Adam Mickiewicz
University
Poznań

THE THIRD INTERNATIONAL WORKSHOP ON DYNAMIC SCHEDULING PROBLEMS

ADAM MICKIEWICZ UNIVERSITY, POZNAŃ
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
JULY 5TH– 6TH, 2021, POZNAŃ, POLAND

EXTENDED ABSTRACTS

IWDSP
POZNAŃ 2021

This book contains extended abstracts of a plenary lecture and papers presented at the Third International Workshop on Dynamic Scheduling Problems, July 5th–6th, 2021, Poznań, Poland.

Layout and cover design by Bartłomiej Przybylski
Edited by Stanisław Gawiejnowicz

ISBN: 978-83-951298-8-9 (printed version)
ISBN: 978-83-951298-6-5 (eBook)
ISBN: 978-83-951298-9-6 (printed version + eBook)

Publisher:

Polish Mathematical Society
Śniadeckich 8, 00-956 Warsaw
Poland

Welcome to IWDSP 2021

Dear participant,

on behalf of the Programme and Local Committees, I am pleased to welcome you to IWDSP 2021, the Third International Workshop on Dynamic Scheduling Problems, and to the Faculty of Mathematics and Computer Science, Adam Mickiewicz University in Poznań, which is the host of this event.

The IWDSP 2021 workshop is the third event in the series started in 2016 with IWDSP 2016 and continued two years later with IWDSP 2018. Both the workshops attracted the authors from Australia, Belarus, Belgium, Canada, France, Germany, Israel, Poland, P. R. China, Russian Federation, Taiwan and United Kingdom. Books of extended abstracts of the papers presented by the authors on the workshops are available at the Web sites `https://iwdsp2016.wmi.amu.edu.pl` and `https://iwdsp2018.wmi.amu.edu.pl`, respectively. Selected revised papers presented at the IWDSP 2018 workshop have also been published in issue no. 6 of volume 23 of the Journal of Scheduling, see `https://link.springer.com/journal/10951/volumes-and-issues/23-6` for details.

The present workshop was originally planned to be held in 2020 but the outbreak of the COVID-19 pandemic postponed it for a year. In view of the uncertainty caused by the pandemic, the workshop is planned as an online event.

IWDSP 2021, similarly as the previous two workshops in the series, focuses on dynamic scheduling problems defined by parameters whose values are varying in time. Problems of this kind appear in many applications. The most common examples are scheduling problems with time-, position- and resource-dependent job processing times. The aim of this workshop is to present the recent research in this important domain of scheduling theory.

The Program Committee, supported by the members of the Advisory Committee and external reviewers, selected for presentation at IWDSP 2021 papers submitted by the authors from Egypt, Germany, India, Israel, Poland, P. R. China and the United States. These papers, together with a plenary lecture on dynamic opponent choice in sport tournaments, allowed the Program Committee to prepare an attractive scientific program of the event.

I wish you a fruitful workshop, expressing the hope that you will find IWDSP 2021 stimulating for your further research.

Stanisław Gawiejnowicz
The Chair of the Program Committee
The Chair of the Local Committee

Committees

Committees

Program Committee

Stanisław GAWIEJNOWICZ (Chair), Adam Mickiewicz University Poznań, Poznań, Poland

Gur MOSHEIOV, Hebrew University of Jerusalem, Jerusalem, Israel

Advisory Committee

Alessandro AGNETIS, University of Siena, Siena, Italy

Evripidis BAMPIS, Sorbonne University, Paris, France

Hans KELLERER, University of Graz, Graz, Austria

Tamás KIS, Institute for Computer Science and Control, Budapest, Hungary

Alexander V. KONONOV, Sobolev Institute of Mathematics, Novosibirsk, Russian Federation

Mikhail Y. KOVALYOV, United Institute of Informatics Problems, Minsk, Belarus

Minming LI, City University of Hong Kong, Hong Kong, P. R. China

Bertrand M-T. LIN, National Chiao Tung University, Hsinchu, Taiwan

Nicole MEGOW, University of Bremen, Bremen, Germany

Michael L. PINEDO, Stern School of Business, New York University, New York, USA

Dvir SHABTAY, Ben-Gurion University of the Negev, Beer Sheva, Israel

Local Committee

Joanna BERLIŃSKA, Adam Mickiewicz University Poznań

Stanisław GAWIEJNOWICZ (Chair), Adam Mickiewicz University Poznań

Bartłomiej PRZYBYLSKI, Adam Mickiewicz University Poznań

Marcin ŻUROWSKI, Adam Mickiewicz University Poznań

Contents

Welcome to IWDSP 2021 3

Committees 7

Programme 13

Plenary lecture 17

Dynamic opponent choice in tournaments
Nicholas G. Hall 17

Extended abstracts 27

Scheduling in data gathering networks with variable communication speed and a processing stage
Joanna Berlińska, Baruch Mor 27

New results in time-dependent open shop scheduling
Stanisław Gawiejnowicz, Marta Kolińska 33

On the complexity of conditional DAG scheduling
Alberto Marchetti-Spaccamela, Nicole Megow, Jens Schlöter, Martin Skutella, Leen Stougie 39

Single machine scheduling with step-learning
Baruch Mor, Gur Mosheiov 47

Consideration of routine losses due to intermittent production in flow shop scheduling
Frederik Ostermeier 51

Dual-criticality scheduling on non-preemptive, dynamic processors using RL agents
Nourhan Sakr, Youssef Hussein, Karim Farid 57

A lower bound for sequentially placing boxes at the moving assembly line to minimize walking time
Helmut A. Sedding 63

Heuristic algorithms for solving hard scheduling problems with positional penalties and controllable processing times
Dvir Shabtay, Baruch Mor, Liron Yedidsion 71

Relative robust total completion time scheduling problem on a single machine
Prabha Sharma, Diptesh Ghosh, Sandeep Singh 77

An exact algorithm for a two-machine time-dependent scheduling problem
Weronika Skowrońska, Stanisław Gawiejnowicz 83

Two-agent scheduling with position-dependent processing times and job rejection
Xiaowen Song, Cuixia Miao, Xiaoxu Song 89

Scheduling with periodic availability constraints to minimize makespan
Lishi Yu, Zhiyi Tan 93

Indexes **101**

Programme

Monday, July 5th, 2021

08:30 – 09:00	Opening
09:00 – 10:20	**Session no. 1**
	Speakers: Dvir Shabtay, Joanna Berlińska
	Chair: Gur Mosheiov
10:20 – 10:40	Coffee break
10:40 – 12:00	**Session no. 2**
	Speakers: Lishi Yu, Jens Schlötter
	Chair: Stanisław Gawiejnowicz
12:00 – 14:00	Lunch break
14:00 – 15:20	**Session no. 3**
	Speakers: Cuixia Miao, Helmut A. Sedding
	Chair: Prabha Sharma
15:20 – 15:40	Coffee break
15:40 – 17:10	**Plenary lecture**
	Speaker: Nicholas G. Hall
	Chair: Stanisław Gawiejnowicz

Tuesday, July 6th, 2021

09:00 – 10:20	**Session no. 4**
	Speakers: Gur Mosheiov, Weronika Skowrońska
	Chair: Stanisław Gawiejnowicz
10:20 – 10:40	Coffee break
10:40 – 12:00	**Session no. 5**
	Speakers: Stanisław Gawiejnowicz, Prabha Sharma
	Chair: Gur Mosheiov
12:00 – 14:00	Lunch break
14:00 – 15:20	**Session no. 6**
	Speakers: Frederik Ostermeier, Nourhan Sakr
	Chair: Dvir Shabtay
15:20 – 15:40	Coffee break
15:40 – 16:00	Closing

Plenary lecture

The Third International Workshop on Dynamic Scheduling Problems
Adam Mickiewicz University, Poznań, July 5th – 6th, 2021
DOI: 10.14708/isbn.978-83-951298-6-5p17-24

Dynamic opponent choice in tournaments[*]

Nicholas G. Hall[**]
Fisher College of Business, The Ohio State University, USA

Keywords: multiple round sports tournament, choice of opponent, performance criteria, professional tennis data

1 Introduction

We consider multiple round tournaments, as used extensively to organize sports events worldwide. Since our work can be applied to both individual and team tournaments, we use the term *player* to describe either an individual player or a team. Many multiple round tournaments consist of a preliminary or group stage, followed by several rounds of single elimination play using a fixed seeding or *bracket*. This design is used by, for example, most U.S. major sports, the FIFA World Football Cup, and the ICC World Cricket Cup. The objectives of the designers of such tournaments include: providing the players with equally fair opportunities, motivating them to perform well, selecting among them, and providing an appealing event for spectators.

In this lecture, we discuss a few deficiencies that arise in such tournaments with respect to these objectives. First, top ranked players randomly incur unfortunate match-ups against other players, which introduces an unnecessary element of luck into the tournament. We believe that the achievement of a top ranking at the preliminary stage should not disadvantage a player based on chance. Second, as documented in the tournament design literature discussed below, various reasonable criteria such as stronger ranked players having a higher probability of winning, are not satisfied. Third, the probability that the top two players meet is not maximized. Fourth, there is the widely observed issue of *shirking* or *tanking* at the preliminary stage, where a player deliberately avoids winning a game, in order to obtain an easier path through the tournament. This occurs where a preliminary stage match can be lost, but the player continues to the next round anyway. This creates situations where a player has an incentive to shirk, in order to play easier opponents at later rounds. Finally, the use of a conventional fixed bracket fails to allow players to consider information that develops during the tournament, such as injuries to other players. The conventional fixed bracket design does not allow for this developing information to be used.

[*]Joint work with Zhixin Liu (College of Business, University of Michigan – Dearborn, USA)
[**]Speaker, e-mail: hall.33@osu.edu

To address these deficiencies of conventional tournament design, we propose a new design under which the players, in ranked order that is at least initially determined at the preliminary stage, *choose their next opponent at each single elimination round*. This choice is made under the two conditions: (i) the opponent has not been previously chosen by a higher seeded player, and (ii) the player has itself not been previously chosen. This opponent choice design allows considerable flexibility in implementation. For example, we study the performance of two versions: under *static ranking*, the ranking of the players when they entered the single elimination stage of the tournament remains fixed; whereas under *dynamic ranking*, a lower ranked player inherits the ranking of a higher ranked player which it beats. Flexibility is also available regarding the number of player(s) from the top of the ranking which are allowed to choose their opponent(s).

Similar solutions are used in contemporary tournament practice. Several tournaments currently include opportunities for opponent choice: The EBEL Austrian Ice Hockey League, The Southern Professional Hockey League (U.S.), The U.S. Bridge Federation, The PRO Chess League. In addition, Inside Hook [5] provides details of a preliminary proposal to expand the U.S. Major League Baseball playoffs to include additional teams, and permit higher ranked teams to choose their opponents.

2 Related literature review

For any tournament, the positioning of the variously ranked players allows for alternative bracket designs. This problem of bracket design, or *seeding problem*, has been extensively discussed in the literature. Below, we briefly review the main related works.

Glenn [1] compares all possible brackets with four players, to evaluate the probability that each player will win, and the expected number of games. Searls [6] extends this analysis to eight players by comparing single and double elimination tournament designs, under single game and best-of-three-games formats. He finds that double elimination with best-of-three-games format provides the highest probability for the best player to win and the highest expected number of games.

Horen and Riezman [4] compare different brackets for multiple round single elimination tournaments where the pairwise win probability matrix satisfies properties that enable the players to be ranked in a natural way, which we term a *medium ranking*. They apply four fairness criteria: (i) does the bracket maximize the probability that the best player wins the tournament? (ii) is it order preserving, i.e., no stronger player has a lower probability of winning the tournament than a weaker player? (iii) does it maximize the probability that the best two teams meet in the final, and (iv) does it maximize the expected value of the winning team? For tournaments with four players, there are only three distinct brackets, and the conventional matchup of the strongest and weakest players at the semifinal round is the unique one that satisfies criteria (i) – (iv). For tournaments with eight players, there are 315 distinct brackets. Criterion (i) is satisfied by eight of them, but for criterion (ii) no bracket is satisfactory.

Vu and Shoham [8] make the stronger assumption that, for any pair of players, player P_i has a higher probability than player P_j of beating every other player. We denote this situation as the existence of a *strong ranking*. They show that, for a tournament with eight or more players, no bracket tournament design can guarantee this criterion for an arbitrary win probability matrix.

Vong [7] considers the problem of strategic manipulation in multiple round tournaments. His definition of strategic manipulation is more general than shirking. Specifically, a tournament that is free of strategic manipulation is characterized as one where (i) full effort exists as a subgame perfect equilibrium, and (ii) each such equilibrium coincides with the outcome from a full-effort social choice function. It is shown that allowing only the top-ranked player from each group to advance to the next round of the tournament is both necessary and sufficient to achieve these properties, under an arbitrary sorting rule that sorts qualifying players from one stage to the next. However, as Vong [7] notes, eliminating all but one player from every group would quickly reduce interest in the tournament and disappoint spectators who have traveled far to watch the event, as at the FIFA World Football Cup.

Guyon [2] proposes the use of a *global ranking* at the elimination stage to remove group advantage, for tournaments where the number of groups is not a power of 2. Under this ranking, all teams that are qualified for the elimination stage are ranked one after another based on performance at the group stage. The proposed model is used by UEFA to modify the elimination bracket in the 2020 UEFA Euro Football Championship to minimize group advantage.

Guyon [3] proposes a tournament design where the players are ranked based on their earlier performance. Then, in ranked order, the surviving players choose their opponents at the next round. He also considers a variant in which players in the top half of the ranking cannot be chosen by others. Within the context of a football tournament, the ranking is established first by points and then in the event of ties, by goal difference. Assuming that the *tournament designer* has the objective of maximizing the number of games that take place within the home country of a player, he applies three variations of this design to data for the 2020 UEFA Euro Football Championship.

The work presented in this lecture differs from that of Guyon [3] in several ways. First, we consider dynamic rankings based on beating higher ranked players during the single elimination stage. Second, we consider various levels of information about pairwise win probabilities. Third, we make the natural assumption that *each player* chooses its opponents to *maximize its tournament win probability*. Fourth, we evaluate the results of the opponent choice design against the three reasonableness criteria described by Horen and Riezman [4]. Finally, we establish anti-shirking results that are maximal, relative to the result of Vong [7].

3 Opponent choice algorithm

We consider a tournament consisting of a preliminary stage, followed by a single elimination stage. The opponent choice is made using the following two rules:

1. Players compete in a preliminary stage, possibly lasting up to an entire season, that establishes a weak ranking among them.

2. Players compete in multiple rounds of single elimination play, by first choosing their opponent in weakly ranked order at each round.

Our algorithm for finding the optimal sequence of opponent choices in a tournament, *Algorithm Opponent Choice*, may be formulated as follows. Let $\mathcal{S}_a$ denote the set of all the players. For a given set $\mathcal{S}$ of players with relative rankings $1, \ldots, |\mathcal{S}|$ and respective original rankings $(1), \ldots, (|\mathcal{S}|)$, let $Q(\mathcal{S})$ denote the corresponding tournament win probabilities of the players, given all players' optimal choices of opponents. Under the notation, at the input of our algorithm are given probabilities p_{ij} for $i, j = 1, \ldots, N$, $N = 2^n$, $i \neq j$. Value function $Q(\mathcal{S}) \equiv \{q_{\mathcal{S},(1)}, \ldots, q_{\mathcal{S},(|\mathcal{S}|)}\}$, and optimal solution value is $Q(\mathcal{S}_a)$.

When $|\mathcal{S}| = 2$, i.e, only two players with relative rankings 1 and 2, we have $q_{\mathcal{S},(1)} = p_{(1)(2)}$ and $q_{\mathcal{S},(2)} = p_{(2)(1)}$, where (1) and (2) are the two players' original rankings. Hence we assume that, for any $\mathcal{S} \in \mathcal{S}_a$, $|\mathcal{S}| = 2, 4, \ldots, N/2$, we have found $Q(\mathcal{S}) = \{q_{\mathcal{S},(1)}, \ldots, q_{\mathcal{S},(|\mathcal{S}|)}\}$.

Further, when $\mathcal{S} = \mathcal{S}_a$ and the opponent of each player is decided, there are $2^{N/2}$ possible sets of the $N/2$ winners. Define ordered sets $\mathbf{X}$ and $\mathbf{X}'$ such that players in set $\mathbf{X}$ play against players in set $\mathbf{X}'$ in the first round, in the same order as in their respective sets, where $|\mathbf{X}| = |\mathbf{X}'| = N/2$.

Let $\Omega(\mathbf{X}, \mathbf{X}')$ denote the collection of all possible sets of winners, and $p_{\mathcal{S}}$ denote the probability of the occurrence of set $\mathcal{S} \in \Omega(\mathbf{X}, \mathbf{X}')$. Then, the tournament win probability $q_j = q_{\mathcal{S}_a,j}$ of player P_j, $j = 1, \ldots, N$, is:

$$q_{\mathcal{S}_a,j} = \sum_{\mathcal{S} \in \Omega(\mathbf{X},\mathbf{X}')} p_{\mathcal{S}}\, q_{\mathcal{S},j}. \tag{1}$$

At any round, the algorithm works by evaluating all subsets of smaller size before all subsets of larger size. When $N - 2$ players have their opponents already decided and two players P_i and P_j, where $i < j$, do not, player P_i will play player P_j as its opponent, for any $1 \leq i < j \leq N$. Next, the algorithm evaluates all subsets of four players, using the previously computed information for any pair of last two players, and so on, until the optimal choices of all N players and their win probabilities are determined. As in the boundary condition, we define ordered sets $\mathbf{X}$ and $\mathbf{X}'$ with $|\mathbf{X}| = |\mathbf{X}'|$, such that players in set $\mathbf{X}$ play against players in set $\mathbf{X}'$, in the same order as in their respective sets. By assumption, the collection of all possible sets of winners $\Omega(\mathbf{X}, \mathbf{X}')$ is known for $|\mathbf{X}| = |\mathbf{X}'| \geq m$. Next, we assume that $2m - 2$ players have their opponents decided in ordered sets $\mathbf{X}$ and $\mathbf{X}'$ with $|\mathbf{X}| = |\mathbf{X}'| = m - 1$.

Let set $\mathcal{U}$ contain all the players with undecided opponents, where player P_j has the highest ranking in $\mathcal{U}$ and $\mathbf{X} \cup \mathbf{X}' \cup \mathcal{U} = \mathcal{S}_a$. Then, player P_j chooses its opponent to maximize its tournament win probability $q_j = q_{\mathcal{S}_a,j}$, as follows:

$$q_{\mathcal{S}_a,j} = \max_{i \in \mathcal{U}} \sum_{\mathcal{S} \in \Omega(\mathbf{X} \cup \{j\}, \mathbf{X}' \cup \{i\})} p_{\mathcal{S}}\, q_{\mathcal{S},j}, \tag{2}$$

where $\Omega(\mathbf{X} \cup \{j\}, \mathbf{X}' \cup \{i\})$ is known by assumption in view of equalities $|\mathbf{X} \cup \{j\}| = |\mathbf{X}' \cup \{j\}| = m$.

During the lecture, we will show that the following result holds.

Theorem 1. *Algorithm Opponent Choice finds the optimal sequence of opponent choices for all $N = 2^n$ players, and their tournament win probabilities, in $O(2^{\sum_{h=2}^{n} 2^{h-1}} \prod_{l=2}^{n} \prod_{h=1}^{2^{l-1}} (2^l - 2h + 1))$ time.*

4 Win probabilities by seeding

Using the seeding of the tennis players as the ranking, we compute the tournament win probabilities of the various players at the semifinal and quarterfinal rounds and round-of-16, as shown in Tables 1, 2 and 3, respectively. These probabilities are computed by assuming that the highest ranked player(s) select the opponent(s) that maximize their tournament win probability. For example, at the semifinal round, the no. 1 seed has a tournament win probability of 0.391 if choosing to play the no. 4 seed, compared to 0.366 against the no. 2 seed, and 0.386 against the no. 3 seed; it therefore chooses to play the no. 4 seed. The other semifinal match is then between the nos. 2 and 3 seeds. In Table 1, the various players have identical tournament win probabilities for the bracket design and the opponent choice design.

Table 1. Tournament win probabilities for seeded tennis players at semifinal round

Seed i	1	2	3	4
Opponent i	4	3	2	1
q_i	.391	.353	.124	.132

Table 2 contains quarterfinal results, where q_i^D and q_i^S are the tournament win probabilities of player i under dynamic and static rankings of players, respectively. For the opponent choice design, each player's quarterfinal round opponent is also shown.

For this example, each player's quarterfinal round opponents are the same under the dynamic and static rankings of players. Further, q_i^B is the tournament win probability of player i using the conventional quarterfinal round bracket $(1, 8, 4, 5, 3, 6, 2, 7)$. We discuss the results in Table 2. For this instance, seeds nos. 1 and 2 both gain substantially from the opponent choice design, by being able to choose their opponents. However, seeds no. 3 and 4 have a lower tournament win probability, due to the chance of being chosen by a higher ranked player which does not occur at the quarterfinal round under a conventional bracket design. Seed no. 8 gains under the opponent choice design because seed no. 1 is not necessarily its quarterfinal opponent.

Table 2. Tournament win probabilities
for seeded tennis players at quarterfinal round

Seed i	1	2	3	4	5	6	7	8
Opponent	7	4	6	2	8	3	1	5
q_i^D	.342	.242	.101	.078	.049	.064	.039	.085
q_i^S	.342	.242	.107	.078	.050	.064	.037	.080
q_i^B	.315	.201	.121	.109	.081	.072	.044	.056

Table 3. Tournament win probabilities
for seeded tennis players at round-of-16

Seed i	1	2	3	4	5	6	7	8
Opponent	14	12	15	16	13	11	10	9
q_i^D	.337	.223	.086	.080	.056	.047	.031	.036
q_i^S	.336	.232	.089	.085	.057	.047	.030	.035
q_i^B	.313	.228	.073	.079	.056	.050	.035	.037

Seed i	9	10	11	12	13	14	15	16
Opponent	8	7	6	2	5	1	3	4
q_i^D	.023	.016	.020	.009	.012	.009	.009	.007
q_i^S	.021	.014	.020	.008	.010	.005	.006	.006
q_i^B	.025	.017	.021	.013	.013	.011	.010	.018

Table 3 contains round-of-16 results in similar format. For this instance, using the opponent choice design, each player's round-of-16 opponents are the same under dynamic and static rankings of players. The main winner from the opponent choice design is seed no. 1, due to its first-mover advantage. Seeds nos. 3 and 4 benefit from the opponent choice design with dynamic ranking, and seeds nos. 2, 3, 4 and 5 benefit from the opponent choice design with static ranking. The benefit which they receive is paid for quite evenly by seeds nos. 6 through 16.

A particularly interesting case is seed no. 14, which is chosen by seed no. 1. This difficult matchup reduces its tournament win probability from .011 using the bracket design to .005 using the opponent choice design under the static ranking, but only to 0.009 under the dynamic ranking. The difference in these last two numbers results from the possibility that, by beating seed no. 1, seed no. 14 achieves the no. 1 ranking at the quarterfinal round under the dynamic ranking.

5 Future research

Several interesting topics remain open for future research. We particularly recommend the following seven. The first topic applies our work, and the other six extend it.

First, it would be valuable to explore the application of the opponent choice design to empirically-based studies of various sports and competitions. Examples where substantial data for pairwise matchups is available include table tennis and chess. Second, in situations where the pairwise win probabilities are hard to estimate, especially when there are many players involved, a player may resort to a myopic strategy: choose an available opponent which it can beat with highest probability at each round. Third, it would be valuable to study how the results of a particular round could be used to modify the win probability matrix, and consequently the choices of the players at later rounds. For example, a player which has won its matches but with unexpectedly poor performances may become a more attractive opponent at a later round. Fourth, as proposed by Guyon [3], prohibiting the choice of other highly ranked players. Fifth, the opponent choice design can also be used where a player's objective, rather than winning the entire tournament, is to reach a particular round. This scenario arises towards the end of a season, where a player which is seeking to be ranked first at the end of the season needs only to reach a particular round. The sequence of choices of opponents that maximizes the probability of reaching a particular round is not, in general, the same as that which maximizes the player's tournament win probability. Sixth, a natural extension of the opponent choice design under dynamic ranking would be to allow the ranking of the players to be adjusted dynamically, based on detailed performance within the tournament, as measured for example by margin of victory. Finally, the group-strategy-proofness of the proposed design can be studied.

References

[1] W. A. Glenn, A comparison of the effectiveness of tournaments, *Biometrika*, **47** (1960), 253–262, doi: 10.2307/2333297.

[2] J. Guyon, What a fairer 24 team UEFA Euro could look like, *Journal of Sports Analytics*, **4** (2018), 297–317, doi: 10.3233/JSA-180219.

[3] J. Guyon, "Choose your opponent": A new knockout format for sports tournaments. Application to the round of 16 of the UEFA Champions League and to maximize the number of home games during the UEFA Euro 2020, *working paper*, Columbia University, Courant Institute of Mathematical Sciences, New York University, NY, 2019, doi: 10.2139/ssrn.3488832.

[4] J. Horen, R. Riezman, Comparing draws for single elimination tournaments, *Operations Research*, **33**(1985), 249–262, doi: 10.1287/opre.33.2.249.

[5] Inside Hook. MLB considering controversial reality TV-inspired playoff expansion. Available at: https://www.insidehook.com/-daily_brief/sports/mlbconsidering-controversial-reality--tv-inspired-playoff-expansion, last accessed February 20, 2020.

[6] D. T. Searls, On the probability of winning with different tournament procedures, *Journal of the American Statistical Association*, **58** (1963), 1064–1081, doi: 10.2307/2283333.

[7] A. I. K. Vong, Strategic manipulation in tournament games, *Games and Economic Behavior*, **102** (2017), 562–567, doi: 10.1016/j.geb.2017.02.011.

[8] T. Vu, Y. Shoham, Fair seeding in knockout tournaments, *ACM Transactions on Intelligent Systems & Technologies*, **3** (2011), article 9, doi: 10.1145/2036264.2036273.

Extended abstracts

The Third International Workshop on Dynamic Scheduling Problems
Adam Mickiewicz University, Poznań, July 5th – 6th, 2021
DOI: 10.14708/isbn.978-83-951298-6-5p27-32

Scheduling in data gathering networks with variable communication speed and a processing stage

Joanna Berlińska*
Adam Mickiewicz University, Poznań, Poland

Baruch Mor
Ariel University, Ariel, Israel

Keywords: scheduling, data gathering networks, variable communication speed

1 Introduction

Data gathering is an important step of many applications running in distributed systems. Computation results scattered on a large number of workers have to be collected for merging and analysis. Algorithms minimizing the total time of data gathering were proposed by Berlińska for networks with limited base station memory in [1], and for networks with dataset release times in [3]. Luo et al. [7] and Luo et al. [8] studied minimizing the data gathering time in networks with data compression. Berlińska and Przybylski [4] analyzed the problem for networks with local computations. All the mentioned articles share the assumption that the communication parameters of the network are constant. However, in reality, the communication speed may change with time, due to sharing the links with other users and applications, maintenance activities, etc. Scheduling in data gathering networks with variable communication speed was studied by Berlińska [2]. The analyzed problem was to transfer the data from the workers to a single base station in the shortest possible time. In this work, we generalize this problem to the case when each dataset has to be processed after being received by the base station. We show that minimizing the total time of data gathering and processing is strongly $\mathcal{NP}$-hard. Polynomial-time algorithms are proposed for several special cases of the problem. For the general case, heuristic algorithms are designed and compared by means of computational experiments.

2 Problem formulation

We study a star data gathering network consisting of m workers $P_1, \dots, P_m$ and a single base station P_0. Each worker P_i holds dataset D_i of size α_i, which has to be

*Speaker, e-mail: Joanna.Berlinska@amu.edu.pl transferred to the base station for processing. At most one node can communicate with the base station at a time. The communication rate of node P_i depends on the corresponding link being used by other applications. We will be calling a link *loaded* if it is used by background communications at a given time, and *free* in the opposite case. Transferring one unit of data from P_i to P_0 over a free link takes time c_i, for $i = 1, \dots, m$. A loaded link becomes δ times slower, for some fixed rational $\delta > 1$. Thus, sending a unit of data over a loaded link between P_i and P_0 takes time δc_i. After being received by the base station, dataset D_i has to be processed, which takes time $a\alpha_i$. At most one dataset can be processed at a time. Preemptions are allowed both in communication and computations.

The maximum time that may be necessary to gather data from all worker nodes is $\overline{T_c} = \delta \sum_{i=1}^{m} c_i \alpha_i$. The communication speed changes are described in the following way. For each node P_i, we are given a set of n_i disjoint time intervals $[t'_{i,j}, t''_{i,j})$ (where $j = 1, \dots, n_i$, $t''_{i,j} < t'_{i,j+1}$ for $j < n_i$, and $t'_{i,n_i} < \overline{T_c}$), in which the corresponding communication link is loaded. The total number of such intervals is $n_1 + \dots + n_m = n$.

The scheduling problem is to minimize the total time T needed to gather and process all data. It is obvious that nothing can be gained by introducing idle times in communication. Moreover, for a fixed communication schedule, the order of processing the datasets and possible processing preemptions do not affect T, as long as no unnecessary idle times appear. Therefore, we assume without loss of generality that the communication network is never idle before transferring all data and that the datasets are processed in the order in which they arrive at the base station.

3 Computational complexity

In this section, we analyze the computational complexity of our problem and its special cases. Due to limited space, longer proofs are omitted.

The following complexity result is achieved by a pseudopolynomial reduction from the strongly $\mathcal{NP}$-complete 3-Partition problem (Garey and Johnson [5]).

Proposition 1. *The analyzed scheduling problem is strongly $\mathcal{NP}$-hard, even if $a = 1$ and $c_i = 1$ for $i = 1, \dots, m$.*

In the next proposition, we show the main difficulty in constructing exact algorithms for our problem.

Proposition 2. *Constructing an optimum schedule for the analyzed problem may require preempting a dataset transfer at a time when no link speed changes.*

Proof. Let $m = 2$, $c_1 = c_2 = 1$, $a = 0$, $\alpha_1 = \alpha_2 = 2$, and let δ be an arbitrary number greater than 1. Suppose the first link is loaded in interval $[2, 3)$, and the second link is loaded in interval $[3, 4)$. The optimum schedule length 4 can be achieved only if all data are transferred over free links. For example, dataset D_1 can be sent in intervals $[0, 1)$ and $[3, 4)$, and D_2 in interval $[1, 3)$. It is easy to check that if no preemption takes place before time 2, then a part of one of the datasets has to be sent over a loaded link. □

According to Proposition 2, it is not known which moments should be taken into account as possible communication preemption points. Thus, constructing a full search or branch-and bound algorithm for our problem is a challenge.

Note that if the communication links are never loaded, our problem reduces to $F2|pmtn|C_{\max}$, and hence, it can be solved in $O(m \log m)$ time using Johnson's algorithm (Johnson [6]). If $a = 0$, which means there is no processing stage, then our problem is also solvable in polynomial time, using the algorithm by Berlińska [2]. In order to present other polynomial special cases, we prove a few structural properties.

Proposition 3. *If all links are loaded in the same intervals, then the time required to transfer a given set of datasets $\{D_{i_1}, \dots, D_{i_k}\}$, starting at time 0, does not depend on the order of communications.*

Proof. Sending data of size α at unit communication time c is equivalent to sending data of size $c\alpha$ at unit comunication time 1. Hence, if all links are loaded in the same intervals, sending datasets $D_{i_1}, \dots, D_{i_k}$ over the respective links is equivalent to sending a single dataset of size $\sum_{j=1}^{k} c_{i_j}\alpha_{i_j}$ over a link with communication speed 1 in the free intervals and $1/\delta$ in the loaded intervals. □

Proposition 4. *If all links are loaded in the same intervals, there exists an optimum non-preemptive schedule.*

Proof. Suppose that dataset D_i is transferred in several pieces in an optimum schedule Σ. We construct a new schedule Σ' by moving all messages containing parts of D_i just before its last piece. The other communications preceding the transfer of D_i are moved to the left. Although the transfer times of individual datasets may change during this process, by Proposition 3 the transfer of each dataset finishes in Σ' not later than in Σ. Hence, dataset processing also finishes in Σ' not later than in Σ, and consequently, Σ' is an optimum schedule. By repeating this procedure for all datasets sent in several messages, we arrive at a non-preemptive optimum schedule. □

Using Propositions 3 and 4, and the interchange argument, we prove that the following two special cases of our problem are polynomially solvable.

Proposition 5. *If $a \le c_i$ for all i, and all links are loaded in the same intervals, then the optimum schedule can be constructed in $O(m \log m + n)$ time by sending the datasets in the order of non-increasing sizes α_i.*

Proposition 6. *If $a \geq \delta c_i$ for all i, and all links are loaded in the same intervals, then the optimum schedule can be constructed in $O(m \log m + n)$ time by sending the datasets in the order of non-decreasing $c_i \alpha_i$.*

4 Heuristics and computational experiments

In this section, we propose greedy heuristics running in $O((m+n)^2)$ time. In each of these algorithms, every time a dataset transfer completes or the speed of some link changes, the dataset to be transferred is selected according to a given rule. Algorithm *gTime* chooses the dataset whose transfer will complete in the shortest time. Heuristic *gRate* selects the dataset which will be sent at the best average communication rate (under the assumption that there will be no preemption). Algorithm *gJohnson* associates with each available dataset a job consisting of two operations: sending the remaining part of this dataset, and processing this dataset. A job is selected using Johnson's rule (Johnson [6]), and the corresponding dataset is transferred.

The makespan obtained by any algorithm that does not introduce communication idle times is at most $\delta \sum_{i=1}^{m} c_i \alpha_i + a \sum_{i=1}^{m} \alpha_i$, while the optimum makespan is not smaller than $\max\{\sum_{i=1}^{m} c_i \alpha_i, a \sum_{i=1}^{m} \alpha_i\}$. Hence, each of our algorithms delivers a $(\delta + 1)$-approximation of the optimum solution.

The quality of the results delivered by the proposed heuristics was analyzed by means of computational experiments. The number of datasets in the test instances was $m \in \{10, 15, \dots, 50\}$. Dataset sizes α_i were chosen randomly from the range $[1, 20]$. Parameter δ was set to 2. The basic communication costs of all links were equal, $c_i = c$ for $i = 1, \dots, m$. We used $c \in \{0.5, 0.75, 1\}$ and $a = 1$, thus representing the three cases of $\delta c \leq a$, $c < a < \delta c$ and $a \leq c$. The communication speed changes were generated similarly as those in Berlińska [2]. Namely, for given values $F, L \in \{1, 5\}$, the length of the first free interval of link i was selected randomly from the range $[0, \frac{F}{m} \sum_{i=1}^{m} c_i \alpha_i]$. Then, the length of the first loaded interval was chosen randomly from the range $[0, \frac{L}{m} \sum_{i=1}^{m} c_i \alpha_i]$. The lengths of consecutive free and loaded intervals were being selected until reaching the communication time horizon $\overline{T_c}$. For each analyzed setting, 100 instances were generated and solved.

As the optimum solutions for the generated instances were not known, schedule quality was measured by the average percentage error with respect to the lower bound LB computed as follows. Let $T_c^{(i)}$ be the time required to transfer dataset D_i (starting at time 0), and let T_c be the minimum time necessary for transferring all data to the base station. Note that T_c can be computed using linear programming as described by Berlińska [2]. Finally, let T_J be the minimum time required for transferring and processing all datasets under the assumption that the communication links are always free, obtained by Johnson's algorithm. We set

$$LB = \max\{LB_1, LB_2, T_J\}, \tag{1}$$

where

$$LB_1 = \max_{i=1}^{m}\{T_c^{(i)} + a\alpha_i\} \tag{2}$$

and

$$LB_2 = T_c + \min_{i=1}^{m}\{a\alpha_i\}. \tag{3}$$

The results of our experiments lead us to the following conclusions:

1. Naturally, the obtained makespans are closest to LB when c and L are small, and F is large.
2. In general, better results are obtained for large instances than for the smaller ones. Indeed, a larger number of datasets gives more opportunities to avoid loaded links, which may result both in finding better solutions and in LB being closer to the actual optimum.
3. When communication is slow and the links are rarely free ($c = 1$ and $F = 1$), the best results are delivered by algorithm *gRate*. The reported errors are below 18% for $L = 1$, and below 40% for $L = 5$.
4. In the remaining settings, the best results are usually obtained either by *gJohnson* or *gTime* (the only exception are the tests with $c = 0.75$, $F = 1$, $L = 5$ and $m \geq 30$, for which *gRate* is the winner). The makespans delivered by the best of algorithms *gJohnson* and *gTime* are, on average, at most 23% from LB for the most difficult instances in this group (i.e., when $c = 0.75$, $F = 1$, $L = 5$, $m = 10$), but less than 3% from LB for all tests with $c = 0.5$.

5 Future research

Future research should include the complexity analysis of the special case when all links are loaded in the same intervals, but the relations between c_i and a are arbitrary, and the case when $a \geq \delta c_i$ for all i, but different links are loaded in different time intervals.

Acknowledgements

The research of the first author was partially supported by the National Science Centre, Poland, grant 2016/23/D/ST6/00410.

References

[1] J. Berlińska, Heuristics for scheduling data gathering with limited base station memory, *Annals of Operations Research*, **285** (2020), 149–159, doi: 10.1007/s10479-019-03185-3.

[2] J. Berlińska, Scheduling in data gathering networks with background communications, *Journal of Scheduling*, **23** (2020), 681–691, doi: 10.1007/s10951-020-00648-5.

[3] J. Berlińska, Makespan minimization in data gathering networks with dataset release times, *Lecture Notes in Computer Science*, **12044** (2020), 230–241, doi: 10.1007/978-3-030-43222-5_20.

[4] J. Berlińska, B. Przybylski, Scheduling for gathering multitype data with local computations, *European Journal of Operational Research*, 2021, doi: 10.1016/j.ejor.2021.01.043.

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

[6] S. M. Johnson, Optimal two- and three-stage production schedules with setup times included, *Naval Research Logistics Quarterly*, **1** (1954), 61–68, doi: 10.1002/nav.3800010110.

[7] W. Luo, Y. Xu, B. Gu, W. Tong, R. Goebel, G. Lin, Algorithms for communication scheduling in data gathering network with data compression, *Algorithmica*, **80** (2018), 3158–3176, doi: 10.1007/s00453-017-0373-6.

[8] W. Luo, B. Gu, G. Lin, Communication scheduling in data gathering networks of heterogeneous sensors with data compression: Algorithms and empirical experiments, *European Journal of Operational Research*, **271** (2018), 462–473, doi: 10.1016/j.ejor.2018.05.047.

The Third International Workshop on Dynamic Scheduling Problems
Adam Mickiewicz University, Poznań, July 5th – 6th, 2021
DOI: 10.14708/isbn.978-83-951298-6-5p33-37

New results in time-dependent open shop scheduling

Stanisław Gawiejnowicz*
Adam Mickiewicz University, Poznań, Poland

Marta Kolińska
Adam Mickiewicz University, Poznań, Poland

Keywords: time-dependent scheduling, deteriorating jobs, open shop, LAPT rule

1 Introduction

We consider open shop scheduling problems with deteriorating jobs, in which we are given machines $M_1, M_2, \ldots, M_m$ and a set $\mathcal{J}$ of independent jobs $J_1, J_2, \ldots, J_n$ to be scheduled on the machines which are available for processing from time $t_0 > 0$, where $m \in \{2, 3\}$ and $n \geq m$. Any job $J_k \in \mathcal{J}$ is composed of m operations, $O_{1k}, O_{2k}, \ldots, O_{mk}$, $1 \leq k \leq n$, and it is completed if all the operations are completed. The processing time of the ith operation of the jth job, O_{ij}, proportionally deteriorates in time and equals $p_{ij} = b_{ij}s_{ij}$, where integer *deterioration rates* $b_{ij} > 0$ for $1 \leq i \leq m$ and $1 \leq j \leq n$, and $s_{ij} \geq t_0$ denotes the starting time of the operation. Any operation sequence satisfying the problem assumptions is feasible, i.e. no precedence constraints exist between operations of the same job. The criterion of schedule optimality is the makespan $C_{\max} = \max\{C_{ij} : 1 \leq i \leq m, 1 \leq j \leq n\}$, where C_{ij} is the completion time of operation O_{ij}. Applying the three-field notation, we will denote the problems as $O2|p_{ij} = b_{ij}t|C_{\max}$ and $O3|p_{ij} = b_{ij}t|C_{\max}$.

2 Related research

Open shop scheduling problems were defined by Gonzales and Sahni [6], who assumed that operation processing times are fixed and proved that the two-machine problem with the makespan objective, $O2||C_{\max}$, is solvable in $O(n)$ time. They also proved that m-machine generalization of the two-machine open shop problem, $Om||C_{\max}$, is $\mathcal{NP}$-hard for every fixed $m \geq 3$. The question whether this problem is strongly $\mathcal{NP}$-hard remains unanswered as noted by Woeginger [15]. A similar question in case when operation processing times satisfy the equality $p_{ij} = p_j$ was answered in the negative by Sevastyanov [13]. We refer the reader to paper by Gawiejnowicz and Kolińska [5],

*Speaker, e-mail: stgawiej@amu.edu.pl

chapter by Gonzales [7], review by Woeginger [15], and to monographs by Pinedo [12] and Tanaev et al. [14] for more details.

The research on open shop scheduling with proportionally deteriorating operation processing times, introduced by Mosheiov [11], is very limited. Kononov [8] and Mosheiov [10] independently proved that problem $O2|p_{ij} = b_{ij}t|C_{\max}$ is solvable in $O(n)$ time by a modification of the Gonzalez-Sahni algorithm. Kononov [8] also proved that m-machine time-dependent open shop problems are intractable for $m \geq 3$, even if job deterioration rates are restricted, since problem $O3|p_{ij} = b_{ij}t|C_{\max}$ and problem $O3|p_{ij} = b_{ij}t, b_{3j} = b|C_{\max}$ both are $\mathcal{NP}$-hard. Gawiejnowicz and Kononov [9] proved that if we change proportional processing times into linear ones, then already two-machine open shop problem with the makespan objective, $O2|p_{ij} = a_{ij}+b_{ij}t|C_{\max}$, is $\mathcal{NP}$-hard. We refer the reader to review by Gawiejnowicz [2], and to monographs by Agnetis et al. [1] and Gawiejnowicz [3] for more details.

3 Our results

Our first result is the following new lower bound on the value of $C_{\max}$ for multi-machine time-dependent open shop problem.

Theorem 1. (Gawiejnowicz and Kolińska [5]) *The minimum makespan of any feasible schedule σ for problem $Om|p_{ij} = b_{ij}t|C_{\max}$ satisfies the inequality*

$$C^{\star}_{\max}(\sigma) \geq \max\left\{ \max_{1\leq i\leq m}\left\{\prod_{j=1}^{n}(1+b_{ij})\right\}, \max_{1\leq j\leq n}\left\{\prod_{i=1}^{m}(1+b_{ij})\right\}\right\}. \quad (1)$$

Lower bound (1) is a generalization of the following lower bound for two-machine time-dependent open shop problem,

$$C^{\star}_{\max}(\sigma) \geq \max\left\{\prod_{j=1}^{n}(1+b_{1j}), \prod_{j=1}^{n}(1+b_{2j}), \max_{1\leq j\leq n}\{(1+b_{1j})(1+b_{2j})\}\right\}, \quad (2)$$

proved by Mosheiov [10].

Our next result concerns problem $O2|p_{ij} = b_{ij}t|C_{\max}$. We propose to solve this problem using the following new scheduling rule, which is a time-dependent counterpart of the LAPT rule by Pinedo [12]:

> *whenever a machine becomes free, assign to the machine this job was not yet processed on either machine and which has the largest deterioration rate on the other machine.*

The new rule, the *Largest Alternate Deterioration Rate first* ($LADR$), assigns a job to the freed machine taking into account the deterioration rate of the job on the other machine. If both machines are idle and deterioration rates of the same job on both machines are equal, this job may be assigned to any of both machines. Jobs that already have been completed on the other machine, will be assign to the machine just freed with the lowest priority.

Theorem 2. (Gawiejnowicz and Kolińska [5]) *Problem* $O2|p_{ij} = b_{ij}t|C_{\max}$ *is solvable by the LADR rule and the makespan of schedule constructed by the rule satisfies formula* (2).

This result may be proved either by adopting the original proof by Pinedo [12] or by applying the notion of *isomorphic scheduling problems* introduced by Kononov and Gawiejnowicz [4].

Example 3. (Gawiejnowicz and Kolińska [5]) Let us consider an instance of problem $O2|p_{ij} = b_{ij}t|C_{\max}$ with $t_0 = 1$ and $n = 3$ jobs with deterioration rates as in Table 1.

Table 1. Example instance of problem $O2|p_{ij} = b_{ij}t|C_{\max}$

j	b_{1j}	b_{2j}
1	3	7
2	7	1
3	15	15

The LADR rule generates for this instance the schedule presented in Fig. 1. Since the lower bound (2) for the instance equals 512 time units and since the makespan for this schedule equals 512 time units as well, the schedule is optimal.

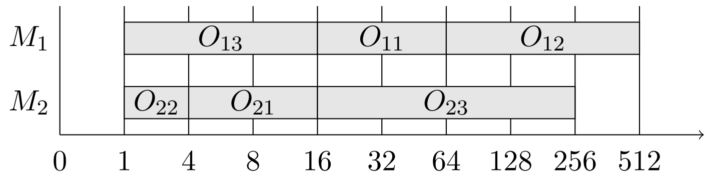


Figure 1. Schedule generated by the LADR rule for instance in Table 1

Our last result concerns problem $O3|p_{ij} = b_{ij}t|C_{\max}$. We propose to solve this problem using a new scheduling rule, the *Largest Total Remaining Deterioration Rate on Other Machines first* ($LTRDROM$). This rule can be formulated as follows:

every time a machine is freed, the job with the largest total remaining deterioration rate on all other machines, among available jobs, is selected for processing.

The quality of schedules generated by the LTRDROM rule was tested in a few numerical experiments. In total, we tested 240 instances. From 120 tested small-size instances with 5, 10 or 15 jobs, optimal schedules were generated for about 55% instances. A better percentage was observed for 120 medium-size instances with 20, 25 or 30 jobs, where more than 60% of instances were solved to optimality. The average computation time varied between 1.1191 ms for instances with $n = 5$ jobs and 7.4223 ms for instances with $n = 30$ jobs. The results (see Gawiejnowicz and Kolińska [5] for details) suggest that the LTRDROM rule generates near-optimal schedules for small- and medium-size instances of problem $O3|p_{ij} = b_{ij}t|C_{\max}$.

4 Future research

One of possible topics for future research is the construction of branch-and-bound algorithms for problem $O3|p_{ij} = b_{ij}t|C_{\max}$. This is a challenge in view of known problems with numerical overflow errors which may appear as a result of multiplicativity of the problem. Another interesting future research topic may be a new scheduling algorithm, combining a good initial solution, e.g. the one generated by the LTRPOM rule, and the lower bound (1).

References

[1] A. Agnetis, J-C. Billaut, S. Gawiejnowicz, D. Pacciarelli, A. Soukhal, *Multi-agent Scheduling: Models and Algorithms*, Springer, Berlin-Heidelberg, 2014, doi: 10.1007/978-3-642-41880-8.

[2] S. Gawiejnowicz, A review of four decades of time-dependent scheduling: main results, new topics, and open problems, *Journal of Scheduling*, **23** (2020), 3–47, doi: 10.1007/s10951-019-00630-w.

[3] S. Gawiejnowicz, *Models and Algorithms of Time-Dependent Scheduling*, Springer, Berlin-Heidelberg, 2020, doi: 10.1007/978-3-662-59362-2.

[4] S. Gawiejnowicz, A. Kononov, Isomorphic scheduling problems, *Annals of Operations Research*, **213** (2014), 131–145, doi: 10.1007/s10479-012-1222-2.

[5] S. Gawiejnowicz, M. Kolińska, Two- and three-machine open shop scheduling using LAPT-like rules, *Computers & Industrial Engineering*, **157** (2021), 107261, `doi: 10.1016/j.cie.2021.107261`.

[6] T. Gonzalez, S. Sahni, Open shop scheduling to minimize finish time, *Journal of the Association for Computing Machinery*, **23** (1976), 665–679, `doi: 10.1145/321978.321985`.

[7] T. F. Gonzalez, Open shop scheduling, in: J. Y-T. Leung, *Handbook of Scheduling: Algorithms, Models and Performance*, Chapman & Hall/CRC Press, Boca Raton, 2004, `doi: 10.1201/9780203489802`.

[8] A. Kononov, Combinatorial complexity of scheduling jobs with simple linear deterioration, *Discrete Analysis and Operations Research*, **3** (1996), 15–32 (in Russian).

[9] A. Kononov, S. Gawiejnowicz, NP-hard cases in scheduling deteriorating jobs on dedicated machines, *Journal of the Operational Research Society*, **52** (2001), 708–718, `doi: 10.1057/palgrave.jors.2601117`.

[10] G. Mosheiov, Complexity analysis of job-shop scheduling with deteriorating jobs, *Discrete Applied Mathematics*, **117** (2002), 195–209, `doi: 10.1016/S0166-218X(00)00385-1`.

[11] G. Mosheiov, Scheduling jobs under simple linear deterioration, *Computers and Operations Research*, **21** (1994), 653–659, `doi: 10.1016/0305-0548(94)90080-9`.

[12] M. L. Pinedo, *Scheduling: Theory, Algorithms and Systems*, 5rd ed., Springer, Berlin-Heidelberg, 2016, `doi: 10.1007/978-3-319-26580-3`.

[13] S. Sevastyanov, Some positive news on the proportionate open shop problem, *Siberian Electronic Mathematical Reports*, **16** (2019), 406–426, `doi: 10.33048/semi.2019.16.023`.

[14] V. S. Tanaev, Y. N. Sotskov, V. A. Strusevich, *Scheduling Theory: Multi-Stage Systems*, Kluwer, Dordrecht, 1994, `doi: 10.1007/978-94-011-1192-8`.

[15] G. Woeginger, The open shop scheduling problem, *The 35th Symposium of Theoretical Aspects of Computer Science*, 2018, 4:1-12, `doi: 10.4230/LIPIcs.STACS.2018.4`.

The Third International Workshop on Dynamic Scheduling Problems
Adam Mickiewicz University, Poznań, July 5th – 6th, 2021
DOI: 10.14708/isbn.978-83-951298-6-5p39-45

On the complexity of conditional DAG scheduling

Alberto Marchetti-Spaccamela
Sapienza University of Rome, Italy

Nicole Megow
University of Bremen, Germany

Jens Schlöter*
University of Bremen, Germany

Martin Skutella
Technical University of Berlin, Germany

Leen Stougie
CWI Amsterdam and Vrije Universiteit Amsterdam, Netherlands

Keywords: parallel processing, makespan, conditional DAG, complexity

1 Introduction

As parallel processing became ubiquitous in modern computing systems, parallel task models have been proposed to describe the structure of parallel applications. A popular model is to represent a task by a DAG (directed acyclic graph). Each node represents the execution of a sub-task and each edge formulates precedence constraints between sub-tasks. The DAG model assumes a fixed structure capturing only straight-line code. While this captures the intra-parallelism of tasks, it does not capture the typical conditional nature of control flow instructions, such as `if-then-else` statements. The presence of conditional constructs within the modeled code may mean that different activations of the task cause different parts of the code to be executed. The *conditional DAG* model generalizes the DAG model by allowing *conditional* nodes (Baruah et al. [3], Melani et al. [9]).

While first algorithmic results have been presented for the conditional DAG model, the complexity of schedulability analysis remains wide open. We perform a thorough analysis on the worst-case makespan of a conditional DAG task under list scheduling. We show several hardness results for the optimization problem on multiple processors, even if the conditional DAG has a well-nested structure. For general conditional DAG tasks, the problem is intractable even on a single processor. Complementing these

*Speaker, email: `jschloet@uni-bremen.de` negative results, we show that certain practice-relevant DAG structures are very well tractable. The full version of this abstract is a part of the proceedings of IPDPS 2020 (Marchetti-Spaccamela et al. [10]).

2 Related work

Earlier proposed parallel task models (fork/join, synchronous parallel, DAG) do not capture control flow information *and* conditional executions. Fonseca et al. [5] propose the *multi-DAG* that represents a task as a collection of DAGs, each representing a control flow. When a task is executed, exactly one of the DAGs is executed. The main issue with this model is the possibly exponential number of control flows.

Chakraborty et al. [4] consider a more restricted variant of the conditional DAG model, which models tasks as a two-terminals DAG and for each node exactly one successor needs to be executed. Additionally, each edge characterizes a delay for the start time of the successor. The authors provide complexity results and exact and approximate schedulability analysis for preemptive and non-preemptive scheduling.

Federated scheduling is a scheduling policy for scheduling a set of recurrent tasks modeled by DAGs; each task has a release time and deadline. In the model, we assign high-demand conditional DAGs to a number of completely dedicated processors. All remaining tasks are assigned to a pool of shared processors. Baruah [2] considered federated scheduling for conditional recurring DAG tasks assuming constrained deadlines. In this case, our results imply complexity bounds on the sub-problem of minimizing the number of processors necessary to schedule a high-demand task.

3 System model and definitions

Let τ be a conditional parallel task (*cp-task*) processed on m identical processors. The task τ is characterized by a conditional DAG $G = (V, E, C)$ where V is a set of nodes, $E \subseteq V \times V$ is a set of directed edges (arcs) and $C \subseteq V \times V$ is a set of distinguished node pairs, the *conditional pairs*. Each $j \in V$ represents a sequential computation unit (sub-task, job) with processing time p_j. Slightly abusing notation, we refer to jobs and nodes equivalently. The arcs describe dependencies between sub-tasks: if $(v_1, v_2) \in E$, then v_2 can only start processing if v_1 has completed, except for endpoints of conditional pairs as explained below. We call v_1 a predecessor of v_2.

```
if $c^1$ then
    if $c^2$ then
        basic block $b_1$;
    else
        basic block $b_2$;
    end if
else
    basic block $b_3$;
end if
```

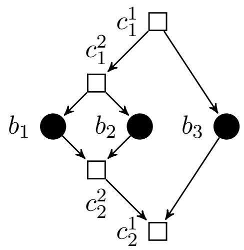


Figure 1. Example of a conditional DAG

A distinguished pair $(c_1, c_2) \in C$ of nodes is a conditional pair which denotes the beginning and ending of a conditional construct such as an `if-then-else` statement. In sub-task c_1, a conditional expression is being evaluated and, depending on the outcome, exactly one out of many possible subsequent successors must be chosen. In our figures, the conditional nodes are depicted by a square and all other nodes are circles; see Figure 1. Following the definitions given in Baruah et al. [3], Melani et al. [9], we formally define a conditional DAG.

Definition 1. A *conditional DAG* $G = (V, E, C)$ is a DAG (V, E) and a set of *conditional pairs* $C \subseteq V \times V$ such that the following holds for each $(c_1, c_2) \in C$:

1. There are multiple outgoing edges from c_1 in E. Suppose that there are exactly k outgoing edges from c_1 to vertices $s_1, s_2, \ldots, s_k$ for some $k > 1$. Then there are exactly k incoming edges into c_2 in E, from the vertices $t_1, t_2, \ldots, t_k$.
2. For each $l \in \{1, \ldots, k\}$ let P_l be the set of all paths from s_l to t_l in G. We define $G_l = (V_l, E_l)$ as the union of all paths from s_l to t_l, i.e., $V_l = \bigcup_{p \in P_l} V(p)$ and $E_l = \bigcup_{p \in P_l} E(p)$, where $V(p)$ and $E(p)$ denote the sets of vertices and edges on path p. We refer to each G_l with $l \in \{1, \ldots, k\}$ as a *conditional branch*.
3. It must hold that $V_l \cap V_{l'} = \emptyset$ for all l, l' with $l \neq l'$. Additionally, with the exception of (c_1, s_l) and (t_l, c_2) there should be no edges in E into vertices in V_l from nodes not in V_l or vice versa for each $l \in \{1, \ldots, k\}$. That is, for all l, $E \cap ((V \setminus V_l) \times V_l) = \{(c_1, s_l)\}$ and $E \cap (V_l \times (V \setminus V_l)) = \{(t_l, c_2)\}$.

For each pair $(c_1, c_2) \in C$ we call c_1 and c_2 *conditional nodes* and refer to the subgraph of G beginning at c_1 and ending at c_2 as *conditional construct* in G. Notice that in the above definition, 3. explicitly rules out any interaction between a node within a conditional branch and any other node outside this branch. The restriction to well-nested structures is very natural when modeling the execution flow of a structured programming language (Melani et al. [9]). We refer to a *conditional DAG with shared nodes* when relaxing restriction 3. and allowing interaction between different conditional branches.

When executing a conditional DAG G, at most one conditional branch per conditional pair is executed. For a $c \in C$ no branch is executed iff the construct of c is nested into a branch that is not executed. Thus, a job j is executed if either (i) node j is not part of any conditional branch, i.e, $j \notin V_l$ for each branch G_l of any conditional pair c, or (ii) the innermost branch G_l with $j \in V_l$ is executed. Let $J \subseteq V$ be a set of jobs obtained by *fully executing* the jobs of the conditional DAG $G = (V, E, C)$ taking into account the outcome of conditional pairs. Let $G_J = (V_J, E_J)$ with $V_J = J$ denote the subgraph of G induced by J, then G_J is a *realization* of G. Let $\mathcal{J}$ be the collection of all sets J for which a realization with $V_J = J$ exists.

We consider the list scheduling of conditional DAGs. Let τ be a cp-task with $G = (V, E, C)$ and with processing times p_j, for each $j \in V$, to be executed on m

parallel identical processors. Let $\prec$ be a given *fixed-priority order (FP-order)* over V. A (non-preemptive) fixed-priority schedule (FP-schedule) starts executing the job with the highest priority according to $\prec$ among the available jobs whenever a processor is idle and processes it until completion. A job is *available* if all predecessors have been completed. For each $J \in \mathcal{J}$ let S_J denote the FP-schedule induced by $\prec$ for the realization G_J. Let C_J denote the latest completion time of any job in G_J in S_J. This is the *makespan* for realization G_J of the cp-task τ. Then, $M(G, \prec) = \max_{J \in \mathcal{J}} C_J$ is the *worst-case makespan* of τ for list scheduling according to the FP-order $\prec$.

Definition 2 (CDAG-MAX)**.** Given a cp-task with a conditional DAG G, processing times p_j, a number m of parallel identical processors and an FP-order $\prec$, the *worst-case makespan problem* (CDAG-MAX) is to compute $M(G, \prec)$. Slightly abusing notation, we use CDAG-MAX also to refer to the following *decision variant of* this problem: for a given CDAG-MAX instance and a parameter D decide whether $M(G, \prec) \leq D$.

4 Our results

We give several hardness results for different CDAG-MAX variants using a general reduction framework that exploits a non-obvious relation between the problem of computing the *maximum* makespan for a conditional DAG and *minimizing* the maximum makespan for a DAG. This framework uses an intermediate problem and we show that there is an approximation preserving reduction from the intermediate problem to CDAG-MAX that also preserves some structural properties of input instances. The reduction framework allows us to show hardness and inapproximability results for CDAG-MAX by proving corresponding results for the intermediate problem.

The used intermediate problem is LS-MAX, where we are given a precedence constraint DAG $G = (V, E)$, processing times p_j for each job $j \in V$, m identical parallel processors and a deadline D. The goal is to decide whether $\overline{C}_{\max} > D$, where $\overline{C}_{\max}$ is the maximum makespan that can be achieved by any list scheduling (FP) order. The main part of our reduction framework is to give a reduction fulfilling Theorem 1.

Theorem 1. *There is an approximation preserving polynomial time reduction from* LS-MAX *to* CDAG-MAX.

The main idea of the reduction that proves Theorem 1 is to use the same graph for the constructed CDAG-MAX instance as in the input LS-MAX instance but replace each original job j with a conditional construct defined by a conditional pair $c_j = (c_1^j, c_2^j)$. Each branch of the conditional construct consists of a single copy

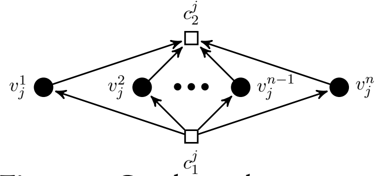


Figure 2. Conditional construct as used in the reduction of Theorem 1

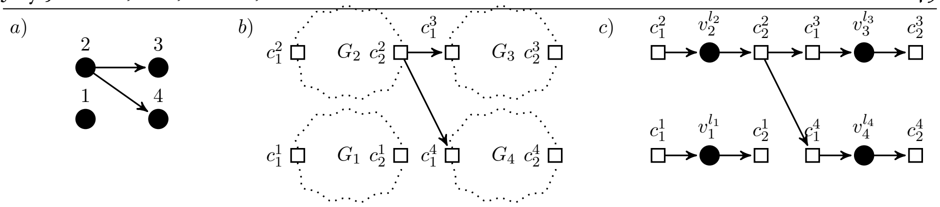


Figure 3. a) Input LS-MAX instance. b) Conditional DAG as constructed by the reduction of Theorem 1. Each G_i represents a conditional construct as illustrated in Figure 2. c) Realization of the constructed conditional DAG.

of job j (see Figure 2) such that each realization of the constructed conditional DAG will execute exactly one copy of each original job using precedence constraints analogous to the original instance. Essentially, each realization of the constructed conditional DAG then is a copy of the input LS-MAX graph (with conditional dummy nodes) as illustrated in Figure 3. The key aspect of the reduction is to define the fixed priority order of the constructed conditional DAG task such that each possible list scheduling order of the LS-MAX instance is used by at least one realization of the constructed conditional DAG. This implies that the makespan of each list scheduling schedule of the LS-MAX instance corresponds to the makespan of a realization of the CDAG-MAX instance and vice versa. A full version of this proof sketch implies Theorem 1. Using Theorem 1 and additional properties of the reduction, we show hardness and inapproximability results for CDAG-MAX by proving the corresponding results for LS-MAX. Table 1 summarizes these results.

Table 1. Complexity results for CDAG-MAX variants without shared nodes

CDAG-MAX Variant	Complexity	Reduction via
General (preemptive and non-preemptive)	co$\mathcal{NP}$-complete	$P\|\|C_{\max}$, [6]
$m = 2$	co$\mathcal{NP}$-complete (Weakly)	$P2\|\|C_{\max}$, [8]
Tree-Realizations	co$\mathcal{NP}$-complete	$P\|\|C_{\max}$
4-Chain-Realizations, $m = 2$	co$\mathcal{NP}$-complete (Weakly)	$P2\|3chains\|C_{\max}$, [1]
Approx. within factor $7/5$ and $6/5$ (preemptive)	$\mathcal{NP}$-hard	Clique, [8]

On the positive side, it is known that the worst-case makespan of a conditional DAG task can be approximated within a factor of 2 (Melani et al. [9]). We show that if the conditional DAG has in each realization bounded width, then the worst-case makespan for non-preemptive FP-scheduling can be computed in pseudo-polynomial time via a dynamic program. A realization has a bounded width, if the size of the longest antichain of the realization is bounded by a constant. We also show that this algorithm can be turned into an FPTAS if a certain monotonicity property holds for the job completion times under FP-scheduling. We prove that the property holds, e.g., for a bounded number of chains. In general, the monotonicity property does not hold as a classical example known as Graham anomaly [7] shows.

Finally, it is known that the worst-case makespan for list scheduling a conditional DAG task with independent conditional constructs on a single processor can be computed in polynomial time (Melani et al. [9]). We show that it is crucial for this result that different conditional constructs are independent of each other.

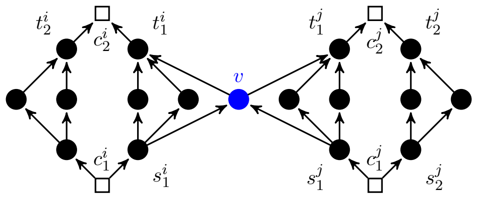


Figure 4. Conditional DAG with shared nodes

Conditional DAGs with shared nodes allow dependencies between different conditional branches in form of shared nodes (see Figure 4). In this relaxed model, a node j is executed if *at least one* of the innermost conditional branches G_l with $j \in V_l$ is executed. We show that in this model computing the worst-case makespan under FP-scheduling is already co$\mathcal{NP}$-hard on a single machine.

References

[1] A. Agnetis, M. Flamini, G. Nicosia, A. Pacifici, Scheduling three chains on two parallel machines, *European Journal of Operational Research*, **202** (2010), 669-674, doi: 10.1016/j.ejor.2009.07.001.

[2] S. Baruah, The federated scheduling of systems of conditional sporadic DAG tasks, *2015 International Conference on Embedded Software*, 2015, 1–10, doi: 10.1109/EMSOFT.2015.7318254.

[3] S. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, The global EDF scheduling of systems of conditional sporadic DAG tasks, *27th Euromicro Conference on Real-Time Systems*, 2015, 222–231, doi: 10.1109/ECRTS.2015.27.

[4] S. Chakraborty, T. Erlebach, S. Kunzli, L. Thiele, Schedulability of event-driven code blocks in real-time embedded systems, *Proceedings of the 39th Annual Design Automation Conference*, 2002, 616–621, doi: 10.1145/513918.514075.

[5] J. C. Fonseca, V. Nélis, G. Raravi, L. M. Pinho, A multi-dag model for real-time parallel applications with conditional execution, *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, 2015, 1925–1932, doi: 10.1145/2695664.2695808.

[6] M. R. Garey, D. S. Johnson, Strong NP-completeness results: motivation, examples, and implications, *Journal of the Association for Computing Machinery*, **25** (1978), 499–508, doi: 10.1145/322077.322090.

[7] R. L. Graham, Bounds on multiprocessing timing anomalies, *SIAM Journal on Applied Mathematics*, **17** (1969), 416–429, doi: 10.1137/0117039.

[8] J. K. Lenstra, A. H. G. Rinnooy Kan, Complexity of scheduling under precedence constraints, *Operations Research*, **26** (1978), 22–35, `doi: 10.1287/opre.26.1.22`.

[9] A. Melani, M. Bertogna, V. Bonifaci, A. Marchetti-Spaccamela, G. C. Buttazzo, Response-time analysis of conditional DAG tasks in multiprocessor systems, *27th Euromicro Conference on Real-Time Systems*, 2015, 211–221, `doi: 10.1109/ECRTS.2015.26`.

[10] A. Marchetti-Spaccamela, N. Megow, J. Schlöter, M. Skutella, L. Stougie, On the complexity of conditional DAG scheduling in multiprocessor systems, *2020 IEEE International Parallel and Distributed Processing Symposium*, 2020, 1061–1070, `doi: 10.1109/IPDPS47924.2020.00112`.

The Third International Workshop on Dynamic Scheduling Problems
Adam Mickiewicz University, Poznań, July 5th – 6th, 2021
DOI: 10.14708/isbn.978-83-951298-6-5p47-50

Single machine scheduling with step-learning

Baruch Mor
Ariel University, Ariel, Israel

Gur Mosheiov*
School of Business Administration, The Hebrew University, Jerusalem, Israel

Keywords: scheduling, single machine, step-learning, makespan

1 Introduction

In traditional scheduling theory, the processing times of the jobs were assumed to be constants dictated in advance. But as scheduling theory evolved, this concept was revised and many studies have proposed variable job processing times to reflect real-life circumstances. As claimed by Gawiejnowicz [2]: "*Scheduling with variable job processing times has numerous applications, e.g., in the modelling of the forging process in steel plants, manufacturing of preheated parts in plastic molding or in silverware production, finance management and scheduling maintenance or learning activities*".

The most prevalent aspects of variable job processing times are deterioration and/or learning effects. In this study, we concentrate on learning effects. The importance of the learning process is its implications on manufacturing routines. Empirical research has confirmed that learning-by-doing increases the productivity on the single worker level and on the team level. The learning effects are manifested by enhanced productivity of the production system, decreased operational expense, and faster time-to-market, thus improving the sustainability of the business. A very recent paper by Azzouz et al. [1] summarizes developments in the concept of learning effects and presents an overview of the significant learning models, a classification scheme for scheduling under learning effects and a mapping of the relations between major models.

An important model for time-dependent job processing times is that of *step-deterioration*, which was first proposed by Mosheiov [4], and suggested that the processing time of a job follows a step function of its starting time. More specifically, the actual processing times of the jobs that start after their deterioration-dates experience a step increase. The author focused on minimizing makespan with a single step-deterioration on a single- and multi-machine settings, presented $\mathcal{NP}$-completeness justification, integer programming formulation and provided a heuristic procedure. An abundance of studies was published subsequently, assuming common deterioration-date or job-specific

*Speaker, e-mail: msomer@huji.ac.il

deterioration-dates, various step-functions and machine settings; see Gawiejnowicz [2] and Strusevich and Rustogi [5].

Considering this wide research on step-deterioration, it is surprising that the complementary phenomenon, i.e. that of *step-learning* has yet to be discussed, let alone studied. Step-learning can be encountered in many real-life situations, e.g. when improved routines are assimilated in the production process, enhanced raw materials are utilized, faster equipment replaces outdated models, and, ultimately, when disruptive technology emerges and revolutionizes industry. Thus, the significance of incorporating step-learning into scheduling theory is two-fold, both theoretical and practical.

In the context of scheduling theory, there are two main approaches to formulate learning-effects, i.e., by time-dependent or by position-dependent job processing times. We focus on a single-machine scheduling setting with both *time-dependent* and *job-dependent* step-learning effect. Each job has its own *Learning-Date* (LD), such that if the starting time of a job is not smaller than this value, its actual processing time is reduced by a job-dependent factor. No idle time between consecutive jobs is permitted, an assumption which is justified in numerous manufacturing systems, due to the cost of stopping and renewing the production process. The objective function is minimum makespan. The problem is proved to be $\mathcal{NP}$-hard, and a pseudo-polynomial dynamic programming (DP) algorithm is introduced and tested. Our numerical tests indicate that medium size problems can be solved in a very reasonable running time. We also study the special case in which all jobs share a *Common Learning-Date*, denoted $CONLD$. This problem is $\mathcal{NP}$-hard as well, but a more efficient dynamic programming is proposed, and larger instances are solved to optimality.

2 Problem formulation

Formally, a set $\mathcal{J}$ containing n jobs is to be processed on a single machine, with the jobs ready for processing at time zero and no idle times and no preemption allowed. The basic (maximal) processing time of job $j \in \mathcal{J}$, is denoted by u_j and the reduced (minimal) processing time is denoted by v_j, such that $u_j, v_j \in \mathbb{Z}^+$. We also denote by $u_{\max} = \max_{j \in J} \{u_j\}$, the maximal basic processing time among all jobs.

For a given schedule, the starting time of job $j \in \mathcal{J}$, is denoted by S_j and the completion time of job $j \in \mathcal{J}$, is denoted by C_j. In this study, we focus on the makespan, i.e., the completion time of the last job to leave the production line, defined as $C_{\max} = \max_{j \in \mathcal{J}} \{C_j\}$.

We denote by $LD_j \in Z$ the learning-date of job $j \in \mathcal{J}$ and by $LD_{\max} = \max_{j \in \mathcal{J}} \{LD_j\}$, the maximal learning-date in set $\mathcal{J}$. If the starting time of job j is strictly less than LD_j, then its processing time is u_j, whereas if the starting time is equal to or greater than LD_j, its processing time is v_j. Eventually, the actual processing time of job $j \in \mathcal{J}$ is defined as

$$p_j = \begin{cases} u_j, \text{if } S_j < LD_j \\ v_j, \text{if } S_j < LD_j. \end{cases}$$

If the processing of job j starts before its job-dependent learning-date, LD_j, it is regarded as an early job, and otherwise if the processing starts at or after the LD_j, it is considered as late job. Consequently, we denote by $\mathcal{J}^E$ and $\mathcal{J}^L$, the subsets of early and late jobs, respectively, such that $\mathcal{J} = \mathcal{J}^E \cup \mathcal{J}^L$ and $\mathcal{J}^E \cap \mathcal{J}^L = \emptyset$.

Utilizing the standard 3-field notation of scheduling problems (Graham et al. [3]), the problem studied here, in short denoted by **Q1**, is

$$1 \left| LD_j, p_j \in \{u_j, v_j : u_j \leq v_j\} \right| C_{\max}.$$

We then focus on the special case of a common learning-date ($CONLD$), i.e., $LD_j = LD$, $j \in \mathcal{J}$. For this setting, the subset of jobs that start their production before LD are regarded as early jobs, whereas the subset of jobs that start their production exactly at or after LD are considered as late jobs. The $CONLD$ problem, in short denoted **Q2**, is the following

$$1 \left| LD_j = LD,\ p_j \in \{u_j, v_j : u_j \leq v_j\} \right| C_{\max}.$$

3 Our results

We first prove that problem **Q1** is $\mathcal{NP}$-hard. In fact, we prove that even the special case of a common learning-date (i.e., problem **Q2**) is $\mathcal{NP}$-hard.

Theorem 1. *Problem* **Q1** *is $\mathcal{NP}$-hard even for a common learning-date.*

Next, we introduce a pseudo-polynomial DP algorithm, thus establishing that problem **Q1** is $\mathcal{NP}$-hard in the ordinary sense. In order to do this, we prove a number of properties of an optimal schedule for the problem.

Property 1. *An optimal schedule exists such that all the early jobs are scheduled prior to the late jobs.*

Property 2. *An optimal schedule exists such that all the early jobs are scheduled according to Earliest Learning-Date first* (ELD) *rule.*

Property 3. *An optimal schedule exists such that all the late jobs are scheduled according to Earliest Learning-Date first* (ELD) *rule.*

The running time of the DP, denoted **DP1**, is provided in the following result.

Theorem 2. *The computational complexity of algorithm* $\mathbf{DP1}$ *is* $O\left(n\left(T^E_{\max}\right)^2\right)$, *where* $T^E_{\max}$ *denotes the maximal completion time of any subset of early jobs.*

A DP for problem $\mathbf{Q2}$, denoted $\mathbf{DP2}$, is introduced as well. Its running time is given in the following result.

Theorem 3. *The computational complexity of* $\mathbf{DP2}$ *is* $O\left(nu_{\max}T^E_{\max}\right)$.

An extensive numerical study was performed in order to evaluate the actual running times of each of algorithms $\mathbf{DP1}$ and $\mathbf{DP2}$ as a function of the input parameters. Instances of up to 175 jobs and 500 jobs were solved by $\mathbf{DP1}$ and $\mathbf{DP2}$, respectively. The worst-case running time of $\mathbf{DP1}$ with $n = 175$ did not exceed 46.1 seconds, while the worst-case running time of $\mathbf{DP2}$ with $n = 500$ did not exceed 12.9 seconds.

4 Future research

Future interesting and challenging topics might be dedicated to the extensions, either to multi-step learning, and to multi-machine settings.

References

[1] A. Azzouz, M. Ennigrou, L. Ben Said, Scheduling problems under learning effects: classification and cartography, *International Journal of Production Research*, 56 (2018), 1642–1661, doi: 10.1080/00207543.2017.1355576.

[2] S. Gawiejnowicz, *Models and Algorithms of Time-Dependent Scheduling*, Springer, Berlin-Heidelberg, 2020, doi: 10.1007/978-3-662-59362-2.

[3] R. L. Graham, 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** (1979), 287–326, doi: 10.1016/S0167-5060(08)70356-X.

[4] G. Mosheiov, Scheduling jobs with step-deterioration: minimizing makespan on a single-and multi-machine, *Computers & Industrial Engineering*, **28** (1995), 869–879, doi: 10.1016/0360-8352(95)00006-M.

[5] V. A. Strusevich, K. Rustogi, *Scheduling with Time-Changing Effects and Rate-Modifying Activities*, Springer, Cham, 2017, doi: 10.1007/978-3-319-39574-6.

The Third International Workshop on Dynamic Scheduling Problems
Adam Mickiewicz University, Poznań, July 5th – 6th, 2021
DOI: 10.14708/isbn.978-83-951298-6-5p51-56

Consideration of routine losses due to intermittent production in flow shop scheduling

Frederik Ostermeier*
BMW Group in Munich, Germany
Institute for Production Systems, Technical University in Dortmund, Germany

Keywords: learning effects, forgetting effects, scheduling, flow shop

1 Introduction

In manual flow shops workers learn and aquire routine by repetitively processing units of the same product leading to a decrease of processing times with increasing number of units processed. However, routine follows an ongoing learn-forget-relearn cycle as it can be lost whenever the production of a product is interrupted (Givi et al. [3]). Forgetting during the interruption leaves the worker at an earlier point of the learning curve from which he/she begins relearning after the interruption (Bailey [1]). Scheduled rest breaks, times for maintenance activities, but also times at which a worker performs tasks at different stations or on different products are relevant interruptions within a shift (Globerson and Levin [4]).

Forgetting models used in short-term scheduling primarily focused on modeling the impact of rest breaks scheduled within a shift (Nembhard and Uzumeri [7], Givi et al. [3]). In the context of dual-resource-constrained production environments some authors considered worker transfers between different stations as causes for forgetting (Jaber at al. [5]). Less research dealt with forgetting due to the production of various products in an intermixed sequence on a line. However, observations in industrial practice clearly indicate that product-dependent routine is at least partially lost if workers process different products alternately. Such intermittent production of products is quite typical in manual mixed-model assembly lines where different products can be launched in any sequence. Consequently there exists a need to model the losses in routine due to intermittent production.

This work aims at providing a modeling formulation for routine losses in intermittent flow shop production that can be used in existing short-term learn-forget-relearn models such as those by McCreery and Krajewski [6], Yue at al. [9] and Ostermeier [8]. Not modeled are transfer effects between shifts or between products from prior production periods as the focus are the effects present within one single shift.

*Speaker, e-mail: frederik.ostermeier@bmw.de

2 Modeling routine losses

To model routine losses we build on the processing time formula derived by Ostermeier [8]. As in (1) the processing time is the sum of a learning term $f_{learning,p}(n_p)$ depending on the current volume n_p processed of product p and a fatigue term $f_{fatigue}(t)$ that depends on the time t already passed within a shift, i.e.

$$p_{ij}(n_p, t) = f_{learning,p}(n_p) + f_{fatigue}(t). \tag{1}$$

This work's fatigue term denoted in (2) is identical to the one presented in [8]. Fatigue can lead to processing time increases from the normal processing time p_{i_j} up to δ percent if the degree of fatigue $F(t)$ reaches its maximum of 100%. The degree of fatigue increases with time in dependence of a fatigue rate FR as depicted in (3). During rest breaks, recovery with a recovery rate RR takes place, leading to a remaining degree of fatigue of $R(t_1, l_b)$ after the break as denoted in (4). With the recovery function we model the impact of rest breaks b of duration l_b starting at time t_1 and ending at time t_2. To ensure that after a break fatigue increases again with the right slope of the fatigue curve, the effective time has to be adjusted by $t - t_2 + t_0$ with t_0 as in (5) being the time at which the original fatigue curve has the exact slope that reflects the degree of fatigue remaining after the rest break at t_2. Thus, the fatigue term is defined stepwise for intervals before a rest break ($t \leq t_1$), during a rest break ($t_1 < t \leq t_2$) and after a rest break ($t \geq t_2$), i.e.

$$f_{fatigue}(t) = \begin{cases} p_{ij} \cdot \delta \cdot F(t), t \leq t_1 \\ p_{ij} \cdot \delta \cdot R(t_1, l_b), t_1 < t \leq t_2 \\ p_{ij} \cdot \delta \cdot F(t - t_2 + t_0), t \geq t_2 \end{cases} \tag{2}$$

$$F(t) = 1 - e^{-FR \cdot t} \tag{3}$$

$$R(t_1, l_b) = F(t_1) \cdot e^{-RR \cdot t} \tag{4}$$

$$t_0 = \lceil (\ln(1 - (1 - e^{-FR \cdot t_1}) \cdot e^{-RR \cdot l_b}))/ - FR \rceil \tag{5}$$

The learning term presented in (6) is altered in contrast to [8] to account for routine losses due to intermittent production of different products. $n_{p,2}$ captures the volume of product p at the last position that a product of this product has been produced and no longer the volume at the beginning of a rest break. Consequently, this work treats rest breaks as the only interruptions relevant for recovery, while interruptions relevant for forgetting can be both rest breaks and the times during which different products are processed. As in (6), the learning term consists of a learning function before an interruption ($n_p < n_{p,2}$) and a relearning term that considers forgetting after the interruption has occurred ($n_p \geq n_{p,2}$). The learning curve with product-dependent learning rate LR_p uses a minimum notation as done by Cheng and Wang [2] in order to guarantee that learning reaches a plateau after a volume $n_{p,0}$. After the interruption the effective volume relevant for learning has to be adjusted by $n_p - n_{p,2} + n_{p,1}$ in order to take the routine losses due to forgetting into account. $n_{p,1}$ constitutes the effective volume with the right slope on the original learning curve after the last interruption of production of product p. Following (7), $n_{p,1}$ is a function of $n_{p,2}$, the volume $x_{\Delta t}$ that could have been produced during the interruption of effective duration Δt computed as in (8), the learning rate LR_p and the forgetting rate VR.

$$f_{learning,p}(n_p) = \begin{cases} (\min\{n_p, n_{p,0}\})^{LR_p} \cdot p_{ij}, n_p < n_{p,2} \\ (\min\{n_p - n_{p,2} + n_{p,1}, n_{p,0}\})^{LR_p} \cdot p_{ij}, n_p \geq n_{p,2} \end{cases} \tag{6}$$

$$n_{p,1} = n_{p,2} \cdot {x_{\Delta t}}^{VR/LR_p} \tag{7}$$

$$x_{\Delta t} = \lfloor \Delta t / p_{ij} \rfloor \tag{8}$$

The main difference to [8] is that the effective duration of the interruption Δt is no longer merely the length of the break l_b but partially considers times during which different products are processed, as denoted by (9). Therefore, for all jobs positioned between the position $r_{n_{p,2}}$, being the last position at which a job of product p has been processed, and the sequence position before the current sequence position r we consider times $s_k - s_{k-1}$. For a job at position k, $s_k - s_{k-1}$ is the time span between the starting times of succeeding jobs and contains both processing times and potential blocking and starving times of the station, i.e. the elapsed time per job at which the product p of interest is not processed. This time is weighted with a product-dependent routine loss factor θ_p. If no routine is lost whenever a different product is produced, $\theta_p = 0$. If routine is lost, θ_p takes a value between 0 and 1. Thus, θ_p constitutes a lever that allows to count the elapsed time partially. It is product-dependent as the amount of forgetting may increase if products with dissimilar tasks are produced. To ensure that the desired θ_p is used for the job at sequence position k, ϑ_{kp} is introduced. $\vartheta_{kp} = 1$ if a unit of product p is processed at position k, 0 otherwise. In order to consider also times of rest breaks $\sum_{b=1}^{B} \omega_{kb} \cdot l_b \cdot (1 - \sum_{p=1}^{P} \theta_p \cdot \vartheta_{kp})$ adds the times associated with rest breaks that occur directly after processing of jobs at position k. ω_{kb} is 1, if break b takes place after the production of the job at sequence position k has started and before start of the job at position $k+1$. Note that the term $(1 - \sum_{p=1}^{P} \theta_p \cdot \vartheta_{kp})$ is used to ensure that the break times are not partially double-counted. As a break time in the interval is included in the starting time difference and weighted with θ_p, this fraction has to be subtracted:

$$\Delta t = \sum_{k=r_{n_{p,2}}+1}^{r} \left(\sum_{p=1}^{P} \theta_p \cdot \vartheta_{kp} \cdot (s_k - s_{k-1}) + \sum_{b=1}^{B} \omega_{kb} \cdot l_b \cdot (1 - \sum_{p=1}^{P} \theta_p \cdot \vartheta_{kp})\right). \tag{9}$$

With the proposed formulation the interruption duration relevant for forgetting is extended by an effective duration that accounts for the times at which different products

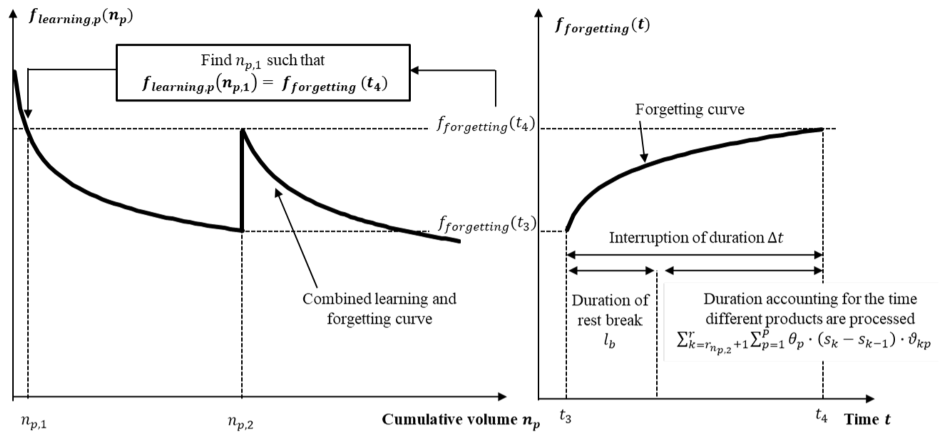

$f_{learning,p}(n_p)$
Find $n_{p,1}$ such that $f_{learning,p}(n_{p,1}) = f_{forgetting}(t_4)$
$f_{forgetting}(t_4)$
$f_{forgetting}(t_3)$
Combined learning and forgetting curve
$n_{p,1}$
$n_{p,2}$
Cumulative volume n_p
$f_{forgetting}(t)$
Forgetting curve
Interruption of duration Δt
Duration of rest break l_b
Duration accounting for the time different products are processed $\sum_{k=r_{n_{p,2}}+1}^{r} \sum_{p=1}^{P} \theta_p \cdot (s_k - s_{k-1}) \cdot \vartheta_{kp}$
t_3
t_4
Time t

Figure 1. Altered forgetting curve based on [8]

are processed. θ_p allows to count these times of production partially. The formulation can be incorporated into the models by McCreery and Krajewski [6] with a different k_n, by Yue at al. [9] with a different ϵ and by Ostermeier [8] with a different Δt. The impact of the incorporation of θ_p in the forgetting curve is illustrated in Fig. 1. The higher θ_p is, the higher is the counted fraction of the time elapsed for the production of different products. Routine is lost in the interval of length Δt from t_3 as time when the last unit of the product has been produced to the time t_4.

3 Numerical results

Numerical studies are carried out to show the appropriateness of the proposed formulation for the consideration of routine losses due to intermittent production. A simulation study is conducted based on data from an industrial mixed-model engine assembly line with seven products. Three different volume mixes are examined differing in the volume mix portion of the main product P1 (25%, 50%, 75%). The volume mix portion of the other six products is evenly distributed among them. The resulting heterogeneous, intermixed and homogeneous products mixes ensure that no special case is treated. For each of the mixes three different sequence types with no grouping and a maximum spacing of jobs, with groups of size four and with groups of size twelve are examined. The routine loss factor θ_p is gradually increased from 0 to 1. Further parameters of the processing time function are kept constant with $LR_p = LR = -0.057$, $n_{p,0} = n_p = 50, VR = 0.01, \delta = 0.15, FR = 0.0053$ and $RR = 0.003$. Fig. 2 depicts the results of the numerical studies for mean gross learning effects, net learning

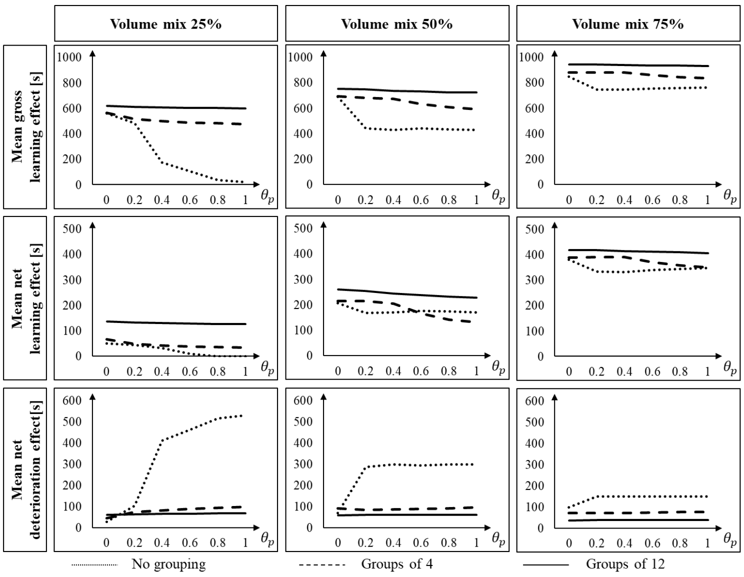


Figure 2. Mean learning and deterioration effects

effects and net deterioration effects. Mean gross learning effects are the average processing time reductions that a job would gain if only learning and forgetting are considered. Mean net gross learning and deterioration effects take into account the net processing time deviations from the normal processing time if next to learning and forgetting also fatigue and recovery are modeled. If a job exhibits a processing time reduction compared to the normal processing time this denotes a learning effect. Otherwise in case of a processing time increase a deterioration effect is counted.

As expected, mean gross and net learning effects decrease with increasing θ_p, whereas net deterioration effects increase. The magnitude of the effect depends strongly on the volume mix and the degree of grouping inherent in the sequence type. The more heterogeneous the volume mix becomes, the less strong are the gains due to learning effects and they more prone is the sequence to deterioration effects. The higher the degree of grouping within the sequence, the more independent the sequence becomes in terms of learning and deterioration effects as a smaller number of product switches is conducted and the runs of consecutive production of the same product are longer. This restricts the occurrence of forgetting. An interesting observation can be made for net learning effects in the 50% volume mix case. As the times during which forgetting occurs for product P1 occur are partially of a longer duration for groups of four than for no grouping where every second product is of product P1, they deliver similar net learning effect values.

4 Future research

Future research may concern how adequate values for θ_p in actual industrial cases can be derived. In general, empirical field studies in industrial practice and focused laboratory experiments are required to assess how much routine is actually lost whenever a different product is processed. These studies have to explore which factors differentiating products in mixed-model lines affect the routine loss and how adequate values for θ_p can be derived based on the factors.

References

[1] C. D. Bailey, Forgetting and the learning curve, *Management Science*, **35** (1989), 340–352, doi: 10.1287/mnsc.35.3.340.

[2] T. C. E. Cheng, G. Wang, Single machine scheduling with learning effect considerations, *Annals of Operations Research*, **98** (2000), 273–290, doi: 10.1023/A:1019216726076.

[3] Z. S. Givi, M. Y. Jaber, W. P. Neumann, Production planning in DRC systems considering worker performance, *Computers & Industrial Engineering*, **87** (2015), 317–327, doi10.1016/j.cie.2015.05.005.

[4] S. Globerson, N. Levin, Incorporating forgetting into learning curves, *International Journal of Operations & Production Management*, **7** (1987), 80–94, doi: 10.1108/eb054802.

[5] M. Y. Jaber, Z. S. Givi, W. P. Neumann, Incorporating human fatigue and recovery into the learning-forgetting process, *Applied Mathematical Modelling*, **37** (2013), 7287–7299, doi: 10.1016/j.apm.2013.02.028.

[6] J. K. McCreery, L. J. Krajewski, Improving performance using workforce flexibility in an assembly environment with learning and forgetting effects, *International Journal of Production Research*, **37** (1999), 2031–2058, doi: 10.1080/002075499190897.

[7] D. A. Nembhard, M. V. Uzumeri, Experiential learning and forgetting for manual and cognitive tasks, *International Journal of Industrial Ergonomics*, **25** (2000), 315–326, doi: 10.1016/S0169-8141(99)00021-9.

[8] F.F. Ostermeier, The impact of human consideration, schedule types and product mix on scheduling objectives for unpaced mixed-model assembly lines, *International Journal of Production Research*, **58** (2020), 4386–4405, doi: 10.1080/00207543.2019.1652780.

[9] H. Yue, J. Slomp, E. Mollemann, D. J. Van der Zee, Worker flexibility in a parallel dual resource constrained job shop, *International Journal of Production Research*, **46** (2008), 451–467, doi: 10.1080/00207540601138510.

The Third International Workshop on Dynamic Scheduling Problems
Adam Mickiewicz University, Poznań, July 5th – 6th, 2021
DOI: 10.14708/isbn.978-83-951298-6-5p57-62

Dual-criticality scheduling on non-preemptive, dynamic processors using RL agents

Nourhan Sakr*
American University in Cairo, Egypt

Youssef Hussein
American University in Cairo, Egypt

Karim Farid
American University in Cairo, Egypt

Keywords: mixed-criticality scheduling, varying-speed processors, reinforcement learning

1 Introduction and related work

Real-time embedded systems have stringent non-functional requirements on cost, weight, and energy that give rise to the study of *mixed-criticality* (MC) systems, where functionalities of different *criticality levels* are consolidated into a shared hardware platform (Barhorst et al. [1], Burns and Davis [6]). The literature discusses the schedulability of MC systems under various conditions and objectives (Baruah et al. [3], Gu et al. [7], Baruah et al. [4]). In this work, we study a dual-criticality, non-preemptive system with a varying-speed uniprocessor.

This varying-speed processor makes MC scheduling dynamic: In real-time systems, it cannot be predetermined if, when or for how long the processor would *degrade*, i.e. its speed would drop. Under speed stochasticity, it is critical to guarantee the running of high (HI) criticality jobs by their deadlines, sometimes even at the cost of not running low (LO) criticality jobs at all, when operating under degradation.

Scheduling MC systems non-preemptively (even without degradation) is $\mathcal{NP}$-hard (Lenstra et al. [9]). Baruah and Guo [5] model an LP to preemptively schedule dual-criticality jobs on a varying-speed processor. Agarwal and Baruah [2] further discuss the online nature of the problem and its intractability. We agree with the authors that MC scheduling is inherently an online problem, as it better depicts real-time scheduling and the dynamic nature of this problem. Therefore, we devise deep reinforcement learning (deep RL or DRL) to tackle the problem presented by Baruah and Guo [5], under both the offline and online setting.

*Speaker, e-mail: n.sakr@columbia.edu

2 Problem formulation

We model the system functionalities by a set of independent jobs J, each job j is defined by its release date r_j, deadline d_j, processing time p_j (representing worst-case execution time) and criticality level $\chi_j \in \{\text{LO}; \text{HI}\}$, describing a dual-criticality system of low and high criticality jobs. A speed-v processor runs a job j in $\frac{p_j}{v}$ time units.

The system assumes two modes of operation: *normal* ($v = 1$) and *degradation* mode ($v < 1$). We assume a self-monitoring system that immediately knows when a degradation occurs during runtime. The degradation speed v is observed then. An MC instance is a set of MC jobs J that are schedulable on a varying-speed processor. According to Lemma 1 in Baruah and Guo [5], J is schedulable if an earliest deadline first (EDF) policy schedules all jobs on a speed-1 processor and all HI jobs on a speed-$v_{\min}$ processor. That is, there is a degradation bound $v_{\min}$, below which J would be no longer schedulable. Finally, a *correct schedule* runs all jobs by their deadlines as long as $v = 1$ and guarantees all HI jobs to meet their deadlines regardless of the speed.

3 Model and evaluation

Our environment is modeled as a Markov decision process (MDP). At each timestep t, a reinforcement learning (RL) agent interacts with an environment by receiving an observation o_t from a state space S and taking a corresponding action a_t from a given action space A. This action is (later) rewarded or penalized (i.e. negative reward) using a reward function $r(a_t, o_t)$. The agent's goal is to maximize the reward in an *episode*.

Episode, states and actions. We consider one episode to be a series of decisions taken until all jobs in J either run (and complete by the deadline) or expire (cannot meet the deadline). At a given time t, an observation o_t is a buffer of jobs $B_t \subseteq J$ and the processor speed v_t. Each job in the buffer is represented by its static tuple (r_j, d_j, p_j, χ_j), and three status parameters that indicate whether the job was released, expired (i.e. missed its deadline) or was scheduled to run. Given o_t, the action function produces a selection for the index of the job to be scheduled at t. We next outline three environments that help us stage the transition into our target online problem.

Offline environment. In an offline setting, all decisions are taken at $t = 0$ when jobs are known but degradation cannot be observed. Jobs can be scheduled at anytime $t \in [\min_j\{r_j\}, \max_j\{d_j - p_j\}]$. We, therefore, set $B_t = J$ and $v_t = 1, \forall t$, thereby assuming normal operation. The the RL agent schedules jobs sequentially, so all job binary status parameters are updated before every new decision. This sequential process also ensures non-preemption, i.e. when a job j is selected for t, the next decision is made for time $t + p_j$. During this time window, some jobs may expire before selection.

Varying buffer (VB). In an online setting, not all jobs are known to the agent a priori. This constraint introduces a modeling issue since the buffer size should be fixed for each observation, yet the number of observable jobs changes at each time step. We create the VB environment as a transitional stage, where we fix the size of the buffer B_t at n. If $|J| > n$, we add the jobs with least laxities to the buffer. (The laxity of a job, defined as $l_j = d_j - p_j$, gives early warnings on expiry. When $l_j \leq t$, this means that even if the agent schedules job j at t, the time would not be sufficient for it to complete before d_j. Thus, the agent is penalized for choosing such jobs.) Otherwise, we add all jobs plus a number of dummy jobs that would bring the buffer size to n. To avoid scheduling dummy jobs, we label them as expired and set their criticality to LO.

Online environment. This environment is closest to mimicking real-time scheduling. Our agent is now able to observe the processor speed, so we relax the constraint on $v_t = 1$. We build B_t similar to that of the VB environment but impose two additional restrictions: To be added to B_t, a job must have been released ($r_j \leq t$) and have enough time to complete ($l_j \geq t$). As such, all dynamic features of our system are captured.

To the best of our knowledge, we are the first to use RL agents for scheduling MC systems and, hence, have no previous RL-based benchmarks. We remedy this limitation by using the staging process above for testing and validation. Although, we are ultimately interested in modeling the online problem (see Sect. 1), we start with an offline environment in order to leverage available offline benchmarks. Once validated, our offline environment became the baseline for the VB environment and subsequently the VB became the benchmark for the online environment. In this abstract, we highlight the reward function and results of the online environment only.

Online reward function. We use the *number of completed jobs* as the main metric of comparison. A secondary goal is to complete all HI jobs in J. The online agent receives a reward equivalent to the number of executed jobs at the end of each episode. This design achieves the highest (over episode) average reward, emphasizing the priority of running HI jobs. Otherwise, the agent is penalized for selecting jobs that have expired or have run before. We test other reward designs, such as applying rewards or penalties instantaneously rather than at the end of an episode, or scaling the reward by v (for higher rewards in case of degradation), but they were all deemed less successful.

Algorithm choice. In general, DRL allows more unstructured input to the agent with larger data. We tested RL algorithms from RLLib and Stable Baselines, where Ape-X DQN (Horgan et al. [8]) produced the best results in scheduling jobs and learning EDF behavior.

4 Our results

Simulation. We generate various instances of J and processor speeds. The details of data generation are masked from this abstract but rely heavily on a modification of Baruah and Guo [5]. It is worth noting that each set J is verified for schedulability before being fed to the RL agent. Our hardware limitations cap the size of our instances, $|J|$ at 30. We also study the settings needed for the warm-up period, number of episodes (steps) and run 500 iterations per experiment.

Evaluation criteria. We evaluate environments based on the general reward evaluation, which assesses the agent behavior against the mean reward over a series of episodes, i.e. the average number of executed jobs per instance. Note that the maximum reward that the agent can receive in any episode i is $|J_i|$, assuming all jobs complete without expiring. We additionally conduct a degradation analysis, which is essential to understanding the sensitivity of the learned policy to the degradation speed.

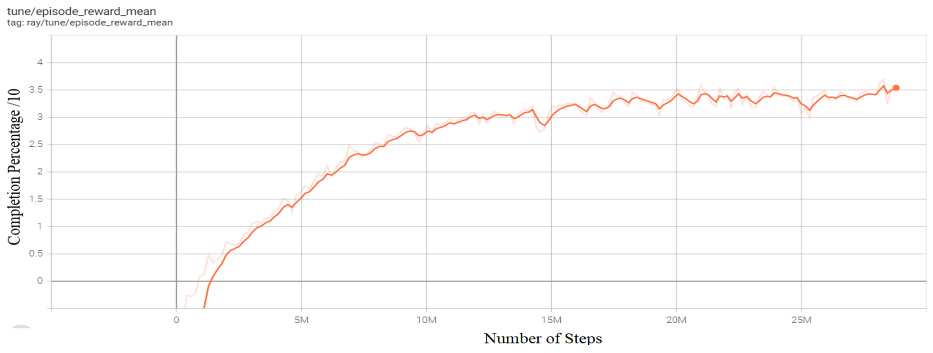


Figure 1. General reward evaluation results of the online environment.

Preliminary results. After its success in the offline environment, Ape-X was capable of correctly scheduling around 30-40% of the provided instances when introduced to the online environment (see Fig. 1). The limitations come from two sources: a problem-specific challenge and a hardware challenge. The online environment is non-clairvoyant and imposes unpredictable degradation in the performance. We believe that this can be improved by augmenting the RL agent with a prediction module that can make forecasts on the expected processor speeds. On the hardware side, we are limited by resources (14 CPUs working in parallel and one empowered GPU), which did not allow us to run large scale examples. The results produced in Fig. 1 are based on instances that are 30 jobs each and a buffer size set at $n = 10$. We believe that obtaining more resources that can run larger examples will yield better training and testing results, as the agent has more data to learn. However, these preliminary results show a promising approach that is worth discussing.

Degradation analysis. When the processor degrades, it is expected that the agent prioritizes completing all HI jobs. The model should learn to sacrifice LO jobs to free resources for the HI ones. We measure the sensitivity of the agent to degradation speeds: In a total of 10,000 job scenarios(episodes), we label the scenarios where the agent completes all HI jobs as successful. These scenarios are fed to the RL agent, and the agent's performance is assessed under varying speeds $v \in [0.1, 1]$. Fig. 2 shows that online environments perform slightly better than the other two environments and that lower speeds yield decreasing performances until it plateaus at almost $15 - 20\%$. This phenomenon is likely attributed to the ability of the RL agent to now observe the processor's speed and dynamically adapt to any disruptions exposing the system.

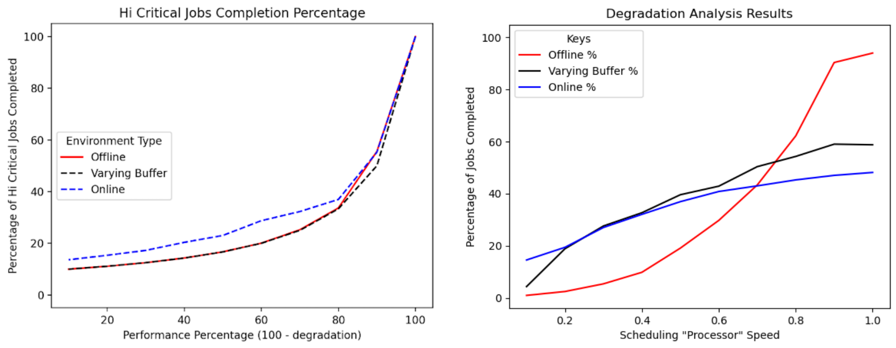


Figure 2. Degradation analysis results for comparing the three environments

5 Future research

For future work, we wish to improve our performance using larger examples and contrast our results against the OCBP benchmark proposed by Agarwal and Baruah [2]. We also wish to extend our work to other variants of the problem, such as preemptive scheduling, multiprocessor systems with resource sharing, or integral data generation. Furthermore, we plan to study other evaluation metrics, conduct an error analysis on unsuccessful schedules, as well as assess the benefit of augmenting a forecasting module for predicting speeds a priori.

References

[1] J. Barhorst, T. Belote, P. Binns, J. Hoffman, J. Paunicka, P. Sarathy, J. Scoredos, P. Stanfill, D. Stuart, R. Urzi, A research agenda for mixed-criticality systems, white paper, 2009.

[2] K. Agarwal, S. Baruah, Intractability issues in mixed-criticality scheduling, *30th EuroMicro Conference on Real-Time Systems*, 2018, article no. 11, 11:1–11:21, doi: 10.4230/LIPIcs.ECRTS.2018.11.

[3] S. Baruah, V. Bonifaci, G. D'Angelo, H-H. Li, A. Marchetti-Spaccamela, N. Megow, L. Stougie, Scheduling real-time mixed-criticality jobs, *IEEE Transactions on Computers*, **61** (2011), 1140–1152, doi: 10.1109/TC.2011.142.

[4] S. Baruah, A. Easwaran, Z. Guo, Mixed-criticality scheduling to minimize makespan, *36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, 2016, article no. 7, 7:1–7:13, doi: 10.4230/LIPIcs.FSTTCS.2016.7.

[5] S. Baruah, Z. Guo, Mixed-criticality scheduling upon varying speed processors, *34th Real-Time Systems Symposium*, 2013, 68–77, doi: 10.1109/RTSS.2013.15.

[6] A. Burns, R. Davis, Mixed criticality systems-a review, Department of Computer Science, University of York, Tech. Rep. 172, 2016.

[7] C. Gu, N. Guan, Q. Deng, W. Yi, Improving OCBP-based scheduling for mixed-criticality sporadic task systems, *19th International Conference on Embedded and Real-Time Computing Systems and Applications*, 2013, 247–256, doi: 10.1109/RTCSA.2013.6732225.

[8] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. van Hasselt, D. Silver, Distributed prioritized experience replay, arXiv preprint arXiv:1803.00933, 2018.

[9] J. K. Lenstra, A. H. G. Rinnooy Kan, P. Brucker, Complexity of machine scheduling problems, *Annals of Discrete Mathematics*, **1** (1977), 343–362, doi: 10.1016/S0167-5060(08)70743-X.

The Third International Workshop on Dynamic Scheduling Problems
Adam Mickiewicz University, Poznań, July 5th – 6th, 2021
DOI: 10.14708/isbn.978-83-951298-6-5p63-69

A lower bound for sequentially placing boxes at the moving assembly line to minimize walking time

Helmut A. Sedding*
ZHAW Institute of Data Analysis and Process Design, Operations Research and Operations Management Group in Winterthur, Switzerland

Keywords: assembly line, line side placement, walking time, moving conveyor, time-dependent scheduling

1 Introduction

An automotive assembly line typically produces a variety of m different car models M in a model-mix (Thomopoulos [9]). Each model $i \in M$ has a production share $r_j \in [0,1]$ such that $\sum_{i \in M} r_j = 1$. The assembly line is divided into stations; we consider only one of them. Then, our objective is to increase its productivity, which is measured by the weighted average makespan. The corresponds to a minimization of $\phi = \sum_{i \in M} r_j C_{i,\max}$, where $C_{i,\max}$ denotes the makespan of model $i \in M$ at the the station. We achieve this through a placement optimization that reduces nonproductive walking times, which belongs to tactical logistics planning (Boysen [1]).

The station's worker assembles a product of model $i \in M$ by processing a nonpermutable list of n_i jobs $J_i = (i,1), (i,2), \ldots, (i,n_i)$. Set $J = J_1 \cup \cdots \cup J_m$ denotes the union of all $n = |J|$ jobs of the worker. At the start of each job (i,j), the worker needs to fetch necessary components from a corresponding box, which we place at $\pi_{i,j}$ along the line. Then the walking time is calculated like as in Sedding [5] by the time-dependent, piecewise-linear function $f_{i,j}(t) = \max\{-a \cdot (t - \pi_{i,j}),\ b \cdot (t - \pi_{i,j})\}$, given slopes $0 \leq a \leq 1, b \geq 0$. Afterwards, the worker proceeds to assemble the job's components, which takes the constant assembly time $\ell_{i,j} \in \mathbb{Q}^+$. Thus, the job's processing time is $p_{i,j}(t) = f_{i,j}(t) + \ell_{i,j}$. Its completion time is $C_{i,j}(t) = t + p_{i,j}(t)$. All jobs of a model $i \in M$ are processed without idle time, starting at 0. Thus, the model's makespan can be calculated by the recurrence $C_{i,\max} = C_{i,n_i}(\cdots(C_{i,1}(0))\cdots)$.

The box of a job $(i,j) \in J$ is given a width $w_{i,j} \in \mathbb{N}$. We are able to decide its position $\pi_{i,j}$ such that the boxes are placed side-by-side along the line. Hence, the boxes are placed in a sequence between 0 and $W = \sum_{(i,j) \in J} w_{(i,j)}$. It follows that $\pi_{i,j} \in [0, W - w_{(i,j)}]$.

*Speaker, e-mail: helmut.sedding@zhaw.ch

Minimizing the objective ϕ by setting the box positions $\{\pi_{(i,j)}\}_{(i,j)\in J}$ is an $\mathcal{NP}$-hard problem, since it includes the $\mathcal{NP}$-hard single-model case $m = 1$ (Sedding [5]). In this abstract, we describe a Lagrangian relaxation based lower bound for ϕ that accepts partial solutions, and an algorithm to solve it in $\mathcal{O}(n^2)$ time. This provides the core of a branch and bound procedure introduced in the author's dissertation [7].

2 Related literature

The case $m = 1$ is studied by Sedding [5]. As well, it uses time-dependent processing times. Even though the job sequence is fixed, it is in a close relationship to time-dependent scheduling problems, on which Gawiejnowicz [3] gives a recent review. Such a problem allows for a variable job sequence. Note this is the case during operative planning, where the box positions are fixed while the assembly job sequence is variable, see Sedding [6] and, for walking in the middle of a job, Jaehn and Sedding [4].

3 Mixed integer linear program

The following mixed integer linear program places the boxes at a discrete number of integral positions along the line. If the binary assignment variable $x_{i,j,s}$ for job $(i, j) \in J$, $s = 0, \dots, W - 1$ is set to one, then its box is at $\pi_{i,j} = s$. Although this position is limited to $s \leq W - w_{(i,j)}$, we just limit it by $s \leq W - 1$, which is necessary for later results. The job's completion time C_{ij} sums the processing times so far, including walking time $\varpi_{i,j}$ with deviation $\delta_{i,j}$ between job start time and box position.

$$\phi^* = \min \sum_{i\in M} r_i\, C_{i,n_i} \tag{1a}$$

subject to

$$C_{i,j} = \sum_{k=1,\dots,j} \ell_{i,k} + \varpi_{i,k}, \qquad (i,j) \in J, \tag{1b}$$

$$\varpi_{i,j} \geq -a \cdot \delta_{i,j}, \qquad (i,j) \in J, \tag{1c}$$

$$\varpi_{i,j} \geq b \cdot \delta_{i,j}, \qquad (i,j) \in J, \tag{1d}$$

$$\max\{-a(C_{i,j-1} - (W-1)),\, b\,C_{i,j-1}\} \geq \varpi_{i,j} \geq 0, \qquad (i,j) \in J, \tag{1e}$$

$$\delta_{i,j} = C_{i,j-1} - \pi_{i,j}, \qquad (i,j) \in J, \tag{1f}$$

$$\pi_{i,j} = \sum_{s=0,\dots,W-w_{i,j}} x_{i,j,s} \cdot s, \qquad (i,j) \in J, \tag{1g}$$

$$\sum_{s=0,\,\dots,\,W-w_{i,j}} x_{i,j,s} = 1, \qquad (i,j) \in J, \tag{1h}$$

$$\sum_{(i,j)\in J,\, s'=\max\{0,\, k-w_{i,j}+1\},\,\dots,\, s} x_{i,j,s'} \leq 1, \qquad s = 0,\dots,W-1, \tag{1i}$$

$$x_{i,j,s} \in \{0,1\}, \qquad (i,j) \in J, s = 0,\dots,W-1. \tag{1j}$$

4 Lower bound by Lagrangian relaxation

We describe a lower bound on ϕ^* in the following. Note that to employ it in a branch and bound procedure, it accepts a partial solution (a *partial box placement*), which provides the positions $\{\pi_j\}_{j\in J_\mathrm{F}}$ for set of *fixed jobs* $J_\mathrm{F} \subseteq J$ placed between 0 and $F = \sum_{j\in J_\mathrm{F}} w_j$. This sets the values for their corresponding assignment variables $x_{i,j,s}$. In particular, $x_{i,j,s} = 0$ for $(i,j) \in J_\mathrm{F}$, $s = F,\dots,W-1$. The remaining set of *open jobs* $J_\mathrm{O} = J \setminus J_\mathrm{F}$ are to be placed between F and W. Therefore, $x_{i,j,s} = 0$ for $(i,j) \in J_\mathrm{O}$, $s = 0,\dots,F-1$.

To devise a lower bound, we perform a Lagrangian relaxation of constraint (1c) and constraint (1d). This introduces the corresponding Lagrangian multipliers $\lambda_{i,j} \geq 0$, $\lambda'_{i,j} \geq 0$ for $(i,j) \in J$. Then, the Lagrangian problem is $\phi^*_\mathrm{Lagr}(L) = \min \phi_\mathrm{Lagr}$ with

$$\phi_\mathrm{Lagr} = \sum_{i\in M} r_i\, C_{i,n_i} + \sum_{(i,j)\in J} \lambda_{i,j}\,(-a\delta_{i,j} - \varpi_{i,j}) + \lambda'_{i,j}\,(b\delta_{i,j} - \varpi_{i,j}) \tag{2}$$

subject to $L = \Big(\big(\lambda_{i,j}, \lambda'_{i,j}\big)\Big)_{(i,j)\in J} \geq 0$ and constraint (1b), (1g) to (1j), and (1e). Note that L can be optimized using a standard subgradient optimization, see Fisher [2]. Also, note that the lower bound inequality $\phi^*_\mathrm{Lagr}(L) \leq \phi^*$ holds for any L.

Substituting the completion time variable in (2) with (1b) gives

$$\begin{aligned}
\phi_\mathrm{Lagr} = &\sum_{(i,j)\in J} r_i\,(\ell_{i,j} + \varpi_{i,j}) + \big(b\lambda'_{i,j} - a\lambda_{i,j}\big)\,\delta_{i,j} - \big(\lambda_{i,j} + \lambda'_{i,j}\big)\,\varpi_{i,j} \\
\iff \phi_\mathrm{Lagr} = &\sum_{(i,j)\in J} \ell_{i,j}\zeta_{i,j} + \underbrace{\sum_{(i,j)\in J} \varpi_{i,j}\theta_{i,j}}_{\psi} + \underbrace{\sum_{(i,j)\in J} \big(a\lambda_{i,j} - b\lambda'_{i,j}\big)\,\pi_{i,j}}_{\gamma}
\end{aligned}$$

$$\text{with constants} \quad \zeta_{i,j} = r_i + \sum_{k=j+1,\dots,n_i} \big(b\lambda'_{i,k} - a\lambda_{i,k}\big), \qquad (i,j) \in J,$$

$$\text{and} \quad \theta_{i,j} = \zeta_{i,j} - \big(\lambda_{i,j} + \lambda'_{i,j}\big), \qquad (i,j) \in J.$$

Observe that the walking time and box placement variables occur only in different constraints.

Property 4.1. *In the Lagrangian problem, the walking time variables* $\varpi_{i,j}$*,* $(i,j) \in J$*, and box position variables* $\pi_{i,j}$*,* $(i,j) \in J_{\mathrm{O}}$*, are independent.*

Thus, it is possible to separately optimize walking times and box positions. This gives us the partial objective $\psi = \sum_{(i,j)\in J} \varpi_{i,j}\theta_{i,j}$ for the walking times and $\gamma = \sum_{(i,j)\in J} \left(a\lambda_{i,j} - b\lambda'_{i,j}\right) \pi_{i,j}$ for the box positions.

4.1 Optimizing box position values

The open job's boxes are to be placed in a sequence between F and W. In γ, each box $(i,j) \in J_{\mathrm{O}}$ contributes with $\left(a\lambda_{i,j} - b\lambda'_{i,j}\right) \pi_{i,j}$. Hence, we get a classic total weighted completion time scheduling problem of the boxes (as jobs), which is solved in polynomial time by sorting them (Smith [8]).

Lemma 4.2. *Optimal* $\pi_{i,j}$*,* $(i,j) \in J_{\mathrm{O}}$*, are obtained by sequencing the boxes in* J_{O} *nonincreasingly by*

$$\frac{a\lambda_{i,j} - b\lambda'_{i,j}}{w_{i,j}}.$$

Thus for $n_{\mathrm{O}} = |J_{\mathrm{O}}|$, optimal box position values are attained in $\mathcal{O}(n_{\mathrm{O}} \log n_{\mathrm{O}})$ time.

4.2 Optimizing walking time values

The walking time variables $\varpi_{i,j}$, $(i,j) \in J$, are optimized by minimizing ψ. For each $(i,j) \in J$, the value range of $\varpi_{i,j}$ is limited by constraint (1e). Substituting $C_{i,j-1}$ with constraint (1b) gives, with constants $q_{i,j} = \sum_{k=1,\dots,j-1} \ell_{i,k}$ and $q'_{i,j} = q_{i,j} - W + 1$, the range

$$0 \le \varpi_{i,j} \le \max\left\{-a\left(1 - W + \sum_{k=1,\dots,j-1} \ell_{i,k} + \varpi_{i,k}\right), b\left(\sum_{k=1,\dots,j-1} \ell_{i,k} + \varpi_{i,k}\right)\right\}$$

$$\iff 0 \le \varpi_{i,j} \le \max\left\{\underbrace{-a\left(q'_{i,j} + \sum_{k=1,\dots,j-1} \varpi_{i,k}\right)}_{\alpha_{i,j}}, \underbrace{b\left(q'_{i,j} + \sum_{k=1,\dots,j-1} \varpi_{i,k}\right)}_{\beta_{i,j}}\right\}. \quad (3)$$

We observe for any $i \in M$ and with increasing j from 1 to n_i that term $\alpha_{i,j}$ (as defined in (3)) is nonincreasing and term $\beta_{i,j}$ (as in (3)) is nondecreasing. Hence, we can find some $\kappa_i \in \{0, \dots, n_i\}$ such that $\alpha_{i,j} > \beta_{i,j}$ for all $j = \kappa_i + 1, \dots, n_i$. Given such a κ_i, we can replace constraint (3) by

$$0 \le \varpi_{i,j} \le \alpha_{i,j} \text{ if } j \le \kappa_i, \qquad 0 \le \varpi_{i,j} \le \beta_{i,j} \text{ if } j > \kappa_i.$$

Hence, depending on the value of κ_i, either of the two range constraints is active for job $(i,j) \in J$.

We replace the upper bound by an equality with slack variable $0 \le y_{i,j} \le 1$. Then,

$$0 \le \varpi_{i,j} = y_{i,j}\alpha_{i,j} \text{ if } j \le \kappa_i, \qquad 0 \le \varpi_{i,j} = y_{i,j}\beta_{i,j} \text{ if } j > \kappa_i. \tag{4}$$

Property 4.3. *Given κ_i, $i \in M$, of an optimal solution. If $\alpha_{i,j} < 0$ for any job $(i,j) \in J$ with $j \le \kappa_i$, then it is possible to decrease κ_i without changing the objective ψ such that $\alpha_{i,j} \ge 0$ for each job $(i,j) \in J$ with $j \le \kappa_i$.*

Proof. Given the described case, then $\kappa_i \ge 1$, and $\alpha_{i,\kappa_i} < 0$ because $\alpha_{i,j}$ is decreasing with j. Hence, $y_{i,\kappa_i} = \varpi_{i,\kappa_i} = 0$ to fulfill constraint (4).

Let us decrease κ_i by one. Then, it is possible to leave $y_{i,\kappa_i+1} = \varpi_{i,\kappa_i+1} = 0$ in the solution. Thus, ψ remains unchanged. We repeat this step until $\alpha_{i,\kappa_i} \ge 0$. □

Corollary 1. *An optimum solution exists where $\alpha_{i,j} \ge 0$ holds for each $(i,j) \in J$ with $j \le \kappa_i$.*

Lemma 4.4. *For each $i \in M$, let κ_i^* be the minimum value for κ_i that permits an optimum solution. Then, there exists such a solution where for each job $(i,j) \in J$ there is*

$$y_{i,j} = \begin{cases} 0, \textit{ if } \theta_{i,j} + \sum\limits_{k=j+1,\dots,n_i} y_{i,k}c_{i,k}\theta_{i,k} > 0, \\ 1, \textit{ else} \end{cases} \quad \textit{with } c_{i,k} = \begin{cases} -a, \textit{ if } k \le \kappa_i^*, \\ b, \quad \textit{else.} \end{cases} \tag{5}$$

Proof. For each model $i \in M$, we show this by induction for $j = n_i, \dots, 1$. Then,

$$\varpi_{i,j} = y_{i,j} \cdot \underbrace{c_{i,j}\left(d_{i,j} + \sum_{k=1,\dots,j-1} \varpi_{i,k}\right)}_{\ge 0} \tag{6}$$

$$\text{with} \quad c_{i,j} = \begin{cases} -a, & \text{if } j \le \kappa_i^*, \\ b, & \text{if } j > \kappa_i^*, \end{cases} \qquad d_{i,j} = \begin{cases} q'_{i,j}, & \text{if } j \le \kappa_i^*, \\ q_{i,j}, & \text{if } j > \kappa_i^*. \end{cases} \tag{7}$$

By choice of κ_i and use of Property 4.3, the slack variable $y_{i,j}$ multiplies a nonnegative value. Hence, $\varpi_{i,j}$ is nonnegative. Moreover, $\varpi_{i,j}$ influences $\varpi_{i,k}$ for each $k = j+1, \dots, n_i$ unless $y_{i,k} = 0$. Thus, $\varpi_{i,j}$ contributes to the objective ψ not only with factor $\theta_{i,j}$, but moreover via $\varpi_{i,k}$ with factor $y_{i,k}c_{i,k}\theta_{i,k}$. The total contribution of $\varpi_{i,j}$ to ψ is thus with factor $\theta_{i,j} + \sum_{k=j+1,\dots,n_i} y_{i,k}c_{i,k}\theta_{i,k}$. If this factor is positive, then the lowest slack value $y_{i,j} = 0$ minimizes ψ. If it is negative, then the highest slack value $y_{i,j} = 1$ is optimal. If the factor is zero, any value for $y_{i,j}$ is optimal. □

Let $\overline{\kappa_i} \in \{0, \dots, n_i\}$ for each $i \in M$ be the maximum κ_i for which inequality $-aq'_{i,\overline{\kappa_i}} \geq 0$ holds.

Property 4.5. *An optimum solution exists where $\kappa_i \leq \overline{\kappa_i}$ for each $i \in M$.*

Proof. Assume we are given an optimum solution. Naturally, all walking time variables $\varpi_{i,j}$, $(i, j) \in J$, are nonnegative. For each $i \in M$, both $\sum_{k=1,\dots,j-1} \varpi_{i,k}$ and $q'_{i,j}$ are nondecreasing with respect to j, while $-aq'_{i,j}$ is nonincreasing. Hence if $-aq'_{i,\kappa'_i} < 0$ for any $(i, \kappa'_i) \in J$ with $\kappa'_i \leq \kappa_i$, then $\alpha_{i,j} < 0$ for $j = \kappa'_i, \dots, \kappa_i$. However, by Property 4.3, it is possible to set $\kappa_i < \kappa'_i$ such that the solution is optimum and $-aq'_{i,j} < 0$ as well as $\alpha_{i,j} \geq 0$ hold for any $(i, j) \in J$ with $j \leq \kappa_i$. □

The above results allow us to describe an algorithm to minimize the walking time variables. In an outer loop, the algorithm sets $\kappa_i = 0, \dots, \overline{\kappa_i}$ for $i \in M$. Given κ_i, recurrence (5) is used to set $y_{i,j}$ for each $(i, j) \in J$, which also sets $\varpi_{i,j}$ and objective value ψ. The smallest obtained objective value is optimal. This takes quadratic time in terms of n_i. Over all m models, the worst case runtime is $\mathcal{O}\big(\sum_{i \in M} n_i^2\big) \leq \mathcal{O}\big(n^2\big)$.

Together with the algorithm in Lemma 4.2 we are thus able to find a solution.

Theorem 4.1. *An optimum solution to $\phi^*_{Lagr}(L)$ can be computed in $\mathcal{O}\big(n^2\big)$ time.*

References

[1] N. Boysen, S. Emde, M. Hoeck, M. Kauderer, Part logistics in the automotive industry: Decision problems, literature review and research agenda, *European Journal of Operational Research*, **242** (2014), 107–120, doi: 10.1016/j.ejor.2014.09.065.

[2] M. L. Fisher, The Lagrangian relaxation method for solving integer programming problems, *Management Science*, 50 (2004), 1861–1871, doi: 10.1287/mnsc.1040.0263.

[3] S. Gawiejnowicz, A review of four decades of time-dependent scheduling: main results, new topics, and open problems, *Journal of Scheduling*, **23** (2020), 3–47, doi: 10.1007/s10951-019-00630-w.

[4] F. Jaehn, H. A. Sedding, Scheduling with time-dependent discrepancy times, *Journal of Scheduling*, **19** (2016), 737–757, doi: 10.1007/s10951-016-0472-2.

[5] H. A. Sedding, Line side placement for shorter assembly line worker paths, *IISE Transactions* **52** (2020), 181–198, doi: 10.1080/24725854.2018.1508929.

[6] H. A. Sedding, Scheduling jobs with a V-shaped time-dependent processing time, *Journal of Scheduling*, **23** (2020), 751–768, doi: 10.1007/s10951-020-00665-4.

[7] H. A. Sedding, *Time-Dependent Path Scheduling: Algorithmic Minimization of Walking Time at the Moving Assembly Line*, Springer Vieweg, Wiesbaden, 2020, doi: 10.1007/978-3-658-28415-2.

[8] W. E. Smith, Various optimizers for single-stage production, *Naval Research Logistics Quarterly*, **3** (1956), 59-66, doi: 10.1002/nav.3800030106.

[9] N. T. Thomopoulos, Line balancing-sequencing for mixed-model assembly, *Management Science*, **14** (1967), B59–B75, doi: 10.1287/mnsc.14.2.B59.

The Third International Workshop on Dynamic Scheduling Problems
Adam Mickiewicz University, Poznań, July 5th – 6th, 2021
DOI: `10.14708/isbn.978-83-951298-6-5p71-76`

Heuristic algorithms for solving hard scheduling problems with positional penalties and controllable processing times

Dvir Shabtay*
Ben-Gurion University of the Negev, Beer-Sheva, Israel

Baruch Mor
Ariel University, Ariel, Israel

Liron Yedidsion
Amazon Research, Amazon, Seattle, USA

Keywords: scheduling, resource-dependent processing times, resource allocation, bi-criteria optimization, heuristics

1 Introduction and problem definition

In this paper we provide a set of heuristic algorithms that is capable of practically solving a large set of $\mathcal{NP}$-hard single-machine scheduling problems involving resource allocation decisions. The set of problems we study share the common property that the scheduling criterion can be represented as one that includes positional penalties. Among the scheduling criteria sharing this property are: the classical problem of minimizing the total completion time, the problem of minimizing the variation in the waiting times, the problem of minimizing the variation in completion times, and a set of at least four earliness/tardiness problems involving due-date assignment decisions.

The set of single-machine scheduling problems we study here can be stated as follows. We are given a set of n jobs, $\mathcal{J} = \{1, ..., n\}$, that is available at time zero and is to be non-preemptively scheduled on a single-machine. The processing time of job j, denoted by $p_j(u_j)$, is a convex decreasing function of the amount of continuous and non-renewable resource, u_j, allocated to its processing operation (u_j is a continuous decision variable to be determined by the scheduler), and is given by:

$$p_j(u_j) = B_j + (w_j/u_j)^k, \tag{1}$$

where for job j ($j \in \{1, 2, ..., n\}$), B_j is a lower bound on the processing time, w_j is the *workload*, and k is a parameter common to all jobs. Various special cases of the processing time function in (1) has been used extensively in continuous resource

*Speaker, e-mail: `dvirs@bgu.ac.il`

allocation theory (see, e.g., Lee and Li [2], Shakhlevich and Strusevich [9], Oron [5], and Shabtay and Zofi [8]) as it captures many real life applications (in most of these cases it is assumed that $B_j = 0$ for $j = 1, ..., n$). For example, Monma [3] pointed out that the time required to perform many actual government and industrial operations can be expressed by (1) with $B_j = 0$ for $j = 1, ..., n$ and $k = 1$, and that the time required to perform very large scale integration (VLSI) circuit design operations may also be represented by (1) with $B_j = 0$ for $j = 1, ..., n$ and $k = 0.5$.

A solution S for our set of single-machine scheduling problems with resource-dependent processing times is defined by a feasible resource allocation vector, $\mathbf{u} = (u_1, u_2, ..., u_n)$, and a job schedule on the single-machine. In the set of problems we consider here, an optimal solution does not include machine idle times. Therefore, a job schedule is defined by a processing permutation, $\phi = (\phi(1), \phi(2), ..., \phi(n))$, of the n jobs on the single-machine, where $\phi(i)$ is the job that is assigned to position i in ϕ. The quality of a solution $S = (\phi, \mathbf{u})$ is measured by two different criteria. The first, F_1, is a *scheduling criterion*. We focus on a set of problems sharing the common property that this criterion can be represented using the following format:

$$F_1 = \sum_{i=1}^{n} \xi_i p_{\phi(i)} \left(u_{\phi(i)}\right), \tag{2}$$

where ξ_i is a positive integer representing the *positional penalty* per unit processing time that results from assigning any job to position i in the job processing order. The second, F_2, is the total resource allocation cost, defined by

$$F_2 = \sum_{j=1}^{n} v_j u_j, \tag{3}$$

where v_j is the cost of assigning one unit of resource to the processing of job j.

Our aim is to find a solution S that minimizes the scheduling criterion, $F_1 = \sum_{i=1}^{n} \xi_i p_{\phi(i)} \left(u_{\phi(i)}\right)$, subject to the condition that $F_2 = \sum_{j=1}^{n} v_j u_j \le U_v$, where U_v is a predefined upper bound on the total resource-consumption cost. Following the three-field notation for scheduling problems by Graham et al. [1], and the extensions for resource-dependent processing times by Shabtay and Steiner [7], we denote this set of problems by $1 \left| conv, \sum_{j=1}^{n} v_j u_j \le U_v \right| \sum_{i=1}^{n} \xi_i p_{\phi(i)} \left(u_{\phi(i)}\right)$.

2 Literature review

Yedidsion et al [10] showed that there are many single-machine scheduling problems in which the scheduling criterion can be represented as a special case of (2). For example, if the scheduling criterion, F_1, is the total completion time, then the corresponding scheduling criterion, $F_1 = \sum_{j=1}^{n} C_j$, can be considered as a special case of F_1 in Eq. (2) with positional penalty $\xi_i = n - i + 1$ for $i = 1, ..., n$. As another example, the problem of minimizing the total completion time deviation, such that $F_1 = \sum_{s=1}^{n} \sum_{t=s}^{n} |C_s - C_t|$, can also be viewed as a special case of F_1 in Eq. (2) with $\xi_i = (n - i)(n - i + 1)$ for $i = 1, ..., n$. As a last example, Yedidsion et al [10] provided a summary of a set of single-machine scheduling problems with assignable due dates, which can also be represented as a special case of F_1. In this set of problems the exact value of ξ_i depends on the method by which due dates are assigned to jobs. For example, Yedidsion et al [10] showed (based on the result by Panwalkar [6]) that when a common due date, d, has to be assigned to all jobs, the problem of minimizing $F_1 = \alpha \sum_{j=1}^{n} E_j + \beta \sum_{j=1}^{n} T_j + n\gamma d$, where E_j and T_j are the earliness and tardiness of job J_j, respectively, and α, β, and γ are non-negative parameters representing one unit of earliness, tardiness and due-date cost, can also be represented in the format of Eq. (2). In this case the positional penalties are given by

$$\xi_i = \begin{cases} \alpha (i-1) + \gamma n & \text{for} \quad i \leq i^* \\ \beta (n - i + 1) & \text{for} \quad i > i^* \end{cases} \tag{4}$$

where position i^* can be computed in constant time.

Shabtay and Steiner [7] studied the *makespan* crierion, i.e., the case where $\xi_i = 1$ for $1, ..., n$. They show that the corresponding $1 \left| conv, \sum_{j=1}^{n} u_j \leq U_v \right| C_{\max}$ problem (or equivalently the $1 \left| conv, \sum_{j=1}^{n} v_j u_j \leq U_v \right| \sum_{i=1}^{n} p_{\phi(i)} \left(u_{\phi(i)} \right)$ problem) is solvable in $O(n)$ time.

Lee and Lei [2] studied the special case where the scheduling criterion is the *total completion time* (and accordingly $\xi_i = n - i + 1$ for $i = 1, ..., n$) problem. They proved that the corresponding $1 \left| conv, \sum_{j=1}^{n} u_j \leq U_v \right| \sum_{j=1}^{n} C_j$ problem (or equivalently the $1 \left| conv, \sum_{j=1}^{n} v_j u_j \leq U_v \right| \sum_{i=1}^{n} (n-i+1) p_{\phi(i)} \left(u_{\phi(i)} \right)$ problem) is solvable in $O(n \log n)$ time if either $B_j = B$ or $w_j = w$ for $j = 1, ..., n$. They also conjectured that the general $1 \left| conv, \sum_{j=1}^{n} u_j \leq U_v \right| \sum_{j=1}^{n} C_j$ problem is $\mathcal{NP}$-hard, but did not provide any formal proof to support their conjecture.

Yedidsion and Shabtay [11] provided adequate proof of the above conjecture by showing that the $1 \left| conv, \sum_{j=1}^{n} v_j u_j \leq U_v \right| \sum_{i=1}^{n} \xi_i p_{\phi(i)} \left(u_{\phi(i)} \right)$ problem is $\mathcal{NP}$-hard for *any* set of ξ_i parameters satisfying $\xi_l \neq \xi_m$ for any $l \neq m$, *even* when $v_j = 1$ for $j = 1, ..., n$. Moreover, they constructed an approximation algorithm for solving the most general $1 \left| conv, \sum_{j=1}^{n} v_j u_j \leq U_v \right| \sum_{i=1}^{n} \xi_i p_{\phi(i)} \left(u_{\phi(i)} \right)$ problem, which is based on solving a weakly polynomial number of $1 \left| conv \right| \sum_{i=1}^{n} \xi_i p_{\phi(i)} \left(u_{\phi(i)} \right) + \sum_{j=1}^{n} v_j u_j$ problems (each of which is solvable in $O(n^3)$ time).

3 Research objectives and our contribution

The fact that the $1\left|conv, \sum_{j=1}^{n} v_j u_j \leq U_v\right| \sum_{i=1}^{n} \xi_i p_{\phi(i)}\left(u_{\phi(i)}\right)$ problem is $\mathcal{NP}$-hard for *any* set of ξ_i parameters satisfying $\xi_l \neq \xi_m$ for any $l \neq m$, implies that there is a need for designing exact algorithms to solve small instances of this set of hard problems, along with heuristic algorithms that will be able to (heuristically) tackle larger instances of this set of problems. The only algorithm exists in the literature, is the approximation algorithm proposed by Yedidsion and Shabtay [11]. However, this algorithm wasn't tested against any other algorithm or against the value of a tight lower bound. Our aim is to help closing this gap in the literature.

We begin by providing the optimal resource allocation strategy as a function of the job sequencing decision. This enable us to reduce the problem into an equivalent (non-linear) problem which involve only sequencing decisions. We then suggest five different heuristics algorithms to solve the reduced sequencing problem:

- Heuristic 1 (**H1**): Based on a simple sorting rule (requires only $O(n \log n)$ time).
- Heuristic 2 (**H2**): A job-insertion heuristics (similar to that used in Nawza et al. [4]), which requires $O(n^3)$ time.
- Heuristic 3 (**H3**): The approximation algorithm by Yedidsion and Shabtay [11] (which requires $O(n^{3+p})$ time, for some predefined constant p).
- Heuristic 4 (**H4**): Implementation of the classical genetic algorithm.
- Heuristic 5 (**H5**): Implementation of the classical simulated annealing algorithm.

We also obtain a method to derive a tight lower bound for the minimal objective value (we leave its explanation in view of space limitations). We then perform an experimental study to test the performance of the heuristic algorithms.

Algorithms H_i for $i = 1, 2, 3, 4, 5$ were implemented in C++ and run on an Intel (R) Core ™ i7-8650U CPU @ 1.90 GHz 16.0 GB RAM platform. The programming platform consisted of the '*Visual Studio*' software. Moreover, to better evaluate the quality of the suggested algorithms when solving large instances, we included another heuristic (referred to hereafter as heuristic H_6) which simply selects the best permutation out of $50,000$ randomly generated permutations. For each out of several combinations of k and n, we randomly generated 50 numerical instances. All parameters were drawn from an integer uniform distribution ranging between 1 and 20 for B_j, w_j and v_j, and between $0.5n10^{2-1/k}$ and $1.5n10^{2-1/k}$ for U_v.

Partial information regrading the results for the total completion time criterion appears in Table 1. It includes the average relative gap of the value of the solution obtained by Heuristic (H_i) from the lower bound value ($avg\delta^i$), and the average running time of each heuristic ($r.t$), except for H_1 and H_6 for which the running time was negligible even for $n = 150$.

Table 1. Comparison between heuristics for large instances

n	k	$\mathbf{H_1}$	$\mathbf{H_2}$		$\mathbf{H_3}$	
		$avg\delta^1$	$avg\delta^2$	$r.t\,(sec)$	$avg\delta^3$	$r.t\,(sec)$
50	0.5	0.0123	0.0112	0.002	0.0047	1.138
50	1	0.0040	0.0040	0.002	0.0011	0.960
50	2	0.0147	0.0120	0.002	0.0001	0.837
100	0.5	0.0114	0.0106	0.012	0.0032	30.743
100	1	0.0063	0.0063	0.013	0.0029	29.518
100	2	0.0149	0.0132	0.013	0.0003	37.825
150	0.5	0.0081	0.0074	0.042	0	224.30
150	1	0.0075	0.0075	0.042	0.0046	306.15
150	3	0.0452	0.0392	0.041	0.0203	464.19

n	k	$\mathbf{H_4}$		$\mathbf{H_5}$		$\mathbf{H_6}$
		$avg\delta^4$	$r.t\,(sec)$	$avg\delta^5$	$r.t\,(sec)$	$avg\delta^6$
50	0.5	0.0049	0.114	0.0063	0.309	0.0906
50	1	0.0013	0.110	0.0028	0.298	0.1057
50	2	0.0002	0.111	0.0032	0.308	0.1331
100	0.5	0.0032	0.212	0.0065	0.833	0.1302
100	1	0.0030	0.212	0.0056	0.811	0.1584
100	2	0.0004	0.212	0.0062	0.834	0.1905
150	0.5	0.0001	0.426	0.0047	1.582	0.1505
150	1	0.0047	0.314	0.0072	1.521	0.1837
150	2	0.0010	0.311	0.0080	1.555	0.2208

It appears that all of our heuristics (especially the three last ones) are capable of solving large instances of the problem with a *negligible* gap between the value of the heuristic's solution and the value of a tight lower bound on the optimal solution value.

Finally, we design exact procedures to optimally solve small instances of the problem. One which is based on a Convex Integer Programming (*CIP*) formulation of the equivalent sequencing problem, and the other based on a branch and bound (*B&B*) formulation of the equivalent sequencing problem.

4 Future research

In future research, we aim to conduct numerical experiments with mentioned above exact algorithms, in order to learn about the exact instance size that they are capable to optimally solve in reasonable time.

References

[1] R. L. Graham, 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** (1979), 287–326, doi: 10.1016/S0167-5060(08)70356-X.

[2] C. Y. Lee, L. Lei, Multiple-project scheduling with controllable project duration and hard resource constraint: some solvable cases, *Annals of Operation Research*, **102** (2001), 287–307, doi: 10.1023/A:1010918518726.

[3] C. L. Monma, A. Schrijver, M. J. Todd, V. K. Wei, Convex resource allocation problems on directed acyclic graphs: duality, complexity, special cases and extensions, *Mathematics of Operations Research*, **15** (1990), 736–748, doi: 10.1287/moor.15.4.736.

[4] M. Nawaz, E. E. Enscore, I. Ham, Heuristic algorithm for the m-machine, n-job flowshop sequencing problem, *Omega*, **11** (1983), 91–95, doi: 10.1016/0305-0483(83)90088-9.

[5] D. Oron, Scheduling controllable processing time jobs with position dependent workload, *International Journal of Production Economics*, **173** (2016), 153–160, doi: 10.1016/j.ijpe.2015.12.014.

[6] S. S. Panwalkar, M. L. Smith, A. Seidmann, Common due date assignment to minimize total penalty for the one machine scheduling problem, *Operations Research*, **30** (1982), 391–399, doi: 10.1287/opre.30.2.391.

[7] D. Shabtay, G. Steiner, Survey of scheduling with controllable processing times, *Discrete Applied Mathematics*, **155** (2007), 1643–1666, doi: 10.1016/j.dam.2007.02.003.

[8] D. Shabtay, M. Zofi, Single machine scheduling with controllable processing times and unavailability period to minimize the makespan, *International Journal of Production Economics*, **198** (2018), 191–200, doi: 10.1016/j.ijpe.2017.12.025.

[9] N. V. Shakhlevich, V. A. Strusevich, Pre-emptive scheduling problems with controllable processing times, *Journal of Scheduling*, **8** (2005), 233–253, doi: 10.1007/s10951-005-6813-1.

[10] L. Yedidsion, D. Shabtay, M. Kaspi, Complexity analysis of an assignment problem with controllable assignment and its applications in scheduling, *Discrete Applied Mathematics*, **159** (2011), 1264–1278, doi: 10.1016/j.dam.2011.04.001.

[11] L. Yedidsion, D. Shabtay, The resource dependent assignment problem with a convex agent cost function, *European Journal of Operational Research*, **261** (2017), 486–502, doi: 10.1016/j.ejor.2017.03.004.

The Third International Workshop on Dynamic Scheduling Problems
Adam Mickiewicz University, Poznań, July 5th – 6th, 2021
DOI: 10.14708/isbn.978-83-951298-6-5p77-81

Relative robust total completion time scheduling problem on a single machine

Prabha Sharma[*]
The NorthCap University, Gurugram, India

Diptesh Ghosh
Indian Institute of Management, Ahmedabad, India

Sandeep Singh
The NorthCap University, Gurugram, India

Keywords: relative robust optimization, discrete scenario, adjacent scenario, adjacent job sequence, local search

1 Introduction

Uncertainties in a production environment occur in the form of machine breakdowns, non-availability of quality tools, unstable work force and in many more other forms as stated by Yang and Yu [1]. Kouvelis and Yu [2] have suggested using a finite discrete set of values or values in a finite interval as processing times for each job to adequately represent these uncertainties. We follow this line of research.

2 Problem formulation

In this paper, we consider the case in which a finite discrete set of processing times is assigned to each of n jobs, which are to be processed on a single machine. Let

$$P_j = (p_{1j}, p_{2j}, \ldots, p_{rj})$$

be the set of r processing times for job j, where $j = 1, 2, \ldots, n$. A scenario s associates a processing time for each job $j$from its respective set P_j. The collection of all the valid scenarios is denoted by S. Since we consider all possible combinations of the processing times as scenarios, hence $|S| = r^n$.

Let us consider a job sequence π and a scenario $s \in S$. Let $C(\pi, s)$ denote total completion time of the job sequence π with respect to scenario s and let $C^*(s)$ denote minimum completion time for scenario s, obtained by using the *Shortest Processing Time first* (SPT) rule.

[*] Speaker, e-mail: prabhasharma@ncuindia.edu

The *deviation* of sequence π for the scenario $s \in S$ is defined as $d(\pi, s) = C(\pi, s) - C^*(s)$. A scenario $s \in S$, for which $d(\pi, s)$ is maximum is called the *worst case scenario* for π. The relative robust total completion time problem is one of finding a job sequence whose maximum deviation is the smallest, i.e., to find

$$\min_{\pi} \max_{s} \left\{ C(\pi, s) - C^*(s) \right\}. \tag{1}$$

Yand and Yu [1] have shown that the problem is $\mathcal{NP}$-complete, even when $|S| = 2$. They have also designed an exact algorithm for solving the robust total completion time problem using dynamic programming and two polynomial time heuristics.

In this work, we propose a local search based algorithm to solve the relative robust total completion time problem. The following lemma characterizes the worst case scenario for any sequence π.

Lemma 1. *For any sequence π, its worst case scenario will either have its job processing times at their maximum or at their minimum.*

Proof. Consider an arbitrary scenario in which the processing time for job j is p_j. Let job j be in the i_1-th position in sequence Π, and be in the i_2-th position in the sequence of non-increasing processing times in that scenario. The contribution of p_j in the relative deviation for Π in this scenario is $((n-i_1+1)-(n-i_2+1))p_j = (i_2-i_1)p_j$. We have to consider two cases: Case 1, when $i_1 < i_2$, and Case 2, when $i_1 > i_2$.

Case 1. In this case, we will show that the relative deviation increases for the scenario in which p_j changes to $\overline{p_j}$ if all other processing times are held constant. Suppose p_j increases to $\overline{p_j}$. This increase either causes the order of jobs in the SPT sequence to remain the same, or to change. If the order of jobs in the SPT sequence remains the same, the relative deviation increases by $(i_2 - i_1)(\overline{p_j} - p_j) > 0$.

Next, suppose the order of jobs change if p_j increases to $\overline{p_j}$. After this change, let the position of job j be $i_2 + k$ in the SPT sequence, with $k > 0$. This means that the jobs in positions $i_2 + 1$ through $i_2 + k$ in the original SPT sequence move one position to the left. After such a change, the contribution of job j to the relative difference is

$$((n - i_1 + 1) - (n - (i_2 + k) + 1))\overline{p_j} = (i_2 - i_1)\overline{p_j} + k\overline{p_j}.$$

The jobs in positions $i_2 + 1$ through $i_2 + k$ in the original SPT sequence reduce the relative deviation by $p_{i_2+1} + \cdots + p_{i_2+k}$. So the net increase in the relative deviation is

$$(i_2 - i_1)\overline{p_j} + k\overline{p_j} - (p_{i_2+1} + \cdots + p_{i_2+k}) = (i_2 - i_1)\overline{p_j} + \sum_{r=1}^{k}(\overline{p_j} - p_{i_2+r})$$

which is positive since $\overline{p_j} \geq p_{i_2+r}$ for $1 \leq r \leq k$.

Case 2. In this case, we will show that the relative deviation increases for the scenario in which p_j changes to $\underline{p_j}$ if all other processing times are held constant. Suppose the processing time for job j reduces from p_j to $\underline{p_j}$. This will cause the sum of completion times for given sequence to reduce by $(n - i_i + 1)(p_j - \underline{p_j})$.

Next, suppose this reduction in processing time causes the sequence of jobs in the SPT sequence to change. Since the processing time of job j reduces, suppose that after the change it occupies the position $i_2 - k$ with $k > 0$. This would cause the jobs in positions $i_2 - 1$ through $i_2 - k$ in the original SPT sequence to move one position to the right in the new SPT sequence. Let us denote the processing times of these jobs as $p_{[i_2-1]}$ through $p_{[i_2-k]}$. After the reduction in processing time of job j, the sum of completion times of the jobs in the SPT sequence reduces by

$$(n - i_2 + 1)p_j - (n - (i_2 - k) + 1)\underline{p_j} + p_{[i_2-1]} + \cdots + p_{[i_2-k]}$$

which can be rewritten as

$$(n - i_2 + 1)(p_j - \underline{p_j}) + \sum_{r=1}^{k}(p_{[i_2-r]} - \underline{p_j}).$$

The increase in relative deviation due to the change in the processing time of job j is obtained by subtracting the reduction in the sum of completion times of the given sequence from the reduction in the sum of completion times of the SPT sequence. So the increase in relative deviation is given by

$$\begin{aligned}&(n - i_2 + 1)(p_j - \underline{p_j}) + \sum_{r=1}^{k}(p_{[i_2-r]} - \underline{p_j}) - (n - i_i + 1)(p_j - \underline{p_j})\\ &= (i_1 - i_2)(p_j - \underline{p_j}) + \sum_{r=1}^{k}(p_{[i_2-r]} - \underline{p_j}).\end{aligned}$$

This increase in relative deviation is positive since $i_1 > i_2$ and each of $p_{[i_2-1]}$ through $p_{[i_2-k]}$ is greater than $\underline{p_j}$. The result follows. $\square$

Lemma 1 simplifies the search for the worst case scenario on the polytope of all scenarios by reducing the number of scenarios to be searched from r^n to 2^n.

We use the following definitions of adjacency to describe neighborhoods for the problem under consideration.

Definition 1 (Adjacency of scenarios). *Two scenarios s^1 and $s^2 \in S$ are adjacent if s^1 and s^2 differ in exactly one component value, i.e., the processing time of only one job is different in the two scenarios.*

Definition 1 implies that for each scenario in the polytope of all scenarios in S we have $n(r-1)$ adjacent scenarios.

Definition 2 (Adjacency of permutations, Gaiha and Gupta [3]). *Two permutations π^1 and π^2 of sequence $(1, 2, \ldots, n)$ are adjacent, if π^2 is obtained from π^1 by interchanging positions of i and j, where if $\pi^1(i) = k$, then $\pi^1(j) = k+1$ and $k = 1, 2, \ldots, n-1$.*

Definition 2 implies that a job sequence has $(n-1)$ adjacent job sequences.

3 Local search algorithm

In this section, we describe a two phase local search algorithm which has been designed for solving the relative robust total completion time problem. This algorithm uses Procedure 1 which computes the best neighbor of a given sequence π_0 and Procedure 2 which finds a heuristic estimate of the maximum deviation for a given sequence π. We start with a given sequence π_0 as the current sequence, and repeat Procedure 1 until the current sequence is locally optimal, i.e., none of its neighbors have a maximum deviation that is smaller than its maximum deviation.

Procedure 1: Consider a sequence π_0 and use Definition 2 to obtain the $n(r-1)$ neighboring (i.e., adjacent) scenarios. For each of the neighboring scenarios, use Procedure 2 to compute the maximum deviation for the scenario, and hence obtain the neighboring scenario π with the minimum of the maximum deviation values. If the maximum deviation of π is less than or equal to that of π_0, replace π_0 with π and repeat Procedure 1.

Procedure 2: Consider a sequence π, and consider an arbitrary scenario s^o as the current scenario. Compute the deviation $d(\pi, s^o)$. Using Definition 1 obtain $n(r-1)$ scenarios adjacent to the current scenario and compute the deviations for each of the neighboring scenarios. Move to the scenario with maximum deviation. In case of tie with the starting scenario s^o, move to the adjacent scenario tied with s^o. Continue till the value of the maximum deviation does not improve for three iterations.

Numerical results are very encouraging. The search appears to be linear time and for small problems exact values of the worst case scenario was obtained.

4 Future research

In future, we wish to make an attempt to fully characterize the worst case scenario for any job sequence so that Phase-I search will not be required. Even if this does not happen using Lemma 1, the search on the polytope of all scenarios can be simplified.

We also plan to conduct extensive numerical experiments in order to determine the time complexity and quality of the solutions obtained.

References

[1] J. Yang, G. Yu, On the robust single machine scheduling problem, *Journal of Combinatorial Optimization*, **6** (2002), 17–33, `doi: 10.1023/A:1013333232691`.

[2] P. Kouvelis, G. Yu, Robust discrete optimization and its applications, Springer Science & Business Media, Berlin, 2013, `doi: 10.1007/978-1-4757-2620-6`.

[3] P. Gaiha, S. K. Gupta, Adjacent vertices on a permutohedron, *SIAM Journal on Applied Mathematics*, **32** (1977), 323–327, `doi: 10.1137/0132025`.

The Third International Workshop on Dynamic Scheduling Problems
Adam Mickiewicz University, Poznań, July 5th – 6th, 2021
DOI: 10.14708/isbn.978-83-951298-6-5p83-88

An exact algorithm for a two-machine time-dependent scheduling problem

Weronika Skowrońska[*]
Adam Mickiewicz University, Poznań, Poland

Stanisław Gawiejnowicz
Adam Mickiewicz University, Poznań, Poland

Keywords: time-dependent scheduling, parallel machines, deteriorating jobs, total completion time, V-shape property

1 Introduction

A number of scheduling problems exists, in which jobs processed later are more time-consuming than those started earlier. In the problems, the variable processing time of a job may be defined as a function of the starting time of the job. Problems of this type are called *time-dependent scheduling problems* and appear in many applications such as scheduling maintenance procedures, multiple loan repayment problems, fire fighting problems etc. Most commonly are studied time-dependent scheduling problems with job processing times defined by non-decreasing functions. In this case, job processing times *deteriorate* in time, i.e., a job started later has a larger processing time compared to case when it starts earlier. Hence, such jobs are called *deteriorating jobs* and are intensively studied recently (see, e.g., Agnetis et al. [1], Gawiejnowicz [5], Strusevich and Rustogi [8]).

A separate group of scheduling problems constitute those in which an optimal job sequence $J_{[1]}, J_{[2]}, \ldots, J_{[n]}$ can be divided into two parts: sequence L such that

$$p_{[1]} \geq p_{[2]} \geq \cdots \geq p_{[k]}$$

for all $J_{[i]} \in L$, followed by sequence R containing remaining jobs such that

$$p_{[k]} \leq p_{[k+1]} \leq \cdots \leq p_{[n]}$$

for all $J_{[i]} \in R$. In other words, in optimal schedules for the problems jobs are scheduled in non-increasing (non-decreasing) order with respect to processing times before the job with the smallest (the largest) processing time. In time-dependent

[*]Speaker, e-mail: wersko3@st.amu.edu.pl

scheduling problems the sequences L and R are defined similarly, with this difference that instead of job processing times $p_{[j]}$ are applied *deterioration rates* $b_{[j]}$ defined as in Section 2. Schedules of this form are called *V-shaped schedules* and are studied in the context of scheduling problems with both fixed and variable job processing times (see, e.g., Eilon and Chowdhury [3], Mosheiov [9], Gawiejnowicz [4, 5]).

In this talk, we consider a parallel machine time-dependent scheduling problem which, to the best of our knowledge, was not studied earlier. We present an algorithm generating all V-shaped schedules for the problem, implemented in Python 3.8.

2 Problem formulation and its properties

The problem under consideration can be formulated as follows. We are given a set of n independent, deteriorating jobs $J_1, J_2, \ldots, J_n$ to be scheduled on two parallel identical machines M_1 and M_2. The processing time of job J_j is equal to

$$p_j(t) = 1 + b_j t, \tag{1}$$

where $t \geq 0$ is the starting time of the job and $b_j > 0$ is *deterioration rate* of J_j, $1 \leq j \leq n$. The aim is to find a schedule minimizing the total completion time of all jobs, $\sum_{j=1}^{n} C_j$, where C_j denotes the completion time of job J_j, $1 \leq j \leq n$. Applying the extended three-field notation (Gawiejnowicz [5]), we will denote the problem as $P2|p_j = 1 + b_j t|\sum C_j$.

Problem $P2|p_j = 1 + b_j t|\sum C_j$ is a generalization of single machine problem $1|p_j = 1 + b_j t|\sum C_j$ which is one of the main open problems in time-dependent scheduling (see, e.g., Gawiejnowicz [4, 6] for a summary of research on this problem). Problem $P2|p_j = 1 + b_j t|\sum C_j$ is also a special case of problem $P2|p_j = a_j + b_j t|\sum C_j$, where $a_j \geq 0$ is the *basic processing time* of job J_j, $1 \leq j \leq n$. The latter problem is $\mathcal{NP}$-hard, since in case when $a_j = 0$ it reduces to problem $P2|p_j = b_j t|\sum C_j$ which is $\mathcal{NP}$-hard (Kononov [7], Chen [2]). Though we suppose that problem $P2|p_j = 1 + b_j t|\sum C_j$ is $\mathcal{NP}$-hard as well, a formal proof of its $\mathcal{NP}$-hardness is not known.

Based on the V-shape property of problem $1|p_j = 1 + b_j t|\sum C_j$ (Mosheiov [9]), one can easily observe than an optimal schedule for problem $P2|p_j = 1 + b_j t|\sum C_j$ is V-shaped on each machine. This property may be used in a recursive algorithm generating all V-shaped schedules for the problem which we present in Section 3. Though it is an exponential algorithm, since there exist $O(2^n)$ V-shaped sequences of length n, it may be useful in studying the structure of optimal schedules. This, in turn, may help to establish its complexity.

3 Results

In this section, we present our exact algorithm for solving the two-machine problem formulated in Section 2.

The main idea of this algorithm is as follows. First, we generate two subsets of jobs assigned to both machines. Next, for such a pair of subsets, we generate all V-shaped sequences on both machines, calculating for each pair of such V-shaped sequences a corresponding value of the criterion function. Comparing the values for every two successively generated V-shaped schedules, we find the optimal schedule.

More formally, our algorithm one can formulate using three main procedures: DIVISIONS, GEN_V_SHAPES and OPTIMUM.

The first of the procedures, DIVISIONS, is responsible for creating every possible division of a set of n jobs with processing times in the form of (1) into two subsets, assigned to machine M_1 and machine M_2, respectively.

Algorithm 1 DIVISIONS(S, F, P, n)

```
1:  $SPLITTED, FUNC\_COEFFS = []$
2:  for int I in $(0, len(p)/2)$ do
3:      $M_1, M_2, F_1, F_2 = []$
4:      $SEQ = P[I]$
5:      for int J in $(0, len(SEQ))$ do
6:          if SEQ[J] = 0 then
7:              APPEND $S[J]$ to $M_1$
8:              APPEND $F[J]$ to $F_1$
9:          else
10:             APPEND $S[J]$ to $M_2$
11:             APPEND $F[J]$ to $F_2$
12:         end if
13:     end for
14:     APPEND $[M_1, M_2]$ to SPLITTED
15:     APPEND $[F_1, F_2]$ to FUNC_COEFFS
16: end for
17: return SPLITTED, FUNC_COEFFS
```

Procedure GEN_V_SHAPES generates all V-shaped sequences out of the two subsets assigned to machines M_1 and M_2, and calculates the value of criterion function, $\sum C_j$, for each such a sequence.

Algorithm 2 GEN_V_SHAPES(n)

```
Input: $I\_ARR, C\_ARR, N = LEN(I\_ARR)$ $RES = []$,
 1:     $OPT = [string$ SEQUENCE = [], $int$ VAL= INFINITY]
 2: function GEN_V_SHAPE($I\_ARR, C\_ARR, N, RES, OPT, F, IND, K$)
 3:     if $N = 0$ then
 4:         return RES, [[],[0]]
 5:     else if $N = 1$ then
 6:         return $I\_ARR$, [[$I\_ARR$][1]]
 7:     end if
 8:     $L = CONCAT\ (I\_ARR[K], C\_ARR)$
 9:     $R = CONCAT\ (C\_ARR, I\_ARR[K])$
10:     if $K = N$ then
11:         $RES = L + R$
12:         for I in RES do
13:             $SUM\_CJ = 0$
14:             for J in I do
15:                 $V = IND[J]$
16:                 $SUM\_CJ+ = 1 + F_V * SUM\_CJ$
17:                 if $SUM\_CJ < OPT[VAL]$ then
18:                     $OPT = [I, SUM\_CJ]$
19:                 end if
20:             end for
21:         end for
22:         return RES, OPT
23:     end if
24:     GEN_V_SHAPE(I_ARR, L, N, RES, OPT, F, IND, K+1)
25:     GEN_V_SHAPE(I_ARR, R, N, RES, OPT, F, IND, K+1)
26:     return RES, OPT
27: end function
```

Finally, a straight-forward procedure OPTIMUM returns the optimal schedule among all generated V-shaped schedules.

We illustrate application of our algorithm using the following numerical example.

Example 1. Let us assume that we have $n = 5$ jobs $J_1, J_2, \dots, J_5$ such that $S = [1, 2, 3, 4, 5]$ and $F = [7, 4, 1, 3, 8]$.

Then P is the list of all 0-1 sequences of length 5, and as result we obtain schedule in which jobs J_3, J_4 and J_5 are scheduled on machine M_1, jobs J_1 and J_2 are scheduled on machine M_2 (see Fig. 1). The value of criterion $\sum C_j$ equals 24.

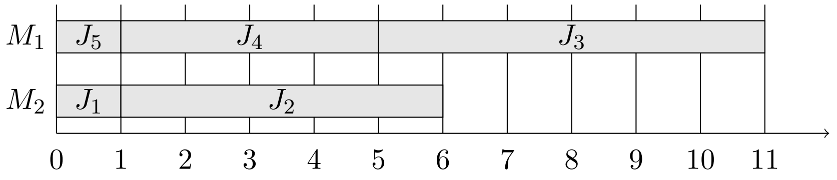


Figure 1. Optimal schedule generated in Example 1

4 Further research

Future research on problem $P2|p_j = 1 + b_j t|\sum C_j$ may concern the following three topics. First, one should establish the time complexity of the problem. Second, one should find other its properties than the one related to the V-shapeness of optimal schedule. Finally, the algorithm presented in the talk, which is purely a combinatorial problem, could be possibly transformed into a branch-and-bound algorithm. This, however, would require finding a good lower bound on the value of the $\sum C_j$ criterion or applying a good heuristic for construction of a near-optimal initial schedule.

References

[1] A. Agnetis, J-C. Billaut, S. Gawiejnowicz, D. Pacciarelli, A. Soukhal, *Multiagent Scheduling: Models and Algorithms*, Springer, Berlin-Heidelberg, 2014, `doi: 10.1007/978-3-642-41880-8`.

[2] Z. L. Chen, Parallel machine scheduling with time dependent processing times, *Discrete Applied Mathematics*, **70** (1996), 81–93, `doi: 10.1016/0166-218X(96)00102-3`. (Erratum: *Discrete Applied Mathematics*, **75** (1997), 103, `doi: 10.1016/S0166-218X(97)00002-4`.)

[3] S. Eilon, I. G. Chowdhury, Minimizing waiting time variance in the single machine problem, *Management Science*, **23** (1977), 567–573, `doi: 10.1287/mnsc.23.6.567`.

[4] S. Gawiejnowicz, A review of four decades of time-dependent scheduling: main results, new topics, and open problems, *Journal of Scheduling*, **23** (2020), 3–47, `doi: 10.1007/s10951-019-00630-w`.

[5] S. Gawiejnowicz, *Models and Algorithms of Time-Dependent Scheduling*, Springer, Berlin-Heidelberg, 2020, `doi: 10.1007/978-3-662-59362-2`.

[6] S. Gawiejnowicz, W. Kurc, New results for an open time-dependent scheduling problem, *Journal of Scheduling*, **23** (2020), 733–744, doi: 10.1007/s10951-020-00662-7.

[7] A. Kononov, Scheduling problems with linear increasing processing times, U. Zimmermann et al. (eds.), *Operations Research 1996*, Springer, Berlin-Heidelberg, 1997, 208–212, doi: 10.1007/978-3-642-60744-8_38.

[8] V. A. Strusevich, K. Rustogi, Scheduling with Time-Changing Effects and Rate-Modifying Activities, Springer, Cham, 2017, doi: 10.1007/978-3-319-39574-6.

[9] G. Mosheiov, V-shaped policies for scheduling deteriorating jobs, *Operations Research*, **39** (1991), 979–991, doi: 10.1287/opre.39.6.979.

The Third International Workshop on Dynamic Scheduling Problems
Adam Mickiewicz University, Poznań, July 5th – 6th, 2021
DOI: 10.14708/isbn.978-83-951298-6-5p89-92

Two-agent scheduling with position-dependent processing times and job rejection

Xiaowen Song
Qufu Normal University, Qufu, P. R. China

Cuixia Miao*
Qufu Normal University, Qufu, P. R. China

Xiaoxu Song
Qufu Normal University, Qufu, P. R. China

Keywords: agent scheduling, position-dependent jobs, single machine, parallel machines, job rejection, dynamic programming, FPTAS

1 Introduction

We consider two-agent scheduling problems with position-dependent processing times and job rejection. The problems can be formulated as follows. There are two agents A and B, each agent has a job family $\mathcal{J}^X = \{J_1^X, J_2^X, \cdots, J_{n_X}^X\}$, $X \in \{A, B\}$, either to be processed on $m \geq 1$ parallel machines or outsourced to a subcontractor with a penalty. The actual processing time of each job J_j^X is defined $p_{jk}^X = \alpha_j^X f(k)$, where $f(k)$ is a function that only depends on the position k. If the function $f(k)$ is an increasing function with respect to k, we will deal with a position-dependent *deterioration effect*, otherwise we deal with a position-dependent *learning effect*. We will denote the effects by symbols DE and LE, respectively. The objective is to minimize the objective value of one agent with the restriction that the other agent objective value cannot exceed a given upper bound.

2 Related research

The two-agent scheduling was first considered by Baker and Smith [3] and Agnetis et al. [2]. Cheng et al. [5] and Gawiejnowicz [8] gave surveys of scheduling with time-dependent processing times. Bachman and Janiak [4] gave a review for the position-dependent scheduling problems on a single machine, and they summarized most of the basic problems with different objectives. Gawiejnowicz [7] and Strusevich and

*Speaker, e-mail: miaocuixia@126.com

Rustogi [12] considered more recent research on time- and position-dependent scheduling. Agnetis et al. [1] discussed the agent scheduling in detail. Gawiejnowicz and Suwalski [9] considered two-agent scheduling problems with linearly deteriorating jobs. Feng et al. [6] and Li and Lu [10] considered two-agent scheduling with rejection on a single machine and on two-parallel-machines. Yang and Lu [13] addressed two-agent scheduling problems with the general position-dependent processing time.

3 Our results

In this paper, we consider two-agent scheduling problems with position-dependent processing times and job rejection. We assume the DE holds and that the makespan of accepted jobs plus the total rejection penalty of rejected jobs of agent B cannot exceed a given upper bound Q. For the minimization of the makespan of accepted jobs plus the total rejection penalty of rejected jobs of agent A, we design pseudo-polynomial dynamic programming algorithms for the single-machine problem and the two-parallel-machine problem, running in $O(n_A^2 n_B^2 P^2 Q^2 E)$ and $O(n_A^3 n_B^3 P^4 Q^3 E)$ time, respectively, where $P = \left(\sum_{j=1}^{n_A} e_j^A + \sum_{j=1}^{n_B} e_j^B\right) f(n_A + n_B)$, $E = \sum_{j=1}^{n_A} e_j^A$ and $f(n_A + n_B)$ is the maximal value of function $f(k)$ for $1 \leq k \leq n$. Applying the geometric rounding technique and inflated rejection penalty (Sengupta [11]), we develop a $O(\frac{n^3 n_A^4 n_B^4}{\epsilon^7} \log^2 E \log^2 P \log^3 Q)$ fully polynomial time approximation scheme (an FPTAS) for the parallel-machine problem. For the minimization of the total completion time of accepted jobs plus the total rejection penalty of rejected jobs of agent A, we present for the single-machine problem a pseudo-polynomial dynamic programming algorithm and an FPTAS running in $O(n_A^2 n_B^2 P_1 Q^2)$ and $O(\frac{n^3 n_A^2 n_B^2}{\epsilon^3} \log P_1 \log^2 Q)$ time, respectively, where $P_1 = \left(\sum_{j=1}^{n_A} e_j^A + \sum_{j=1}^{n_B} e_j^B\right) f(1)$.

4 Future research

In future research, it would be worth to consider scheduling problems in uniform and flow shop machine environments. Analysis of other objectives such as the total weighted completion time is another interesting topic for future research.

Acknowledgements

This research was supported by the National Natural Science Foundation of China (11801310, 11771251), the Province Natural Science Foundation of Shandong (ZR2017-MA031, ZR2019MA061), the Project of Shandong Province Graduate Supervisor Ability Promotion (SDYY17139).

References

[1] A. Agnetis, J-C. Billaut, S. Gawiejnowicz, D. Pacciarelli, A. Soukhal, *Multiagent Scheduling: Models and Algorithms*, Springer, Berlin-Heidelberg, 2014, doi: 10.1007/978-3-642-41880-8.

[2] A. Agnetis, P. B. Mirchandani, D. Pacciarelli, A. Pacifici, Scheduling problems with two competing agents, *Operations Research*, **52** (2004), 229–242, doi: 10.1287/opre.1030.0092.

[3] K. R.Baker, J. C. Smith, A multiple-criterion model for machine scheduling, *Journal of Scheduling*, **6** (2003), 7–16, doi: 10.1023/A:1022231419049.

[4] A. Bachman, A. Janiak, Scheduling jobs with position-dependent processing times, *Journal of Operations Research Society*, **55** (2004), 257–264, doi: 10.1057/palgrave.jors.2601689.

[5] T. C. E. Cheng, Q. Ding, B. M. T. Lin, A concise survey of scheduling with time-dependent processing times, *European Journal of Operational Research*, **152** (2004), 1–13, doi: 10.1016/S0377-2217(02)00909-8.

[6] Q. Feng, B-Q. Fan, S-S. Li, W-P. Shang, Two-agent scheduling with rejection on a single machine, *Applied and Mathematical Modelling*, **39** (2015), 1183–1193, doi: 10.1016/j.apm.2014.07.024.

[7] S. Gawiejnowicz, *Models and Algorithms of Time-Dependent Scheduling*, Springer, Berlin-Heidelberg, 2020, doi: 10.1007/978-3-662-59362-2.

[8] S. Gawiejnowicz, *Time-Dependent Scheduling*, Springer, Berlin-Heidelberg, 2008, doi: 10.1007/978-3-540-69446-5.

[9] S. Gawiejnowicz, C. Suwalski, Scheduling linearly deteriorating jobs by two agents to minimize the weighted sum of two criteria, *Computer and Operation Research*, **52** (2014), 135–146, doi: 10.1016/j.cor.2014.06.020.

[10] D-W. Li, X-W. Lu, Two-agent parallel-machine scheduling with rejection, *Theoretical Computer Science*, **703** (2017), 66–75, doi: 10.1016/j.tcs.2017.09.004.

[11] S. Sengupta, Algorithms and approximation schemes for minimum lateness/tardiness scheduling with rejection, *Lecture Notes in Computer Science*, **2748** (2003), 79–90, doi: 10.1007/978-3-540-45078-8_8.

[12] V. A. Strusevich, K. Rustogi, *Scheduling with Times-Changing Effects and Rate-Modifying Activities*, Springer, Cham, 2017, doi: 10.1007/978-3-319-39574-6.

[13] L-Y. Yang, X-W. Lu, Two-agent scheduling problems with the general position-dependent processing time, *Theoretical Computer Science*, **796** (2019), 90–98, doi: 10.1016/j.tcs.2019.08.023.

The Third International Workshop on Dynamic Scheduling Problems
Adam Mickiewicz University, Poznań, July 5th – 6th, 2021
DOI: 10.14708/isbn.978-83-951298-6-5p93-98

Scheduling with periodic availability constraints to minimize makespan

Lishi Yu*
School of Mathematical Science, Zhejiang University, P. R. China

Zhiyi Tan
School of Mathematical Science, Zhejiang University, P. R. China

Keywords: scheduling, worst-case ratio, periodic availability, LPT, FFD

1 Introduction

In the real environment, we often encounter the situation that the production of a company is suspended, or the work is not carried out as planned due to various reasons. Some of them are routine arrangements such as employees taking weekends off and machines being maintained regularly, while others may be caused by emergencies, such as sudden breakdown of a machine and isolation measures taken to fight infectious diseases. Therefore, it is significant to study the scheduling problems in which the machine environment changes in time. Time intervals during which machines cannot process jobs are usually called *unavailability periods*. Accordingly, we call the remaining time intervals *availability periods*.

There are usually two assumptions on the unavailability periods of the machines. One is that the start and end time of the unavailability periods have a certain regularity, generally manifested as periodic. The second is that the unavailability periods appear without any obvious regularity. This will bring some difficulties in design and analysis of exact or approximation algorithms. Therefore, some restrictions on the unavailability periods are usually added. For example, there is at most one unavailability period on each machine, or the unavailability periods on different machines do not overlap, which are often far away from reality. On the other hand, the periodic unavailability periods is closer to the real situation.

In this paper, we study scheduling problems with periodic unavailability periods. Given a set of n independent jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$, which are to be processed on $m \geq 1$ parallel identical machines $M_1, M_2, \ldots, M_m$. All the jobs are available at time zero, and no preemption is allowed. The processing time of J_j is p_j, where $j = 1, \ldots, n$. Each machine is periodically unavailable. In other words, available

*Speaker, e-mail: 12035036@zju.edu.cn

periods and unavailable periods appear alternately on each machine. The duration of each unavailable period and available period is t and T, respectively. Denote by $\beta = \frac{t}{T}$ the ratio between the length of an unavailable period and available period. In most real settings, $\beta < 1$. Without loss of generality, we assume that each machine just finishes its maintenance at time 0 and $p_j \leq T$, for $j = 1, 2, \ldots, n$. The objective is to minimize the makespan. Following Ji et al. [5], the problem will be denoted in the extended three-field notation as $Pm|nr - pm|C_{\max}$.

2 Related research

There are plenty of papers on scheduling with unavailability periods. We refer to Lee [7] and Ma et al. [9] for the surveys on this topic. However, very few papers dealt with periodic unavailability periods. For $1|nr - pm|C_{\max}$, Ji et al. [5] proposed an algorithm LPT, and showed that its worst-case ratio is 2. Moreover, there is no polynomial time approximation algorithm with a worst-case ratio of less than 2 unless $\mathcal{P} = \mathcal{NP}$. For $P2|nr - pm|C_{\max}$, Sun and Li [12] introduced an algorithm and proved that its worst-case ratio is at least $\max\left\{\frac{14}{11} + \frac{12}{11}\beta, 2\right\}$ and at most $\max\left\{\frac{8}{5} + \frac{6}{5}\beta, 2\right\}$. Results on corresponding problems with different objectives and other variations can be found in Gawiejnowicz [2, 3], Lu and Li [8], Qi et al. [11], Qi [10], Sun and Li [12], Xu et al. [13, 14, 15].

In the scheduling literature, LPT is usually referred to as a classical algorithm for parallel machine scheduling. Algorithm *Longest Processing Time first* (LPT for short, Graham [4]) first sorts the jobs in non-increasing order by processing times, then always assigns the first unprocessed job in the sequence to the machine which can complete it as early as possible. Algorithm LPT is also applicable for single machine and machines with unavailability periods, as shown by Ji et al. [5].

In fact, LPT for $1|nr - pm|C_{\max}$ is more intuitive to be interpreted using the bin-packing terminology. Namely, in the *one-dimensional bin packing* problem, we are given a sequence of jobs each associated with a size. Jobs are required to be packed into a minimum number of bins with identical capacity. *First Fit Decreasing* (FFD for short) is a fundamental algorithm for bin-packing problems (Johnson [6]). Algorithm FFD first sorts the jobs in non-increasing order by size, then always packs the next unpacked jobs in the sequence into the first opened bin that has enough room to accommodate it. If no opened bin is suitable for this, a new bin is opened and the job is packed there. For any instance of $Pm|nr - pm|C_{\max}$, we can construct a companion one-dimensional bin packing instance by setting the size of a job as its processing time and the capacity of the bin as t. In fact, the essential idea of algorithm LPT by Ji et al. [5] is first using FFD to pack jobs into bins and then processing jobs of each bin as a whole in an available period.

In the first glance, it seems that Ji et al. [5] complete the study on $1|nr-pm|C_{\max}$. However, both the tightness of LPT and the non-approximability only valid when β tends to infinity, which falls into the relatively unrealistic situation. Adopting the reduction given in [5], it is not difficult to see that there is no polynomial time approximation algorithm with a worst-case ratio of less than $\frac{2\beta+2}{\beta+2}$ unless $\mathcal{P}=\mathcal{NP}$, but the performance of LPT when β is small remains unexplored. To learn more about LPT, Yu et al. [16] presented worst-case ratios of LPT and algorithms based on other bin-packing algorithms as functions of b^*, the minimum number of availability periods that at least one job is processed on in any schedule. However, the parameter b^* is instance-dependent and it is $\mathcal{NP}$-hard to obtain its exact value.

3 Our results

For a companion bin-packing instance constructed by an instance of scheduling problems with periodic unavailability periods, denote by b^{FFD} (b for short) and b^*_{BP} the number of bins created by FFD and in an optimal packing, respectively. Let B_i be the ith bin created by the FFD algorithm, $i=1,\dots,b$. By abuse of notation, we also use B_i to denote the set of jobs that are packed in it.

For an instance of $Pm|nr-pm|C_{\max}$, let b^* be the number of availability periods that at least one job is processed on in the optimal schedule. Clearly, $b^*_{BP}=b^*$ when $m=1$. When $m=2$, $b^*_{BP}\le b^*$, and there exists instance such that the strict inequality holds.

Lemma 1. (Dósa [1]) $b\le\frac{11}{9}b^*_{BP}+\frac{6}{9}$.

Lemma 1 is a fundamental result on FFD. Though the bound is tight, and even the additive term can not be improved, it is still inadequate to prove our result.

Let $\mathcal{B}=\{B_1,B_2,\dots,B_{b-1}\}$, and $\mathcal{B}_I$, $\mathcal{B}^1_{II}$, $\mathcal{B}^2_{II}$, $\mathcal{B}_{III}$ be disjoint subsets of $\mathcal{B}$. Concretely,

(i) $\mathcal{B}_I$ consists of bins of $\mathcal{B}$ which contains exactly one job.

(ii) $\mathcal{B}^1_{II}$ consists of bins of $\mathcal{B}$ which contains exactly two jobs with one of them has a size greater than $\frac{T}{2}$.

(iii) $\mathcal{B}^2_{II}$ consists of bins of $\mathcal{B}$ which contains exactly two jobs with none of them has a size greater than $\frac{T}{2}$.

(iv) $\mathcal{B}_{III}$ consists of bins of $\mathcal{B}$ which contains exactly three jobs.

If $p_n>\frac{T}{4}$, then any bin can contain at most three jobs. Therefore, $\mathcal{B}=\mathcal{B}_I\cup\mathcal{B}^1_{II}\cup\mathcal{B}^2_{II}\cup\mathcal{B}_{III}$.

Let y_0 be the number of jobs packed to B_b. We have the following non-trivial result on the FFD algorithm for the bin-packing.

Lemma 2. *Suppose that $b > b^*_{BP} \geq 2$ and J_n is packed in B_b with $p_n > \frac{T}{4}$.*

(i) If $b - b^*_{BP} = 2k + 1$, where k is an integer, then $|\mathcal{B}^2_{II}| \geq 6k + y_0$ and $|\mathcal{B}_{III}| \geq 8k + y_0$.

(ii) If $b - b^*_{BP} = 2k + 2$, where k is an integer, then $|\mathcal{B}^2_{II}| \geq 6k + 3 + y_0$ and $|\mathcal{B}_{III}| \geq 8k + 4 + y_0$.

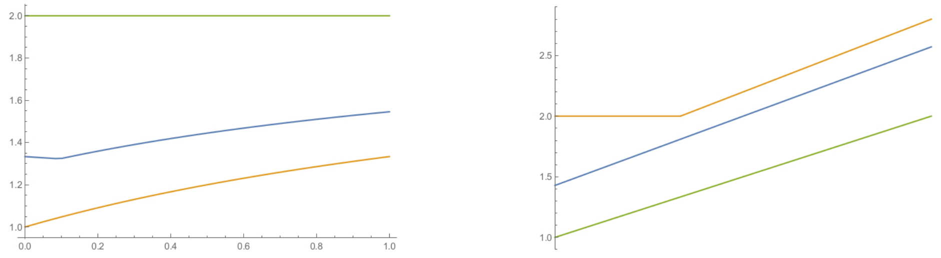
Note that given b and b^*_{BP}, we can enumerate all possible triples $(|\mathcal{B}^2_{II}|, |\mathcal{B}_{III}|, y_0)$ by Lemma 2. For example, if $b^*_{BP} = 2$ and $b = 3$, then the only possibility is $(1, 1, 1)$. It follows that $n = 6$. Similarly, if $b^*_{BP} = 3$ and $b = 4$, then $(1, 1, 1)$, $(1, 2, 1)$, $(2, 1, 1)$ are all three possibilities.

Our main result for problem $1|nr - pm|C_{\max}$ is the following theorem (see Fig. 1a, where from top to bottom are given bounds by Ji et al. [5], bounds given in this paper, and the theoretic lower bound, respectively).

Theorem 3. *The worst-case ratio of algorithm LPT for problem $1|nr - pm|C_{\max}$ is no more than*

$$r(\beta) = \begin{cases} \frac{44+44\beta}{33+36\beta}, & \beta \in (0, \frac{\sqrt{313}-15}{32}] \approx (0, 0.0841], \\ \frac{29+28\beta}{22+20\beta}, & \beta \in (\frac{\sqrt{313}-15}{32}, \frac{\sqrt{181}-11}{24}] \approx (0.0841, 0.1022], \\ \frac{9+8\beta}{7+4\beta}, & \beta \in (\frac{\sqrt{181}-11}{24}, \infty) \approx (0.1022, \infty), \end{cases}$$

and the bound is tight when $\beta \geq 0.1022$.



(a) Worst-case ratios of algorithm LPT and theoretic lower bounds for problem $1|nr - pm|C_{\max}$

(b) Worst-case ratios of algorithms and theoretic lower bounds for problem $P2|nr - pm|C_{\max}$

Figure 1. Worst case ratios and theoretic lower bounds

For problem $P2|nr - pm|C_{\max}$, we first give a non-approximability result (see Fig. 1b, where from top to bottom are given bounds by Sun and Li [12], bounds given in this paper, and the theoretic lower bound, respectively).

Theorem 4. *For problem $P2|nr - pm|C_{\max}$, there is no polynomial time approximation algorithm with a worst-case ratio less than $1 + \beta$ unless $\mathcal{P} = \mathcal{NP}$.*

Next, we propose a new algorithm $DFFD$ problem $P2|nr - pm|C_{\max}$, which beats the algorithm proposed by Sun and Li [12].

Algorithm $DFFD$

1. Apply algorithm FFD for the companion bin-packing instance. If $b = 2k+1$, where k is an integer, Go to Step 2. If $b = 2k$, where k is an integer, Go to Step 3.
2. For $i = 1, \ldots, k$, process jobs in B_{2i-1} on the ith availability period of M_1, and process jobs in B_{2i} on the ith availability period of M_2. Process the jobs in B_b on two machines by algorithm LPT. Output the resulting schedule. Stop.
3. For $i = 1, \ldots, k$, process the jobs in B_{2i-1} on the ith availability period of M_1, and process jobs in B_{2i} on the ith availability period of M_2. Denote the resulting schedule by σ_1.
4. For $i = 1, \ldots, k - 1$, process the jobs in B_{2i-1} on the ith availability period of M_1, and process jobs in B_{2i} on the ith availability period of M_2. Process the jobs in $B_{b-1} \cup B_b$ on two machines by algorithm LPT. Denote the resulting schedule by σ_2.
5. Select the better schedule of σ_1 and σ_2 as output. Stop.

Theorem 5. *The worst-case ratio of algorithm $DFFD$ for problem $P2|nr-pm|C_{\max}$ is $\frac{10}{7} + \frac{8}{7}\beta$, and the bound is tight.*

4 Further research

Tighten the worst-case ratio of LPT for $1|nr - pm|C_{\max}$ when $\beta \in [0, 0.0841)$ is a tedious and hard work. Generalizing the algorithms and their analysis for two machines case to m machines case is more interesting and also more challenging.

Acknowledgement. Supported by the National Natural Science Foundation of China (12071427, 11671356).

References

[1] G. Dósa, The tight bound of first fit decreasing bin-packing algorithm is $FFD(I) \leq \frac{11}{9}OPT(I) + \frac{6}{9}$, *Lecture Notes in Computer Science*, **4614** (2007), 1–11, `doi: 10.1007/978-3-540-74450-4_1`.

[2] S. Gawiejnowicz, A review of four decades of time-dependent scheduling: main results, new topics, and open problems, *Journal of Scheduling*, **23** (2020), 3–47, `doi: 10.1007/s10951-019-00630-w`.

[3] S. Gawiejnowicz, *Models and Algorithms of Time-Dependent Scheduling*, Springer, Berlin-Heidelberg, 2020, doi: 10.1007/978-3-662-59362-2.

[4] R. L. Graham, Bounds on multiprocessing timing anomalies, *SIAM Journal on Applied Mathematics*, **17** (1969), 416–429, doi: 10.1137/0117039.

[5] M. Ji, Y. He, T-C. E. Cheng, Single-machine scheduling with periodic maintenance to minimize makespan, *Computers & Operations Research*, **34** (2007), 1764–1770, doi: 10.1016/j.cor.2005.05.034.

[6] D. S. Johnson, *Near-Optimal Bin Packing Algorithms*, Ph. D. thesis, Massachusetts Institute of Technology, Cambridge, MA, 1973.

[7] C-Y. Lee, Machine scheduling with an availability constraint, J. Y-T. Leung (ed.), *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, Chappman & Hall/CRC, Boca Raton-London-New York, 2004, doi: 10.1201/9780203489802.

[8] G. Li, X. Lu, Two-machine scheduling with periodic availability constraints to minimize makespan, *Journal of Industrial and Management Optimization*, **11** (2015), 685–700, doi: 10.3934/jimo.2015.11.685.

[9] Y. Ma, C-B. Chu, C-R. Zuo, A survey of scheduling with deterministic machine availability constraints, *Computers & Industrial Engineering*, **58** (2010), 199–211, doi: 10.1016/j.cie.2009.04.014.

[10] X. Qi, A note on worst-case performance of heuristics for maintenance scheduling problems, *Discrete Applied Mathematics*, **155** (2007), 416–422, doi: 10.1016/j.dam.2006.06.005.

[11] X. Qi, T. Chen, F. Tu, Scheduling the maintenance on a single machine, *Journal of the Operational Research Society*, **50** (1999), 1071–1078, doi: 10.1057/palgrave.jors.2600791.

[12] K. Sun, H. Li, Scheduling problems with multiple maintenance activities and non-preemptive jobs on two identical parallel machines, *International Journal of Production Economics*, **124** (2010), 151–158, doi: /10.1016/j.ijpe.2009.10.018.

[13] D. Xu, Z. Cheng, Y. Yin, H. Li, Makespan minimization for two parallel machines scheduling with a periodic availability constraint, *Computers & Operations Research*, **36** (2009), 1809–1812, doi: 10.1016/j.cor.2008.05.001.

[14] D. Xu, Y. Yin, H. Li, A note on 'scheduling of nonresumable jobs and flexible maintenance activities on a single machine to minimize makespan', *European Journal of Operational Research*, **197** (2009), 825–827, doi: 10.1016/j.ejor.2008.07.021.

[15] D. Xu, K. Sun, H. Li, Parallel machine scheduling with almost periodic maintenance and non-preemptive jobs to minimize makespan, *Computers & Operations Research*, **35** (2008), 1344–1349, doi: 10.1016/j.cor.2006.08.015.

[16] X. Yu, Y. Zhang, G. Steiner, Single-machine scheduling with periodic maintenance to minimize makespan revisited, *Journal of Scheduling*, **17** (2014), 263–270, doi: 10.1007/s10951-013-0350-0.

Indexes

Index of authors

Berlińska, Joanna, 27

Farid, Karim, 57

Gawiejnowicz, Stanisław, 33, 83
Ghosh, Diptesh, 77

Hall, Nicholas G., 17
Hussein, Youssef, 57

Kolińska, Marta, 33

Liu, Zhixin, 17

Marchetti-Spaccamela, Alberto, 39
Megow, Nicole, 39
Miao, Cuixia, 89
Mor, Baruch, 27, 47, 71
Mosheiov, Gur, 47

Ostermeier, Frederik, 51

Sakr, Nourhan, 57
Schlöter, Jens, 39
Sedding, Helmut A., 63
Shabtay, Dvir, 71
Sharma, Prabha, 77
Singh, Sandeep, 77
Skowrońska, Weronika, 83
Skutella, Martin, 39
Song, Xiaowen, 89
Song, Xiaoxu, 89
Stougie, Leen, 39

Tan, Zhiyi, 93

Yedidsion, Liron, 71
Yu, Lishi, 93

Index of keywords

adjacent job sequence, 77
adjacent scenario, 77
assembly line, 63

bi-criteria optimization, 71

choice of opponent, 17
complexity, 39
conditional DAG, 39

data gathering networks, 27
deteriorating jobs, 83
discrete scenario, 77
dynamic programming, 89

FFD, 93
flow shop, 51
forgetting effects, 51
FPTAS, 89

heuristics, 71

job rejection, 89
jobs
    position-dependent, 89

LAPT rule, 33
learning effects, 51
line side placement, 63
local search, 77
LPT, 93

makespan, 39, 47
mixed-criticality scheduling, 57
moving conveyor, 63
multiple round sports tournament, 17

open shop, 33

parallel machines, 83, 89
parallel processing, 39
performance criteria, 17
periodic availability, 93
professional tennis data, 17

reinforcement learning, 57
relative robust optimization, 77
resource allocation, 71
resource-dependent processing times, 71

scheduling, 27, 47, 51, 71, 93
    agent, 89
    time-dependent, 33, 63, 83
single machine, 47, 89
step-learning, 47

total completion time, 83

V-shape property, 83
variable communication speed, 27
varying-speed processors, 57

walking time, 63
worst-case ratio, 93