technology Development Zone, Dalian, P.R. China, 116620.

incoming job must be assigned, or if it is stored, then another job from the buffer must be taken off, and be assigned to some machine. At the end, when no more job comes, the content of the buffer must be also assigned to the machines. We denote this problem as BB (bounded buffer).

3. *Rearrangement of any K jobs at any time when a new job comes* [6] (without knowledge that the sequence is ended or not), denoted by BR for short (bounded rearrangement). This problem BR is a natural relaxation of the buffer problem BB (i.e. theoretically it allows more). We can show this in the next way: In case of the buffer problem at the end, when there is no more job, all jobs being in the buffer can be assigned together to the machines optimally. Thus in the BR problem at any time when a new job comes we can choose exactly that jobs which would be stored in the buffer in case of problem BB, and we temporarily but optimally reassign these chosen jobs. But, in problem BR we can reassign *any* job, while in case of problem BB if a job is taken from the buffer it cannot be put back, so we can not change the assignment of a job what is taken from the buffer.

This means that problem BR is equivalent to such a problem where we have a buffer of size K, and at any moment when a new job comes, any K jobs can be put into the buffer, no matter that the job is just being in the buffer, it is just revealed or it is already assigned to some machine. (At the end when there is no job, the content of the buffer must be also assigned to the machines.) On the other hand, in most cases the condition of problem BR provides better chance to get better schedule (than condition of problem BB) only theoretically. For example in many cases turns out that the biggest K jobs (so far) are worth to be stored in the buffer, so we can not get better competitive ratios for $s \geq 2$, neither for $K \geq 2$, see for details [4, 6].

There are three further problems, defined in [8], where rearrangement can be done only after all jobs are revealed and scheduled. These are the next:

4. The last job *of any machine* can be moved to the other machine.

5. The last K jobs *of the sequence* can be moved.

6. *After the sequence ended any K jobs can be rearranged.* Here first all jobs are (temporarily) scheduled, then we are informed that there is no further job (i.e. the sequence is ended). Then at most K already scheduled jobs can be removed from the schedule, and then all removed jobs must be (re)assigned to the machines. We denote this version as problem BFR (bounded final rearrangement). This problem *seems to be* the relaxation of the similar problem BR, where at any moment the rearrangement is allowed. But on the other hand in case of BFR we are informed about the end of the sequence while in case of BR we are not, so in the latter case any rearrangement must be made in such a way that "it would not be too bad" if the sequence will just end, and we are not allowed to make further change at all, but it also must be quite good if the sequence follows [7].

2 Results

We will mainly concentrate on three versions, the buffer problem BB, the rearrangement model BR, and the final rearrangement problem BFR. With respect to the best possible worst case ratio, we obtain that

i) For $s \geq 2$, all three models BB, BR, and BFR are equivalent, they all allow to get $C(s) = \frac{s+2}{s+1}$ as optimal competitive ratio, and $K = 1$ is enough to get this optimal ratio

(we get no advantage having $K > 1$, but with $K = 0$ we can get only weaker competitive ratio.)

ii) For $1 \leq s \leq 2$, for all three problems $K = 2$ is enough to get the best competitive ratio

$$C(s) = \begin{cases} \frac{(s+1)^2}{s^2+s+1} & 1 \leq s \leq \frac{1+\sqrt{5}}{2} \\ \frac{s^2}{s^2-s+1} & \frac{1+\sqrt{5}}{2} \leq s \leq 2 \end{cases}$$

what can be achieved by arbitrary big value of K.

iii) By getting $1 \leq s \leq 2$ and $K = 1$, all three problems seems to be more difficult: it is very hard to obtain the best possible competitive ratios. But we still can establish some results: there exists at least some subinterval of $1 \leq s \leq 2$, where using $K = 1$ does not allow to get as good competitive ratio than the help of $K = 2$. To get the optimal competitive ratios remains open, at least for some subinterval of $[1, 2]$.

References

[1] N. Sivadasan, P. Sanders, M. Skutella, Online scheduling with bounded migration, Math. Oper.Res. 34 (2) (2009) 481-498.

[2] H. Kellerer, V. Kotov, M. G. Speranza, Z. Tuza, Semi on-line algorithms for the partition problem. Operations Research Letters, 21 (1997) 235-242.

[3] G. C. Zhang, A simple semi-online algorithm for $P2||C_{\max}$ with a buffer. Information Processing Letters, 61 (1997), 145-148.

[4] G. Dosa, L. Epstein, Preemptive online scheduling with reordering, SIAM Journal on Discrete Mathematics, Volume 25, Issue 1, pp. 21-49 (2011)

[5] G. Dosa, L. Epstein, Online scheduling with a buffer on related machines, J.Comb.Optim. 20(2) 2010, 161-179.

[6] G. Dosa, Y. Wang, X. Han, H. Guo, Online scheduling with rearrangement on two related machines, Theoretical Computer Science, 412(8-10): 642-653 (2011)

[7] A. Benko, X. Chen, G. Dosa, X. Han, Online scheduling with bounded rearrangement at the end, to be submitted, 2011.

[8] Z. Tan, S. Yu, Online scheduling with reassignment, Oper.Res.Lett. 36(2) 2008, 250-254.

Author Index

A
Aggoune, 137
Akker, 73
Alfares, 250
Almgren, 171
Anand, 119
Andersson, 198
Antoniadis, 32
Artigues , 156
Ásgeirsson, 153
Atkin, 61
Azar, 1
Azem, 137

B
Bansal, 46
Barketau, 145
Barták, 8
Baruah, 122
Bäumelt, 70
Benkő, 256
Berlińska, 162
Beygang, 88
Bini, 103
Biondi, 201
Bludau, 140
Bonifaci, 20, 122, 204
Borndörfer, 219
Borndörfer, 17
Bougeret, 125
Briand, 156
Briskorn, 231
Brunsch, 38
Burke, 61

C
Čapek, 180
Carrasco, 116
Chang, 131
Chen, 52
Cichenski, 97
Coene, 91
Cole, 29
Correa, 29
Cucu-Grosjean, 110

D
Dahms, 88
D'Angelo, 122, 204
Dauzère-Pérès, 137
Davis, 195
Dey, 242
Dobbs, 192
Dokka, 216
Dósa, 256
Drozdowski, 162
Dugarzhapov, 174
Dutot, 125

E
Ebenlendr, 82
Epstein, 148, 183, 253

F
Fang, 168
Fliedner, 231
Fırat, 134, 222
Fung, 233

G
Gabow, 131
Garaix, 156
Garg, 119
George, 100, 106
Geraerts, 73
Gkatzelis, 29
Gonzales, 245
Goossens, 228
Grigoriev, 189
Guinand, 210
Günther, 23

H
Han, 256
Hanen, 165
Hanzálek, 70
Hanzálek, 180
Harjunkoski, 201
Heinz, 67
Hermant, 106
Hoeksma, 151
Hoogeveen, 73
Hüffner, 32
Hurkens, 134, 222

I
Imreh, 64
Iyengar, 116

J
Jansen, 186
Jarus, 97
Jeż, 85

K
Kedad-Sidhoum, 245
Khuller, 131
Kleiman, 148
Kononov, 174, 189
Kovalyov, 145
Kozlov, 248
Krumke, 88, 236
Küfer, 140
Kuhn, 49

L
Lenté, 76
Lenzner, 32
Levin, 183, 253
Li, 122, 159
Liedloff, 76
Lin, 177
Lübbecke, 23

M
Machowiak, 145
Mahjoub, 210
Marbán, 113
Marchetti-Spaccamela, 20, 122, 204
Marouf, 100
Mastrolilli, 49
Megow, 119, 122
Mirrokni, 29
Mitra, 153
Möhring, 23
Moldenhauer, 32
Mourtos, 216

N
Németh, 64
Nonner, 35
Nowicki, 192

O
Olver, 29
Oulamara, 207

P
Patriksson, 171
Pawlak, 97
Phillips, 242
Poon, 233
Prädel, 186
Prins, 73
Pruhs, 26, 46

R
Ravizza, 61
Rudová, 213
Rutten, 38, 113

S
Sadykov, 55
Sagnol, 219
Saliba, 201
Sanlaville, 210
Schranzhofer, 52
Schulz, 67
Schulze, 225
Schwarz, 186
Seddik, 245
Sevastyanov, 177
Sgall, 82
Shakhlevich, 41
Shmoys, 15
Sitters, 44
Sorel, 100
Soukhal, 76, 207
Souza, 32
Spieksma, 91, 216, 228

Steele, 242
Stein, 26, 116
Ster, van der, 204
Stougie, 122, 204
Strömberg, 171
Strusevich, 79
Šůcha, 70, 143, 165, 180
Sutherland, 168
Svensson, 186
Sviridenko, 189, 192
Swarat, 219
Swirszcz, 192

T
Thiele, 52
Thielen, 236
Thierry, 106
Thörnblad, 171
T'kindt, 76
Troubil, 213
Trystram, 125, 239

U
Uetz, 151
Uhan, 168

V
Vanderbeck, 55
Verschae , 128
Vredeveld, 38, 113

W
Wagner, 239
Walkowski, 97
Węglarz, 145
Wein, 159
Westphal, 236
Wiese, 128
Woeginger, 134
Wong, 58

X
Xu, 58, 239

Y
Yu, 58, 94
Yung, 233

Z
Zaid, 250
Zajíček, 143
Zhang, 94, 239
Zhao, 168
Zimmermann, 225
Zwart, 4