European Journal of Operational Research 316 (2024) 976–987

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Production, manufacturing, transportation and logistics

Exploiting symmetry for the job sequencing and tool switching problem

Najmaddin Akhundov *, James Ostrowski

Department of Industrial and Systems Engineering, University of Tennessee, Knoxville, 37966, TN, USA

ARTICLE INFO

Keywords:
Integer programming
Tool switching
Sequencing
Symmetry-breaking
Combinatorial optimization

ABSTRACT

The Job Sequencing and Tool Switching Problem (SSP), a well-known combinatorial optimization problem in the domain of Flexible Manufacturing Systems (FMS), is studied in this article. The aim of this research is to improve the currently known exact solution methodology for this $\mathcal{NP}$-hard problem. We propose a new integer linear programming approach with symmetry-breaking and tightening cuts that provably outperformed the existing methodology described in the literature. The computational study conducted on a data set with 1,050 instances showed that the developed method can solve hard instances that were not solvable with the state-of-the-art methods presented in the literature.

1. Introduction

Flexible Manufacturing System (FMS) is an automated production method that was developed to increase efficiency and flexibility in production planning. FMS has been applied in a wide variety of industries, from metal working industries to microelectronics (McGeoch & Sleator, 1991; Tang & Denardo, 1988a).

Tool management plays a critical role for achieving high productivity in FMS (Konak et al., 2008). Early researches refer to this class of problem as a Loading problem (Eilon & Christofides, 1971; Rupe & Kuo, 1997; Stecke & Solberg, 1981), many recent studies refer to it as a Job Sequencing and Tool Switching problem (SSP) (Calmels, 2019, 2022; da Silva et al., 2021). In a general SSP, different jobs are processed on a single machine. Each job requires a different set of tools and the machine's tool magazine needs to be adjusted accordingly. It is not possible to process all jobs without switching the tools, because of the limited magazine capacity. Therefore, the jobs need to be ordered in a way that minimizes the total number of tool switch. The sequencing problem (SP) was proven to be $\mathcal{NP}$-hard (Crama et al., 1994), therefore SSP also belongs to $\mathcal{NP}$-hard problem class, as it is more a general form of SP.

In the semiconductor industry, flexible component placement machines are used for assembling components onto a printed circuit board (PCB). These machines are highly automatized and equipped with a modular feeder unit to improve versatility (Hirvikorpi, Salonen, et al., 2006). Efficient utilization of placement machines can reduce the process cost significantly. It was reported that retrieval, transportation, loading, and calibration of tools from storage are responsible for 25%–30% of the total fixed and variable costs in an FMS (Ayres, 1987; Beezão et al., 2017; Cumings, 1986; Tomek, 1986).

Symmetry in SSP was first studied by Ghiani et al. (2007). They observed that a sequence of jobs and its reverse order sequence requires the same number of tool switches. They used this property to improve the branch-and-bound procedure developed by Laporte et al. (2004). The same symmetry property was used by Ghiani et al. (2010) to improve the branch-and-cut procedure to generate an improved lower bound. da Silva et al. (2021) introduced a symmetry-breaking constraints that enforces that the job that requires the most number of tools is processed in the first half of the sequence. This constraint helped to partially eliminate the symmetrical property introduced by Ghiani et al. (2007), and reduced the search space.

In this study, we improve symmetry exploitation by introducing new symmetry-breaking cuts. The SSP is reformulated as a job grouping and sequencing as a multi-commodity flow. One source of symmetry is when multiple jobs are performed between tool switches. Any permutation of these jobs will result in a feasible solution with the same number of switches. Further, there may be jobs that can be performed both before and after a given tool switch. In this case, the task of performing the job and performing the tool switch can be permuted without changing the feasibility or cost of the solution. Symmetry-breaking cuts are added to the model to eliminate these symmetries to reduce the search space and improve the computation time. A computational study is conducted by using the available data in the literature which has 1050 test instances to test the performance of the proposed approach.

The remainder of the paper is organized as follows. The problem is described in Section 2 followed by the review of the existing literature in Section 3. The proposed approach is presented in Section 4. The performance of the new method is tested in the computational study presented in Section 5. Finally, a short conclusion is given in Section 6.

* Corresponding author.
E-mail address: nakhundo@central.uh.edu (N. Akhundov).

https://doi.org/10.1016/j.ejor.2024.02.030
Received 2 December 2022; Accepted 28 February 2024
Available online 1 March 2024

Table 1
Example 1, SSP instance with solution.

Jobs	1	2	3	4	5	6	7	8	9	10	11	12
Tools	1	1	1	1	1	1	2	2	2	2	2	3
	2	2	2	2	2	3	3	3	3	3	4	4
	3	3	3	3	4	4	4	4	4	4	5	5
	4	4	4	5	5	5	5	5	5	6	6	6
	5	5	6	6	6	6	6	6	7	7	7	7
	6	7	7	7	7	7	7	**8**	8	8	8	8
Switch								**1 → 8**				

2. Problem description

Let $\mathcal{N} = \{1, \dots, N\}$ represent the set of jobs that need to be processed in a single machine. Each job (i) requires a set of tools ($\mathcal{T}_i$) from the tool set $\mathcal{T} = \{1, \dots, M\}$ to be loaded into the magazine before processing the job. The machine has tool capacity C, and for feasibility, it is assumed that $|\mathcal{T}_i| \leq C$. The objective is to find the sequence of jobs that minimizes the number of tool switches.

The assumptions are the following: (I) the tool sockets are indifferent and each tool requires a single slot, (II) tool changing times are constant and are identical for all tools, (III) the jobs and the required tools for each job are known in advance, (IV) the required tools for any job do not exceed the magazine capacity, (V) the tools do not break or wear out, (VI) at the start of the planning horizon, the magazine capacity is empty and can be filled with no cost.

An example of an SSP instance is shown in Table 1. The first row shows the jobs and columns represent the required tools for each job ($\mathcal{T}_i$). In this example, there are 8 tools in total and the magazine capacity is 7. The unloaded tools in the optimal solution are underlined and the loaded tools are given in bold. First, the magazine needs to be loaded with the first seven tools and the first seven jobs can be processed. Then after job 7, tool 1 is switched with tool 8. The remaining five jobs can be processed without any tool switches. The last row in Table 1 shows the required tools switch. In total, one switch is required to process all jobs.

In fact, the solution presented in Table 1 is only one of the alternative solutions among many. It can be clearly seen that any permutation of the first six or last five jobs within itself would lead to the same results. Switching these orders does not change the total number of switches. Similarly, one can process the latter jobs before the first seven jobs. Thus, this simple rearranging can be used to create $6! \cdot 5! \cdot 2$-many distinct solutions. Moreover, observe that job 7 can also be performed in the same path as jobs 9–12. Permuting and rearranging would yield an additional $6! \cdot 6! \cdot 2$-many distinct optimal solutions.

3. Literature review

3.1. The evolution of job sequencing and tool switching problem

The SSP was formally introduced by Tang and Denardo (1988a). They demonstrated an exact method to solve SSP and tightened the constraints for better performance. Upon the slow performance of the exact method, they developed a Keep Tool Needed Soonest (KTNS) algorithm to find minimum tool switches. This algorithm works under the condition that the order of jobs is known. By combining this method with a Greedy Perturbation (GP) heuristic they could solve small instances very fast. However, it was inefficient for large instances.

Following this study, different variants of SSP were studied in the literature. In tool size variation non-uniform tool sizes were considered with the relaxed capacity constraints (Crama et al., 2007; Hirvikorpi, Salonen, et al., 2006; Matzliach & Tzur, 2000; Rupe & Kuo, 1997). Multiple machine variations specially focused on parallel machines with uniform or non-uniform magazine capacities (Fathi & Barnette, 2002; Gökgür et al., 2018; Özpeynirci et al., 2016; Van Hop & Nagarur, 2004). Multi-objective variation considers the objective of minimizing tool switching instants and minimizing the number of tool switches simultaneously (Baykasoğlu & Ozsoydan, 2017, 2018; Mauergauz, 2017; Solimanpur & Rastgordani, 2012). In tool wear variation tool lifetimes are assumed to be either deterministic (Dadashi et al., 2016; Hirvikorpi, Nevalainen, & Knuutila, 2006) or stochastic (Farughi et al., 2017; Hirvikorpi et al., 2007). In this study, we will focus on uniform SSP, which is the most popular variation of SSP (Calmels, 2019). In this variation tool sizes are uniform and sequence-independent set-up times are used (Chaves et al., 2016; Paiva & Carvalho, 2017; Schwerdfeger & Boysen, 2017; Tang & Denardo, 1988a).

In terms of solution methods used in the uniform SSP literature, heuristic algorithms were used in the majority of the studies (Ahmadi et al., 2018; Al-Fawzan & Al-Sultan, 2003; Amaya et al., 2020; Chaves et al., 2016; Mecler et al., 2021; Paiva & Carvalho, 2017; Salonen et al., 2006; Schwerdfeger & Boysen, 2017). However, these methods do not guarantee the optimality of the solution, and in general, they are focused on finding a better sub-optimal solution in a reasonably short time frame. Therefore, in this study, we focus on finding an efficient exact method and compared it with the existing literature.

In the uniform SPP literature, there are very few studies that proposed an exact method to solve the problem. Following the mentioned work of Tang and Denardo (1988a), Laporte et al. (2004) proposed a reformulation of SSP that provides a better lower bound compared to Tang 1988. Their model is effective with instances up to 25 jobs.

Some of the studies used branch-and-bound methods to improve the solution process. Karakayalı and Azizoğlu (2006) proposed a branch-and-bound algorithm that is refined by precedence relations and combined with lower and upper-bound improvement techniques. Ghiani et al. (2007) explored symmetrical property of SSP to improve branch-and-bound algorithm developed by Laporte et al. (2004). The same authors proposed a reformulation of SSP as a nonlinear least cost Hamiltonian cycle problem and generated a branch-and-cut algorithm (Ghiani et al., 2010). This method was able to solve several instances with 45 jobs that could not be solved by the previous methods presented in the literature.

More recent studies focused on developing a tighter formulation of SSP. Catanzaro et al. (2015) proposed three MIP models that were tighter than the model proposed by Laporte et al. (2004). They showed that their approach improved the lower bound, compared to the previous studies. However, their models were unable to solve large instances, due to an increased number of variables and valid constraints. da Silva et al. (2021) proposed a multi-commodity flow model presented for SSP. Their approach helped to improve Linear Programming (LP) relaxation and therefore has a better lower bound compared to previous studies.

Please note that minimizing the number of tool switching instants problem (TSIP) is different than SSP. This class of problem also is referred to as the job grouping problem in the earlier literature (Ham, 1985; Kusiak, 1986; Tang & Denardo, 1988b). TSIP is a problem in $\mathcal{P}$, and a polynomial algorithm for TSIP was developed by Adjiashvili et al. (2015). The branch-and-bound based algorithm developed by Furrer and Mütze (2017) could solve TSIP with 2650 jobs in less than 1 s.

Contrary to TSIP, SSP is an $\mathcal{NP}$-hard problem (Crama et al., 1994; Karakayalı & Azizoğlu, 2006). The exact formulations used for solving SSP are not efficient in terms of solution time. There is a still need for improving existing exact methods and tighter formulations (Calmels, 2019).

3.2. Merits of applying symmetry breaking cuts

Symmetry breaking is a technique used in mathematical optimization to reduce the computational complexity of solving mixed-integer linear programming (MILP) problems. Symmetry arises in MILP when there are multiple solutions that are equivalent under certain permutations or transformations. This can lead to large search spaces, making

it difficult for standard optimization algorithms to find an optimal solution within a reasonable amount of time.

To reduce the size of the search space, symmetry breaking cuts, which can be referred as additional constraints, can be added to the MILP problem. Symmetry breaking cuts can be derived from the properties of the problem structure or can be generated algorithmically during the optimization process. Sherali and Smith (2001) showed that the use of symmetry breaking cuts has been shown to significantly improve the performance of MILP solvers, particularly for problems with a large degree of symmetry. An extensive computational experiment was conducted by Pfetsch and Rehn (2019) to test performance of different methods for handling symmetries by using a common implementation framework. They showed that by using symmetry breaking cuts the solution time could be improved by 15%.

Some of researchers have focused on the development of efficient algorithms for generating symmetry breaking cuts. Pemmaraju and Srinivasan (2008) presented an algorithm for generating symmetry breaking cuts based on vertex coloring. Verschae et al. (2023) used geometrical group theory to understand minimal and closed symmetry breaking polyhedral. Ostrowski et al. (2011) introduced orbital branching based on group of decision variables that are equivalent with respect to symmetry in branching. They showed that exploiting symmetries is as competitive as isomorphism pruning.

In addition to theoretical developments, symmetry breaking cuts have also been applied in a wide range of practical applications. For example, Lima and Novais (2016) and Ostrowski et al. (2015) used symmetry breaking cuts to solve unit commitment problem, a large-scale scheduling problem in the field of power systems. Jans (2009) studied symmetry in the lot-sizing problem on parallel machines and proposed symmetry breaking cuts by imposing hierarchical constraints. Anjos and Vieira (2017) showed that many versions of facility layout problem (FLP) have symmetric solution and they reviewed symmetry breaking strategies for FLP. Li et al. (2019) developed symmetry breaking cuts based for grid-based multi-agent path finding (MAPF) problem. Denton et al. (2010) used symmetrical property of surgeon-to-ORs assignment problem to generate cuts for deterministic and stochastic version of the problem. Similarly, Ghoniem and Farhadi (2015) used hierarchical symmetry breaking constraints to improve performance of branch-and-bound tree of aircraft sequencing problem.

4. New MILP reformulation for the SSP

This section provides an initial description of the new symmetry-breaking formulation. Following that, the *SSPMF* model, as introduced by da Silva et al. (2021), is presented as a reference. Subsequently, the adaptation of the new model to the *SSPMF* is discussed.

4.1. Job grouping and sequencing

In contrast to the previous SSP literature, the jobs that can be processed with the same magazine configuration are grouped together in the new reformulation. This is the first step in reducing the number of non-dominating solutions. Assume that in the optimal solution the first a jobs, where $a \leq N$, are set to be processed without changing the tools in the magazine. In this case, all permutations of these a jobs are alternative optimal solutions. Grouping the jobs in bins helps to avoid such cases.

Let $\mathcal{B} = \{1, \ldots, K\}$ represent a set of job groups or bins where the jobs are clustered. First, the main decision variables and constraints of the proposed *JGS* model are presented. Then, *Symmetry Breaking Cuts* and *Performance Improvement Cuts*, which are added to the *JGS* model to reduce the search space and speed up the solution process, are shown.

Decision variables:

x_{ik} Binary, 1 if job i is placed at bin k; 0 otherwise.

y_{tk} Binary, 1 if tool t is in bin k; 0 otherwise.

v_{tk} Binary, 1 if tool t is loaded to bin k; 0 otherwise.

w_{tk} Binary, 1 if tool t is removed from bin k; 0 otherwise.

JGS model

$$\min \quad \sum_{t=1}^{T} \sum_{k=2}^{K} v_{tk} \tag{1a}$$

$$\text{s.t.} \quad \sum_{k \in \mathcal{B}} x_{ik} = 1 \qquad \forall i \in \mathcal{N} \tag{1b}$$

$$\sum_{t \in \mathcal{T}} y_{tk} = C \qquad \forall k \in \mathcal{B} \tag{1c}$$

$$x_{ik} \leq y_{tk} \qquad \forall t \in \mathcal{T}_i, \forall k \in \mathcal{B}, \forall i \in \mathcal{N} \tag{1d}$$

$$y_{tk} = y_{t(k-1)} + v_{tk} - w_{tk} \qquad \forall t \in \mathcal{T}, \forall k \in \mathcal{B}. \tag{1e}$$

Here, the objective function (1a) minimizes the number of tool switches starting from the second bin. Constraint (1b) ensures every job is placed in one of the bins, Constraint (1c) ensures the number of tools in the magazine is equal to the capacity, Constraint (1d) ensures that if a job is placed in a bin then all necessary tools required for that job are available, Constraint (1e) is used for ensuring the relationship between loaded, unloaded and existing tools.

Symmetry breaking cuts

If a job could be done earlier then it should not be delayed. The job needs to be processed as soon as all necessary tools are available. Consider a case that job i could be placed in bin k or $k+1$, as both bins have the necessary tools to process job i. In this case, we are forcing job i to be placed in bin k. Constraint (1f) ensures this condition:

$$x_{ik} \leq \sum_{t \in \mathcal{T}_i} v_{tk} \qquad \forall k \in \mathcal{B}, \forall i \in \mathcal{N}. \tag{1f}$$

If no tool is added to a bin then that bin is redundant. Constraint (1g) ensures that if there is no tool added to a bin then the following bins should not receive any tools. It forces empty bins to be gathered at the end of the sequence indirectly:

$$C \sum_{t \in \mathcal{T}} v_{tk} \geq \sum_{t \in \mathcal{T}} v_{t(k+1)} \qquad \forall k \in \mathcal{B}. \tag{1g}$$

Similarly, If no tool is removed from a bin, then it means that the following bin is redundant. Constraint (1h) ensures that, if no tool is removed from the bin at position k then do not remove any tool from later bins:

$$C \sum_{t \in \mathcal{T}} w_{tk} \geq \sum_{t \in \mathcal{T}} w_{t(k+1)} \qquad \forall k \in \mathcal{B}. \tag{1h}$$

Consider a case that the tool t could be removed from the magazine after bin k or bin $k+1$, and both lead to the same objective function value. If a tool could be removed from the magazine then it should be removed as soon as possible to avoid having an alternative solution. Constraint (1i) ensures that, if tool t is removed from bin $k+1$ then there should be at least one job that uses tool t in bin k. In other words, tool t should be removed from the magazine if there is no job in a bin that requires this tool.

$$\sum_{(i \mid t \in \mathcal{T}_i)} x_{ik} \geq w_{t(k+1)} \qquad \forall t \in \mathcal{T}, \forall k \in \mathcal{B}. \tag{1i}$$

If job i is not done until bin k and at bin k all necessary tools for job i are available, then job i should be placed at bin k. Otherwise, it would lead to multiple optimal solutions. Constraint (1j) ensures this condition:

$$x_{ik} \geq 1 - \sum_{b=1}^{k-1} x_{ib} + \sum_{t \in \mathcal{T}_i} y_{tk} - |\mathcal{T}_i| \qquad \forall k \in \mathcal{B}, \forall i \in \mathcal{N}. \tag{1j}$$

Constraint (1j) could also be written alternatively, as:

$$\sum_{b=1}^{k} x_{ib} \geq \sum_{t \in \mathcal{T}_i} y_{tk} - |\mathcal{T}_i| + 1 \qquad \forall k \in \mathcal{B}, \forall i \in \mathcal{N}. \tag{1k}$$

Performance improvement cuts

Assume that a binary decision variable z_k is equal to 1 if, there is at least one job placed at bin k, 0 otherwise. Two more constraints are required for linking this new decision variable to the rest of the model:

$$z_k \geq x_{ik} \qquad \forall k \in \mathcal{B}, \forall i \in \mathcal{N} \tag{1l}$$

$$z_k \leq \sum_{t \in \mathcal{T}} v_{tk} \qquad \forall k \in \mathcal{B}. \tag{1m}$$

Constraint (1l) makes sure that if there is at least one job in bin k, then bin k should be available, Constraint (1m) ensures that if no tools are added to bin k, then bin k is not available. After taking care of the new decision variable, the following constraints could be considered for tightening the proposed formulation:

$$z_k \geq z_{k+1} \qquad \forall k \in \mathcal{B}. \tag{1n}$$

Constraint (1n) ensures that the bins are utilized in an alphanumeric order. This constraint prevents empty bins from scattering among other bins, thereby enabling multiple non-dominating solutions. As a result of this constraint, empty bins will tend to accumulate at the latter part of the order.

Please note that Constraint (1n) acts as symmetry breaking cut as well. We have already considered similar logic in Constraint (1g). Therefore it is listed as a performance improvement cut, because it would help to put an upper bound for z values during the first step of the solution process, where LP relaxation is solved by Gurobi.

The next two cuts are generated by using graph theory. Assume that, $G = (V, E)$ has $V = \mathcal{N}$ and edges between vertices if two jobs could not be placed in the same bin. The limited magazine capacity does not allow job i and j to be placed into the same bin if the $|\mathcal{T}_i \cup T_j| > C$ condition is satisfied. Assume that $\mathcal{L} = \{1, \ldots, L\}$ represents a set of cliques, and O_l represents set of jobs in the clique l, where $l \in \mathcal{L}$. Let $\mathcal{Q} = \{1, \ldots, Q\}$, where Q represents the maximal clique size, $Q = max(\{O_l \mid l \in \mathcal{L}\})$. By using these notations the following cuts could be defined:

$$z_k \geq \sum_{i \in O_l} x_{ik} \qquad \forall l \in \mathcal{L}, \forall k \in \mathcal{B} \tag{1o}$$

$$z_k = 1 \qquad \forall k \in \mathcal{Q}. \tag{1p}$$

Constraint (1o) ensures that the jobs that form a clique in graph G are not placed in the same bin. Constraint (1p) ensures that the number of available bins is at least equal to the maximal clique size, and the available bins are positioned first among all bins. All decision variables in this model are set to binary:

$$x_{ik}, y_{tk}, v_{tk}, w_{tk}, z_k \in \{0, 1\} \qquad \forall t \in \mathcal{T}, \forall k \in \mathcal{B}, \forall i \in \mathcal{N}. \tag{1q}$$

To avoid loss of generality assume that $v_{t(K+1)} = w_{t(K+1)} = y_{t0} = 0$ for all tool $t \in \mathcal{T}$ and $z_{(K+1)} = 0$.

4.2. SSPMF model

The Job Sequencing and Tool Switching Problem modeled as a Multicommodity Flow Problem (*SSPMF*) model developed by da Silva et al. (2021) is described in this section. To illustrate the flow of tools, a capacitated graph was employed, as depicted in Fig. 1. In this graph, the capacity on arc $(i, i+1)$, where i ranges from 0 to $N-1$, was set at C, while the other arcs were granted unlimited capacity. Node 0 and node $N+1$ served as the respective origin and destination points, and node $N+2$ was an auxiliary node. The remaining nodes within the graph signified the order of job processing.

To gain a clearer understanding of the arcs, let us examine node 2 as an illustrative example. At this juncture, arc $(1, 2)$ reveals the composition of tools in the magazine prior to the processing of job 2. Subsequently, arcs $(2, N+2)$ and $(2, N+1)$ represent the tools extracted from the magazine following the completion of job 2. The choice of whether a tool is routed to node $N+2$ or node $N+1$ hinges on its intended usage in subsequent jobs. In essence, if a tool is directed to node $N+2$, it indicates that it will be employed in forthcoming tasks, while tools dispatched to node $N+1$ are not slated for later use. Lastly, arc $(N+2, 2)$ delineates the tools replenished into the magazine in preparation for the execution of job 3.

Table 2
Example 2, SSP instance with solution.

Jobs	1	2	3	4	5
Tools	1	1	2	3	3
	2	2	3	**4**	4
	3	**5**	5	**6**	5
	4				
Switch		**4 → 5**		**1 → 4**	
				2 → 6	

The costs for incoming flow to nodes $N+1$ and $N+2$ were set to one, while all other flow costs within the network were set to zero. This design choice ensured that the overall minimization of flow in this graph aligns with the overarching objective of minimizing the number of tool switches in the context of the SSP, making it a coherent and effective strategy for addressing the problem.

To gain a deeper understanding of the *SSPMF* approach, consider the example outlined in Table 2. Here, the task is to process 5 jobs with a magazine capacity limited to 4 tools. The optimal execution of these jobs necessitates the utilization of 3 switches. The solution is depicted in Fig. 2, visualizing the flow of tools on a graph.

For this specific problem, the jobs are assigned to the nodes based on their numbering. Initially, the tools required for job 1 are placed in the magazine (sent to node 1), while the remaining tools are sent to node 7. The tools dispatched to node 7 are employed in subsequent jobs. Those no longer required for later jobs are forwarded to node 6. Here, the incoming flow to nodes 7 and 6 (excluding flow from node 5) indicates the total switch count.

Two sets of binary decision variables were used in this formulation. X_{ik} was equal to 1 if job i is placed in order k. $Y_{ii't}$ was equal to 1 If a tool t is sent from node i to i'. By representing the set $1, \ldots, N-2$ as $\mathcal{N}'$, the *SSPMF* can be expressed as follows:

$$\min \quad \sum_{t=1}^{M} \sum_{i=1}^{N-1} Y_{i(N+1)t} + \sum_{t=1}^{M} \sum_{i=1}^{N-2} Y_{i(N+2)t} \tag{2a}$$

$$\text{s.t.} \quad \sum_{k=1}^{N} X_{ik} = 1 \qquad \forall i \in \mathcal{N} \tag{2b}$$

$$\sum_{i=1}^{N} X_{ik} = 1 \qquad \forall k \in \mathcal{N} \tag{2c}$$

$$Y_{01t} + Y_{0(N+2)t} = 1 \qquad \forall t \in \mathcal{T} \tag{2d}$$

$$Y_{(i-1)it} + Y_{(N+2)it} - Y_{i(N+1)t} - Y_{i(i+1)t} - Y_{i(N+2)t} = 0 \qquad \forall t \in \mathcal{T}, i \in \mathcal{N}' \tag{2e}$$

$$Y_{(N-2)(N-1)t} + Y_{(N+2)(N-1)t} - Y_{(N-1)Nt} + Y_{(N-1)(N+1)t} = 0 \qquad \forall t \in \mathcal{T} \tag{2f}$$

$$Y_{(N-1)Nt} - Y_{N(N+1)t} = 0 \qquad \forall t \in \mathcal{T} \tag{2g}$$

$$\sum_{i=1} Y_{i(N+1)t} = 1 \qquad \forall t \in \mathcal{T} \tag{2h}$$

$$\sum_{k=1}^{N-2} Y_{i(N+2)t} - \sum_{i=1}^{N-1} Y_{(N+2)it} = 0 \qquad \forall t \in \mathcal{T} \tag{2i}$$

$$X_{ik} \leq Y_{(k-1)kt} \qquad \forall t \in \mathcal{T}_i, \ \forall i, k \in \mathcal{N} \tag{2j}$$

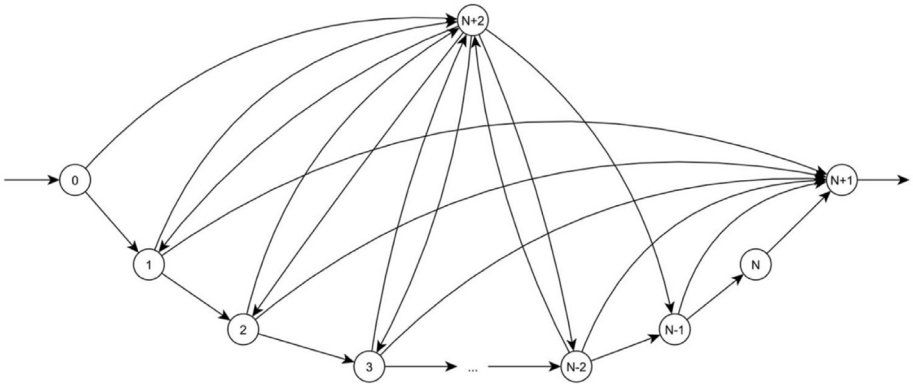

Fig. 1. Capacitated graph presented by da Silva et al. (2021).

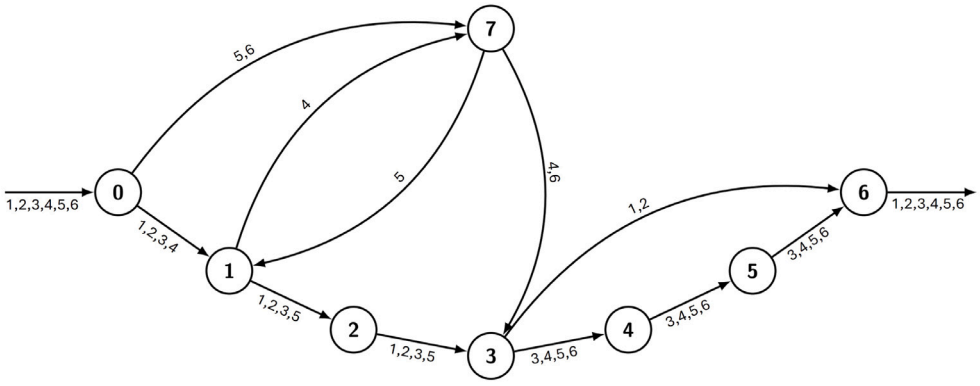

Fig. 2. Graphical Representation of the Optimal Result for Example 2.

$$\sum_{t=1}^{M} Y_{(k-1)kt} = C \qquad \forall k \in \mathcal{N} \qquad (2k)$$

$$\sum_{k=1}^{\lceil N/2 \rceil} X_{pk} = 1 \qquad (2l)$$

$$Y_{k(N+1)t} = 0 \qquad \forall t \in \mathcal{T}, \forall k \in J' \qquad (2m)$$

$$X_{ik}, Y_{ii't} \in \{0,1\} \qquad i' = 0, \ldots, N, \qquad \forall t \in \mathcal{T}, \forall i, k \in \mathcal{N}. \qquad (2n)$$

The objective function (2a) minimized the cost of the flow on arcs $(i, N+2)$, $i = 0, 1, \ldots, N-2$, and $(i, N+1)$, $i = 1, 2, \ldots, N-1$. Constraints (2b) and (2c) ensured all N jobs to be sequenced in same order. Constraints (2d)–(2i) were flow conservation constraints. Constraints in (2j) required that, if job i was designated as kth job to be processed, then the flow on arc $(k-1, k)$ must include a unit flow of the commodities corresponding to tools required to process job i. Constraints in (2k) ensured that the flow in arc $(i, i+1)$, where $i = 0, 1, \ldots, N$ is equal to magazine capacity (C). Constraint (2l) imposed that the job p to be processed in the first $\lceil N/2 \rceil$ position, here $p = arg\ max_{j \in J} |\mathcal{T}_i|$. Constraint (2m) ensured that if a tool t cannot leave the magazine if the number of processed jobs is less than the number of jobs need tool t. Here J' is $\{1, 2, \ldots, |J_t - 1|\}$, $|J_t|$ represented the number of jobs that needs tool t. All decision variables were binary (2n).

In this formulation Constraint (2l) was a symmetry-breaking cut introduced by da Silva et al. (2021). Constraint (2m) can be considered as a performance improvement cut as it helps to set some of the decision variables' values to zero and tighten the formulation.

This approach was evaluated by da Silva et al. (2021) in relation to three earlier works (Catanzaro et al., 2015; Laporte et al., 2004; Tang & Denardo, 1988a) within the domain of finding an exact solution for the SPP. The computational analysis unveiled the superior performance of their method in comparison to other techniques. This superiority could be attributed to their pioneering graph-based approach, which adeptly established a more robust lower bound on the linear relaxation of the problem.

4.3. Job grouping and sequencing as a multi-commodity flow

The incorporation of symmetry-breaking and tightening cuts within the *JGS* model yields enhancements to the lower bound through a reduction in the search space. Further enhancement of this model can be achieved by adopting the multi-commodity flow approach developed by da Silva et al. (2021). This implementation enhances the robustness of the lower bound within the linear programming (LP) relaxation, which Gurobi leverages to determine an initial solution. In this section, we synergize these two approaches to leverage the advantages of the novel symmetry-breaking cuts from the *JGS* model and the graph utilized in the *SSPMF* model.

The capacitated graph approach introduced by da Silva et al. (2021) serves as the foundation for constructing the multi-commodity flow graph depicted in Fig. 3. In this illustration, the blue nodes $\{H, R, D\}$ correspond to starting tools, the tool repository, and detached tools, respectively. The red nodes are designated for bins, while the green nodes represent jobs. The solid arrows symbolize the flow of tools, and the dashed arrows illustrate the potential allocation of jobs $(\mathcal{N})$ to bins $(\mathcal{B})$. It is worth noting that tools dispatched to the repository (R) can be reused, whereas detached tools (D) are not eligible for reuse.

To provide a clearer insight into the arcs, let us take red node 2 as an illustrative example. The flow between red node 1 and red node 2 represents the inventory of tools in the magazine before initiating the processing of jobs in bin 2. The two-sided arc connecting red node 2 to node R illustrates the flow of tools after completing the jobs in bin 2. Meanwhile, the one-sided arc linking red node 2 to node D signifies the removal of tools from the magazine that will no longer be utilized in subsequent jobs. The arc between red node 2 and red node 3 delineates

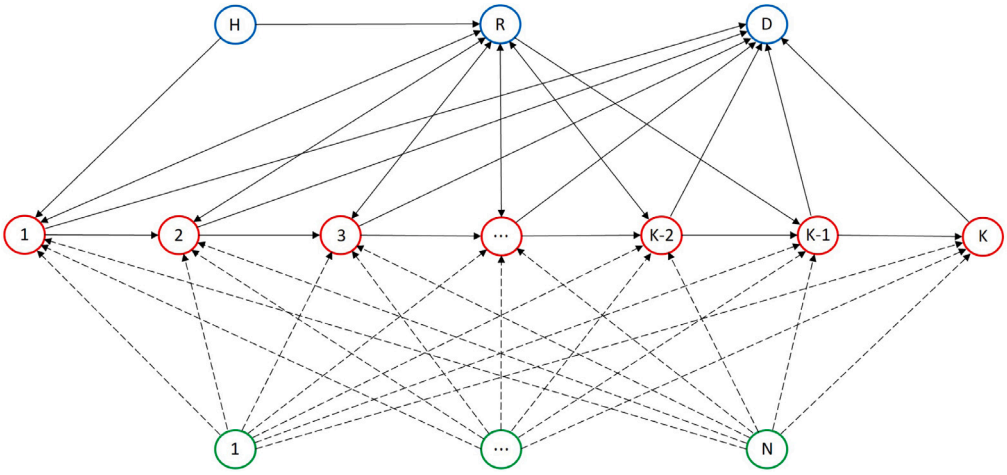

Fig. 3. Multi-Commodity Flow Graph.

the set of tools available in the magazine before the commencement of jobs in bin 3. Lastly, the dashed arcs between green nodes and red node 2 depict the allocation of jobs to the respective bins.

The capacity of the arcs connecting the red nodes equals the magazine capacity (C), while the capacity of the dashed arcs is set to one. All remaining arcs have unbounded capacity. The cost associated with the flow from red nodes to nodes R and D is set to one, whereas all other flow costs are set to zero.

Assume that a binary decision variable f_{abt} is equal to 1, if tool t is transferred from node a to b, where $a \in \{H \cup R \cup \mathcal{B}\}$ and $b \in \{R \cup D \cup \mathcal{B}\}$; 0 otherwise. The new Job Grouping and Sequencing as a Multi-commodity Flow model could be written as follows:

JGSMF model

$$\min \quad \sum_{t\in\mathcal{T}}\sum_{k=1}^{K-1} f_{kDt} + \sum_{t\in\mathcal{T}}\sum_{k=1}^{K-2} f_{kRt} \tag{3a}$$

$$\text{s.t.} \quad \sum_{k\in\mathcal{B}} x_{ik} = 1 \qquad \forall i \in \mathcal{N} \tag{3b}$$

$$\sum_{t\in\mathcal{T}} f_{(k-1)kt} = C \qquad \forall k \in \mathcal{B} \tag{3c}$$

$$x_{ik} \le f_{(k-1)kt} \qquad \forall t \in \mathcal{T}_i, \forall k \in \mathcal{B}, \forall i \in \mathcal{N} \tag{3d}$$

$$f_{S1t} + f_{SRt} = 1 \qquad \forall t \in \mathcal{T} \tag{3e}$$

$$\sum_{k\in\mathcal{B}} f_{kDt} = 1 \qquad \forall t \in \mathcal{T} \tag{3f}$$

$$f_{(k-1)kt} + f_{Rkt} = f_{k(k+1)t} + f_{kDt} + f_{kDt} \qquad \forall t \in \mathcal{T}, \forall k \in \mathcal{B} \tag{3g}$$

$$f_{(K-2)(K-1)t} + f_{R(K-1)t} = f_{(K-1)Kt} + f_{(K-1)Dt} \qquad \forall t \in \mathcal{T} \tag{3h}$$

$$f_{(K-1)Kt} = f_{KDt} \qquad \forall t \in \mathcal{T} \tag{3i}$$

$$f_{SRt} + \sum_{k=1}^{K-2} f_{kRt} = \sum_{k=1}^{K-1} f_{Rkt} \qquad \forall t \in \mathcal{T} \tag{3j}$$

$$x_{ik} \le \sum_{t\in\mathcal{T}_i} f_{R(k-1)t} \qquad \forall k \in \mathcal{B}, \forall i \in \mathcal{N} \tag{3k}$$

$$C \sum_{t\in\mathcal{T}} f_{R(k-1)t} \ge \sum_{t\in\mathcal{T}} f_{Rkt} \qquad \forall k \in \mathcal{B} \tag{3l}$$

$$C \sum_{t\in\mathcal{T}} f_{(k-1)Dt} \ge \sum_{t\in\mathcal{T}} f_{kDt} \qquad \forall k \in \mathcal{B} \tag{3m}$$

$$\sum_{(i|\, t\in\mathcal{T}_i)} x_{ik} \ge f_{kDt} \qquad \forall t \in \mathcal{T}, \forall k \in \mathcal{B} \tag{3n}$$

$$\sum_{b=1}^{k} x_{ib} \ge \sum_{t\in\mathcal{T}_i} f_{(k-1)kt} - |\mathcal{T}_i| + 1 \qquad \forall k \in \mathcal{B}, \forall i \in \mathcal{N} \tag{3o}$$

$$z_k \ge x_{ik} \qquad \forall k \in \mathcal{B}, \forall i \in \mathcal{N} \tag{3p}$$

$$z_{k+1} \le \sum_{t\in\mathcal{T}} f_{kDt} \qquad \forall k \in \mathcal{B} \tag{3q}$$

$$z_k \ge z_{k+1} \qquad \forall k \in \mathcal{B} \tag{3r}$$

$$z_k \ge \sum_{i\in O_l} x_{ik} \qquad \forall l \in \mathcal{L}, \forall k \in \mathcal{B} \tag{3s}$$

$$z_k = 1 \qquad \forall k \in \mathcal{Q} \tag{3t}$$

$$x_{ik}, z_k, f_{abt} \in \{0,1\} \quad \forall (a,b) \in \{H \cup R \cup D \cup \mathcal{B}\}, \quad \forall t \in \mathcal{T}, \forall k \in \mathcal{B}, \forall i \in \mathcal{N}. \tag{3u}$$

Here, the objective function (3a) minimizes the flow on arcs (k, D) and (k, R), $k \in \mathcal{B}$, which corresponds to tool switches. Constraint (3b) ensures that every job is placed into one of the bins, Constraint (3c) enforces the capacity limit, Constraint (3d) makes sure that if a job placed into a bin then all necessary tools are available. Constraint (3e) enforces that all the tools are available at the start, similarly, Constraint (3f) enforces that all tools are removed at the end. Constraint (3g), (3h), (3i), (3j) are used for ensuring the flow balance in intermediate nodes. Similar to the JGS model, Constraints (3k), (3l), (3m), (3n), (3o) are used as symmetry-breaking cuts, and Constraints (3p), (3q), (3r), (3s) are used for tightening the formulation. Finally, Constraint (3u) enforces that all decision variables are binary.

The Constraint (2l) of *SSPMF* cannot be used in this formulation. Because the reverse sequence of the bins is not feasible as a result of using symmetry-breaking cuts in *JGSMF*. The jobs are grouped in *JGSMF* therefore, Constraint (2m) cannot be used.

Please note that in *JGSMF* k represents a group of jobs, which is referred as a bin. Here a bin contains tools that do not exceed the capacity, the number of bins can be any number between 1 to N. On the contrary, k represents the order of a single job in *SSPMF* and there is no need for a procedure to find the maximum value k can get, it is set to be equal to the number of jobs.

The resulting model has the ability to eliminate symmetries through the utilization of *Symmetry-Breaking Cuts* and *Performance Improvement Cuts* as detailed in Section 4.1. Simultaneously, the graph-based approach enhances solution efficiency by offering an improved lower bound for the linear programming (LP) relaxation of the problem.

4.4. Solution procedure

The *JGSMF* model has $K(3M + N + K + 5) - 2$ decision variables and $K(MN + 3N + 2M + L + 5) + 5M + N + Q$ constraints. In this formulation, all parameters are fixed, except the number of bins (K), which affects problem size significantly. Having N bins ($K = N$) would be sufficient to solve the problem, but it will be computationally difficult. On the other hand, having very few bins may lead to an infeasible or non-optimal solution because of the capacity constraint.

Therefore, a procedure is needed to find preferable low but sufficient bin count to be used in *JGSMF* model.

This problem is solved by dividing the solution procedure into three phases (P_1, P_2, P_3). Assume that the optimal count of switches is denoted as S. In the first phase (P_1), K_1 bins are used in *JGSMF* model and solve it to optimality to find an initial solution. In the beginning K_1 is set to K_0, where $K_0 = max(5, \min(Q, N))$. Due to the limited number of bins, the problem has a small scale, allowing for a quick solution. However, a reduction in the number of bins may lead to a higher frequency of switches, potentially compromising the optimality of the solution. In this case, the found number of the switches in phase 1 (S_1) may surpass the optimal count (S), where we have sufficient bins. This can be leveraged to establish an upper bound (UB) for S. If it is infeasible, K_1 is incremented by 2 and solved repetitively until the feasible solution is found or the time limit (T_1) is reached.

In the second phase (P_2), the number of bins are set to K_2, where $K_2 = K_1 + 1$. If optimal switches in the second phase (S_2) are equal to S_1 then it is terminated, otherwise, UB is updated. The problem is resolved by incrementing K_2 by 1 and corresponding S_2 is found. The process resumes until the time limit (T_2) is reached or there is no reduction in optimal switch (S_2) in comparison to the previous run.

In the third phase (P_3), a slightly different model is solved to ensure the optimality of the solution. It is important to ascertain that a further increase in the number of bins will not lead to an improved solution. In order to achieve this, additional constraints are introduced into the *JGSMF* model, which serve to regulate the flow between each bin while also mandating that all bins remain open. Constraint (4a) and (4b) are used to fix the flow to 1, Constraint (4c) is used to set all bins to be open:

$$\sum_{t \in \mathcal{T}} f_{kRt} + f_{kDt} = 1 \qquad \forall k \in \{1, \dots, K-2\} \tag{4a}$$

$$\sum_{t \in \mathcal{T}} f_{(K-1)Dt} = 1 \tag{4b}$$

$$z_k = 1 \qquad \forall k \in \mathcal{B}. \tag{4c}$$

If it is known that s switch is a feasible answer to an SSP problem, then $s+1$ bins should be sufficient enough to solve *JGSMF* model. At the end of phase two, it is clear that $S_2 + 1$ bins would be sufficient enough. Therefore, the bins in P_3 are fixed to K_3, where $K_3 = S_2$, to test the feasibility of solving the problem with 1 less switch. If it is infeasible, then it means that K_3 switches are optimal, otherwise K_3 is decremented by 1, UB is updated, and the modified model is solved again. The third phase is terminated if an infeasible solution is found or the time limit (T_3) is reached. The solution procedure is described in Fig. 4. Here, $\{t_1, t_2, t_3\}$ represents solution time of an instance of procedure $\{P_1, P_2, P_3\}$ respectively.

Among the three phases, P_1 uses the lowest number of bins, therefore it is the fastest. The time limit of P_1 is set to the maximum given time (1 h). If a test instance is not solved within this time then it is reported as unsolved, the remaining time is divided equally between P_2 and P_3.

5. Computational analysis

5.1. Comparison overview

We demonstrate a performance comparison of the proposed reformulation supported by the solution process described in Fig. 4. From now on, we will be referring to the proposed approach as *JGSMF*. It will be compared with the fastest known method in the literature (*SSPMF*) developed by da Silva et al. (2021).

All computational experiments were performed on a server with AMD Ryzen Threadripper 2950X 16-Core processor and 64 GB RAM running Ubuntu 18.04.6 LTS. Gurobi 8.0.1 with Python interface is used for solving both models with 3600 s solution time limit. The remaining Gurobi parameters were set to default for solving all instances. If an instance is not solved within the given time the optimality gap is reported.

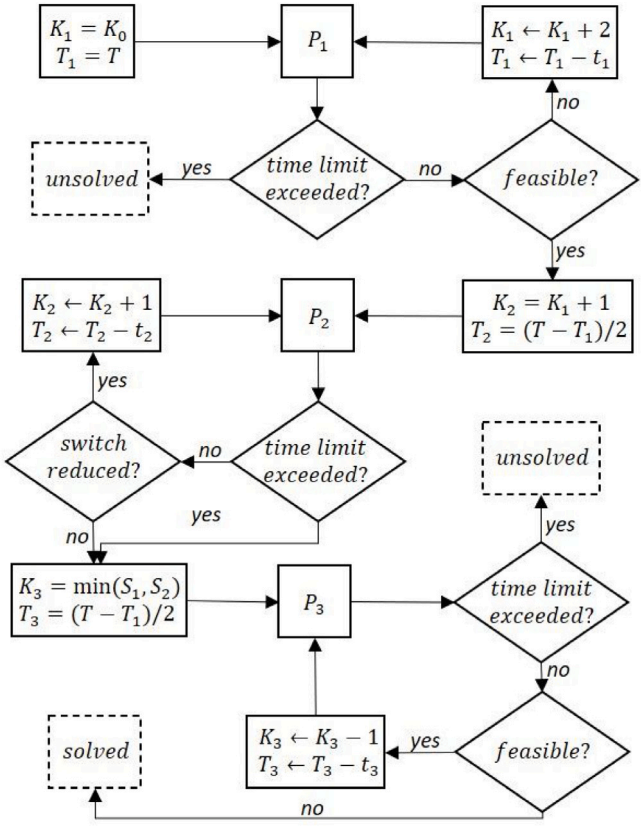

Fig. 4. The Flow of Solution Process.

5.2. Data set description

The dataset presented by Yanasse et al. (2009) was used to test the effectiveness of the proposed approach. In this database, there are five clusters, and in each cluster, there are 8–15 groups, and in each group, there are 10–130 instances. Among the five clusters, the one with the fewest number of jobs was eliminated as it could be easily solved (within one second) by the two compared methods and the previous methods presented in the literature. The resulting data set has 1050 instances in total. These test instances are available for access via the following link, located within the folder named "Laporte"(https://sites.google.com/site/antoniochaves/publications/data).

There is a condition that could improve the solution time of a test instance. If a set of required tools for job i is a subset of required tools of job j $(\mathcal{T}_i \subseteq T_j)$, then job j is referred as the *dominating* job. In such a case, all *dominated* jobs could be emitted from the job set $(\mathcal{N})$ prior to the modeling in order to reduce the complexity of the problem. Then, after finding the optimal job processing sequence they could be added to the sequence by placing it right after the *dominating* job. It is worth mentioning that, none of the tested instances has such a condition.

5.3. Performance comparison

The computational study was divided into two sections. In the first section, a comparison was made between the proposed *JGSMF* model and *SSPMF* in terms of the number of instances solved to optimality and the run times. If an instance could not be solved within the one-hour time limit, the optimality gaps were compared.

In the second section, a closer examination was undertaken on the *JGSMF* model, focusing on the analysis of its components and the gradual improvement in performance they exhibited.

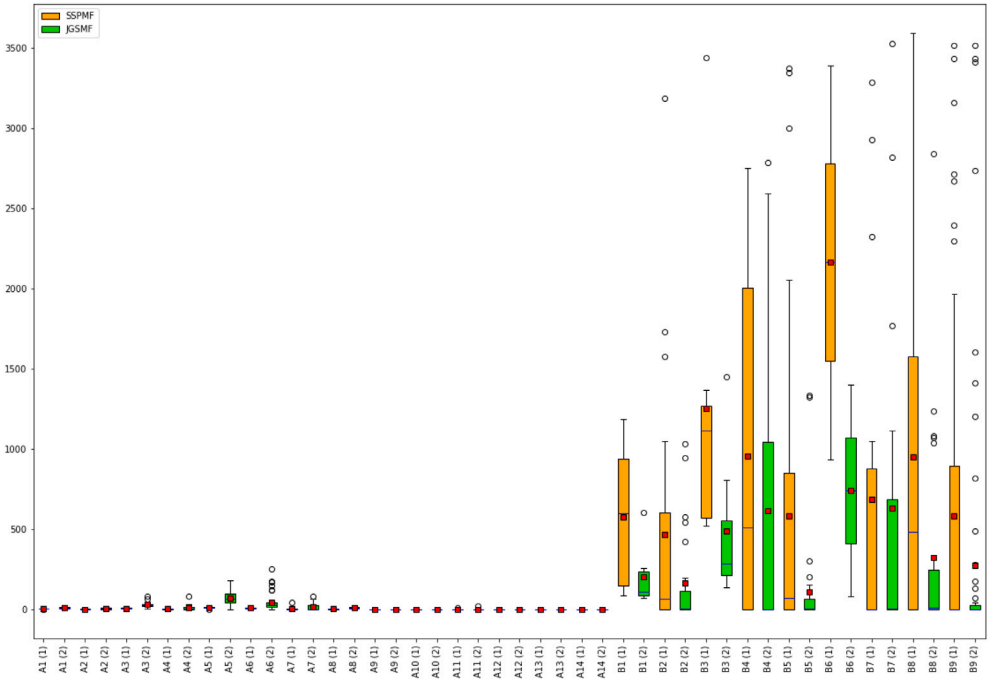

Fig. 5. Comparison of the solution times for the instances in cluster A and B, that are solved with both methods.

5.3.1. Comparative analysis

The comparison was initiated with an assessment of the number of instances that were solved to optimality (*Opt.*) out of the total I instances. If the number of solved instances were the same then the average run times of commonly solved instances (*Time*) were compared. The superior result was presented in bold font. In cases where certain instances could not be solved to the optimality within the specified time limit, the optimality gaps for the common instances that remained unsolved (referred to as *Gap*) were reported.

The size of the problem in each test cluster and an overview of the computational study is presented in Table 3. Here, the range of jobs (N), required tools (M), and magazine capacity (C) are given in respective order. The total number of instances in each group is represented by *TI*, and *TO* shows the total number of optimally solved instances.

If an instance is not solved within one hour, the optimality gap reported by Gurobi is used for *SSPMF*. For *JGSMF* an instance reported to be unsolved either in Phase 1 or 3. Note that, in Phase 2 half of the remaining time is used, and then it is passed to Phase 3. If an instance is found to be unsolved in Phase 1, then the upper bound (UB) reported by Gurobi is used, and the lower bound (LB) is set to $K_1 - 1$. Because if K_1 bins are used and none of them is redundant then there have to be at least $K_1 - 1$ switches. If the limit is reached at Phase 3 then K_3 is used as UB, and the LP relaxation of the *JGSMF* model with K_3 bins is reported as LB.

Both methods were able to solve all instances in data cluster A. On average, the *SSPMF* method was faster than the *JGSMF* method, see Table 4. However, this difference was usually a few seconds where almost all instances were solved within a minute. *JGSMF* solves the model in each phase (P_1, P_2, P_3), therefore, solution speed slowed down for the easy instances.

As the problem size gets slightly larger *JGSMF* was able to solve more test instances, see Table 5. Approximately 16% more instances could be solved with *JGSMF* and the average solution time was faster for these solved instances for data cluster B. Among 340 instances in this cluster, 227 instances were solved with both methods. On average, *JGSMF* was able to solve these instances approximately 1.02 times faster than *SSPMF*. In terms of the average optimality gap for common instances that were not solved to optimality, *JGSMF* was slightly (1.4%) better than *SSPMF*. In comparison to data cluster A, the run times difference was getting significantly bigger for data cluster B, see Fig. 5. The yellow and green colored box plots in Figs. 5, 6, 7 represents *SSPMF* and *JGSMF*, respectively.

Table 3
Overview of the problem size and number of solved instances.

					SSPMF	*JGSMF*
Cluster	N Range	M Range	C Range	*TI*	*TO*	*TO*
A	9–9	15–25	5–20	370	370	370
					difference:	**+0%**
B	15–15	15–25	5–20	340	227	263
					difference:	**+16%**
C	20–25	15–25	5–20	260	114	169
					difference:	**+33%**
D	10–15	10–20	4–12	80	71	75
					difference:	**+5%**

In data cluster C, 33% more instances were solvable by *JGSMF*, see Table 6. Out of 260 instances in this cluster, 112 instances were solved with both methods within an hour. On average, the run time of *JFSMF* was 48.4 times faster than *SSMPF* for these 112 instances. The optimality gap for common instances that were not solved to optimality with *JGSMF* was 39% less than *SSPMF*.

The performance differences for data cluster D were almost identical where only four (5%) more instances were solved by *JGSMF*. In this cluster, there were 80 instances in total and 71 instances were solved with both methods. The average solution time of commonly solved instances was improved by 2.07 times by using *JGSMF*. The run time distribution of commonly solved instances in Fig. 6 shows that the majority of run times were close to zero for data clusters C and D. In one test case (C7) the average run time of commonly solved instances by *JGSMF* was significantly higher than *SSPMF*, however, in the rest of the cases their performance was either very close or *JGSMF* performed better. Across all data groups, a superior optimality gap was achieved, attributed to the enhanced lower bound (LB). For a detailed report on the lower and upper bounds of instances that were

Table 4
Computational results for the data cluster A.

					SSPMF			*JGSMF*		
Group	*N*	*M*	*C*	*I*	*Opt.*	*Time*	*Gap (%)*	*Opt.*	*Time*	*Gap (%)*
A1	9	15	5	10	10	**4.7**	–	10	11.8	–
A2	9	15	10	30	30	**3.0**	–	30	6.9	–
A3	9	20	10	20	20	**8.7**	–	20	31.4	–
A4	9	20	15	50	50	**4.6**	–	50	11.0	–
A5	9	25	10	20	20	**12.5**	–	20	73.2	–
A6	9	25	15	40	40	**9.6**	–	40	45.8	–
A7	9	25	20	130	130	**5.0**	–	130	16.1	–
A8	9	20	5	10	10	**6.2**	–	10	14.4	–
A9	9	20	10	10	10	**0.1**	–	10	**0.1**	–
A10	9	20	15	10	10	0.2	–	10	**0.04**	–
A11	9	25	5	10	10	**1.9**	–	10	3.3	–
A12	9	25	10	10	10	**0.1**	–	10	**0.1**	–
A13	9	25	15	10	10	**0.03**	–	10	0.1	–
A14	9	25	20	10	10	**0.02**	–	10	0.1	–
[Total] /(Average):				[370]	[370]	**(5.2)**	(-)	[370]	(19.1)	(-)

Table 5
Computational results for the data cluster B.

					SSPMF			*JGSMF*		
Group	*N*	*M*	*C*	*I*	*Opt.*	*Time*	*Gap (%)*	*Opt.*	*Time*	*Gap (%)*
B1	15	15	5	10	7	579	0.12	**9**	316	0.12
B2	15	15	10	30	25	470	–	**30**	284	–
B3	15	20	5	10	7	1107	0.05	**8**	527	0.05
B4	15	20	10	30	22	957	0.19	22	**661**	0.11
B5	15	20	15	60	44	949	0.20	**50**	463	0.19
B6	15	25	5	10	**5**	1934	0.14	3	1639	0.16
B7	15	25	10	30	16	802	0.24	**19**	827	0.34
B8	15	25	15	60	30	952	0.26	**40**	540	0.23
B9	15	25	20	100	71	584	0.24	**82**	452	0.22
[Total] /(Average):				[340]	[227]	(788)	(0.22)	**[263]**	(22)	(0.21)

Table 6
Computational results for the data cluster C.

					SSPMF			*JGSMF*		
Group	*N*	*M*	*C*	*I*	*Opt.*	*Time*	*Gap (%)*	*Opt.*	*Time*	*Gap (%)*
C1	20	15	5	10	0	–	0.43	**1**	–	0.26
C2	20	15	10	20	10	138	0.50	**17**	1	0.08
C3	20	20	5	10	0	–	0.35	**1**	–	0.27
C4	20	20	10	10	6	33	0.14	**7**	9	0.08
C5	20	20	15	30	12	71	–	**30**	1	–
C6	20	25	5	10	0	–	0.36	**1**	–	0.29
C7	20	25	10	10	6	74	0.06	**8**	310	0.06
C8	20	25	15	40	10	1	0.44	10	**0.2**	0.33
C9	20	25	20	40	26	4	0.41	**37**	0.5	0.12
C10	25	15	10	10	8	993	–	**10**	3	–
C11	25	20	10	10	1	19	0.21	**2**	5	0.13
C12	25	20	15	10	1	3485	0.39	**7**	2873	0.24
C13	25	25	10	10	**3**	136	0.19	1	69	0.15
C14	25	25	15	10	10	7	–	10	**0.8**	–
C15	25	25	20	30	21	8	0.37	**27**	0.6	0.14
[Total] /(Average):				[260]	[114]	(132)	(0.36)	**[169]**	(135)	(0.24)

not optimally solved by both methods, the report can be accessed at [https://github.com/nakhundo/SSP] (see Table 7).

In general, there were few instances in which *JGSMF* was outperformed by *SSPMF*. The optimality gap of common cases that were not solved to optimality shows a similar result, see Fig. 7. By analyzing these instances it was observed that the higher run time of *JGSMF* was caused by assigning a small number to the initial bin count (K_0). As a result, additional time was required to finalize P_1 in order to achieve a feasible bin count, leaving little to no time available for processing P_3.

5.3.2. Component analysis

In this section, the focus is on comparing the gradual improvement of each component within the *JGSMF* model. As previously mentioned, the performance of the *JGSMF* model is influenced by the number of bins used. However, determining the precise number of bins that strikes a balance between solving the problem efficiently and maintaining optimality is challenging. To address this, the solution procedure is divided into three phases, and the problem is solved multiple times (refer to Section 4.4).

For testing purposes, our focus lies on a single run of the *JGSMF* model, aiming to evaluate its performance while incorporating the symmetry-breaking cuts (3k)–(3o). We will refer to this model as *JGSMF′*. Subsequently, we enhance the *JGSMF′* model by introducing performance improvement cuts (3p)–(3r), resulting in the modified model referred to as *JGSMF″*. Finally, we augment the *JGSMF′* model with clique-related cuts (3s)–(3t), and denote this further enhanced

Table 7
Computational results for the data cluster D.

					SSPMF			*JGSMF*		
Group	*N*	*M*	*C*	*I*	*Opt.*	*Time*	*Gap (%)*	*Opt.*	*Time*	*Gap (%)*
D1	10	10	4	10	10	**6**	–	10	10	–
D2	10	10	5	10	10	**4**	–	10	**4**	–
D3	10	10	6	10	10	0.4	–	10	**0.2**	–
D4	10	10	7	10	10	**0.04**	–	10	**0.04**	–
D5	15	20	6	10	8	1255	0.25	8	**840**	0.25
D6	15	20	8	10	5	444	0.20	**7**	85	0.06
D7	15	20	10	10	9	4	–	**10**	4	–
D8	15	20	12	10	9	98	–	**10**	7	–
[Total] /(Average):				[80]	[71]	(187)	(0.22)	**[75]**	(101)	(0.14)

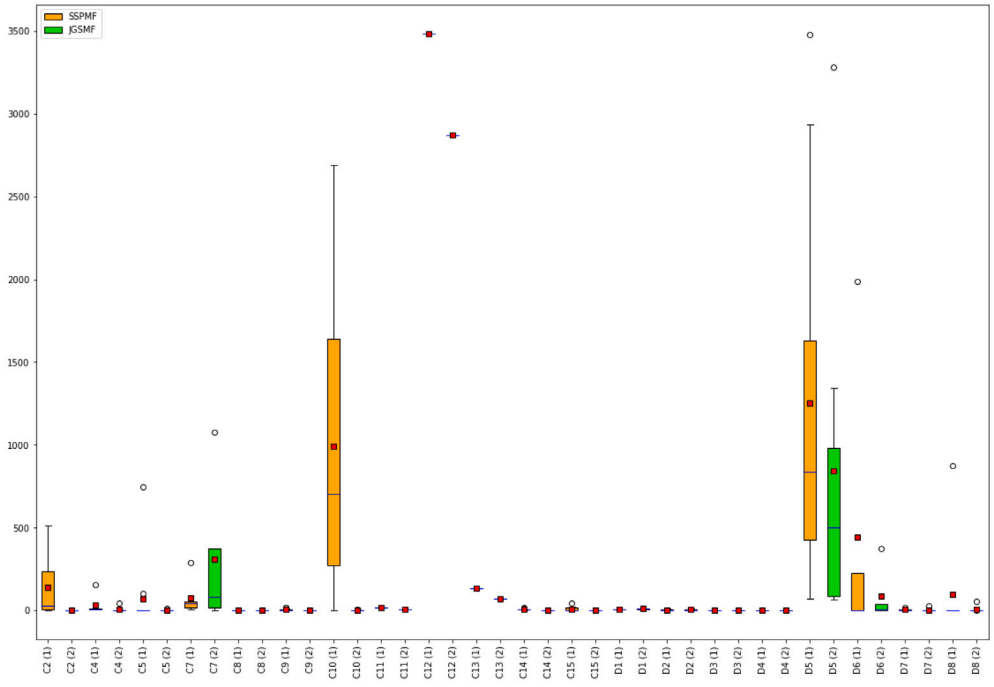

Fig. 6. Comparison of the solution times for the instances in clusters C and D, that are solved with both methods.

model as *JGSMF'''*. In order to maintain uniformity across the three models, a constant number of bins is used, which corresponds to half of the total number of jobs. Specifically, the number of bins, designated as K, is defined as $K = \lceil N/2 \rceil$.

For this test, a total of twenty-five sets were selected, each comprising ten instances (I) with various parameters, see Table 8. In the majority of cases, the addition of extra components proved beneficial, either by achieving optimality (O) for more instances or by reducing the overall run time (R). However, it is worth noting that there were instances where the solution time did not demonstrate improvement. These instances typically constituted the easier cases, solvable within a second. Essentially, by introducing additional constraints to these straightforward instances, the model's size slightly increased without yielding any computational advantage.

By excluding these easy instances from consideration, it became evident that the *JGSMF'* model outperformed *SSPMF*. Furthermore, the *JGSMF'* model exhibited even better performance in terms of either solving more instances or reducing the run time, with only a few exceptions. The introduction of clique-related cuts further enhanced the overall performance. It is important to note that the run time encompasses the process of finding the maximal clique, facilitated by the Python package "networkx". Given the small scale of the resulting graph in all instances, the maximal clique finding step typically required less than 0.1 s.

6. Conclusion

In this research, we studied an $\mathcal{NP}$-hard combinatorial optimization problem arising from manufacturing systems, the Job Sequence and Switching Problem. A new reformulation was proposed to model SSP as a Job Grouping and Sequencing with a Multi-commodity Flow. The symmetrical property of SSP was exploited and symmetry-breaking cuts were added to the model to reduce the search space and speed up the solution process. The formulation was tightened further by converting SSP to an undirected graph and using maximum clique size as an initial bin count. To get the maximum performance from the proposed reformulation the solution procedure was divided into three phases and in each phase, the model was solved iteratively until the optimality of the solution was guaranteed.

The conducted computational study showed that as the size of the problem grows the proposed method was able to solve up to 33% more instances compared to the state-of-the-art method. The solution time for the hard instances that were solvable with the currently known methods in the literature is improved up to 48 times. Furthermore, The optimality gap was reduced up to 39% for the large instances that were not solvable within the given time threshold.

Acknowledgments

Both authors were supported by the United States Department of Energy (DOE), USA grant DE-SC0018175.

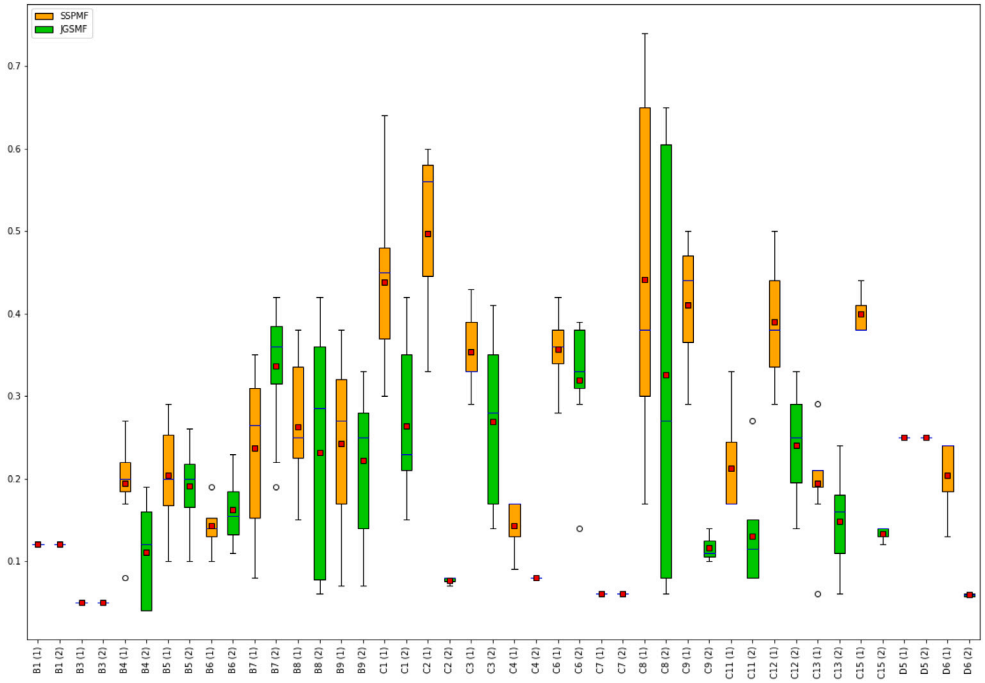

Fig. 7. Comparison of the optimality gaps for the instances that could not be solved with either method.

Table 8
Component analysis of the *JGSMF* model.

					SSPMF		*JGSMF′*		*JGSMF″*		*JGSMF‴*	
Set	*N*	*M*	*C*	*I*	*Opt.*	*Time*	*Opt.*	*Time*	*Opt.*	*Time*	*Opt.*	*Time*
1	10	10	5	10	10	4.0	10	2.3	10	1.7	10	**0.5**
2	15	15	5	10	7	578.9	10	31.4	10	34.5	10	**10.7**
3	15	15	10	10	10	1.0	10	0.8	10	**0.6**	10	0.7
4	15	20	5	10	7	1107.0	10	36.9	10	39.0	10	**4.0**
5	15	20	10	10	10	2.5	10	1.1	10	**1.0**	10	**1.0**
6	15	20	15	10	10	**0.1**	10	0.3	10	**0.1**	10	**0.1**
7	15	25	5	10	5	1933.9	10	7.7	10	6.5	10	**0.1**
8	15	25	10	10	10	2.7	10	1.2	10	**1.0**	10	1.1
9	15	25	15	10	10	0.3	10	**0.2**	10	**0.2**	10	**0.2**
10	15	25	20	10	10	**0.1**	10	**0.1**	10	**0.1**	10	**0.1**
11	20	15	5	10	0	–	9	1048.0	9	1107.4	**10**	94.6
12	20	15	10	10	10	138.1	10	5.2	10	**3.1**	10	5.6
13	20	20	5	10	0	–	8	616.0	9	800.7	**10**	216.3
14	20	20	10	10	6	32.7	8	48.9	8	**45.6**	8	56.1
15	20	20	15	10	10	**0.3**	10	0.7	10	0.4	10	0.5
16	20	25	5	10	0	–	10	285.7	10	302.9	10	**33.0**
17	20	25	10	10	6	74.3	7	102.2	10	537.5	10	**181.8**
18	20	25	15	10	10	1.3	10	0.8	10	**0.2**	10	0.9
19	20	25	20	10	10	0.2	10	**0.1**	10	**0.1**	10	**0.1**
20	25	15	10	10	8	992.6	9	38.7	10	228.3	10	**162.1**
21	25	20	10	10	1	19.2	4	550.0	4	622.4	4	**443.0**
22	25	20	15	10	1	3485.1	1	132.5	1	115.7	**2**	639.1
23	25	25	10	10	3	136.0	3	10.3	3	8.1	**4**	33.7
24	25	25	15	10	10	7.0	10	6.6	10	5.0	10	**3.7**
25	25	25	20	10	10	0.3	10	**0.2**	10	0.3	10	**0.2**
[Total] /(Average):				[250]	[174]	(204)	[219]	(100)	[224]	(142)	**[228]**	(47)

References

Adjiashvili, D., Bosio, S., & Zemmer, K. (2015). Minimizing the number of switch instances on a flexible machine in polynomial time. *Operations Research Letters*, *43*(3), 317–322.

Ahmadi, E., Goldengorin, B., Süer, G. A., & Mosadegh, H. (2018). A hybrid method of 2-TSP and novel learning-based GA for job sequencing and tool switching problem. *Applied Soft Computing*, *65*, 214–229.

Al-Fawzan, M., & Al-Sultan, K. (2003). A tabu search based algorithm for minimizing the number of tool switches on a flexible machine. *Computers & Industrial Engineering*, *44*(1), 35–47.

Amaya, J., Cotta, C., Fernández-Leiva, A., & García-Sánchez, P. (2020). Deep memetic models for combinatorial optimization problems: application to the tool switching problem. *Memetic Computing*, *12*, 3–22.

Anjos, M. F., & Vieira, M. V. (2017). Mathematical optimization approaches for facility layout problems: The state-of-the-art and future research directions. *European Journal of Operational Research*, *261*(1), 1–16.

Ayres, R. U. (1987). Future trends in factory automation.

Baykasoğlu, A., & Ozsoydan, F. B. (2017). Minimizing tool switching and indexing times with tool duplications in automatic machines. *International Journal of Advanced Manufacturing Technology*, *89*(5), 1775–1789.

Baykasoğlu, A., & Ozsoydan, F. B. (2018). Minimisation of non-machining times in operating automatic tool changers of machine tools under dynamic operating conditions. *International Journal of Production Research*, *56*(4), 1548–1564.

Beezão, A. C., Cordeau, J.-F., Laporte, G., & Yanasse, H. H. (2017). Scheduling identical parallel machines with tooling constraints. *European Journal of Operational Research*, *257*(3), 834–844.

Calmels, D. (2019). The job sequencing and tool switching problem: state-of-the-art literature review, classification, and trends. *International Journal of Production Research*, *57*(15–16), 5005–5025.

Calmels, D. (2022). An iterated local search procedure for the job sequencing and tool switching problem with non-identical parallel machines. *European Journal of Operational Research*, *297*(1), 66–85.

Catanzaro, D., Gouveia, L., & Labbé, M. (2015). Improved integer linear programming formulations for the job sequencing and tool switching problem. *European Journal of Operational Research*, *244*(3), 766–777.

Chaves, A. A., Lorena, L. A. N., Senne, E. L. F., & Resende, M. G. (2016). Hybrid method with CS and BRKGA applied to the minimization of tool switches problem. *Computers & Operations Research*, *67*, 174–183.

Crama, Y., Moonen, L. S., Spieksma, F. C., & Talloen, E. (2007). The tool switching problem revisited. *European Journal of Operational Research*, *182*(2), 952–957.

Crama, Y., Oerlemans, A. G., & Spieksma, F. C. (1994). Minimizing the number of tool switches on a flexible machine. In *Production planning in automated manufacturing* (pp. 165–195). Springer.

Cumings, S. (1986). Developing integrated tooling systems: A case study at garrett turbine engine company.. In *1986 fall industrial engineering conference* (pp. 21–26).

Dadashi, H., Moslemi, S., & Mirzazadeh, A. (2016). Optimization of a new tool switching problem in flexible manufacturing systems with a tool life by a genetic algorithm. *International Journal of Industrial and Manufacturing Systems Engineering*, *1*(3), 52.

Denton, B. T., Miller, A. J., Balasubramanian, H. J., & Huschka, T. R. (2010). Optimal allocation of surgery blocks to operating rooms under uncertainty. *Operations Research*, *58*(4-part-1), 802–816.

Eilon, S., & Christofides, N. (1971). The loading problem. *Management Science*, *17*(5), 259–268.

Farughi, H., Dolatabadiaa, M., Moradi, V., Karbasi, V., & Mostafayi, S. (2017). Minimizing the number of tool switches in flexible manufacturing cells subject to tools reliability using genetic algorithm. *Journal of Industrial and Systems Engineering*, *10*(special issue on Quality Control and Reliability), 17–33.

Fathi, Y., & Barnette, K. (2002). Heuristic procedures for the parallel machine problem with tool switches. *International Journal of Production Research*, *40*(1), 151–164.

Furrer, M., & Mütze, T. (2017). An algorithmic framework for tool switching problems with multiple objectives. *European Journal of Operational Research*, *259*(3), 1003–1016.

Ghiani, G., Grieco, A., & Guerriero, E. (2007). An exact solution to the TLP problem in an NC machine. *Robotics and Computer-Integrated Manufacturing*, *23*(6), 645–649.

Ghiani, G., Grieco, A., & Guerriero, E. (2010). Solving the job sequencing and tool switching problem as a nonlinear least cost hamiltonian cycle problem. *Networks: An International Journal*, *55*(4), 379–385.

Ghoniem, A., & Farhadi, F. (2015). A column generation approach for aircraft sequencing problems: a computational study. *Journal of the Operational Research Society*, *66*(10), 1717–1729.

Gökgür, B., Hnich, B., & Özpeynirci, S. (2018). Parallel machine scheduling with tool loading: a constraint programming approach. *International Journal of Production Research*, *56*(16), 5541–5557.

Ham, I. (1985). An expanding role for group technology. *Computer Integrated Manufacturing Review*, *1*, 21–25.

Hirvikorpi, M., Knuutila, T., Leipälä, T., & Nevalainen, O. S. (2007). Job scheduling and management of wearing tools with stochastic tool lifetimes. *International Journal of Flexible Manufacturing Systems*, *19*(4), 443–462.

Hirvikorpi, M., Nevalainen, O., & Knuutila, T. (2006). Job ordering and management of wearing tools. *Engineering Optimization*, *38*(2), 227–244.

Hirvikorpi, M., Salonen, K., Knuutila, T., & Nevalainen, O. S. (2006). The general two-level storage management problem: A reconsideration of the KTNS-rule. *European Journal of Operational Research*, *171*(1), 189–207.

Jans, R. (2009). Solving lot-sizing problems on parallel identical machines using symmetry-breaking constraints. *INFORMS Journal on Computing*, *21*(1), 123–136.

Karakayalı, İ., & Azizoğlu, M. (2006). Minimizing total flow time on a single flexible machine. *International Journal of Flexible Manufacturing Systems*, *18*(1), 55–73.

Konak, A., Kulturel-Konak, S., & Azizoğlu, M. (2008). Minimizing the number of tool switching instants in flexible manufacturing systems. *International Journal of Production Economics*, *116*, 298–307.

Kusiak, A. (1986). Application of operational research models and techniques in flexible manufacturing systems. *European Journal of Operational Research*, *24*(3), 336–345.

Laporte, G., Salazar-Gonzalez, J. J., & Semet, F. (2004). Exact algorithms for the job sequencing and tool switching problem. *IIE Transactions*, *36*(1), 37–45.

Li, J., Harabor, D., Stuckey, P. J., Ma, H., & Koenig, S. (2019). Symmetry-breaking constraints for grid-based multi-agent path finding. In *Proceedings of the AAAI conference on artificial intelligence, vol.33, no. 01* (pp. 6087–6095).

Lima, R. M., & Novais, A. Q. (2016). Symmetry breaking in MILP formulations for unit commitment problems. *Computers & Chemical Engineering*, *85*, 162–176.

Matzliach, B., & Tzur, M. (2000). Storage management of items in two levels of availability. *European Journal of Operational Research*, *121*(2), 363–379.

Mauergauz, Y. (2017). Job and tool group scheduling for a machining center. *International Journal of Management Science and Engineering Management*, *12*(4), 280–287.

McGeoch, L. A., & Sleator, D. D. (1991). A strongly competitive randomized paging algorithm. *Algorithmica*, *6*(1), 816–825.

Mecler, J., Subramanian, A., & Vidal, T. (2021). A simple and effective hybrid genetic search for the job sequencing and tool switching problem. *Computers & Operations Research*, *127*, Article 105153.

Ostrowski, J., Anjos, M. F., & Vannelli, A. (2015). Modified orbital branching for structured symmetry with an application to unit commitment. *Mathematical Programming*, *150*, 99–129.

Ostrowski, J., Linderoth, J., Rossi, F., & Smriglio, S. (2011). Orbital branching. *Mathematical Programming*, *126*, 147–178.

Özpeynirci, S., Gökgür, B., & Hnich, B. (2016). Parallel machine scheduling with tool loading. *Applied Mathematical Modelling*, *40*(9–10), 5660–5671.

Paiva, G. S., & Carvalho, M. A. M. (2017). Improved heuristic algorithms for the job sequencing and tool switching problem. *Computers & Operations Research*, *88*, 208–219.

Pemmaraju, S., & Srinivasan, A. (2008). The randomized coloring procedure with symmetry-breaking. In *Automata, languages and programming: 35th international colloquium, ICALP 2008, reykjavik, iceland, July 7-11, 2008, proceedings, part i 35* (pp. 306–319). Springer.

Pfetsch, M. E., & Rehn, T. (2019). A computational comparison of symmetry handling methods for mixed integer programs. *Mathematical Programming Computation*, *11*, 37–93.

Rupe, J., & Kuo, W. (1997). Solutions to a modified tool loading problem for a single FMM. *International Journal of Production Research*, *35*(8), 2253–2268.

Salonen, K., Raduly-Baka, C., & Nevalainen, O. S. (2006). A note on the tool switching problem of a flexible machine. *Computers & Industrial Engineering*, *50*(4), 458–465.

Schwerdfeger, S., & Boysen, N. (2017). Order picking along a crane-supplied pick face: The SKU switching problem. *European Journal of Operational Research*, *260*(2), 534–545.

Sherali, H. D., & Smith, J. C. (2001). Improving discrete model representations via symmetry considerations. *Management Science*, *47*(10), 1396–1407.

da Silva, T. T., Chaves, A. A., & Yanasse, H. H. (2021). A new multicommodity flow model for the job sequencing and tool switching problem. *International Journal of Production Research*, *59*(12), 3617–3632.

Solimanpur, M., & Rastgordani, R. (2012). Minimising tool switching and indexing times by ant colony optimisation in automatic machining centres. *International Journal of Operational Research*, *13*(4), 465–479.

Stecke, K. E., & Solberg, J. J. (1981). Loading and control policies for a flexible manufacturing system. *The International Journal of Production Research*, *19*(5), 481–490.

Tang, C. S., & Denardo, E. V. (1988a). Models arising from a flexible manufacturing machine, part I: minimization of the number of tool switches. *Operations Research*, *36*(5), 767–777.

Tang, C. S., & Denardo, E. V. (1988b). Models arising from a flexible manufacturing machine, part II: Minimization of the number of switching instants. *Operations Research*, *36*(5), 778–784.

Tomek, P. (1986). Tooling strategies related to FMS management. *The FMS Magazine*, *5*(4), 102–107.

Van Hop, N., & Nagarur, N. N. (2004). The scheduling problem of PCBs for multiple non-identical parallel machines. *European Journal of Operational Research*, *158*(3), 577–594.

Verschae, J., Villagra, M., & von Niederhäusern, L. (2023). On the geometry of symmetry breaking inequalities. *Mathematical Programming*, *197*(2), 693–719.

Yanasse, H. H., Rodrigues, R. d. C. M., & Senne, E. L. F. (2009). Um algoritmo enumerativo baseado em ordenamento parcial para resolução do problema de minimização de trocas de ferramentas. *Gestão & Produção*, *16*, 370–381.