Computers & Operations Research 170 (2024) 106780

Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/cor

Scheduling jobs with shared additional operations on parallel identical machines

Yakov Zinder [a], Joanna Berlińska [b,*], Bertrand M.T. Lin [c]

[a] School of Mathematical and Physical Sciences, University of Technology Sydney, Australia
[b] Faculty of Mathematics and Computer Science, Adam Mickiewicz University, Poznań, Poland
[c] Institute of Information Management, National Yang Ming Chiao Tung University, Hsinchu, Taiwan

ARTICLE INFO

Keywords:
Parallel-machine scheduling
Computational complexity
Approximation algorithms
Scheduling with communication delay
Duplication

ABSTRACT

The paper is concerned with assigning jobs, each with associated additional operations, to identical machines. A machine, allocated to a job, must also process all the additional operations associated with this job. An additional operation that is associated with several jobs assigned to the same machine needs to be processed by this machine only once. The goal is to minimise the time needed for the completion of all jobs and their additional operations. It is shown that even very restricted particular cases of the considered problem remain NP-hard in the strong sense. For the general case, the paper introduces two mixed integer linear programs as well as a broad class of approximation algorithms and a performance guarantee that is valid for any algorithm in this class. It is shown that, for one of the above-mentioned NP-hard particular cases, the considered class contains the best possible approximation algorithm. The performance of the mixed integer linear programs and several approximation algorithms is compared by means of computational experiments.

1. Introduction

The research presented below was triggered by the needs of a company designing integrated circuits. In order to test the conformance of design to the specifications, the company must run various computer programs, each on one workstation from a cluster of identical workstations. To be prepared for running a program, a workstation has to complete some additional operations associated with this program, such as the installation of special libraries, parameter setting and activation of background processes, and delineation of data sets. The sets of associated operations for different computer programs assigned to the same workstation may overlap. In this case, any such common operation needs to be completed on this workstation only once. The objective is to determine an assignment of computer programs (and their associated additional operations) to the workstations that allows to finish the entire testing in a minimal time.

The research presented below is also applicable far beyond the scope of software testing. One of such applications is production planning when it is necessary to minimise the time needed for producing certain products on identical machining centres. A machining centre permits the installation of all tools prior to the start of processing and after this installation produces products by applying the installed tools one by one in the required order. If several products, assigned to the same machining centre, need the same tool (maybe in conjunction with some other tools), this common tool needs to be installed only once.

The analysed problem can be stated as follows. Consider a finite set of jobs N, each of which is to be assigned to one of m identical machines, where $m < |N|$. It is convenient to assume that the machines are numbered from 1 to m and to denote the set of jobs assigned to machine $k \in \{1, \ldots, m\}$ by N^k. Each $i \in N$ has a finite set of associated additional operations $M(i)$. A machine can process a job i only if it also processes all operations in $M(i)$. If, for two jobs i and j assigned to the same machine, $M(i) \cap M(j) \neq \emptyset$, then this machine needs to complete each operation in $M(i) \cap M(j)$ only once. Denote the set of all additional operations by M, i.e. $M = \cup_{i \in N} M(i)$, and the processing time of each $g \in N \cup M$ by p_g. In what follows, it is assumed that all processing times are positive integers. The goal is to find a partition of N into N^1, $\ldots$, N^m with the smallest makespan, i.e. with the smallest value of the function

$$f(N^1, \ldots, N^m) = \max_{1 \leq e \leq m} \left(\sum_{i \in N^e} p_i + \sum_{j \in \cup_{i \in N^e} M(i)} p_j \right). \quad (1)$$

In what follows, this problem will be referred to as Assignment with Sharing Additional Operations (ASAO).

* Corresponding author.
E-mail addresses: Yakov.Zinder@uts.edu.au (Y. Zinder), Joanna.Berlinska@amu.edu.pl (J. Berlińska), bmtlin@nycu.edu.tw (B.M.T. Lin).

https://doi.org/10.1016/j.cor.2024.106780
Received 20 February 2024; Received in revised form 22 May 2024; Accepted 14 July 2024
Available online 17 July 2024

To the best of the authors' knowledge, the ASAO problem has never been studied previously. Saying this, it is important to stress that the ASAO problem closely relates to scheduling with communication delay and duplication — an active research area in computer science motivated by the necessity to take into account the time needed for transferring data between processors in multiprocessor computer systems.

For the purpose of this discussion, the scheduling problem with communication delay and duplication can be stated as follows. Consider a finite set of tasks J that are to be processed on $m > 1$ parallel identical machines. Each machine can process at most one task at a time. Each task $i \in J$ can be processed by any machine and its processing time p_i remains the same regardless of the machine. Since duplication is allowed, any task can be processed more than once, each time on a different machine. Such a machine processes its own copy (duplicate) of this task; all these duplicates are identical; and the start of processing a copy does not depend on the start of processing any other copy of the same task. The tasks are to be processed subject to precedence constraints in the form of a directed acyclic graph $G(J, A)$, where J is the set of nodes and A is the set of arcs. For any $(i, j) \in A$, machine k can start processing a copy of j at time S_j^k only if either the same machine starts processing a copy of i at time S_i^k satisfying the inequality $S_i^k + p_i \le S_j^k$, or k does not process a copy of i and there exists machine k' that starts processing a copy of i at $S_i^{k'}$ satisfying the inequality $S_i^{k'} + p_i + c_{ij} \le S_j^k$, where c_{ij} is a non-negative constant called communication delay. The goal is to minimise the time needed to complete all tasks. This time is commonly called the makespan and is denoted by $C_{\max}$.

It is easy to see that the optimum solution of the ASAO problem can be found by solving the stated-above scheduling problem with the bipartite directed acyclic graph $G(M \cup N, A)$ where the set of arcs is comprised of all arcs (i, j) such that i is in M, j is in N, and i is an operation associated with task j, and where the communication delay $c_{ij} = \infty$ for all $(i, j) \in A$. Conversely, the scheduling problem with the bipartite graph $G(M \cup N, A)$ and infinite communication delay is equivalent to the ASAO problem.

The rest of the paper is organised as follows. Section 2 outlines the contribution of this paper together with a survey of the relevant publications. Section 3 shows that even some very restricted particular cases of the ASAO problem are NP-hard in the strong sense. Section 4 introduces a family of approximation algorithms and a performance guarantee which is valid for all the algorithms in this family. Section 5 presents two alternative mixed integer linear programs, while Section 6 describes randomised constructive algorithms. Section 7 presents the results of computational experiments with the solution methods discussed in this paper. The conclusions can be found in Section 8.

2. Contribution of the paper and related publications

Since the pioneer publications (Rayward-Smith, 1987; Papadimitriou and Yannakakis, 1990), scheduling with communication delay has remained a subject of extensive research (Giroudeau and König, 2007; Drozdowski, 2009). This research is largely motivated by the necessity to schedule operations in the multiprocessor computer systems.

Using the three-field notation adopted in scheduling theory (Graham et al., 1979), the above-stated scheduling problem with communication delay and duplication can be denoted by $P|prec, c_{ij}, dup|C_{max}$. If, in the precedence constraints (in the acyclic directed graph), a path cannot contain more than one arc, then the attribute $prec$ is replaced by $1prec$. If, in addition, this directed graph is an out-tree or in-tree, then the attribute $1prec$ is replaced by $1out\text{-}tree$ or $1in\text{-}tree$, respectively. If all instances of this scheduling problem have the same number of machines, then the attribute P in the first field, indicating that the number of parallel identical machines is part of the input and may vary from instance to instance, is replaced by Pm, indicating that all instances have the same number of machines m. Similarly, $p_i = 1$ and $c_{ij} = \infty$ in the second field of the three-field notation indicate that the processing of each task takes one unit of time and that the communication delay is infinitely large, respectively. If the second field does not contain c_{ij}, then $c_{ij} = 0$ for all $(i, j) \in A$. The attribute dup indicates the possibility that a task can have multiple copies across different machines. If the second field does not contain the attribute dup, then duplication is not allowed.

Section 3 shows that even some very restricted versions of the ASAO problem are NP-hard in the strong sense. In terms of scheduling with communication delay and duplication these results are:

- strong NP-hardness of $P2|1prec, c_{ij} = \infty, p_i = 1, dup|C_{\max}$;
- strong NP-hardness of $P|1in\text{-}tree, c_{ij} = \infty, p_i = 1, dup|C_{\max}$;
- strong NP-hardness of $P|1out\text{-}tree, c_{ij} = \infty, p_i = 1, dup|C_{\max}$.

The known results in this realm are as follows. According to Zinder and Roper (1995), $P|1prec, p_i = 1|C_{\max}$ is NP-hard in strong sense, while Papadimitriou and Yannakakis (1990) proved that scheduling unit execution time tasks on an infinite number of machines with a constant communication delay and duplication, i.e., the problem $P\infty|prec, p_i = 1, c_{ij} = c, dup|C_{\max}$, is NP-hard. Jakoby and Reischuk (1992) proved that this problem remains NP-hard even if the precedence constraints graph is a binary in-tree. Chrétienne (1994) showed that when task processing times and communication delays are arbitrary, then the problem with unlimited number of processors is NP-hard even when the precedence graph is an in-tree of depth two. Strong NP-hardness of problem $P|prec, p_i = 1, c_{ij} = 1|C_{\max}$ has been proven by Rayward-Smith (1987). Hoogeveen et al. (1994) showed that even the problem $P|prec, p_i = 1, c_{ij} = 1|C_{\max} = 4$ is strongly NP-complete. Lenstra et al. (1996) proved the strong NP-hardness of problem $P|in\text{-}tree, p_i = 1, c_{ij} = 1|C_{\max}$. In all these cases, allowing duplication does not affect the proofs, and hence, the corresponding problems with duplication are also strongly NP-hard (Drozdowski, 2009). Bampis et al. (1996) proved that problem $P|prec, p_i = 1, c_{ij} = c, dup|C_{\max} = c + 3$ is NP-complete, which also follows from the result by Hoogeveen et al. (1994) mentioned above.

Sections 4–6 propose algorithms for solving the general case of the ASAO problem and contribute:

- a performance guarantee for a family of approximation algorithms,
- two mixed integer linear programs,
- randomised constructive algorithms.

In terms of scheduling with communication delay and duplication, these results complement the following existing literature. Papadimitriou and Yannakakis (1990) designed a 2-approximation algorithm for $P\infty|prec, p_i = 1, c_{ij} = c, dup|C_{\max}$. Colin and Chrétienne (1991) proposed an algorithm that generates optimal schedules for problem $P\infty|prec, c_{ij}, dup|C_{\max}$ if communication delays are smaller than the minimum task execution time. Jung et al. (1989) solved problem $P\infty|prec, p_i = 1, c_{ij} = c, dup|C_{\max}$ with an $O(|J|^{c+2})$ dynamic programming algorithm. For a finite number m of machines, unit task execution times, and unit communication delay, Munier and Hanen (1997) derived the $2 - \frac{1}{m}$ performance guarantee for a list scheduling algorithm.

Various heuristic algorithms without a performance guarantee were designed for the general scheduling problem with communication delay and duplication by Kruatrachue and Lewis (1988), Darbha and Agrawal (1997), Ahmad and Kwok (1998), Park et al. (1998), Bansal et al. (2003). Maiti et al. (2020) proposed a polynomial-time approximation algorithm for scheduling on uniform machines with a fixed communication delay. Orr and Sinnen (2020) devised a branch-and-bound algorithm for problem $P|prec, c_{ij}, dup|C_{\max}$ and investigated the number of duplicated tasks in the produced schedules. Tang et al. (2020) formulated the problem as a mixed integer linear program. Ahmad and Alam (2021) proposed a list scheduling algorithm incorporating task duplication for heterogeneous computing environments executing scientific big data workflow applications.

3. Computational complexity

An instance of the decision version of the ASAO problem requires an answer to the question: for a given positive integer C, does there exist a partition of N into m non-empty subsets $N^1, \dots, N^m$ for which $f(N^1, \dots, N^m)$ does not exceed C?

The proof below is a reduction from the Clique problem (CLIQUE) that is strongly NP-complete (Garey and Johnson, 1979) and can be stated as follows:

CLIQUE Given an integer $k > 1$ and a graph $G(V, E)$, where V is the set of nodes and E is the set of edges, such that

$$k < |V| \quad \text{and} \quad \frac{k^2 - k}{2} < |E|. \tag{2}$$

Does $G(V, E)$ contain k nodes that induce a subgraph where any two nodes are linked by an edge (such a subgraph is called complete or a k-clique)?

Observe that the largest number of edges in a graph with k nodes is $\frac{k^2-k}{2}$, and a graph with k nodes has this number of edges if and only if it is complete (is a k-clique).

Theorem 1. *The ASAO problem is NP-hard in the strong sense even when $m = 2$ and $p_i = 1$ for all $i \in N \cup M$.*

Proof. Let integer $k > 1$ and graph $G(V, E)$ be an instance of the CLIQUE problem, where $V = \{v_1, \dots, v_n\}$ is the set of nodes and E is the set of edges. The edge that links nodes v and u will be denoted by $\{v, u\}$. The corresponding instance of the decision version of the ASAO problem is constructed as follows.

For each $\{v, u\} \in E$, introduce a job $x_{\{v,u\}}$. Denote the set of all these jobs by X. Observe that $|X| = |E|$. Introduce two additional disjoint sets of jobs, T and T_0, such that

$$|T| = \frac{k^2 + k}{2} \quad \text{and} \quad |T_0| = 1. \tag{3}$$

The set N is a union of these three disjoint sets, X, T, and T_0. For each $v \in V$, introduce an operation y_v. Denote the set of all these operations by Y. Observe that $|Y| = |V|$. The set M is a union of two disjoint sets of operations, Y and an additional set S such that

$$|S| = n + |E| - \frac{k^2 - k}{2} - 1. \tag{4}$$

For each $j \in N$,

$$M(j) = \begin{cases} S & \text{if } \{j\} = T_0 \\ \{y_v\} \cup \{y_u\} & \text{if } j = x_{\{v,u\}} \in X \\ Y & \text{if } j \in T \end{cases}. \tag{5}$$

Finally,

$$C = n + |E| + k, \quad m = 2, \quad \text{and} \quad p_i = 1 \quad \text{for all} \quad i \in N \cup M.$$

Suppose that there exists a partition of N into 2 non-empty subsets N^1 and N^2 such that

$$f(N^1, N^2) \le C = n + |E| + k. \tag{6}$$

Without loss of generality assume that $T_0 \subseteq N^1$. Then, by virtue of (5), $S \subseteq \cup_{i \in N^1} M(i)$. Furthermore, this assumption leads to the inequality

$$T \cap N^2 \neq \emptyset, \tag{7}$$

because otherwise $T \subset N^1$, which, by virtue of (5), implies $Y \subset \cup_{i \in N^1} M(i)$, and consequently,

$$f(N^1, N^2) \ge \sum_{i \in N^1} p_i + \sum_{j \in \cup_{i \in N^1} M(i)} p_j \ge |T_0| + |T| + |S| + |Y|$$

$$= 1 + \frac{k^2 + k}{2} + n + |E| - \frac{k^2 - k}{2} - 1 + n > C,$$

which contradicts (6). This, in turn, implies that $T \cap N^1 = \emptyset$, because otherwise, by virtue of (5) and (7),

$$Y \subset \cup_{i \in N^1} M(i) \quad \text{and} \quad Y \subseteq \cup_{i \in N^2} M(i),$$

and consequently, taking into account (2),

$$f(N^1, N^2) \ge \frac{2|Y| + |S| + |X| + |T| + |T_0|}{2}$$

$$= n + \frac{n}{2} + \frac{|E|}{2} - \frac{k^2 - k}{4} - \frac{1}{2} + \frac{|E|}{2} + \frac{k^2 + k}{4} + \frac{1}{2} > C,$$

which contradicts (6).

Summarising the above,

$$T_0 \subseteq N^1, \quad S \subseteq \cup_{i \in N^1} M(i), \quad T \subseteq N^2, \quad Y \subseteq \cup_{i \in N^2} M(i), \tag{8}$$

and hence,

$$|X \cap N^2| \le C - |Y| - |T| = n + |E| + k - n - \frac{k^2 + k}{2} = |E| - \frac{k^2 - k}{2},$$

which, since $|E| = |X| = |X \cap N^2| + |X \cap N^1|$, gives

$$\frac{k^2 - k}{2} \le |X \cap N^1|. \tag{9}$$

Furthermore, by (6) and (8),

$$|\cup_{i \in N^1} M(i) \cap Y| + |X \cap N^1| \le C - |S| - |T_0|$$

$$= n + |E| + k - \left(n + |E| - \frac{k^2 - k}{2} - 1\right) - 1 = \frac{k^2 - k}{2} + k.$$

Moreover, by adding

$$|\cup_{i \in N^1} M(i) \cap Y| + |X \cap N^1| \le \frac{k^2 - k}{2} + k$$

and (9), we get

$$|\cup_{i \in N^1} M(i) \cap Y| \le k. \tag{10}$$

Denote by $\widetilde{V}$ the set of all nodes v in the graph $G(V, E)$ such that the corresponding operation $y_v \in \cup_{i \in N^1} M(i) \cap Y$, and denote by $\widetilde{E}$ the set of all edges $\{v, u\}$ in the graph $G(V, E)$ such that the corresponding job $x_{\{v,u\}} \in X \cap N^1$. According to (5), for each $\{v, u\} \in \widetilde{E}$, the nodes v and u are in $\widetilde{V}$. Hence, by virtue of (9) and (10) and by taking into account that the number of edges in a graph containing $|\widetilde{V}|$ nodes cannot exceed $\frac{|\widetilde{V}|^2 - |\widetilde{V}|}{2}$,

$$\frac{k^2 - k}{2} \le |X \cap N^1| = |\widetilde{E}| \le$$

$$\le \frac{|\widetilde{V}|^2 - |\widetilde{V}|}{2} = \frac{|\cup_{i \in N^1} M(i) \cap Y|^2 - |\cup_{i \in N^1} M(i) \cap Y|}{2} \le \frac{k^2 - k}{2}$$

which implies

$$|\widetilde{E}| = \frac{k^2 - k}{2}. \tag{11}$$

The right-hand side of (11) is the largest number of edges in a graph with k nodes. Therefore, the number of nodes in $\widetilde{V}$ cannot be less than k. This observation, together with (10), gives

$$k \le |\widetilde{V}| = |\cup_{i\in N^1} M(i) \cap Y| \le k,$$

and consequently, $|\widetilde{V}| = k$, which, by virtue of (11), implies that the subgraph induced by $\widetilde{V}$ is a k-clique.

Conversely, assume that the graph $G(V, E)$ has a k-clique. Without loss of generality, let $v_1, \dots, v_k$ be the nodes in this clique. Consider the partition of N into N^1 and N^2 where

$$N^1 = T_0 \cup \{x_{\{v_i,v_j\}} : 1 \le i < j \le k\}. \tag{12}$$

Then, according to (5),

$$\cup_{i\in N^1} M(i) = \{y_{v_1}, \dots, y_{v_k}\} \cup S,$$

and taking into account (3) and (4),

$$\sum_{i\in N^1} p_i + \sum_{j\in\cup_{i\in N^1} M(i)} p_j = 1 + \frac{k^2-k}{2} + k + n + |E| - \frac{k^2-k}{2} - 1 = C.$$

Furthermore, (12) implies that

$$N^2 = T \cup (X \setminus \{x_{\{v_i,v_j\}} : 1 \le i < j \le k\}), \tag{13}$$

which, in turn, by virtue of (5), implies

$$\cup_{i\in N^2} M(i) = Y.$$

Consequently, taking into account (3),

$$\sum_{i\in N^2} p_i + \sum_{j\in\cup_{i\in N^2} M(i)} p_j = |T| + |X \setminus \{\{v_i, v_j\} : i < j \le k\}| + |Y|$$

$$= \frac{k^2+k}{2} + |X| - \frac{k^2-k}{2} + |V| = k + |E| + n = C.$$

Hence, the partition of N, defined by (12) and (13), satisfies (6). □

Two particular cases of the ASAO problem, studied below, will be referred to as ASAO_M and ASAO_N. Observe that, for both problems, ASAO_M and ASAO_N, the number of machines m can vary from instance to instance.

For each instance of ASAO_M,

- $p_i = 1$ for all $i \in N \cup M$;
- the number of machines m is part of the input;
- $|M(i)| = 1$ for all $i \in N$.

Thus, each job has only one associated additional operation, although the same additional operation can be associated with several jobs.

As far as ASAO_N is concerned, for any $j \in M$, denote by $N(j)$ the set of all $i \in N$ such that $j \in M(i)$. For each instance of ASAO_N,

- $p_i = 1$ for all $i \in N \cup M$;
- the number of machines m is part of the input;
- $|N(j)| = 1$ for all $j \in M$.

Thus, each additional operation is associated with only one job, although the same job can be associated with several additional operations.

The proof of the following theorem is a reduction from the 3-partition problem (3-PARTITION), which is strongly NP-complete (Garey and Johnson, 1979) and can be stated as follows:

3-PARTITION Given $3r$ integers $a_1, \dots, a_{3r}$, each greater than one, and an integer B such that

$$\sum_{k=1}^{3r} a_k = rB, \tag{14}$$

and, for each k,

$$\frac{B}{4} < a_k < \frac{B}{2}. \tag{15}$$

Does there exist a partition of the set $\{1, 2, \dots, 3r\}$ into r subsets $Z_1, \dots, Z_r$ such that, for each Z_e,

$$\sum_{k\in Z_e} a_k = B? \tag{16}$$

Observe that (15) implies that if the desired partition exists, the cardinality of each Z_k is equal to three.

Theorem 2. *The ASAO_M problem is NP-hard in the strong sense.*

Proof. Consider an instance of the 3-PARTITION problem, i.e. $3r$ integers $a_1, \dots, a_{3r}$, each greater than one, and an integer B, satisfying (14) and (15). The corresponding instance of the decision version of the ASAO_M problem is constructed as follows. Set

- $m = r$ and $C = B$;
- $M = \{j_1, \dots, j_{3r}\}$ and $N = \cup_{k=1}^{3r} A_k$ where $A_1, \dots, A_{3r}$ are disjoint sets such that $|A_k| = a_k - 1$ for each $1 \le k \le 3r$;
- $N(j_k) = A_k$ for each $1 \le k \le 3r$, and $p_i = 1$ for all $i \in N \cup M$.

Observe that for this instance of the decision version of the ASAO_M problem

$$\sum_{i\in N\cup M} p_i = \sum_{k=1}^{3r} \sum_{g\in A_k\cup\{j_k\}} p_g = \sum_{k=1}^{3r} a_k = rB = mB. \tag{17}$$

Suppose that there exists a partition of N into m non-empty subsets $N^1, \dots, N^m$ such that

$$f(N^1, \dots, N^m) \le C = B. \tag{18}$$

Then, taking into account (1), for each $1 \le e \le m$,

$$B \ge \sum_{i\in N^e} p_i + \sum_{j\in\cup_{i\in N^e} M(i)} p_j,$$

which together with (17) gives

$$mB \ge \sum_{e=1}^{m} \left(\sum_{i\in N^e} p_i + \sum_{j\in\cup_{i\in N^e} M(i)} p_j \right) \ge \sum_{i\in N\cup M} p_i = mB,$$

and consequently,

$$\sum_{e=1}^{m} \left(\sum_{i\in N^e} p_i + \sum_{j\in\cup_{i\in N^e} M(i)} p_j \right) = \sum_{i\in N\cup M} p_i \tag{19}$$

and

$$\sum_{i\in N^e} p_i + \sum_{j\in\cup_{i\in N^e} M(i)} p_j = B \quad \text{for each } 1 \le e \le m. \tag{20}$$

For each $1 \le k \le 3m$, there exists a unique c such that $j_k \in \cup_{i\in N^c} M(i)$, because otherwise p_{j_k} will appear in the left-hand side of (19) more than once, which contradicts this equality. This implies that if index e is not equal to this c, then $A_k \cap N^e = \emptyset$, and consequently $A_k \subseteq N^c$. For each $1 \le e \le m$, denote by K_e the set of all k such that $j_k \in \cup_{i\in N^e} M(i)$. Then, taking into account (20),

$$B = \sum_{i\in N^e} p_i + \sum_{j\in\cup_{i\in N^e} M(i)} p_j = \sum_{k\in K_e} \sum_{g\in A_k\cup\{j_k\}} p_g = \sum_{k\in K_e} a_k.$$

So, $Z_1 = K_1, \dots, Z_r = K_r$ is a required partition of the set $\{1, 2, \dots, 3r\}$.

Conversely, let $Z_1, \dots, Z_r$ be a partition of the set $\{1, 2, \dots, 3r\}$ such that each Z_e satisfies (16). Then, taking into account that $m = r$,

$$N^1 = \cup_{k\in Z_1} A_k, \quad \dots, \quad N^m = \cup_{k\in Z_r} A_k$$

is a partition of N which satisfies (18) because, for each $1 \le e \le m$,

$$\sum_{i\in N^e} p_i + \sum_{j\in\cup_{i\in N^e} M(i)} p_j = \sum_{k\in Z_e} |A_k| + |Z_e| =$$

$$\sum_{k\in Z_e} (a_k - 1) + |Z_e| = \sum_{k\in Z_e} a_k = B. \quad \square$$

It is easy to see that the ASAO_N problem is also NP-hard in the strong sense. Indeed, the condition that, for each $j \in M$, $|N(j)| = 1$ implies that, for any two $i \in N$ and $u \in N$, $M(i) \cap M(u) = \emptyset$. So, there exists an optimal partition $N^1, \ldots, N^m$ such that, for any $i \in N$, there exists N^e such that $M(i) \subset N^e$ and $M(i) \cap N^k = \emptyset$ for all $k \neq e$. On the other hand, problem $P \parallel C_{max}$ is NP-hard in the strong sense (Garey and Johnson, 1979) and, for any instance of $P \parallel C_{max}$, its optimal makespan is equal to the optimal value of (1) for the instance of ASAO_N where the set $|N|$ is obtained by introducing for each $g \in J$ a job $i(g)$ such that $|M(i(g))| = p_g - 1$.

Recall that the ASAO problem with $m = 2$ and $p_i = 1$ for all $i \in N \cup M$ is equivalent to $P2|1prec, c_{ij} = \infty, p_i = 1, dup|C_{\max}$, and problems ASAO_N and ASAO_M are equivalent to $P|1in\text{-}tree, c_{ij} = \infty, p_i = 1, dup|C_{\max}$ and $P|1out\text{-}tree, c_{ij} = \infty, p_i = 1, dup|C_{\max}$, correspondingly. Thus, the three scheduling problems with communication delay and duplication are NP-hard in the strong sense.

4. Approximation algorithms

There exist a number of approximation algorithms for the $P \parallel C_{\max}$ scheduling problem (Pinedo, 2012; Leung, 2004). In the light of the discussion above, each of these algorithms is also applicable to the ASAO_N problem.

Consider the general case of the ASAO problem. For any partition $R^1, \ldots, R^r$ of N, the algorithm presented below and referred to as Partition Conversion constructs a partition of N into m subsets $P^1, \ldots, P^m$. For this partition, Theorem 3 below establishes an upper bound in terms of the partition $R^1, \ldots, R^r$ on the deviation of $f(P^1, \ldots, P^m)$ from the optimal value of this function. This leads to a two-phase optimisation procedure where the first phase is a choice of a partition $R^1, \ldots, R^r$ and the second phase is the conversion of $R^1, \ldots, R^r$ into $P^1, \ldots, P^m$. In other words, this two-phase procedure can be viewed as a family of approximation algorithms which differ from each other by the method of initial partitioning of N.

If $r = 1$ and consequently $R^1 = N$, i.e. no partition has been actually made, Partition Conversion is applied to the original set of jobs. Other two extreme choices of the initial partition are $R^1, \ldots, R^r$ where each $|R^k| = 1$ and consequently $r = |N|$, and, for ASAO_M, $R^1, \ldots, R^{|M|}$ where each R^k is $N(j)$ for some $j \in M$. It will be shown that, unless $P = NP$, the latter choice of the initial partition for ASAO_M leads to the best possible polynomial-time algorithm in terms of the guaranteed upper bound on the deviation from the optimal value of (1).

Partition Conversion constructs the required partition using the value

$$D = \left\lceil \frac{1}{m} \left(\sum_{j \in N} p_j + \sum_{u=1}^{r} \sum_{g \in \cup_{j \in R^u} M(j)} p_g \right) \right\rceil + \max_{1 \le u \le r} \left\{ \max_{j \in R^u} p_j + \sum_{g \in \cup_{j \in R^u} M(j)} p_g \right\} - 1$$

which is computed based on the input partition $R^1, \ldots, R^r$. Partition Conversion constructs P^1 by consecutively assigning to P^1 the jobs from R^1, then from R^2, and so on as long as the total processing time of the assigned jobs and the required associated additional operations does not exceed D. After that, in the same manner, using the remaining jobs, Partition Conversion constructs P^2 and so on. The choice of D guarantees that at most m sets P^e will be constructed. Since the ASAO problem requires to construct a partition consisting of exactly m sets, in the case when less than m sets were constructed, the final part of the algorithm redistributes some jobs from the already constructed sets P^e to new sets of the partition.

Partition Conversion

1: $P^1 = \emptyset$, $k = 1$, $e = 1$, $H^1 = R^1, \ldots, H^r = R^r$
2: **while** $k \le r$ **do**
3: $H = H^k$
4: **while** $H \neq \emptyset$ **do**
5: choose arbitrary $i \in H$
6: **if** $\sum_{j \in P^e \cup \{i\}} p_j + \sum_{j \in \cup_{g \in P^e \cup \{i\}} M(g)} p_j \le D$ **then**
7: $P^e = P^e \cup \{i\}$
8: $H^k = H^k \setminus \{i\}$
9: **end if**
10: $H = H \setminus \{i\}$
11: **end while**
12: **if** $H^k \neq \emptyset$ **then**
13: $e = e + 1$
14: $P^e = \emptyset$
15: **else**
16: $k = k + 1$
17: **end if**
18: **end while**
19: **while** $e < m$ **do**
20: $k = 1$
21: **while** $|P^k| = 1$ **do**
22: $k = k + 1$
23: **end while**
24: choose arbitrary $i \in P^k$
25: $P^k = P^k \setminus \{i\}$
26: $e = e + 1$
27: $P^e = \{i\}$
28: **end while**
29: **return** $P^1, \ldots, P^m$

Theorem 3. *Partition Conversion constructs a partition of N into m subsets $P^1, \ldots, P^m$ such that*

$$f(P^1, \ldots, P^m) - f^* \le \left\lceil \frac{1}{m} \left(\sum_{j \in N} p_j + \sum_{u=1}^{r} \sum_{g \in \cup_{j \in R^u} M(j)} p_g \right) \right\rceil + \max_{1 \le u \le r} \left\{ \max_{j \in R^u} p_j + \sum_{g \in \cup_{j \in R^u} M(j)} p_g \right\} - \max \left\{ \left\lceil \frac{1}{m} \sum_{i \in N \cup M} p_i \right\rceil, \max_{i \in N} \left(p_i + \sum_{j \in M(i)} p_j \right) \right\} - 1,$$

where f^* is the optimal value of (1).

Proof. Observe that if $P^e = \emptyset$, then the condition in line 6 is satisfied, which leads to the expansion of P^e by job i, chosen in line 5. Each time when P^e is expanded by job i, chosen in line 5, this job is eliminated from H^k and from H. Observe also that if $H^k \neq \emptyset$ after the execution of the while loop 4–11, then the next iteration of the while loop 2–18 commences with this H^k and $P^e = \emptyset$. Furthermore, index k is increased according to line 16, and consequently H^k is eliminated from further consideration, only when $H^k = \emptyset$. Hence, the while loop 2–18 terminates after a finite number of iterations with some partition $P^1, \ldots, P^{e'}$ of N.

In order to prove that $e' \le m$, assume to the contrary that $e' > m$. For each $1 \le e < e'$, denote by k_e the largest among all k such that $P^e \cap R^k \neq \emptyset$ and denote by K_e the set defined as follows. If the increase of e in line 13 (and consequently the termination of the construction of P^e) is triggered in line 12 by set H^{k_e+1}, then K_e is the set of all k such that $P^e \cap R^k \neq \emptyset$. Otherwise, if the increase of e in line 13 is triggered in line 12 by set H^{k_e}, then K_e is the set of all k such that $P^e \cap R^k \neq \emptyset$ and $k < k_e$.

In the former case, i.e. in the case when $k_e \in K_e$, the while loop 4–11 failed to add to P^e an element of H^{k_e+1}. So, for any $i \in H^{k_e+1}$,

$$\sum_{j\in P^e} p_j + \sum_{k\in K_e} \sum_{j\in \cup_{g\in R^k} M(g)} p_j + p_i + \sum_{j\in M(i)} p_j \geq \sum_{j\in P^e\cup\{i\}} p_j + \sum_{j\in \cup_{g\in P^e\cup\{i\}} M(g)} p_j \geq D+1$$

and consequently

$$\sum_{j\in P^e} p_j + \sum_{k\in K_e} \sum_{j\in \cup_{g\in R^k} M(g)} p_j \geq \frac{1}{m}\left(\sum_{j\in N} p_j + \sum_{u=1}^{r} \sum_{g\in \cup_{j\in R^u} M(j)} p_g\right). \tag{21}$$

In the latter case, i.e. in the case when $k_e \notin K_e$, the while loop 4–11 failed to add to P^e the element $i \in R^{k_e}$, chosen as a result of the last execution of line 5. Hence,

$$\sum_{j\in P^e} p_j + \sum_{k\in K_e} \sum_{j\in \cup_{g\in R^k} M(g)} p_j + p_i + \sum_{j\in \cup_{g\in R^{k_e}} M(g)} p_j \geq \sum_{j\in P^e\cup\{i\}} p_j + \sum_{j\in \cup_{g\in P^e\cup\{i\}} M(g)} p_j$$

$$\geq D+1,$$

which again leads to (21). Taking into account that, for any $1 \leq u \leq m$ and $1 \leq v \leq m$ such that $u \neq v$, $P^u \cap P^v = \emptyset$ and $K_u \cap K_v = \emptyset$, we obtain by adding (21) for all $1 \leq e \leq m$,

$$\sum_{e=1}^{m} \sum_{j\in P^e} p_j \geq \sum_{j\in N} p_j + \sum_{u=1}^{r} \sum_{g\in \cup_{j\in R^u} M(j)} p_g - \sum_{e=1}^{m} \sum_{k\in K_e} \sum_{j\in \cup_{g\in R^k} M(g)} p_j \geq \sum_{j\in N} p_j,$$

which contradicts the assumption that $P^{m+1} \neq \emptyset$.

If $e' < m$, then the partition $P^1, \ldots, P^{e'}$, constructed by the while loop 4–11, becomes the initial current partition for the while loop 19–28. At each iteration, the while loop 19–28 constructs a new current partition of N by finding a set in the current partition, containing more than one element (such a set exists by virtue of $m < |N|$), removing one element from this set, and introducing a new set that is comprised of only this one element.

Taking into account that

$$f^* \geq \max\left\{\left\lceil \frac{1}{m} \sum_{i\in N\cup M} p_i \right\rceil, \max_{i\in N}\left(p_i + \sum_{j\in M(i)} p_j\right)\right\}$$

and that $f(P^1, \ldots, P^m) \leq D$, we have

$$f(P^1, \ldots, P^m) - f^* \leq D - \max\left\{\left\lceil \frac{1}{m} \sum_{i\in N\cup M} p_i \right\rceil, \max_{i\in N}\left(p_i + \sum_{j\in M(i)} p_j\right)\right\}$$

$$= \left\lceil \frac{1}{m}\left(\sum_{j\in N} p_j + \sum_{u=1}^{r} \sum_{g\in \cup_{j\in R^u} M(j)} p_g\right)\right\rceil + \max_{1\leq u\leq r}\left\{\max_{j\in R^u} p_j + \sum_{g\in \cup_{j\in R^u} M(j)} p_g\right\} - 1$$

$$- \max\left\{\left\lceil \frac{1}{m} \sum_{i\in N\cup M} p_i \right\rceil, \max_{i\in N}\left(p_i + \sum_{j\in M(i)} p_j\right)\right\}$$

which completes the proof. □

Recall that Partition Conversion specifies a family of algorithms, where a particular algorithm is defined by the choice of the initial partition $R^1, \ldots, R^r$. The performance guarantee presented in Theorem 3 is rather complicated, and it is not known in general what initial partition to choose in order to obtain a good approximation of the optimal partition. However, as stated in the corollary below, the family contains an algorithm with the best possible performance guarantee for problem ASAO_M.

Corollary 1. *If, for ASAO_M, $r = |M|$ and each R^u is $N(j)$ for some $j \in M$, then*

$$f(P^1, \ldots, P^m) - f^* \leq 1$$

where f^* is the optimal value of (1).

Proof. From Theorem 3,

$$f(P^1, \ldots, P^m) - f^*$$

$$\leq \left\lceil \frac{1}{m}(|N| + |M|) \right\rceil + 2 - 1 - \max\left\{\left\lceil \frac{1}{m}(|N| + |M|) \right\rceil, 2\right\} = 1. \quad \square$$

As an example, consider the following instance of the ASAO_M problem. Let $m = 3$, $N = \{J_1, \ldots, J_{10}\}$, $M = \{O_1, O_2, O_3, O_4\}$, $N(O_1) = \{J_1, J_2\}$, $N(O_2) = \{J_3, J_4\}$, $N(O_3) = \{J_5\}$, $N(O_4) = \{J_6, \ldots, J_{10}\}$. Let $r = 4$ and $R^k = N(O_k)$ for $1 \leq k \leq 4$. Then, $D = \left\lceil \frac{1}{3}(10+4) \right\rceil + \max\{2,2,2,2\} - 1 = 6$. Assume that in line 5 of Partition Conversion, the job J_i with the smallest i is always chosen. In the first iteration of the while loop 2–18, jobs J_1 and J_2 are assigned to subset P^1. In the second iteration, jobs J_3 and J_4 are also assigned to subset P^1. In the third iteration, it turns out that adding the job from R^3 to P^1 results in exceeding the bound D on the total time required to process the jobs and additional operations in P^1. Therefore, lines 13–14 are executed, such that a new subset P^2 is created. In the fourth iteration of the while loop 2–18, job J_5 is added to P^2. In the next iteration, jobs J_6, J_7, J_8 are added to P^2. Afterwards, since adding either J_9 or J_{10} to P^2 would make the total time required to process the jobs in P^2 and their additional operations larger than D, a new subset P^3 is constructed. In the last iteration of the while loop 2–18, jobs J_9 and J_{10} are added to P^3. Since m subsets were created, the while loop 19–28 is not executed. Thus, the obtained partition is $P^1 = \{J_1, J_2, J_3, J_4\}$, $P^2 = \{J_5, J_6, J_7, J_8\}$, $P^3 = \{J_9, J_{10}\}$, and its makespan is $\max\{6, 6, 3\} = 6$. It is easy to check that the optimum makespan for the considered instance is 5, and it can be achieved by choosing $P^1 = \{J_1, J_2, J_5\}$, $P^2 = \{J_3, J_4, J_6\}$ and $P^3 = \{J_7, J_8, J_9, J_{10}\}$.

5. Mixed integer linear programming formulations

In this section, two mixed integer linear programming formulations are proposed for the ASAO problem. For the first formulation, denoted by ILP1, define for all $i \in M$ and $k = 1, \ldots, m$ binary variables x_{ik} such that $x_{ik} = 1$ if additional operation i is executed on machine k, and $x_{ik} = 0$ otherwise. Similarly, for all $j \in N$ and $k = 1, \ldots, m$, define binary variables y_{jk} such that $y_{jk} = 1$ if job j is executed on machine k, and $y_{jk} = 0$ otherwise. Let $C_{\max}$ be a variable denoting the value of the makespan. The problem can be stated as follows.

$$\text{(ILP1)} \quad \text{minimise} \quad C_{\max} \tag{22}$$

$$\text{s.t.} \quad \sum_{k=1}^{m} y_{jk} = 1 \quad \forall\, j \in N \tag{23}$$

$$y_{jk}|M(j)| \leq \sum_{i\in M(j)} x_{ik} \quad \forall\, j \in N,\ k = 1, \ldots, m \tag{24}$$

$$\sum_{i\in M} x_{ik} p_i + \sum_{j\in N} y_{jk} p_j \leq C_{\max} \quad \forall\, k = 1, \ldots, m \tag{25}$$

$$y_{jk} = 0 \quad \forall\, j \in N,\ k = j+1, \ldots, m \tag{26}$$

$$x_{ik} \in \{0,1\} \quad \forall\, i \in M,\ k = 1, \ldots, m \tag{27}$$

$$y_{jk} \in \{0,1\} \quad \forall\, j \in N,\ k = 1, \ldots, m \tag{28}$$

In the above formulation, constraints (23) guarantee that each job is assigned to exactly one machine. The additional operations required by each job j are performed on the machine executing j by (24). Inequalities (25) ensure that all machines finish processing by time $C_{\max}$. Equalities (26) are symmetry breaking constraints. Namely, it is assumed that job j can only be assigned to a machine with number not exceeding j. In other words, the numbering of the machines is not arbitrary. The machine executing job 1 is always considered the first machine. If job 2 is assigned to a different machine than job 1, then the machine executing job 2 is considered the second machine, and so on. The formulation (22)–(28) contains $O(m(|N| + |M|))$ variables and $O(m|N|)$ constraints.

Although the symmetry breaking constraints (26) are used in ILP1, there may still exist many optimum solutions that differ only by the numbering of the machines. Therefore, the second formulation, denoted by ILP2, where the machines are not numbered at all, is proposed. A job i will be called an anchor if no job $j > i$ is assigned to the same machine as i. For each $i, j \in N$ such that $j \leq i$, define a binary variable x_{ij} such that $x_{ij} = 1$ if and only if, for some $1 \leq e \leq m$, i is an anchor assigned to machine e, and j is also assigned to machine e. Moreover, for each $i \in N$ and $j \in M$, define a variable y_{ij} such that $y_{ij} = 1$ if, for some $1 \leq e \leq m$, i is an anchor assigned to machine e and j is an additional operation required by some job assigned to machine e, and $y_{ij} = 0$ otherwise. The problem can be formulated as follows.

$$\text{(ILP2)} \quad \text{minimise} \quad C_{\max} \tag{29}$$

$$\text{s.t.} \quad \sum_{i \in N} x_{ii} = m \tag{30}$$

$$\sum_{i=j}^{|N|} x_{ij} = 1 \quad \forall\, j \in N \tag{31}$$

$$\sum_{j=1}^{i} x_{ij} \leq |N| x_{ii} \quad \forall\, i \in N \tag{32}$$

$$|M(j)| x_{ij} \leq \sum_{g \in M(j)} y_{ig} \quad \forall\, \{i, j\} \subset N,\ j \leq i \tag{33}$$

$$\sum_{j=1}^{i} p_j x_{ij} + \sum_{g \in M} p_g y_{ig} \leq C_{\max} \quad \forall\, i \in N \tag{34}$$

$$x_{ij} \in \{0, 1\} \quad \forall\, \{i, j\} \subset N,\ j \leq i \tag{35}$$

$$y_{ij} \in \{0, 1\} \quad \forall\, i \in N,\ j \in M \tag{36}$$

In the above program, constraints (30) guarantee that there are exactly m anchors. Each job is assigned to exactly one machine by (31). Inequalities (32) ensure that if $x_{ij} = 1$ for some $j \leq i$ then job i is an anchor. By constraints (33), all additional operations required by job j are executed on the machine to which j is assigned. Inequalities (34) guarantee that all machines finish their computations by time C_{max}. The formulation (29)–(36) contains $O(|N|(|N|+|M|))$ variables and $O(|N|^2)$ constraints.

6. Randomised constructive algorithms

In this section, randomised constructive algorithms are proposed. As a subroutine, the randomised algorithms use a greedy approach, where each consecutive job is assigned to the machine which is considered the best. If the partial solution resulted from assigning the job to machine k_1 is shorter than the partial solution obtained by choosing machine k_2, then machine k_1 is considered better than k_2. Moreover, if the two solutions have the same length and the additional processing time required to process the job on machine k_1 is shorter than the additional processing time required to process the job on machine k_2, then machine k_1 is also considered better than k_2. Note that the two processing times may be different because some of the additional operations associated with the current job may already be present on machine k_1 or machine k_2.

Clearly, the performance of the above greedy method depends on the order in which the jobs are processed. Unfortunately, it is not known what job sequence could be beneficial. Therefore, the randomised constructive algorithms run the greedy algorithm many times, using random job orders, and return the best solution found. The algorithm that analyses K random job orders will be denoted by Greedy(K).

Assigning a job to a machine requires checking which of its associated additional operations are already assigned to the machine. Using a hash table based set data structure, the presence of a given operation can be checked in average $O(1)$ time. Hence, assigning a single job to a given machine takes $O(|M|)$ time. In order to find the best machine for a job, it is necessary to assign it temporarily to each machine. This operation is executed by the greedy method for all jobs, and hence, a solution is constructed from a given job sequence in $O(m|N||M|)$ time. In consequence, the complexity of the randomised Greedy(K) algorithm is $O(Km|N||M|)$.

It should be noted that an attempt at solving the ASAO problem using metaheuristics, such as a genetic algorithm and a variable neighbourhood search, was also made. Unfortunately, the properties of the problem make it very difficult to solve with such methods. When a solution with a balanced machine load is changed by relocating a single job or a group of jobs to a different machine, the makespan rarely becomes better, and even when it does, the improvement is usually very small. Indeed, in order to significantly improve such a solution, an algorithm would have to find a group of jobs that share associated additional operations but are assigned to different machines. Then, the group of jobs should be reassigned so that they are all executed on the same machine k, and in consequence, some of their additional operations would be executed on a smaller number of machines than before the change. Moreover, some of the other jobs previously assigned to machine k should be moved to the other machines in order to balance the load. Finding such a complex beneficial solution change in a metaheuristic algorithm is very improbable. The conducted computational experiments showed that when K is large enough, the Greedy(K) algorithms obtain better results than metaheuristics, and still have a shorter execution time. Therefore, the designed genetic algorithm and variable neighbourhood search are not presented in the paper.

7. Computational experiments

The proposed algorithms were tested in a series of computational experiments. The algorithms were implemented in C++ and run on an Intel Core i7-7820HK CPU @ 2.90 GHz with 32 GB RAM. The mixed integer linear programs were solved using Gurobi.

It turned out that solving the mixed integer linear programs may take many hours even for moderate size instances. Therefore, a time limit of one hour was imposed on models ILP1 and ILP2. Since the optimal solutions were not always known, the obtained results were compared to the larger among the two lower bounds found by Gurobi while solving ILP1 and ILP2. Solution quality was measured by the average relative error with respect to this lower bound.

7.1. The general case

In the first set of experiments, the behaviour of the integer linear programming formulations, several approximation algorithms based on Partition Conversion and the randomised constructive algorithms for the general case of the problem was analysed. For heuristics Greedy(K), many values of parameter K were studied. Naturally, choosing a larger K results in obtaining better solutions, but also in a longer computation time. In what follows, the results delivered by Greedy(1) and Greedy($10|N|$) will be presented. These two algorithm variants show what can be achieved with a minimal effort, i.e. when only one random job sequence is selected, and what results can be obtained when a significant amount of time is used to produce many solutions.

In the generated test instances, the number of machines was $m \in [2, 16]$, and the numbers of jobs and additional operations were between 10 and 50. The execution times p_j were selected randomly from the range $[1, 20]$. The numbers of additional operations required by individual jobs were controlled by the density parameter $d \in [0.1, 0.6]$. Precisely, for each job i and additional operation j, a random number $r_{ij} \in [0, 1]$ was generated. If $r_{ij} < d$, then operation j was associated with job i. If after selecting all values r_{ij} some operation j was not associated with any job, then a job was chosen randomly, and j was added to the list of operations associated with this chosen job. Unless

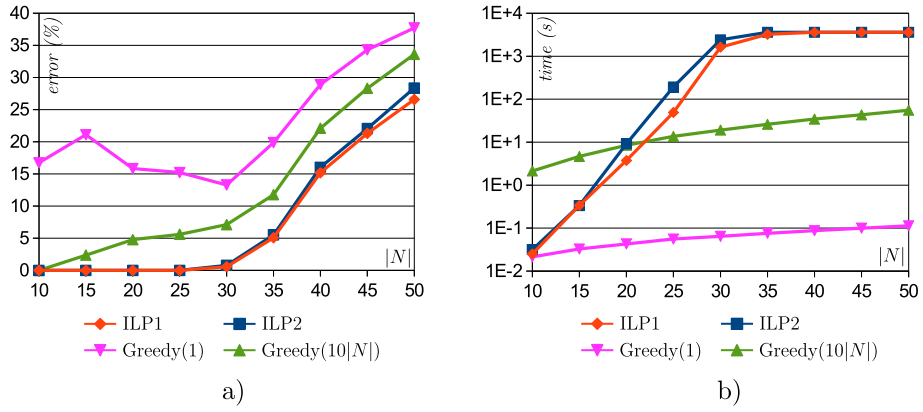

Fig. 1. Algorithm performance vs. $|N|$, for $|M| = |N|$, $m = 5$. (a) Average quality, (b) average execution time.

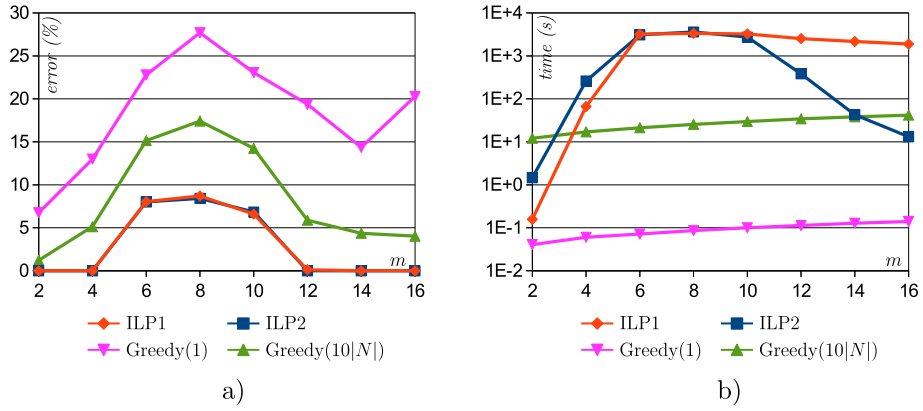

Fig. 2. Algorithm performance vs. m, for $|N| = |M| = 30$. (a) Average quality, (b) average execution time.

stated otherwise, $d = 0.25$ was used. For each analysed combination of parameter values, 10 instances were generated and solved.

Fig. 1 presents the results obtained for tests with $m = 5$, variable $|N|$ and $|M| = |N|$. All instances with $|N| \leq 25$ were solved to optimality by both ILP1 and ILP2. However, the time limit of one hour was reached by these algorithms for some instances with $|N| \in \{30, 35\}$, and for all tests with $|N| \geq 40$. The differences between the performance of the two ILPs are not large, but ILP1 is slightly faster than ILP2 for $|N| \leq 35$, and it obtains slightly better results than ILP2 for $|N| \geq 30$. The relative errors reported for both algorithms grow fast when $|N|$ becomes large, and exceed 25% for $|N| = 50$. However, this effect may be to a large degree caused by the increasing distance between the lower bound found and the actual optimum. The results delivered by Greedy(1) for instances with $|N| \leq 25$ are not good, as the average errors are between 15% and 21%. However, heuristic Greedy(10|N|) performs much better, with errors below 6% for $|N| \leq 25$. For instances with $|N| > 30$, the performance of Greedy(10|N|) in comparison to the ILPs is rather stable, and the difference between Greedy(1) and Greedy(10|N|) decreases. Standard deviations of the percentage errors obtained by the algorithms were also computed, but for the sake of readability, they are not presented in the charts. Of course, the standard deviations of ILP1 and ILP2 errors are 0 for $|N| \leq 25$. For larger instances, the standard deviations of ILP1, ILP2 and Greedy(10|N|) are very similar, and they are in the range between 1% and 4%. The results obtained by Greedy(1) are less stable, especially for instances with small $|N|$. The standard deviation of Greedy(1) errors is more than 13% for $|N| = 10$, and it decreases to around 4% when $|N|$ becomes large. Naturally, Greedy(1) is the fastest among the analysed algorithms (see Fig. 1b). Greedy(10|N|) is faster than ILP1 and ILP2 for $|N| \geq 25$.

The behaviour of several algorithms using the Partition Conversion method was also tested. Recall that Partition Conversion requires an initial partition of the set of jobs as part of input. The analysed partitions were: a single set containing all jobs, a partition consisting of N sets having 1 job each, and different randomly generated partitions. Unfortunately, the results obtained by this group of algorithms were not good. The best solutions were usually obtained using an initial partition consisting of a single set, but even in this case the errors were between 60% and 80%. Therefore, the results delivered by the algorithms based on Partition Conversion are not included in Fig. 1 and the following charts, in order to better expose the differences between the other algorithms. It seems that Partition Conversion may not be a good tool for solving the general version of the ASAO problem in practice. Indeed, it is not certain whether an initial partition leading to obtaining a good assignment of jobs to machines always exists, and even if it does, it is not known how to find it.

Fig. 2 shows the algorithms' performance for $|N| = |M| = 30$ and variable m. The quality of the solutions delivered by ILP1 and ILP2 is almost identical for all values of m. If m is very small or very large, the optimum solution can be found within the one hour time limit by both algorithms. The most difficult instances are the ones with $m \in [6, 10]$. When m is small, ILP1 is faster than ILP2. However, optimum solutions for instances with $m \geq 12$ are found by ILP2 in a much shorter time than by ILP1. Similarly as in the previous experiment, heuristic Greedy(10|N|) delivers much better results than Greedy(1). Moreover, it is significantly faster than both ILPs for $m \in [4, 12]$. The distance between the quality of the solutions produced by the ILPs and Greedy(10|N|) is between 5% and 9% of the lower bound for such tests. The standard deviations of the percentage errors delivered by ILP1, ILP2 and Greedy(10|N|) are again very similar for the cases where ILP1 and ILP2 do not always find optimum solutions, i.e. when $m \in [6, 10]$. Their values are between 5% and 9%, which means they are substantially larger than in the previous experiment, where

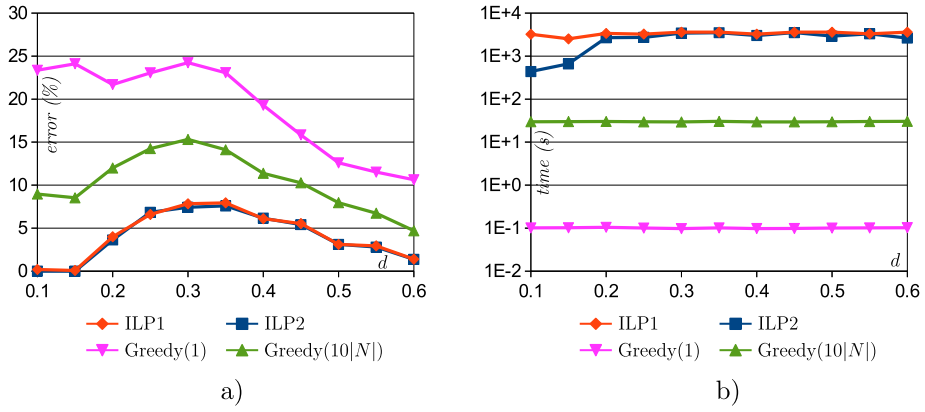

Fig. 3. Algorithm performance vs. d, for $|N| = |M| = 30$, $m = 10$. (a) Average quality, (b) average execution time.

the number m of machines was smaller. The standard deviations of Greedy(1) errors increase with growing m and are close to 10% for $m \geq 12$. Greedy(10|N|) is again much more stable than Greedy(1), with standard deviations smaller than 4% for such tests.

The experiments with changing density parameter d show that ILP1 and ILP2 deliver very similar results for all analysed values of $d \in [0.1, 0.6]$ (see Fig. 3). The obtained errors are the largest for $d \in [0.2, 0.4]$. Both algorithms reach the time limit for a majority of tests with $|N| = |M| = 30$, $m = 10$ and $d \geq 0.2$. However, ILP2 is faster than ILP1 for $d \in \{0.1, 0.15\}$. The randomised constructive algorithms perform better for large values of d than for the small ones. This is probably caused by the fact that when d is large, then many additional operations have to be executed on multiple machines in the optimal solutions. Hence, the optimum makespans are larger than for small d, and in consequence, it is easier to produce high quality solutions. Once again, the standard deviations of the errors obtained by ILP1, ILP2 and Greedy(10|N|) are similar, and Greedy(1) has larger standard deviations, especially when d is small.

In summary, for the general version of the ASAO problem, ILP1 and ILP2 obtain very similar results. Their running times are usually also similar, but ILP2 is faster than ILP1 when m is large or d is small. For solving small instances, using one of the ILPs seems the best option. For larger instances, Greedy(10|N|) or some other variant of the randomised constructive algorithm can be used to obtain a reasonably good solution in a shorter time. Algorithms based on Partition Conversion do not perform well, at least for the input partitions analysed in the conducted experiments.

7.2. Problem ASAO_M

Recall that for problem ASAO_M, Partition Conversion using the initial partition $N(1), \ldots, N(|M|)$ constructs a schedule for which the makespan is greater than the optimum by at most one. From now on, this particular algorithm will be called PC_M. The goal of the experiments presented here was to check how the remaining proposed algorithms perform in comparison to PC_M for ASAO_M.

In order to generate demanding instances, a procedure inspired by the transformation used in the proof of Theorem 2 was designed. For a given number of machines $m \in \{5, 10\}$, the number of jobs was $|N| = lm$, where $l \in \{20, 30, 40, 50\}$, and the number of additional operations was set to $|M| = 3m$. Then, a set of $3m$ random numbers $a_1, \ldots, a_{3m}$ greater than 1, such that $\sum_{k=1}^{3m} a_k = |N| + 3m$, was generated. The kth additional operation, where $k \in \{1, \ldots, |M|\}$, was associated with $a_k - 1$ unique jobs. For each pair of m and l values, 10 instances were generated and solved. The results obtained for such tests are summarised in Tables 1 and 2. A set of random instances constructed without using the above procedure, similarly as for the general case of the ASAO problem, was also generated. A comparison of the results

Table 1
Average solution quality for ASAO_M problem.

m	$\|N\|$	ILP1	ILP2	PC_M	Greedy(1)	Greedy(10|N|)
5	100	0.00%	3.91%	3.91%	40.72%	12.57%
5	150	0.00%	2.12%	2.12%	33.35%	11.13%
5	200	0.00%	1.16%	1.16%	26.45%	9.66%
5	250	0.00%	0.94%	0.94%	21.51%	8.42%
10	200	0.00%	4.35%	4.35%	53.91%	21.74%
10	300	0.00%	3.93%	2.73%	48.96%	17.83%
10	400	0.00%	4.41%	2.09%	48.50%	16.01%
10	500	0.00%	3.20%	1.51%	47.00%	14.67%

Table 2
Average algorithm execution time for ASAO_M problem (seconds).

m	$\|N\|$	ILP1	ILP2	PC_M	Greedy(1)	Greedy(10|N|)
5	100	1.18E−1	3.60E+3	4.49E−3	1.92E−1	1.91E+2
5	150	1.94E−1	3.60E+3	5.89E−3	2.86E−1	4.29E+2
5	200	2.84E−1	3.60E+3	7.51E−3	3.79E−1	7.64E+2
5	250	3.67E−1	3.60E+3	8.90E−3	4.72E−1	1.19E+3
10	200	6.91E−1	3.60E+3	7.98E−3	5.83E−1	1.16E+3
10	300	1.39E+0	3.60E+3	1.08E−2	8.66E−1	2.60E+3
10	400	2.28E+0	3.60E+3	1.36E−2	1.15E+0	4.60E+3
10	500	2.87E+0	3.61E+3	1.66E−2	1.41E+0	7.16E+3

obtained for the two sets confirmed that the method based on the proof of Theorem 2 produced instances which were more difficult for the ILPs and Greedy(K) algorithms.

Recall that the performance guarantee of PC_M pertains to absolute error. Hence, the relative error of this algorithm is smaller for the instances with larger optimum values of the makespan. It is worth noting that also for the remaining algorithms, the errors reported in Table 1 decrease with increasing $|N|$ for a fixed m. This is caused by the fact that when $|N|$ becomes larger, but $|M| = 3m$ is fixed, the number of jobs associated with a given additional operation increases. In consequence, many additional operations have to be duplicated on several machines in the optimum solutions, and hence, the suboptimal assignments found by the algorithms are relatively closer to the optimum. Although the instances analysed in this subsection are much larger than the ones representing the general case of the ASAO problem, ILP1 was able to solve them to optimality in a very short time. On average, its computations took 0.367 s for the largest tests with $m = 5$, and less than 3 s for the largest tests with $m = 10$ (see Table 2). Contrarily, ILP2 did not finish computations on any instance within the time limit. Indeed, its average running time is at least one hour for each group of tests. Moreover, the makespans found by ILP2 were up to several percent longer than the optimum. The results delivered by the randomised constructive algorithms show that they do not benefit from the special structure of the instances of ASAO_M. Indeed, the errors obtained by Greedy(1) are very large. The results obtained by

Greedy(10|N|) are substantially better, but they are not comparable to the ones returned by PC_M. Moreover, the long running time of Greedy(10|N|), which exceeds one hour for the largest tests, makes it unsuitable for solving instances with hundreds of jobs.

All in all, it seems that ILP1 can efficiently handle even very large instances of the ASAO_M problem. PC_M always delivers almost optimal solutions and is guaranteed to have a very low runtime. ILP2 and the randomised constructive algorithms should not be used for solving ASAO_M.

8. Conclusions

This paper considers a combinatorial optimisation problem ASAO, inspired by applications in software testing and manufacturing, and proves that even very restricted particular cases of this problem are NP-hard in the strong sense. These computational complexity results are complemented by a method of designing approximation algorithms and an upper bound on the deviation from the optimal value of the objective function which is valid for all algorithms constructed by this method. Furthermore, two mixed integer linear programming formulations and randomised constructive algorithms are proposed and tested by means of computational experiments. The obtained results show that both ILPs produce good solutions for the general version of the problem, and that ILP1 quickly delivers optimum solutions even for very large instances of ASAO_M. The Greedy(10|N|) algorithm may be used for finding reasonably good solutions of the general ASAO problem when the ILP1 and ILP2 running times are too long. The results presented in the paper also contribute to the realm of scheduling with communication delay, since the studied combinatorial optimisation problem can be viewed as a parallel machine scheduling problem with infinitely large communication delay, duplication and the partial order on the set of jobs in the form of a graph where a path cannot contain more than one arc. Given the computational complexity of the considered problem, a logical continuation of the presented research would be the design and analysis of various further heuristic and approximation algorithms, including but not limited to the algorithms based on the method presented in the paper.

CRediT authorship contribution statement

Yakov Zinder: Writing – review & editing, Writing – original draft, Methodology, Conceptualization, Formal analysis. **Joanna Berlińska:** Writing – review & editing, Writing – original draft, Visualization, Software, Methodology, Conceptualization, Investigation. **Bertrand M.T. Lin:** Writing – review & editing, Writing – original draft, Methodology, Conceptualization.

Declaration of competing interest

None.

Data availability

Data will be made available on request.

References

Ahmad, W., Alam, B., 2021. An efficient list scheduling algorithm with task duplication for scientific big data workflow in heterogeneous computing environments. Concurr. Comput.: Pract. Exper. 33 (5), e5987. http://dx.doi.org/10.1002/cpe.5987.

Ahmad, I., Kwok, Y.-K., 1998. On exploiting task duplication in parallel program scheduling. IEEE Trans. Parallel Distrib. Syst. 9 (9), 87–892. http://dx.doi.org/10.1109/71.722221.

Bampis, E., Giannakos, A., König, J.-C., 1996. On the complexity of scheduling with large communication delays. European J. Oper. Res. 94 (2), 252–260. http://dx.doi.org/10.1016/0377-2217(96)00124-5.

Bansal, S., Kumar, P., Singh, K., 2003. An improved duplication strategy for scheduling precedence constrained graphs in multiprocessor systems. IEEE Trans. Parallel Distrib. Syst. 14 (6), 533–544. http://dx.doi.org/10.1109/TPDS.2003.1206502.

Chrétienne, P., 1994. Tree scheduling with communication delays. Discrete Appl. Math. 49 (1), 129–141. http://dx.doi.org/10.1016/0166-218X(94)90205-4.

Colin, J.Y., Chrétienne, P., 1991. C.P.M. scheduling with small communication delays and task duplication. Oper. Res. 39 (4), 680–684. http://dx.doi.org/10.1287/opre.39.4.680.

Darbha, S., Agrawal, D.P., 1997. A task duplication based scalable scheduling algorithm for distributed memory systems. J. Parallel Distrib. Comput. 46 (1), 15–27. http://dx.doi.org/10.1006/jpdc.1997.1376.

Drozdowski, M., 2009. Scheduling for Parallel Processing. Springer.

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

Giroudeau, R., König, J.-C., 2007. Scheduling with communication delays. In: Levner, E. (Ed.), Multiprocessor Scheduling: Theory and Applications. IntechOpen.

Graham, R.L., Lawler, E.L., Lenstra, J.K., Kan, A.R., 1979. Optimization and approximation in deterministic sequencing and scheduling: a survey. In: Hammer, P., Johnson, E., Korte, B. (Eds.), Discrete Optimization II. In: Annals of Discrete Mathematics, vol. 5, pp. 287–326.

Hoogeveen, J., Lenstra, J.K., Veltman, B., 1994. Three, four, five, six, or the complexity of scheduling with communication delays. Oper. Res. Lett. 16 (3), 129–137. http://dx.doi.org/10.1016/0167-6377(94)90024-8.

Jakoby, A., Reischuk, R., 1992. The complexity of scheduling problems with communication delays for trees. In: Proceedings of the 3rd Scandinavian Workshop on Algorithm Theory. pp. 165–177.

Jung, H., Kirousis, L., Spirakis, P., 1989. Lower bounds and efficient algorithms for multiprocessor scheduling of dags with communication delays. In: Proceedings of the First Annual ACM Symposium on Parallel Algorithms and Architectures. pp. 254–264.

Kruatrachue, B., Lewis, T., 1988. Grain size determination for parallel processing. IEEE Softw. 5 (1), 23–32. http://dx.doi.org/10.1109/52.1991.

Lenstra, J.K., Veldhorst, M., Veltman, B., 1996. The complexity of scheduling trees with communication delays. J. Algorithms 20 (1), 157–173. http://dx.doi.org/10.1006/jagm.1996.0007.

Leung, J.Y., 2004. Handbook of Scheduling: Algorithms, Models, and Performance Analysis. CRC Press.

Maiti, B., Rajaraman, R., Stalfa, D., Svitkina, Z., Vijayaraghavan, A., 2020. Scheduling precedence-constrained jobs on related machines with communication delay. In: 2020 IEEE 61st Annual Symposium on Foundations of Computer Science. FOCS, pp. 834–845.

Munier, A., Hanen, C., 1997. Using duplication for scheduling unitary tasks on m processors with unit communication delays. Theoret. Comput. Sci. 178 (1), 119–127. http://dx.doi.org/10.1016/S0304-3975(97)88194-7.

Orr, M., Sinnen, O., 2020. Integrating task duplication in optimal task scheduling with communication delays. IEEE Trans. Parallel Distrib. Syst. 31 (10), 2277–2288. http://dx.doi.org/10.1109/TPDS.2020.2989767.

Papadimitriou, C.H., Yannakakis, M., 1990. Towards an architecture-independent analysis of parallel algorithms. SIAM J. Comput. 19 (2), 322–328. http://dx.doi.org/10.1145/62212.62262.

Park, G.-L., Shirazi, B., Marquis, J., 1998. Mapping of parallel tasks to multiprocessors with duplication. In: Proceedings of the Thirty-First Hawaii International Conference on System Sciences, Volume VII. pp. 96–105. http://dx.doi.org/10.1109/HICSS.1998.649185.

Pinedo, M.L., 2012. Scheduling: Theory, Algorithms, and Systems. Springer.

Rayward-Smith, V.J., 1987. UET scheduling with unit interprocessor communication delays. Discrete Appl. Math. 18 (1), 55–71. http://dx.doi.org/10.1016/0166-218X(87)90042-4.

Tang, Q., Zhu, L.-H., Zhou, L., Xiong, J., Wei, J.-B., 2020. Scheduling directed acyclic graphs with optimal duplication strategy on homogeneous multiprocessor systems. J. Parallel Distrib. Comput. 138, 115–127. http://dx.doi.org/10.1016/j.jpdc.2019.12.012.

Zinder, Y., Roper, D., 1995. A minimax combinatorial optimisation problem on an acyclic directed graph: polynomial-time algorithms and complexity. In: Proceedings of the A.C. Aitken Centenary Conference. Dunedin, pp. 391–400.