XLVIII Symposium on Operational Research
SYM-OP-IS 2021
Banja Koviljača, 20-23. September 2021.
MATΦ University of Belgrade Faculty of Mathematics
Mathematical Institute SANU

MATHEMATICAL MODELS FOR THE WEIGHTED SCHEDULING PROBLEM WITH DEADLINES AND RELEASE TIMES

UNA STANKOVIĆ[1], LUKA MATIJEVIĆ[1], TATJANA DAVIDOVIĆ[1]

[1] Mathematical Institute, Kneza Mihaila 36, Belgrade, Serbia, {una, luka, tanjad}@mi.sanu.ac.rs

Abstract: *We consider a non-preemptive version of the scheduling problem with multiple identical machines, deadlines, release times, and task weights. The main goal is to select a subset of tasks to be scheduled for execution in such a way to maximize the total sum of weights of all scheduled tasks. For this purpose, we devise two different mathematical formulations and test them on a set of randomly generated instances using the GNU Linear Programming Kit as a solver.*

Keywords: *Task scheduling, Multiple machines, Mixed-Integer linear program, Combinatorial optimization, Exact solver*

1. INTRODUCTION

The task scheduling problem is a widely known problem in the field of discrete mathematical optimization. It was first introduced by Graham [5] in 1966. Over the course of years, many different variations have been proposed, considering different aspects and applications of the problem. Some of the variations are described in a survey [6]. In general, these variations can be classified according to several criteria [8]: *task data* (such as task length, release time, deadline, etc.), *task characteristics* (such as precedence relations, preemption, etc.), *machine environment* (single or multiple machines, homogeneity of machines, etc.), and *optimality criteria*. Scheduling problems generally belong to the class of NP-hard problems, but in specific variants they can be NP-complete [16]. More details about scheduling problem can be found in books [3, 14].

In this paper, we address the problem of scheduling a set of tasks over several identical machines. This problem can be specified with Graham's notation [6] as $(P|r_j,d_j|\sum w_jX_j)$, where X_j is a binary variable which takes the value of 1 if task j is scheduled, or 0 otherwise. Each task has its length, which determines how many time slots a task will take to complete. A non-preemptive version of a problem is considered, meaning that the tasks cannot be interrupted before they are completed. Additional constraints are imposed on tasks, such as release times and deadlines. The release time of a task prevents its execution until the specified number of time units has passed, while deadlines assure that no task finishes its execution after a specified time unit. Together, release times and deadlines can be used to implicitly enforce some form of precedence relation among tasks. Unlike many other similar variants of the problem, which allow missing the deadline with some penalty, the considered version of the problem strictly forbids this situation. In our version, tasks that cannot be scheduled in such a way that both deadline and release time constraints are met, are not going to be scheduled at all. This adds additional complexity to the problem because we need to determine a proper subset of tasks that are going to be executed, so as to fulfill all the constraints and maximize the total sum of weight of these tasks.

Similar variants of this problem have already been investigated. In paper [18], a single-machine problem with release times, deadlines, setup times, and rejection has been described. Missing a deadline was allowed with a certain penalty, and the authors proposed a dynamic programming method for finding the solution. In paper [4], an approximation algorithm was proposed for scheduling tasks with deadlines and release times on a minimal number of machines. The authors of paper [17] considered scheduling with deadlines and release times, precedence, and exclusion relation on a single machine. A weighted version of the scheduling problem with release times and deadlines on a single machine was considered in [15] in the case of unit tasks (i.e., when the lengths of all tasks are the same). The authors proposed a greedy algorithm for efficiently finding a good solution.

The scheduling problems have many practical applications, from the use in multiprocessor systems, scheduling talks at conferences, production processes in the industry, assigning resources in a cloud environment, etc. The application that we are primarily interested in is the use of a scheduler as a part of *Proof-of-Useful-Work* (**PoUW**) in blockchain systems. In recent years, many concerns were raised about a huge amount of energy being spent on mining new blocks. The main problem is that most of that energy is used for calculating hash values, which is not in any way useful by itself. Because of that, Proof-of-Useful-Work [2] is suggested for overcoming this issue, by allowing new blocks to be mined by doing something "useful". Many recent papers addressed PoUW, including [1, 9, 10, 11].

As an application of our models, we consider PoUW that allows users to submit their own tasks to be solved, specifying the price they are willing to pay, together with deadlines and release times. In that case, miners need to choose which tasks they are going to solve and in which order, with the goal of maximizing their reward (profit, expressed as a sum of task prices). Selecting these tasks intuitively may lead to a suboptimal reward, and our model can help miners to make more proper selection. Especially, if miner explores multiple machines, it can decide to use additional software, which can determine the best possible subset of tasks (and their order) to be executed and maximize the profit. As we always need to be able to add new blocks into a blockchain, and miners do not want to lose any potential profit by letting their machines be idle, there should not be any empty timeslots between two tasks. To fill all of the potential empty timeslots, some dummy tasks (with zero weights) are added to the pool of tasks. More precisely, if there are no user-submitted tasks to be scheduled, miners can solve a classical *Proof-of-Work* (**PoW**) hash-based puzzle. A dummy task defined this way has a release time of 0, and a large enough deadline so that it could never be missed. Even though dummy tasks have a weight of 0 (they do not explicitly bring any reward to a miner), a miner can still benefit from solving them if his block gets accepted into a blockchain, in the same way as in the classical PoW.

Another potential usage of our models can be found in blockchain systems that use some forms of centralization. Even though blockchain is intrinsically a decentralized system, some authors investigated the possibility of introducing some level of centralized control, in order to reduce the energy wasted by preventing multiple miners from solving the same problem. In [7], the authors proposed a system in which a set of transactions for each block is determined by a manager, as well as a set of allowed nonce values for each miner (in contrast to classical PoW in which miners can choose transactions and nonce for themselves). This approach ensures that each miner examines only his portion of the search space, thus preventing unnecessary calculations. We further expand on this idea by suggesting that a manager can also assign the user-submitted task to miners. In addition, miners can be organized into groups in such a way that the miners from the same group solve different tasks from the assigned subset, which simulates scheduler with multiple machines. Admittedly, these groups would not be identical, but it would not be overly demanding to modify our model to work with heterogeneous machines. In this situation, the manager would solve our scheduling problem, and assign the appropriate subset of tasks to each of the mining groups.

The main contribution of this paper are two new mathematical formulations that we propose for the considered scheduling problem in the form of MILP. The experimental evaluation is carried out on a set of randomly generated instances. These instances vary in the number of available machines and the number of tasks that need to be scheduled. We tested the proposed models within *GNU Linear Programming Kit* **GLPK** solver [1]. As it turns out, the problem is highly complex because of the fact that we need to find the optimal subset of tasks to be executed. Under a time limit, GLPK solver is unable to prove the optimality for most of the instances, while in the case of very large instances finding even the first feasible solution proved to be a challenge for GLPK.

This paper is organized as follows. The introduction and the review of the relevant literature are presented in Section 1. In Section 2, we formulate the problem as the Mixed-Integer Linear Program. In Section 3, the experimental evaluation is presented, performed on a set of test instances using GLPK solver. Concluding remarks are given in Section 4, together with the ideas for further research.

2. PROBLEM DESCRIPTION

We can formulate the problem in the following way. Let us assume that we have n tasks that we want to schedule, as well as K identical machines on which we can execute the tasks. Each task is characterized by a deadline, release time, task length, and task weight. A non-preemptive version of the problem is considered. The deadlines cannot be exceeded, but a task can be omitted from the schedule without penalty. Empty time slots, in which no task is scheduled, are not permitted. This is justified by the intended application in PoUW. The goal is to maximize the total sum of all the scheduled tasks, with respect to constraints imposed by deadlines and release times. To unify the problem description, we include two dummy tasks. The task with index 0 is considered to be the *starting task*, in the sense that every machine has to start with this task. Similarly, a task with index $n+1$ is the *ending task*, so the execution on every machine has to end with this task. Naturally, both of these tasks have the 0 weight, or in other words, they do not contribute to the value of the objective function. For simplicity, we will denote with N the total number of tasks, the real ones plus starting and ending tasks ($N = n+2$).

[1] https://www.gnu.org/software/glpk/

Let us introduce the necessary notation:

- N - The number of tasks,
- K - The number of available machines,
- l_i - The length of task i,
- d_i - The deadline of task i,
- r_i - The release time task i,
- w_i - The weight of task i,
- α - The parameter that plays the role of a large enough constant, however, it is enough to set it to the maximal deadline: $\max_i\{d_i\}$.

We have two different sets of decision variables, binary variables x and real variables t defined as:

$$x_{i,j,k} = \begin{cases} 1, & \text{if the task } j \text{ is executed immediately after task } i \text{ on a machine } k, \\ 0, & \text{otherwise;} \end{cases}$$

$$t_{i,j,k} = \text{the starting time of the task } j \text{ on a machine } k\text{, corresponding to } x_{i,j,k}.$$

The proposed MILP for the considered problem can be defined as follows:

$$\max \sum_{i=0}^{N} \sum_{j=1}^{N+1} \sum_{k=1}^{K} w_j \cdot x_{i,j,k} \tag{1}$$

s.t.

$$\sum_{i=0}^{N} \sum_{k=1}^{K} x_{i,j,k} \leq 1, \quad 1 \leq j \leq N,\ i \neq j \tag{2}$$

$$x_{i,i,k} = 0, \quad 1 \leq i \leq N, \quad 1 \leq k \leq K \tag{3}$$

$$\sum_{i=0}^{N} x_{i,j,k} = \sum_{i=1}^{N+1} x_{j,i,k}, \quad 1 \leq j \leq N,\ i \neq j\ 1 \leq k \leq K \tag{4}$$

$$\sum_{j=1}^{N+1} x_{0,j,k} \leq 1, \quad 1 \leq k \leq K \tag{5}$$

$$t_{0,j,k} = 0, \quad 1 \leq j \leq (N+1),\ 1 \leq k \leq K \tag{6}$$

$$t_{i,j,k} \leq x_{i,j,k} \cdot \alpha, \quad 0 \leq i \leq N,\ 1 \leq j \leq (N+1),\ 1 \leq k \leq K \tag{7}$$

$$\sum_{i=1}^{N+1} t_{j,i,k} - \sum_{i=0}^{N} t_{i,j,k} = l_j \cdot \sum_{i=0}^{N} x_{i,j,k}, \quad 1 \leq j \leq N,\ 1 \leq k \leq K \tag{8}$$

$$t_{i,j,k} + l_j \leq d_j, \quad 0 \leq i \leq N,\ 1 \leq j \leq (N+1),\ 1 \leq k \leq K \tag{9}$$

$$t_{i,j,k} \geq r_j \cdot x_{i,j,k}, \quad 0 \leq i \leq N,\ 1 \leq j \leq (N+1),\ 1 \leq k \leq K \tag{10}$$

$$x_{i,j,k} \in \{0,1\}, \quad 0 \leq i \leq N,\ 1 \leq j \leq (N+1),\ 1 \leq k \leq K \tag{11}$$

$$t_{i,j,k} \geq 0, \quad 0 \leq i \leq N,\ 1 \leq j \leq (N+1),\ 1 \leq k \leq K \tag{12}$$

The objective function (1) represents the total weight of all the scheduled tasks. Constraints (2) and (3) ensure that no task is executed more than once. Constraints (4) ensure that every scheduled task, except for the first and the last, must have a preceding and subsequent task. This way, we can form an ordered list of tasks for each of the machines. Every machine has at most one start task, as suggested by constraints (5). Of course, there could be some machines without any scheduled tasks. As the list of tasks for each machine always starts with the dummy task 0, we initialize its starting time to zero (constraints (6)). Constraints (7) ensure that if task j is not scheduled after task i, the corresponding starting time is set to zero. Constraints (8) establish the rule that after executing a task j, the starting time is increased exactly for the length of that task. The set of constraints (9) impose that all of the deadlines for the scheduled tasks must be met, while constraints (10) are related to the release times of the scheduled tasks. Finally, constraints (11) and (12) determine the type of decision variables.

The proposed model (which we call *MODEL I*) has N^2K binary and N^2K real variables. A closer inspection reveals that not all of those variables are strictly necessary. Specifically, we do not necessarily need information about the previous task for variables that contain starting times, so we can try to reformulate the model in order to reduce the number of real variables to $N \cdot K$. Consequently, we are changing this set of decision variables to be:

$$t_{j,k} = \text{the starting time of the task } j \text{ on a machine } k.$$

The new formulation can be written as:

$$\max \sum_{i=0}^{N} \sum_{j=1}^{N+1} \sum_{k=1}^{K} p_j \cdot x_{i,j,k} \tag{13}$$

s.t.
(2) - (5), (11),

$$t_{0,k} = 0; \quad 1 \leq k \leq K \tag{14}$$

$$t_{i,k} + l_i - t_{j,k} \leq \alpha \cdot (1 - x_{i,j,k}), \quad 1 \leq i \leq N,\ 1 \leq j \leq N,\ 1 \leq k \leq K \tag{15}$$

$$t_{j,k} + l_j \leq d_j; \quad 1 \leq j \leq N,\ 1 \leq k \leq K \tag{16}$$

$$t_{j,k} \geq r_j \cdot \sum_{i=0}^{N-1} x_{i,j,k}; \quad 1 \leq j \leq N,\ 1 \leq k \leq K \tag{17}$$

$$t_{j,k} \geq 0, \quad 0 \leq j \leq (N+1),\ 1 \leq k \leq K \tag{18}$$

This model, referred to as MODEL II, shares constraints (2), (4), (3), (5), and (11) with the previous model. Constraints (14) set the starting time of the first task to 0 for every machine. Constraints (15) serve the same purpose as constraints (8) from *MODEL 1.* The set of constraints (16) guarantee that all of the deadlines for the scheduled tasks will be met, while constraints (17) are related to the release times of the scheduled tasks. The type of decision variables related to release times is given by constraints (18).

3. EXPERIMENTAL EVALUATION

For the purpose of testing our models, we randomly generated a set of ten instances, with a different number of tasks and available machines. The number of machines is ranging from 2 to 4, while the number of tasks takes values between 10 and 45. The instances can be downloaded from `http://www.mi.sanu.ac.rs/`

`~luka/resources/scheduling/instances.zip`. All the experiments were performed on an Intel i7-10750H processor with 32GB of RAM memory, under a Linux operating system. GLPK v4.65 [12] is used as an exact solution method. For practical reasons, the runtime of GLPK for each instance was set to 30 minutes (1800 seconds).

We present the results obtained by using our two models within GLPK on all ten instances in Table 1 . The first three columns contain some details about test instances, i.e., the name, the number of machines *K* and the number of tasks *N*. In the fourth and fifth columns, we display the results obtained using *MODEL I*, consisting of the objective value provided by GLPK (together with the corresponding percentage gap in parentheses), and the CPU time in seconds that GLPK required to obtain the reported solution. Similarly, the last two columns show the results provided by *MODEL II*.

Table 1: GLPK results for tested instances

			MODEL I		**MODEL II**	
Example	**Num. of machines**	**Num. of tasks**	**Obj. value (% gap)**	**CPU time (s)**	**Obj. value (% gap)**	**CPU time (s)**
Ex1	2	10	**27 (0%)**	3.64	**27(0%)**	2.98
Ex2	2	15	23 (69%)	1800	**34 (14.7%)**	1800
Ex3	2	15	**33.0 (21.2%)**	1800	**33 (21.2%)**	1800
Ex4	3	15	35 (8.6%)	1800	**37 (2.7%)**	1800
Ex5	2	20	**30 (56.7%)**	1800	N/A	1800
Ex6	3	20	**42 (21.4%)**	1800	N/A	1800
Ex7	3	20	**44.0 (11.4%)**	1800	N/A	1800
Ex8	4	20	**59.0 (0%)**	1.18	**59.0 (0%)**	12.56
Ex9	4	40	N/A	1800	N/A	1800
Ex10	4	45	N/A	1800	N/A	1800

As we can see from this table, GLPK was not able to provide the optimal solution for most of the instances within the given runtime limit, even though most of the tested instances are small in size. It may happen that by increasing this limit it would be possible to obtain the optimal solution, however, it cannot be guaranteed, and moreover, that could lead to the limited usefulness of the model since these types of problems often need to be solved quickly and on a regular basis. In the case of the two largest instances (Ex9 and Ex10), GLPK was not even able to find the first feasible solution. We can also notice that the instance Ex8 was solved to optimality in a short amount of time, despite having more tasks and machines than previous instances. This is because all of the tasks in this test instance can be scheduled on one of the available machines, without the need to omit some of the tasks. This observation indicates that the most complex part of this problem is finding a subset of tasks that are going to be scheduled. *MODEL II* has shown some promises for the first 4 instances, finding better or equal solutions than *MODEL I*. Despite that, MODEL II requires much more time to find a first feasible solution, and even was not able to find one at all in five out of ten instances. The main conclusion of this study is that this problem is too complex for being solved exactly, so we need to consider a metaheuristic approach to this problem, which we did in our subsequent paper [13].

4. CONCLUSION

In this paper, we presented two new mathematical formulations for the non-preemptive version of the scheduling problem with multiple identical machines, in which each task is characterized with length, weight, deadline, and release time. The goal is to maximize the total sum of weights of all the scheduled tasks, with the possibility to omit any number of tasks from the schedule. Therefore, the considered problem consists of finding the most suitable subset of tasks to be executed, assigning those tasks to the appropriate machine and determining the order in which selected tasks are going to be executed.

The experimental evaluation of these models was performed using GLPK solver, on a set of ten randomly generated instances. The obtained results suggest that the problem is too complex to be solved to optimality in a reasonable amount of time by an exact method. Consequently, applying metaheuristic methods seems to be a more promising approach. We applied one of such methods to this problem and presented the results in a paper [13].

Acknowledgement

This work was supported by the Serbian Ministry of Education, Science and Technological Development, Agreement No. 451-03-9/2021-14/200029, and by the Science Fund of the Republic of Serbia, Grant AI4TrustBC: *Advanced Artificial Intelligence Techniques for Analysis and Design of System Components Based on Trustworthy BlockChain Technology*.

REFERENCES

[1] Baldominos, A., & Saez, Y. (2019). *Coin. AI: A proof-of-useful-work scheme for blockchain-based distributed deep learning*. Entropy, 21(8), 723.

[2] Ball, M., Rosen, A., Sabin, M., & Vasudevan, P. N. (2017). *Proofs of Useful Work*. IACR Cryptol. ePrint Arch., 2017, 203.

[3] Brucker, P. (2007). *Scheduling algorithms - Fifth edition*. Springer.

[4] Cieliebak, M., Erlebach, T., Hennecke, F., Weber, B., & Widmayer, P. (2004). *Scheduling with release times and deadlines on a minimum number of machines*. In Exploring new frontiers of theoretical informatics (pp. 209-222). Springer, Boston, MA.

[5] Graham, R. L. (1966). *Bounds for certain multiprocessing anomalies*. Bell system technical journal, 45(9), 1563-1581.

[6] Graham, R. L., Lawler, E. L., Lenstra, J. K., & Kan, A. R. (1979). *Optimization and approximation in deterministic sequencing and scheduling: a survey*. In Annals of discrete mathematics (Vol. 5, pp. 287-326). Elsevier.

[7] Hazari, S. S., & Mahmoud, Q. H. (2019, January). *A parallel proof of work to improve transaction speed and scalability in blockchain systems*. In 2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC) (pp. 0916-0921). IEEE.

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

[9] Li, B., Chenli, C., Xu, X., Jung, T., & Shi, Y. (2019). *Exploiting computation power of blockchain for biomedical image segmentation*. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (pp. 0-0).

[10] Li, B., Chenli, C., Xu, X., Shi, Y., & Jung, T. (2019). *DLBC: A Deep Learning-Based Consensus in Blockchains for Deep Learning Services*. arXiv preprint arXiv:1904.07349.

[11] Loe, A. F., & Quaglia, E. A. (2018, June). *Conquering generals: an np-hard proof of useful work*. In Proceedings of the 1st Workshop on Cryptocurrencies and Blockchains for Distributed Systems (pp. 54-59).

[12] Makhorin, A. (2017). *GLPK (GNU linear programming kit)*. Reference Manual Version 4.6. Draft Edition; http://www.gnu.org/s/glpk/glpk.html

[13] Matijević, L., Stanković, U., Davidović, T., *General variable neighborhood search for the weighted scheduling problem with deadlines and release times* , Proc. XLVIII Symposium on Operational Research, SYMOPIS 2021, Banja Koviljača, Sept. 20-23, 2021 (submitted)

[14] Michael, L. P. (2018). *Scheduling: theory, algorithms, and systems*. Springer.

[15] Plaxton, C. G. (2008, July). *Fast scheduling of weighted unit jobs with release times and deadlines*. In International Colloquium on Automata, Languages, and Programming (pp. 222-233). Springer, Berlin, Heidelberg.

[16] Ullman, J. D. (1975). *NP-complete scheduling problems*. Journal of Computer and System sciences, 10(3), 384-393.

[17] Xu, J., & Parnas, D. L. (1990). *Scheduling processes with release times, deadlines, precedence and exclusion relations*. IEEE Transactions on software engineering, 16(3), 360-369.

[18] de Weerdt, M., Baart, R., & He, L. (2021). *Single-machine scheduling with release times, deadlines, setup times, and rejection*. European Journal of Operational Research, 291(2), 629-639.