Computers and Operations Research 88 (2017) 208–219

Contents lists available at ScienceDirect

Computers and Operations Research

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

Improved heuristic algorithms for the Job Sequencing and Tool Switching Problem

Gustavo Silva Paiva*, Marco Antonio M. Carvalho

Federal University of Ouro Preto, Ouro Preto, Minas Gerais, Brazil

ARTICLE INFO

Article history:
Received 14 November 2016
Revised 15 April 2017
Accepted 17 July 2017
Available online 18 July 2017

Keywords:
Scheduling
Job Sequencing and Tool Switching Problem
Iterated Local Search

ABSTRACT

In flexible manufacturing systems, a single machine can be configured with different tools for processing different jobs, each requiring a specific set of tools. There is a limit to the maximum number of tools that can be loaded simultaneously in the machine; between the processing of two different jobs, it may be necessary to switch these tools, causing interruptions in the production line. The *Job Sequencing and Tool Switching Problem* (SSP) aims to determine a job sequence and the tool-loading order for a flexible machine, in order to minimize the number of tool switches. These two tasks can be separated into *Sequencing*, an $\mathcal{NP}$-hard problem, and *Tooling*, which is a $\mathcal{P}$ problem if the job sequence is given. This paper proposes a methodology that uses graph representations, heuristic methods, and local search methods to solve the sequencing problem. These contributions are combined in an Iterated Local Search scheme, which is then combined with a classical tooling method, in order to solve the SSP. Comprehensive computational experiments show that the resulting method is competitive and can establish new best solutions for literature instances, while outperforming the current state-of-the-art method.

1. Introduction

A Flexible Manufacturing System (Crama, 1997; Gray et al., 1993), or FMS, provides versatility and flexibility in production planning with respect to the variety of manufactured products, and fast adaptability in case of unexpected problems. A very common type of FMS, mainly employed in metallurgic and microelectronic companies, uses *flexible machines*.

A flexible machine is able to perform different types of operations (such as cutting, drilling, etc.) without requiring an abrupt change in context between two different operations, making the production more dynamic. For this purpose, these machines have a magazine with a fixed capacity into which different *tools* may be loaded. Each product to be manufactured requires a specific set of tools (e.g., cut blades, drill bits, etc.) to be loaded into the magazine. The tool magazine is able to store all the necessary tools for each product separately, but it is usually not able to hold all tools needed by all products simultaneously. The manufacturing of a specific product can be interpreted as a *job* to be processed.

In order to take full advantage of FMS, it is necessary to correctly plan the production. To process several jobs in sequence in the same production line, it becomes imperative to perform tool switches when the size of the set of required tools for processing all jobs is larger than the magazine's capacity – which is a common case. To perform those switches, the machine must be shut down, interrupting the production line. To increase productivity, the number of tool switches must be minimized in order to decrease the flexible machine's idle time.

In this paper, we consider that (i) the setup times to switch tools are uniform, (ii) tools have uniform size, (iii) job processing times are unitary, and (iv) each tool may be loaded into any magazine slot. Simultaneously determining the job sequence and the tool loadings is known as the *Job Sequencing and Tool Switching Problem* (SSP). As showed by Tang and Denardo (1988), the SSP can be decomposed in two subproblems:

1. The *Sequencing Problem* (SP), which involves determining a job sequence that minimizes the number of tool switches.
2. The *Tooling Problem* (TP), which involves scheduling the loading and unloading of tools for a fixed job sequence such that the number of switches is minimized.

The Sequencing Problem is an $\mathcal{NP}$-Hard problem as demonstrated by Crama et al. (1994). On the other hand, the Tooling Problem is trivial and can be accomplished in deterministic polynomial time by the *Keep Tool Needed Soonest* (KTNS) policy, as shown in Tang and Denardo (1988).

An instance of the SSP is composed of a set of jobs to be processed $J = \{1, \dots, n\}$, a set of available tools $T = \{1, \dots, m\}$, a set

* Corresponding author.
E-mail addresses: guustavo.paiva@gmail.com (G.S. Paiva), mamc@iceb.ufop.br (M.A.M. Carvalho).

http://dx.doi.org/10.1016/j.cor.2017.07.013
0305-0548/© 2017 Elsevier Ltd. All rights reserved.

Table 1
Example of an SSP instance.

Jobs	1	2	3	4	5
Tools	1	1	3	2	1
	2	3	4	3	4
	4		5	5	6
Magazine capacity = 3					

Table 2
Example of the matrix representation and possible SSP solutions.

(a)						(b)					
ϕ	1	2	3	4	5	ϕ	3	4	1	5	2
	1	1	0	0	1		0	0	1	1	1
	1	0	0	1	0		0	1	1	0	0
	0	1	1	1	0		1	1	0	0	1
	1	$\underline{1}$	1	0	1		1	0	1	1	$\underline{1}$
	0	0	1	1	0		1	1	0	0	0
	0	0	0	0	1		0	0	0	1	0

of tools T_k required to process a job $k \in J$, and the capacity C of the magazine. A solution for the SSP is expressed by a job sequence (i.e., a permutation) ϕ of the set J and also by the schedule of tool switches.

The schedule of tool switches is represented by a binary matrix $S^{\phi}_{m,n}$, in which the columns follow the order of ϕ and each entry $s^{\phi}_{i,j} = 1$ if the tool i is loaded in the machine before the processing of the jth job. Otherwise, $s^{\phi}_{i,j} = 0$. The number of tool switches corresponds to the number of inversions such that $s^{\phi}_{i,j-1} = 0$ and $s^{\phi}_{i,j} = 1$ ($\forall i \in T,\ \ j \in \{2, \dots, n\}$), i.e., the ith tool was not loaded in the machine before the processing of the jth $-$ 1 job but was loaded before the processing of the j^{th} job. The initial insertions of the tools are also counted as tool switches.

For a fixed sequence of jobs, the optimal schedule for tool switches is trivially generated in deterministic polynomial time by the KTNS policy (Tang and Denardo, 1988). Briefly, this policy minimizes the total number of switches using a simple rule: when tool switches are necessary, the tools that will be needed soonest by the next jobs in the sequence are kept on the machine.

Table 1 presents a numerical example of an SSP instance. The first row represents the jobs (enumerated from 1 to 5) and the four following rows represent the tools required to process the jobs in each column. The final row presents the tool magazine capacity.

Table 2 presents two possible solutions for the instance presented in Table 1. In (a), matrix S^{ϕ} follows the job sequence $\phi = [1, 2, 3, 4, 5]$. In this example, it is noticeable that tool 4 (underlined) is not required in the processing of job 2, but was kept in the magazine to avoid unnecessary switches in the following stages. This job sequence implies a total of nine tool switches, as follows: three switches to insert the initial tools (1, 2, and 4); one switch before the processing of job 2 (tool 2 for tool 3); another switch before job 3 (tool 1 for tool 5); one more to process job 4 (tool 4 for tool 2), and three switches before job 5 (tools 2, 3, and 5 for tools 1, 4, and 6, respectively). Table 2(b) presents a second possible job sequence, $\phi = [3, 4, 1, 5, 2]$, which results in eight tool switches.

In this work we focus on the Sequencing subproblem of the SSP; to solve it, we propose (i) modifying the SSP representation using graphs, (ii) a constructive heuristic, (iii) a local search method, and (iv) a metaheuristic implementation. The principle of the constructive heuristic is to avoid addressing the problem in terms of job permutation. Instead, the relationship between tools is analyzed for further insights into how the jobs may be sequenced. In order to do so, a graph representation is adopted and modified. A problem-specific local search method based on the reordering of jobs is also proposed. These components are combined in an Iterated Local Search metaheuristic, along with the KTNS procedure. The proposed methods are then compared to the state-of-the-art SSP solution method, using a wide set of literature instances.

The remainder of this work is organized as follows. Section 2 presents a literature review on the SSP and some of its variations. Section 3 describes in detail the proposed contributions; comprehensive computational experiments are reported in Section 4. Concluding remarks are presented in Section 5.

2. Related works

The SSP has been addressed in the literature either as a single problem or explicitly decomposed into its subproblems. Very often, the SP is the problem considered, and the KTNS procedure is employed for the TP solution; this gives rise to SSP solution methods, as is the case in this study. The first work to address the SSP was Tang and Denardo (1988), which formally introduced the SSP and the KTNS policy for the Tooling Problem. Later that year, Bard (1988) proposed non-linear and linear formulations to solve the SSP, along with a heuristic based on the Lagrangian relaxation.

Gray et al. (1993) provides an integrated resource planning hierarchy and a classification of tool management issues in flexible manufacturing systems, including the SSP. Later, Crama (1997) also discussed problems in flexible manufacturing systems and the SSP as well. A brief classification of approaches is given, and the heuristics available for the SSP at that time are pointed out.

A formal proof that the SSP is $\mathcal{NP}$-Hard when $C \geq 2$ is found in Crama et al. (1994). This result stems from the complexity of the Sequencing Problem. In this same work, they proposed a multistart greedy heuristic and a Traveling Salesman Problem (TSP)-based formulation to solve the SSP. This formulation uses a complete graph in which nodes represent jobs and the cost of each edge represents the cost of sequencing two adjacent jobs. Proceeding with the same idea, Hertz et al. (1998) proposed several metrics to estimate edge cost on a TSP graph. Additionally, various TSP heuristic methods were adapted to reflect some peculiarities of the SSP, including the GENI and GENIUS algorithms (Gendreau et al., 1992), which delivered the best results.

Privault and Finke (1995) modeled the tooling problem with nonuniform switching times as a network flow problem on acyclic graphs; this strategy was also applied in a case with uniform switching times as an alternative to the KTNS policy. It also models the SSP as the K-Server Problem and proposes an adaptation of the partitioning algorithm (McGeoch and Sleator, 1991). Later, Privault and Finke (2000) reported that this algorithm outperformed the farthest insertion heuristic and a job grouping heuristic based on the shortest edge and 2-opt heuristics.

Hertz and Widmer (1996) considered an n-job shop environment with nonidentical machines and tooling constraints. A tabu search was proposed to minimize the makespan in such environments. Avci and Akturk (1996) considered precedence constraints among jobs, tool sharing, and tool lifetime while addressing the tooling problem using an integer programming model. The same work also proposed a heuristic for solving the SP. Other SSP variations described in Matzliach (1998) considered tools of nonuniform sizes in an online environment, where complete information about the jobs to be processed is not available. Three heuristics based on a *least recently used* policy were proposed in order to avoid the premature removal of tools from the magazine.

Djellab et al. (2000) introduced an SSP representation using hypergraphs and proposed the iterative best insertion heuristic. The reported results improved on the previous results of Crama et al. (1994), although at a cost of increased computational time.

Heuristic methods previously proposed in the literature were applied to real-world industries by Shirazi and Frizelle (2001). They concluded that real instances were easier to process than artificial ones because the instances were smaller, the magazine capacities were larger, and there was redundancy regarding tools requirements.

An SSP with identical parallel machines and job processing times was addressed by Fathi and Barnette (2002). To minimize the makespan, they developed local search procedures and a heuristic method to determine the machines' job assignments, the job sequences for each machine, and the respective tool switching plans.

Song and Hwang (2002) considered and formally described a TP case in which the objective is to minimize the number of tool transporter movements between the flexible machine and the tools' storage location, rather than minimizing the number of tool switches. In this case, the KTNS policy does not guarantee optimality; thus, an optimal policy is proposed for this case.

A production scheduling problem in the context of printed circuit board manufacturing, equivalent to the SSP, was considered in Ghrayeb et al. (2003). Specifically, the problem addressed the mounting of miniature electronic components into circuit boards. These electronic components are loaded in a fixed number of dispensers, and, depending on the circuit being mounted, it is necessary to switch the components in the dispensers. The objective is to reduce the setup time by reducing the number of switches. This problem (and these methods) can be transported to the SSP context if we consider the circuits as jobs and the electronic components as tools. A mathematical model and a fast heuristic method were developed for this problem.

Al-Fawzan and Al-Sultan (2003) developed five different versions of tabu search, using long- and short-term memory with strategic and probabilistic oscillations to guide the heuristic. The methods were compared using randomly generated instances; the results showed that the long-term memory strategy had a more significant effect on the metaheuristic than the other strategies, although they also seemed promising.

Denizel (2003) presented an integer linear programming formulation, a branch-and-bound procedure, and a lower bounding procedure using Lagrangean decomposition (later transformed into a Lagrangian decomposition-based heuristic) for grouping jobs that share tools, such that each group does not exceed the capacity of the magazine. These groups were then sequenced, in order to minimize the tool switches between them.

An SSP offline case in which each tool may occupy more than one slot of the tool magazine (i.e., the tool sizes were non-uniform) was considered for the first time in Tzur and Altman (2004). Thus, an additional decision in the SSP becomes to determine where to locate each tool on the magazine; this is known as *slot loading*. An integer programming formulation and a heuristic method were proposed for the new problem. Additionally, a variation of the KTNS, named KSTNS, was introduced. This new policy also considers the smallest tool as a criterion for choosing which tools to keep in the machines' magazine.

Laporte et al. (2004) proposed an integer linear programming model and used a branch-and-cut algorithm to solve it. A second approach was proposed based on branch-and-bound and was able to solve small instances with up to 25 tools and 25 jobs, to optimality.

Zhou et al. (2005) applied a beam-search metaheuristic based on a new branch-and-bound scheme for the SSP. This metaheuristic limits the number of explored nodes for each depth of the search tree. This was the first time that a beam-search was applied to the problem, and the method improved the results found by Bard (1988).

The GENIUS method, previously applied to the SSP by Hertz et al. (1998), received a preprocessing phase in Salonen et al. (2006); this phase grouped jobs with similar tools into super jobs. Additionally, GENIUS was transformed into a multistart method and compared to seven other heuristics, including GENIUS itself and other variations. The new method could not outperform its original version when considering random problems and real-world data.

Revisiting the SSP, Crama et al. (2007) proved that considering non-uniformly sized tools and fixed job sequences is also an NP-Hard problem. However, with a fixed value for C, it is possible to solve the problem in polynomial deterministic time, albeit with a large exponential running time.

Ghiani et al. (2007) proved that the SP is a symmetric problem and improved the previous branch-and-bound method of Laporte et al. (2004) by exploiting such a property in the model.

Ant Colony Optimization approaches were presented by Konak and Kulturel-Konak (2007) to minimize the number of tool switching instants, rather than the number of tool switches. The computational experiments considered the instances from Denizel (2003) and reported that the method was able to match a large portion of the optimal solutions available and find near-optimal solutions in the other cases. Later, two tabu search implementations (Konak et al., 2008) were proposed to address the same problem. Larger instances were considered and high-quality solutions were generated in reasonable running times.

Amaya et al. (2008) were the first to approach the SSP with a memetic algorithm. This work presented a local search procedure, and genetic and memetic algorithms – an adaptation of the genetic algorithm with a local search. The memetic, being the best performer among the three proposed algorithms, was able to match the results from the beam-search of Zhou et al. (2005).

A new set of instances for the SSP was introduced by Yanasse et al. (2009), along with a new branch-and-bound algorithm. Although this algorithm obtained better solutions than Laporte et al. (2004) on problems the latter failed to solve, this new algorithm could not generate solutions for instances with 25 jobs and 15, 20, or 25 tools. Senne and Yanasse (2009) presented a new beam search based on Yanasse et al. (2009) branch-and-bound, which also outperformed the results achieved by Zhou et al. (2005).

Zeballos (2010) considered an SSP variation using parallel machines, due dates, part routing, tool lifetimes, and dependent processing times. The objective functions included makespan, tardiness, and machining costs. Different search strategies were compared on randomly generated instances.

Ghiani et al. (2010) proposed an SSP formulation based on the Minimum Cost Hamiltonian Cycle problem with a nonlinear objective function. A branch-and-cut algorithm was proposed to solve this formulation, and experiments showed that it could solve problems with up to 45 jobs and 30 tools in a relatively short computational time compared with Laporte et al. (2004) branch-and-bound.

Again, Amaya et al. (2011) approached the SSP with memetic cooperative algorithms that compose a chain of memetic agents, which communicate with each other using classic network topologies such as ring, broadcast, and random. This new memetic algorithm obtained better results than Amaya et al. (2008) and, consequently, better results than Zhou et al. (2005).

A two-phase heuristic was proposed by Chaves et al. (2012), comprising a construction phase where an initial feasible solution is built, and an improvement phase that employs a standard implementation of the Iterated Local Search metaheuristic. The solution found by this method was applied as an upper bound to Yanasse et al. (2009) branch-and-bound; this reduced computational time by up to 83%, and reduced the number of explored nodes on many instances. Additionally, it was possible to solve optimally some instances that were still unsolved.

A third version of the memetic algorithm, proposed by Amaya et al. (2013), included a refinement based on the cross-entropy technique, which is a Monte Carlo approach to combinatorial optimization. This new version of the memetic algorithm improved on the results achieved by Amaya et al. (2011).

An SSP with tools of non-uniform size was studied by Marvizadeh and Choobineh (2013). The problem was addressed with an integer programming formulation, two job grouping heuristics, and a genetic algorithm. Overall, the genetic algorithm was able to obtain better solutions than the proposed heuristics in a reasonable amount of time on real industry instances.

Catanzaro et al. (2015) surveyed integer programming models for the SSP, highlighting the main features of each one. Furthermore, they proposed three new models that achieved better linear relaxation than the literature models, and consequently were able to solve more instances with the fastest running time among the compared models.

Raduly-Baka and Nevalainen (2015) presented the *Modular* SSP. This variant considers the existence of tool modules that support sets of tools and can be switched at a unary cost. Moreover, they proved that the Modular SSP is also an NP-Hard problem.

Adjiashvili et al. (2015) considered a version of the TP with setup times for loading tools, processing times for jobs, and the ability to perform tool switches without stopping the flexible machine. This type of operation is possible if there are enough empty slots in the magazine, as in (for example) mailroom inserting systems. This problem was named Machine Stopping Minimization and proved to be solvable in deterministic-polynomial time. This work also introduced a new objective function that minimizes the number of times a tool is allocated to different slots of the magazine.

Currently, the state-of-the-art method for solving the SSP is represented by a hybrid metaheuristic proposed by Chaves et al. (2016). A clustering search is used to find promising search space clusters, i.e., regions of the search space with a high probability of containing good solutions. The recent biased random-key genetic algorithm was employed as the search metaheuristic component. Those promising clusters are then intensively examined using variable neighborhood descent as a local search procedure. This method, named CS+BRKGA, recently obtained the best results in many instances reported in the literature, including those proposed by Yanasse et al. (2009) and Crama et al. (1994).

Burger et al. (2015) approached the Color Print Scheduling Problem, which can be modeled as the SSP. This problem arises in the packaging industry, where wrappers require overlay painting with multiple colors. The printing machines also have a magazine with limited capacity for storing colors, and two consecutive print jobs requiring a substantial number of different colors imply changes in the magazine configuration and sequence-dependent setup times for cleaning ink cartridges. The SSP formulations of Tang and Denardo (1988) and Laporte et al. (2004) were applied in a real-world case study. The authors concluded that the model of Tang and Denardo (1988) performed better for small instances and the model of Laporte et al. (2004) performed better for larger instances. Additionally, solution methods based on the decomposition of the SSP into job grouping and group sequencing, as proposed by Salonen et al. (2006), were implemented and compared to each other.

Recently, Beezão et al. (2017) considered the SSP on identical parallel machines, and proposed an adaptive large neighborhood search implementation and two mathematical formulations. These formulations had difficulties solving even small instances with 15 jobs, and the metaheuristic was able to outperform the two heuristics of Fathi and Barnette (2002) on randomly created instances that represents six different scenarios.

Further, Furrer and Mütze (2017) recently presented a branch-and-bound based algorithmic framework for the SSP that considers setup and processing times, tool switches, and switching instants. The framework is flexible and can be adjusted to address the original SSP and its variations. The computational experiments used real-world instances from the Mailroom Insert Planning Problem (Adjiashvili et al., 2015) and other randomly generated problems, which were solved optimally in a few seconds.

3. Methods

This section describes the new contributions in detail: i.e., the adopted graph representation, the search performed using this representation, the transition between rows' and columns' perspectives, the local search procedure, and the Iterated Local Search implementation.

3.1. A graph search-based heuristic

The core idea of the proposed heuristic resides in the notion that column permutation problems may be approached first from the rows' perspective, i.e., first the rows are analyzed to gain some insights, and then these gathered insights are used to solve the problem from the columns' perspective. Therefore, we again decompose the SSP and solve it using a two-step heuristic, named BFS-SSP, that uses the mentioned KTNS as the evaluation function. This is a new heuristic in the SSP context; however, it was inspired by a method for solving a related problem (Carvalho and Soma, 2015).

Tang and Denardo (1988) designed a complete graph $G = (V, E)$, in which the set nodes represent the set of jobs J and the weight of edges $(i, j) \in E$ is an upper bound to the number of tool switches that occur when this pair of jobs is processed consecutively. However, this upper bound considers only the two consecutive jobs, and not the entire sequence of jobs. The global analysis would give rise to a dynamic value, depending on the previously sequenced jobs. The SSP is then solved heuristically as a Traveling Salesman Problem. Other authors (Catanzaro et al., 2015; Crama et al., 1994; Hertz et al., 1998; Privault and Finke, 2000) have also adopted this graph representation, although edge weight estimatives and solution methods vary. A second type of graph was proposed by Chaves et al. (2012), in which nodes represent tools and edges connect tools required by the same job, inducing cliques (i.e., complete subgraphs) on the graph. This graph has parallel edges and does not consider weights on edges; instead, edges have an index that identifies which job requires the connected nodes. A heuristic removes cliques from the graph and sequences the related jobs.

We define a *Tool Graph* as an undirected graph $G = (V, E)$, in which the set of nodes V represents the tools in T and an edge $\{i, j\}$ represents a pair of tools required by the same job. No parallel edges or loops are considered. The weight of the edge $\{i, j\} \in E$ is the number of jobs that require both tools i and j. Considering the example presented in Table 1, job 1 induces the edges {1, 2}, {1, 4}, {2, 4}, and so on. This representation is not dependent on the job sequencing; thus, it is generated considering all jobs at the same time and reflects an SSP instance completely, as every job $k \in J$ induces a clique that contains its required tools. To the best of our knowledge, there is no report of such modifications to the graph representation.

In order to determine a tool sequence T_ϕ, we identify which tools tend to be required together more frequently. A Breadth-First Search (BFS) is performed in the tool graph, using an additional criterion to guide the search: the order of node exploration is based on the cost of the edges. Edges with higher costs denote tools that are used more frequently in pairs by jobs; then, according to a greedy criterion, they should be kept next to each other in

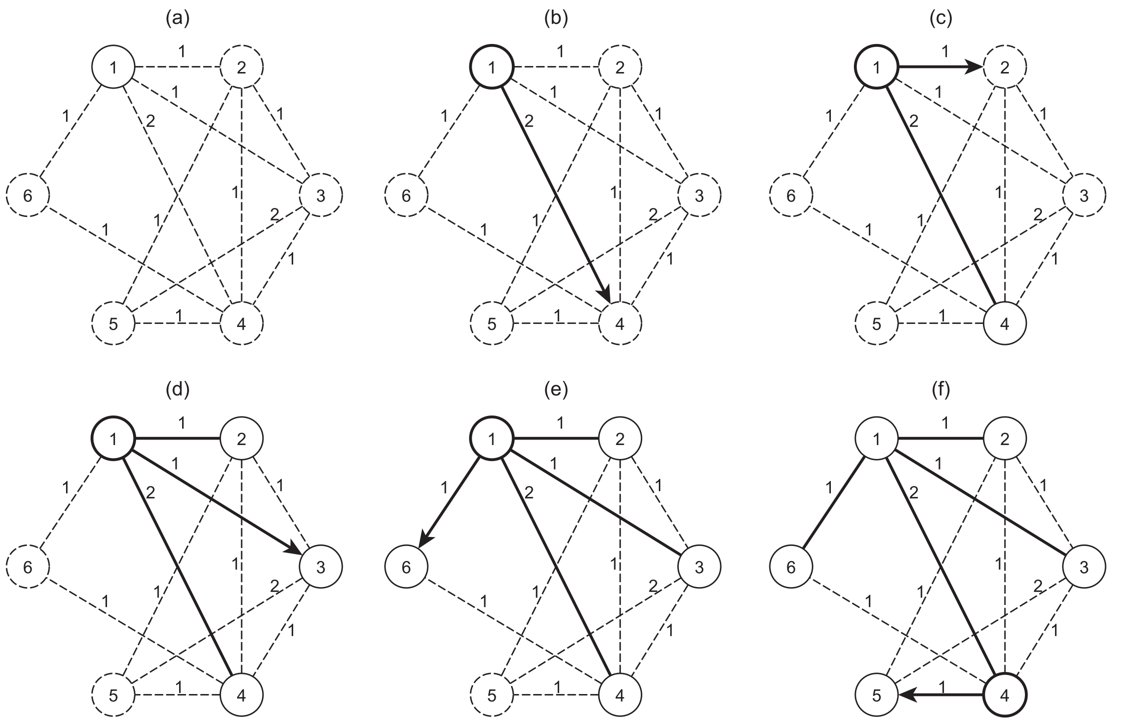

Fig. 1. Step-by-step BFS execution of the instance presented in Table 1.

a tool sequence. The order of exploration of the nodes obtained by the BFS defines the tool sequence T_ϕ.

Algorithm 1 presents the pseudocode for the above-mentioned modified BFS; Fig. 1 illustrates a step-by-step execution, considering the example provided in Table 1.

Considering node 1 as the starting point; it is adjacent to four nodes (2, 3, 4, and 6), which will be explored in the sequence [4, 2, 3, 6], as illustrated in the sequence from (a) to (e) in Fig. 1. In sequence (f), node 4 will have its adjacency explored, thus, node 5 is inserted in the queue. At this point, all nodes have been explored, and $T_\phi = [1, 4, 2, 3, 6, 5]$ is the tool sequence obtained. This sequence of tools gives us insights on how jobs are related to other jobs and how they could be sequenced in order to minimize the number of tool switches.

In order to return to the columns' perspective, the job sequence ϕ is derived from T_ϕ by a greedy algorithm in the following manner: a set A of available tools is gradually built by adding tools in the sequence determined by T_ϕ and according to the number of additional tool switches required, given by the KTNS policy. At each inclusion in A, we check if $T_i \subseteq A$ for each job $i \notin \phi$. If more than one job meets this criterion, we choose the one that results in fewer tool switches and insert it at the end of ϕ. Algorithm 2 presents the pseudocode for the proposed algorithm, where A is the set of all tools available and L is the list of candidate jobs (i.e., jobs for which tools are all already available). The constructive heuristic relies on the KTNS for respecting the magazine capacity.

Algorithm 1: Tool sequencing with BFS.

```
   Input: tool graph $G = (V, E)$, initial node $x$
1  create an empty list $T_\phi$; Create an empty queue $Q$;
2  insert $x$ in $T_\phi$; Insert $x$ in $Q$;
3  while exists not visited nodes in $V$ do
4      while $Q \neq \emptyset$ do
5          $f \leftarrow$ remove element from $Q$;
6          mark $f$ as visited;
7          neighbors $\leftarrow$ neighbors of $f$ in decreasing order
           according to the cost of the edges;
8          foreach $k \in$ neighbors do
9              if $k \notin T_\phi$ then
10                 insert $k$ in $Q$;
11                 insert $k$ at the end of $T_\phi$;
12             end
13         end
14     end
15     if exists a node $k \in V$ not visited then
16         insert $k$ in $Q$;
17     end
18 end
   Output: $T_\phi$
```

Algorithm 2: Greedy Job Sequencing.

```
   Input: set $J$, set $T$, tool sequence $T_\phi$
1  $A \leftarrow \emptyset$;
2  $\phi \leftarrow \emptyset$;
3  foreach $x \in T_\phi$ do
4      $A \leftarrow A \cup x$;
5      $L \leftarrow \{j \in J \mid T_j \subseteq A$ and $j \notin \phi\}$;
6      while $L \neq \emptyset$ do
7          if $\phi = \emptyset$ then
8              $k \leftarrow$ job $j \in L$ with largest $|F_j|$;
9          else
10             $k \leftarrow$ job $j \in L$ that minimizes $KTNS(\phi \cup j)$;
11         end
12         remove $k$ from $L$;
13         insert $k$ at the end of $\phi$;
14     end
15 end
   Output: $\phi$
```

Consider the example in Table 1 and the tool sequence $T_\phi = [1, 4, 2, 3, 6, 5]$. In the first iteration of the greedy Job Sequencing, tool 1 becomes available; however, there is no candidate job. In the second iteration, tool 4 becomes available, but again no candidate job exists. After the third iteration, tool 2 becomes available and job 1 is a candidate. As ϕ is empty, the first tie break criterion (i.e., the job that requires the largest number of tools) is applied, but L only contains one job and thus it is added to ϕ. Subsequently, tool 3 becomes available and job 2 is consequently added to ϕ. In the fifth iteration, tool 6 becomes available and job 5 is the only candidate; thus, it is added to ϕ. In the last iteration, tool 5 becomes available and therefore jobs 3 and 4 are in the candidate list. Because job 3 adds fewer tool switches to the partial solution, it is added to ϕ, and lastly job 4 is added to the solution. In the end, we obtained $\phi = [1, 2, 5, 3, 4]$.

3.2. A new local search method

Crama et al. (1994) presented a definition for *0-blocks* and *1-blocks* while studying the SSP: a maximal sequence of consecutive entries (either 0 or 1, respectively) in the same row of a binary matrix. As a result, the number of tool switches of a solution S^ϕ is exactly the number of 1-blocks in S^ϕ. Crama et al. (1994) explored a particular situation in which each row of S^ϕ has exactly one 1-block. One can easily verify that, if each tool is loaded only once, then the solution is optimal and S^ϕ is an *interval matrix*. Following this idea, Crama et al. (1994) attempted to find submatrices of S^ϕ holding this property. As mentioned, the same work also models the SSP as the Traveling Salesman Problem and introduces *block minimization heuristics*, which consists of traditional TSP heuristics such as nearest neighbor and farthest insertion. The "block minimization" refers to the objective function minimization, rather than to a particular analysis of the structure of S^ϕ. In contrast, based on the 1-blocks definition, we propose a specific local search procedure that aims to reduce the structure and number of 1-blocks by attempting to group their entries.

This local search consists of, iteratively, analyzing each row of S^ϕ once and in random order, searching for two or more 1-blocks. For each pair of 1-blocks, the local search attempts to contiguously group them by moving the related columns from the first 1-block, one by one, to before or after the columns of the second 1-block. The moves are performed in a best insertion fashion; thus, the best move in terms of the number of tool switches (given by the KTNS policy) is performed. Alternatively, in the absence of improving moves, a non-worsening move is performed, if available. This local search was named 1BG-SSP and its pseudocode is presented in Algorithm 3, which, given two 1-blocks i and j, and functions *before*(k, j) and *after*(k, j), returns the resulting solutions if column k from 1-block i is inserted before and after 1-block j, respectively. Function *no_move()* returns the original solution if no move is performed. The move associated with the lowest SSP solution value is performed, observing the priority established in case of ties: *after*, *before* and *no_move*.

Algorithm 3: *1-blocks* grouping (1BG-SSP).

```
   Input: Matrix $S^\phi$
1  foreach randomly selected row $r \in S^\phi$ do
2      find the first 1-block $i$ in $r$;
3      while exists a next 1-block $j$ in $r$ do
4          foreach column $k$ in $i$ do
5              moves ←{after(k, j), before(k, j), no_move()};
6              next_move ← move ∈ moves that results in
                   minimum KTNS value;
7              perform(next_move);
8          end
9          $i \leftarrow j$;
10     end
11 end
   Output: $S^\phi$
```

To illustrate the operation of the 1-block grouping local search, consider the instance presented in Table 1 and the initial solution $\phi = [1, 2, 3, 4, 5]$, presented in Table 2(a), requiring nine tool switches. Initially, the first row is examined and the 1-block i is found, composed of columns 1 and 2. Subsequently, the second 1-block j is found, composed only of column 5. All possible moves of columns from j to i are evaluated. First, column 1 is considered: the possible moves generate the sequences [2, 3, 4, 5, 1] and [2, 3, 4, 1, 5], resulting in nine and eight switches, respectively. Because there is an improving move, it is performed and the solution is updated to [2, 3, 4, 1, 5]. Next, column 2 is examined. The possible moves result in solutions [3, 4, 1, 2, 5] and [3, 4, 1, 5, 2], both requiring eight tool switches. Because there is no worsening of the solution value, the second move is performed, resulting in the solution presented in Table 2(b). The method examines the remaining rows in the same fashion, until the matrix is fully examined.

3.3. Iterated Local Search

After the design and implementation of the aforementioned contributions, we decided to combine them in order to systematically search the solution space. Owing to the nature of these newly proposed methods, we chose the Iterated Local Search (ILS) metaheuristic (Lourenço et al., 2003) to this end. Given an initial solution, this metaheuristic alternates between intensification and diversification of the search. More specifically, the ILS iteratively applies local search and perturbation methods to a current solution until a predetermined stopping criterion is met.

Although a standard implementation of this method was recently used (Chaves et al., 2012) to address the SSP, we believe that the use of tailored components contributes to improved metaheuristic performance. Considering the ILS in particular, our components provide a good starting point for the metaheuristic and help in the intensification of the search. In order to match the ILS requirements, we incorporated a classical diversification strategy, which is also used in a local search method. Therefore, our ILS implementation is composed of (i) the initial solution generated by the heuristic BFS-SSP described in Section 3.1, (ii) a job swap method as a perturbation mechanism, and (iii) two local search methods: a steepest descent method and the 1-block grouping local search described in Section 3.2. Solutions are evaluated by each component using the KTNS policy.

The job swap neighborhood was chosen for the steepest descent method, which swaps the positions of pairs of jobs, generating solutions that differ by exactly two elements from the previous configuration. The steepest descent method only accepts moves that result in an improved solution value. In order to reduce the running time, we chose to partially explore the swap neighborhood, restricting its application to a small portion of the search space and randomly selecting β pairs of jobs to be swapped. These β pairs of jobs are generated beforehand and then evaluated, in order to avoid repetitions.

During the algorithm tuning, we learned that applying the steepest descent before applying the 1BG-SSP method achieved the best solutions. We believe that the reason for this behavior is the complementarity between such methods, i.e., the combination of simple random moves and problem-specific moves provided a robust method.

A portion α of the current solution undergoes perturbation. The number of swaps to be performed is defined over a percentage of

n, and the jobs to be swapped are randomly selected. The pseudocode for the proposed ILS is presented in Algorithm 4.

The acceptance criterion (i.e., the criterion for accepting a new

Algorithm 4: Iterated Local Search.

```
   Input: $\phi$, $\alpha$, $\beta$
1  perform the 1-block grouping local search in $\phi$;
2  while stopping criteria not met do
3      $\phi' \leftarrow \phi$;
4      swap $\alpha$ random jobs in $\phi'$;
5      perform the restricted job swap local search in $\phi'$,
       according to $\beta$;
6      perform the 1BG-SSP local search in $\phi'$;
7      if $KTNS(\phi') < KTNS(\phi)$ then
8          $\phi \leftarrow \phi'$;
9      end
10 end
   Output: $\phi$
```

solution) is defined as the improvement of the solution value, and the stopping criterion is a defined number of iterations. Parameters α and β, as well as the order of the local search procedures and the number of iterations, were defined using the irace package (López-Ibáñez et al., 2016). These values are $\alpha = \lfloor 0.2 \times n \rfloor$, $\beta = \left\lfloor 0.25 \times \binom{n}{2} \right\rfloor$, (job swap, 1BG-SSP) and 200 iterations. The ILS tuning setup is described in Table 3 and considered 10% of representative instances from each set described in Section 4.

For the irace scenario configuration, we set the maximum number of experiments, i.e., the budget, to 1000 runs and the confidence level to 95%. The remaining configurations were set to their default values.

4. Computational experiments

The proposed ILS was implemented in C++ and compiled using the g++ 4.4.1 compiler with the –O3 compilation flag. The experiments were conducted on a 3.2 GHz Intel i5 Quad Core computer with 8 GB of RAM under the Ubuntu 15.10 operating system.

The reported results are compared to the state-of-the-art CS+BRKGA method proposed by Chaves et al. (2016) briefly described in Section 2. The results of the compared method were obtained on a 3.4 GHz Intel i7 computer with 16 GB of RAM. It is important to notice that, although different, the architectures of both computers have comparable processor speeds.

The following tables present the number of jobs (n), the number of tools (m), the magazine capacity (C), the number of instances (i), and the solution found by the proposed BFS-SSP heuristic (S_0). For CS+BRKGA and the proposed ILS implementation, the tables also present the best solution value found (S^*), the average solution value (S), the average running time in seconds (T), and the gap (or percentage distance) to the best known solution, calculated as $100 \times$ (obtained value – best value)/best value. Note that *best_value* considers the solution values found by the proposed ILS. Bold values identify solutions that reached the best known solution values or optimal values (when available).

Table 3
ILS tuning setup.

Parameter	Values
Local search order	{(job swap, 1BG-SSP), (1BG-SSP, job swap)}
α	[0.2–0.8]
β	[0.2–0.8]
Number of iterations	{100, 125, 150, 175, 200}

Owing to the use of randomized components, the methods were independently run 20 times for each instance. The standard deviation (σ) is also presented for each set of instances. Notice that the standard deviation of the CS+BRKGA method was originally reported only for groups C_3 and C_4 of Crama et al. (1994).

4.1. *Yanasse et al. (2009) instances*

The 1350 instances proposed by Yanasse et al. (2009) were divided into five groups (A, B, C, D, and E). Groups A, B, C, and D differ by the number of tools and number of instances. Group E differs from the others by the capacity of the machine.

Table 4 presents the results for group A. Because this group is a trivial one, both methods were able to find all optimal solutions proved by Yanasse et al. (2009). Nonetheless, one can observe that the average running time of the proposed ILS ($T = 0.11$ s) is shorter than the average running time of the CS+BRKGA ($T = 3.71$ s). The ILS robustness in this first group is also noticeable, as the average solution values are identical to the optimal ones.

As for Group A, both methods achieved the optimal solution values for all instances in Group B, as shown in Table 5. Again, the proposed ILS required a shorter average running time ($T = 0.19$ s) than CS+BRKGA ($T = 4.05$ s). Moreover, the average standard deviation of the ILS is small ($\sigma = 0.0018$).

As presented in Table 6, both methods achieved the optimal solution values in the same manner as for the previous groups, and again, the average running time of the ILS ($T = 1.88$ s) was shorter than that of the CS+BRKGA method ($T = 9.83$ s). Further, the ILS average standard deviation is small once again ($\sigma = 0.0009$). For this group of instances, the ILS average solution values match the best known solution values in six out of nine groups of instances.

Of the 260 instances in Group D, the optimal solution values are proved for only 190 of them. The ILS was able to match all these solution values and establish new best solution values, as shown in Table 7.

Considering the instances from group D, the performance difference between the ILS ($T = 5.13$ s and $\sigma = 0.0083$) and the CS+BRKGA ($T = 27.66$ s) becomes more apparent. Once again, in most of the cases, the ILS average solution values also match the best known solutions. We also compared the *ILS* results to the lower and upper bounds from Chaves et al. (2012). As shown in Table 8, one instance was solved to optimality for the first time and two upper bounds were improved. Column e presents the names of these instances. All the upper bounds reported in the literature were matched in the remaining cases.

The previous behavior observed for groups A, B, and C repeats for Group E, as shown in Table 9: both methods achieved all the optimal solutions and the ILS average standard deviation is zero, i.e., the ILS average solution value matches the best known value for all instances in this group. This time, the ILS has an average running time of 0.51 s while the CS+BRKGA has an average time of 6.54 s.

The BFS-SSP heuristic ran in less than 0.03 s for each instance of the five groups considered and, for 75.58% of the instances, the results matched the best ones from the literature. On average, the ILS was able to improve the initial solutions for groups A, B, C, D, and E by 0.60%, 0.65%, 2.07%, 2.27% and 2.05%, respectively.

The reported data for this set of instances indicates the proposed ILS methods' performance improvement over the compared method, as well as its robustness. However, in order to better analyze the method's performance, additional sets of instances were considered.

4.2. *Crama et al. (1994) instances*

The 160 instances proposed by Crama et al. (1994) are divided into four groups (C_1, C_2, C_3, and C_4), which differ from each other

Table 4
Results for group *A*.

n	*m*	*C*	*i*	CS+BRKGA				Proposed ILS					
				S^*	S	*gap* (%)	T	S_0	S^*	S	*gap* (%)	T	σ
8	15	5	10	**17.00**	**17.00**	0.00	2.39	**17.00**	**17.00**	**17.00**	0.00	0.04	0.000
8	15	10	30	**16.83**	**16.83**	0.00	2.65	**16.83**	**16.83**	**16.83**	0.00	0.07	0.000
8	20	5	10	**21.80**	**21.80**	0.00	2.97	22.10	**21.80**	**21.80**	0.00	0.06	0.000
8	20	10	30	**23.07**	**23.07**	0.00	3.54	23.23	**23.07**	**23.07**	0.00	0.13	0.000
8	20	15	60	**22.08**	**22.08**	0.00	3.57	22.15	**22.08**	**22.08**	0.00	0.10	0.000
8	25	5	10	**25.10**	**25.10**	0.00	4.54	**25.10**	**25.10**	**25.10**	0.00	0.03	0.000
8	25	10	30	**28.20**	**28.20**	0.00	4.37	28.37	**28.20**	**28.20**	0.00	0.12	0.000
8	25	15	60	**27.95**	**27.95**	0.00	4.61	28.05	**27.95**	**27.95**	0.00	0.14	0.000
8	25	20	100	**26.61**	**26.61**	0.00	4.72	26.71	**26.61**	**26.61**	0.00	0.11	0.000

Table 5
Results for group *B*.

n	*m*	*C*	*i*	CS+BRKGA				Proposed ILS					
				S^*	S	*gap* (%)	T	S_0	S^*	S	*gap* (%)	T	σ
9	15	5	10	**17.20**	**17.20**	0.00	2.72	**17.20**	**17.20**	**17.20**	0.00	0.07	0.000
9	15	10	30	**17.37**	**17.37**	0.00	3.15	17.40	**17.37**	**17.37**	0.00	0.11	0.000
9	20	5	10	**22.40**	**22.40**	0.00	3.13	22.70	**22.40**	**22.40**	0.00	0.08	0.000
9	20	10	30	**24.17**	**24.17**	0.00	3.92	24.33	**24.17**	**24.17**	0.00	0.19	0.000
9	20	15	60	**22.60**	**22.60**	0.00	3.99	22.70	**22.60**	**22.60**	0.00	0.18	0.000
9	25	5	10	**25.40**	**25.40**	0.00	4.08	25.50	**25.40**	**25.40**	0.00	0.05	0.000
9	25	10	30	**28.77**	**28.77**	0.00	5.08	28.97	**28.77**	28.77	0.00	0.21	0.010
9	25	15	50	**29.74**	29.75	0.00	5.10	30.08	**29.74**	**29.74**	0.00	0.33	0.000
9	25	20	100	**27.19**	**27.19**	0.00	5.26	27.27	**27.19**	27.19	0.00	0.20	0.003

Table 6
Results for group *C*.

n	*m*	*C*	*i*	CS+BRKGA				Proposed ILS					
				S^*	S	*gap* (%)	T	S_0	S^*	S	*gap* (%)	T	σ
15	15	5	10	**21.60**	21.69	0.00	5.31	22.20	**21.60**	**21.60**	0.00	0.53	0.000
15	15	10	30	**19.80**	19.88	0.00	7.10	20.13	**19.80**	**19.80**	0.00	0.89	0.000
15	20	5	10	**25.60**	25.77	0.00	7.26	26.60	**25.60**	**25.60**	0.00	0.67	0.000
15	20	10	30	**28.33**	28.52	0.00	8.93	28.93	**28.33**	**28.33**	0.00	1.59	0.000
15	20	15	60	**25.52**	25.65	0.00	9.61	25.97	**25.52**	25.52	0.00	1.71	0.002
15	25	5	10	**32.50**	32.70	0.00	9.30	33.90	**32.50**	**32.50**	0.00	0.65	0.000
15	25	10	30	**35.07**	35.30	0.00	13.52	35.87	**35.07**	35.07	0.00	1.92	0.009
15	25	15	60	**34.07**	34.27	0.00	13.63	34.85	**34.07**	**34.07**	0.00	3.02	0.000
15	25	20	100	**29.66**	29.79	0.00	13.82	30.07	**29.66**	29.66	0.00	2.06	0.002

Table 7
Results for group *D*.

n	*m*	*C*	*i*	CS+BRKGA				Proposed ILS					
				S^*	S	*gap* (%)	T	S_0	S^*	S	*gap* (%)	T	σ
20	15	5	10	26.10	26.58	0.77	10.78	27.00	**25.90**	**25.90**	0.00	1.39	0.000
20	15	10	20	**18.20**	18.44	0.00	12.34	18.70	**18.20**	18.21	0.00	2.20	0.029
20	20	5	10	29.30	29.93	0.17	14.84	30.30	**29.20**	29.24	0.00	1.85	0.058
20	20	10	10	**20.60**	20.76	0.00	16.08	21.10	**20.60**	**20.60**	0.00	2.88	0.000
20	20	15	30	**21.67**	21.79	0.00	24.66	21.93	**21.67**	**21.67**	0.00	3.32	0.000
20	25	5	10	**35.10**	35.74	0.00	19.16	36.00	**35.10**	**35.10**	0.00	2.10	0.000
20	25	10	10	**25.40**	25.47	0.00	21.49	25.80	**25.40**	**25.40**	0.00	3.73	0.000
20	25	15	40	**36.25**	36.75	0.00	28.11	37.15	**36.25**	36.26	0.00	7.41	0.008
20	25	20	40	**26.15**	26.28	0.00	35.53	26.48	**26.15**	**26.15**	0.00	3.45	0.000
25	15	10	10	**15.90**	16.00	0.00	21.14	16.10	**15.90**	**15.90**	0.00	3.97	0.000
25	20	10	10	**21.60**	22.05	0.00	27.48	22.20	**21.60**	21.61	0.00	9.73	0.044
25	20	15	10	**22.60**	22.82	0.00	25.66	22.90	**22.60**	**22.60**	0.00	9.93	0.000
25	25	10	10	**26.60**	27.06	0.00	36.88	27.70	**26.60**	26.67	0.00	11.68	0.057
25	25	15	10	**25.00**	**25.00**	0.00	54.70	**25.00**	**25.00**	**25.00**	0.00	7.83	0.000
25	25	20	30	**25.50**	25.59	0.00	66.10	25.67	**25.50**	**25.50**	0.00	6.89	0.000

by the number of jobs, tools, and machine capacities. In groups C_1 and C_2 from this second set of instances, both methods repeat the predominant behavior in the previous set of instances. This is explained by the fact that those instances have small dimensions (10 and 15 jobs, respectively). The results for groups C_1 and C_2 are summarized in Table 10, where the groups are separated by a horizontal line.

As mentioned, for groups C_1 and C_2, both methods achieved all the optimal solution values, although the ILS ($T = 0.075$ s and $T = 1.033$ s) obtained solutions in less time compared to CS+BRKGA

Table 8
Comparison with the lower bounds and upper bounds for group *D*.

n	*m*	*c*	*e*	Lower bound	Proposed ILS	Upper bound
20	25	5	L13-8	20	20	21
20	25	10	L14-2	60	63	64
20	25	10	L14-9	45	47	48

($T = 2.418$ s and $T = 11.578$ s, respectively). Also, the ILS average solution values match the optimal ones, implying a standard deviation equal to zero on the two groups of instances except for a single subgroup of instances with a small standard deviation ($\sigma = 0.047$). The same pattern occurs, in a smaller proportion, in the CS+BRKGA results. The BFS-SSP heuristic was able to match the best solution values for 91.25% of these instances in a maximum running time of 0.01 s. In the other cases, the ILS improved the initial solution by 0.92%, on average.

Groups C_3 and C_4 required more effort from both methods when compared to the first two groups. The optimal solution values of these instances are not known; however, the proposed ILS was able to determine new best solution values for some of them. It is not possible to determine the number of solution values that were improved, because only the average solutions are reported in the literature. The results are detailed in Table 11. Again, the groups are separated by a horizontal line.

The BFS-SSP heuristic did not match any of the best solution values for groups C_3 and C_4. However, good solutions were generated in less than five seconds for all instances, which were later improved by the ILS by an average of 4.21%.

Considering group C_3, the proposed ILS demonstrates more robustness ($\sigma = 0.129$ and $T = 134.78$ s) than the CS+BRKGA method ($\sigma = 0.680$ and $T = 123.15$ s), although at the cost of a longer running time. For group C_4, this trend is emphasized: a relatively small error ($\sigma = 0.214$) separated the ILS average solution values from the best solution values found, in contrast to CS+BRKGA ($\sigma = 1.215$), which was unable to achieve the best known solution values in any subgroup of instances. The average ILS running time ($T = 902.03$ s) was approximately twice that of the CS+BRKGA ($T = 541.10$ s), although this is an acceptable running time in practice. Table 12 presents the average standard deviation of both methods for groups C_3 and C_4, separated by a horizontal line.

A statistical analysis was performed to further compare the proposed ILS and the CS+BRKGA method. The average values presented in the previous tables were considered, as individual solutions for each run of the CS+BRKGA are not available. We first applied the Shapiro–Wilk normality test (Shapiro and Wilk, 1965), which rejected the null hypothesis stating that the ILS results could be modeled according to a normal distribution ($W = 0.510$ and $p = 6.0e-6$). After that, we applied the nonparametric Wilcoxon signed-rank test (Rey and Neuhäuser, 2011) in order to investigate whether there is a significant difference between the solution values generated by the ILS and those generated by the CS+BRKGA. The test indicated that there is statistically significant evidence of a difference in the methods' results ($V = 55$, and $p = 0.005889$ for a significance level of 0.05).

We also compared the convergence rate of the two methods. On small instances (up to 25 jobs) both methods converged in 2% of the total computational time on average; however, in instances with 30 or more jobs, CS+BRKGA converged in 10% of the total computational time, while the proposed ILS converged in 33% of the total computational time.

In order to make the comparison between CS+BRKGA and ILS as fair as possible, we ran the ILS for groups C_3 and C_4 again using the same previous standards, while limiting its computational time according to CS+BRKGA's reported time (even though these times were measured using different architectures). The results are shown in Table 13.

The ILS running times were reduced to approximately one-third of the original values. Even with the limited running times, the ILS was able to find solutions close to the previous ones with a maximum gap of 0.91% for any individual instance, thereby confirming the convergence rate analysis while still outperforming the CS+BRKGA method. Indeed, the average solution values of the limited ILS are smaller than the CS+BRKGA's best solution values for five subgroups out of eight. Despite that, the average gap between the methods was only 0.46%.

The reported data for this set of instances strengthens the quality and robustness claims of ILS, as it was able to improve previous best known solutions found in the literature. Additionally, it proves the quality of the initial solutions generated by the BFS-SSP heuristic, which in some cases were superior to those obtained by the CS+BRKGA method.

4.3. *Catanzaro et al. (2015) instances*

Catanzaro et al. (2015) introduced a new set of 160 instances, divided into four groups (*datA, datB, datC*, and *datD*) that differ from each other by the number of jobs, tools, and machine capacities.

Table 14 summarizes the ILS results for the entire set. Column BKS represents the best known solution, reported by Catanzaro et al. (2015) along with the instances. It did not specify the method and running time used to achieve these solutions, although the original article compares integer and linear programming models of Tang and Denardo (1988), Laporte et al. (2004), and three other new models. As can be noticed, this set of instances is quite similar to the set of instances proposed by Crama et al. (1994). Furthermore, it is also possible to notice a resemblance between the results of the proposed ILS for both sets.

For groups *datA* and *datB*4, the ILS results match the reported best known solution values. For the remaining groups (*datB*1, *datB*2, *datB*3, *datC*, and *datD*), the proposed ILS was able to determine new best solutions for 87 instances. Once again, the ILS method proved to be robust, with a maximum average standard

Table 9
Results for group *E*.

n	*m*	*C*	*i*	CS+BRKGA				Proposed ILS					
				S^*	S	*gap* (%)	*T*	S_0	S^*	S	*gap* (%)	*T*	σ
10	10	4	10	**13.50**	**13.50**	0.00	1.55	13.70	**13.50**	**13.50**	0.00	0.08	0.000
10	10	5	10	**11.20**	11.21	0.00	1.96	11.30	**11.20**	**11.20**	0.00	0.08	0.000
10	10	6	10	**10.30**	**10.30**	0.00	2.88	**10.30**	**10.30**	**10.30**	0.00	0.05	0.000
10	10	7	10	**10.00**	**10.00**	0.00	3.45	**10.00**	**10.00**	**10.00**	0.00	0.03	0.000
15	20	6	10	**27.40**	27.71	0.00	7.09	28.40	**27.40**	**27.40**	0.00	0.77	0.000
15	20	8	10	**22.20**	22.33	0.00	7.68	22.80	**22.20**	**22.20**	0.00	0.90	0.000
15	20	10	10	**20.30**	20.34	0.00	12.71	20.40	**20.30**	**20.30**	0.00	0.69	0.000
15	20	12	10	**20.20**	**20.20**	0.00	14.97	**20.20**	**20.20**	**20.20**	0.00	0.62	0.000

Table 10
Results for groups C_1 and C_2.

n	*m*	*C*	*i*	*CS+BRKGA*				Proposed ILS					
				S^*	S	*gap* (%)	T	S_0	S^*	S	*gap* (%)	T	σ
10	10	4	10	**13.10**	13.11	0.00	1.57	13.20	**13.10**	**13.10**	0.00	0.08	0.000
10	10	5	10	**11.20**	**11.20**	0.00	2.05	11.30	**11.20**	**11.20**	0.00	0.08	0.000
10	10	6	10	**10.30**	**10.30**	0.00	2.64	**10.30**	**10.30**	**10.30**	0.00	0.06	0.000
10	10	7	10	**10.10**	**10.10**	0.00	3.41	**10.10**	**10.10**	**10.10**	0.00	0.04	0.000
15	20	6	10	**26.60**	26.87	0.00	8.18	27.00	**26.60**	26.63	0.00	0.84	0.047
15	20	8	10	**21.70**	21.72	0.00	8.88	21.90	**21.70**	**21.70**	0.00	1.09	0.000
15	20	10	10	**20.10**	**20.10**	0.00	11.20	**20.10**	**20.10**	**20.10**	0.00	0.87	0.000
15	20	12	10	**19.60**	**19.60**	0.00	18.05	**19.60**	**19.60**	**19.60**	0.00	0.60	0.000

Table 11
Results for groups C_3 and C_4.

n	*m*	*C*	*i*	*CS+BRKGA*				Proposed ILS				
				S^*	S	*gap* (%)	T	S_0	S^*	S	*gap* (%)	T
30	40	15	10	106.80	108.03	0.38	140.05	111.70	**106.40**	106.70	0.00	86.81
30	40	17	10	88.70	89.98	0.45	127.09	91.80	**88.30**	88.59	0.00	132.01
30	40	20	10	70.70	71.85	0.43	121.43	74.40	**70.40**	70.71	0.00	173.47
30	40	25	10	53.10	53.97	0.38	104.02	55.30	**52.90**	53.14	0.00	146.81
40	60	20	10	199.80	202.25	0.71	599.34	208.50	**198.40**	199.00	0.00	441.66
40	60	22	10	175.30	177.28	1.04	557.97	182.30	**173.50**	174.04	0.00	665.13
40	60	25	10	147.50	149.35	1.03	533.52	154.00	**146.00**	146.52	0.00	1,016.37
40	60	30	10	114.50	116.94	0.53	473.56	118.70	**113.90**	114.18	0.00	1,484.97

Table 12
Standard deviation for groups C_3 and C_4.

n	*m*	*C*	*i*	*CS+BRKGA*	Proposed ILS
				σ	σ
30	40	15	10	0.730	0.147
30	40	17	10	0.810	0.117
30	40	20	10	0.690	0.143
30	40	25	10	0.490	0.109
40	60	20	10	1.310	0.199
40	60	22	10	1.130	0.244
40	60	25	10	1.100	0.235
40	60	30	10	1.310	0.179

Table 13
Results for groups C_3 and C_4, with limited computational time for the ILS.

n	*m*	*C*	*i*	Proposed ILS		
				S^*	S	*gap* (%)
30	40	15	10	106.5	106.69	0.09
30	40	17	10	88.40	88.53	0.11
30	40	20	10	70.50	70.77	0.14
30	40	25	10	53.00	53.25	0.19
40	60	20	10	198.60	198.96	0.10
40	60	22	10	173.60	174.24	0.06
40	60	25	10	146.50	146.92	0.34
40	60	30	10	114.10	114.81	0.18

deviation of 0.290. Also, the average solution values match the best known solution values for all instances in group *datA*, and for 97.50% of the instances in group *datB*. On four subgroups (*datA*2, *datA*3, *datA*4 and *datB*4), the initial solution also matches the best known solution, while in the others, the ILS was able to improve the solution values in up to 17.40% of instances.

Despite the short ILS running time for the first two groups of this set, it is evident that the running time increases according to the input size. Nevertheless, the running time is still acceptable in practice and is justified by the quality of the generated solutions. Furthermore, the initial solution quality is also remarkable. Although it is a greedy method, BFS-SSP was able to match the best known solution values for 37.50% (60 out of 160) of the instances, including new best values, in a maximum running time of 0.26 s.

The reported data for this third set of instances supports the claims that BFS-SSP generates high-quality solutions and that the proposed ILS outperforms the current state-of-the-art method for solving the SSP, consistently and robustly generating high-quality solutions such that the best known results are matched or supplanted.

4.4. Initial solution sensitivity

One of the key aspects of good ILS implementations is that they must not be sensitive to the quality of initial solutions, i.e., the method must be robust enough to converge even from bad starting points. In order to assess the influence of the BFS-SSP heuristic on the ILS results and judge the effectiveness of our ILS implementation, we ran the metaheuristic over solutions generated randomly. Except for this aspect, the standards of the previous experiments were retained. Table 15 presents a summary of the results obtained in this experiment. In this table, the values reported in columns *gap* and *T* are related to the results generated by the original ILS implementation.

The results show that the ILS is indeed robust as it was able to converge from randomly generated solutions and achieve solutions close to those of the original method. Nonetheless, the convergence took longer (approximately 123.78% of the original running time), confirming the importance of the high-quality solutions generated by the BFS-SSP heuristic, as it also helps to speed the overall method.

5. Conclusions and future work

The Job Sequencing and Tool Switching Problem (SSP) is a combinatorial problem of practical interest, resulting from its direct industrial applicability in flexible manufacturing systems. The Sequencing Problem (SP) is an $\mathcal{NP}$-Hard subproblem of the SSP that seeks job sequences that minimize production line interruptions for loading tools into flexible machines. This work proposed a graph representation for the SSP, a constructive heuristic method, and a local search based on grouping of 1-blocks. These methods

Table 14
Results for the instances introduced by Catanzaro et al. (2015).

Group	n	m	C	i	Proposed ILS					BKS
					S_0	S^*	S	T	σ	
*datA*1	10	10	4	10	12.70	**12.50**	**12.50**	0.09	0.000	**12.50**
*datA*2	10	10	5	10	**10.80**	**10.80**	**10.80**	0.09	0.000	**10.80**
*datA*3	10	10	6	10	**10.10**	**10.10**	**10.10**	0.05	0.000	**10.10**
*datA*4	10	10	7	10	**10.00**	**10.00**	**10.00**	0.04	0.000	**10.00**
*datB*1	15	20	6	10	29.20	**26.50**	**26.50**	1.09	0.000	26.90
*datB*2	15	20	8	10	22.60	**21.70**	**21.70**	1.38	0.000	22.00
*datB*3	15	20	10	10	20.10	**19.70**	**19.70**	1.121	0.000	19.80
*datB*4	15	20	12	10	**19.20**	**19.20**	**19.20**	0.714	0.000	**19.20**
*datC*1	30	40	15	10	119.90	**98.90**	99.09	99.471	0.150	102.00
*datC*2	30	40	17	10	101.10	**82.50**	82.82	137.084	0.184	85.90
*datC*3	30	40	20	10	79.70	**66.60**	66.78	172.586	0.112	69.40
*datC*4	30	40	25	10	58.20	**51.30**	51.47	137.789	0.104	53.60
*datD*1	40	60	20	10	239.20	**198.00**	198.36	488.661	0.268	203.20
*datD*2	40	60	22	10	211.40	**173.60**	174.05	706.126	0.223	179.00
*datD*3	40	60	25	10	178.20	**146.20**	146.68	1,069.98	0.240	152.50
*datD*4	40	60	30	10	140.30	**115.20**	115.42	1,518.80	0.199	120.90

Table 15
Results for all sets of instances with random initial solutions.

set	Random initial solutions ILS				
	S^*	S	σ	*gap* (%)	T (%)
Yanasse et al. (2009)	**23.40**	**23.40**	0.004	0.00	+43.20
Crama et al. (1994)	**67.66**	67.85	0.106	0.00	+24.47
Catanzaro et al. (2015)	66.43	6.57	0.092	0.02	+3.67

were embedded in an Iterated Local Search implementation together with the known swap method, to be used as a perturbation and also as a local search mechanism.

The computational experiments considered 1510 instances from different authors, and compared the proposed constructive heuristic (BFS-SSP), the proposed Iterated Local Search (ILS) implementation, and the current state-of-the-art, represented by the Clustering Search and Biased Random-Key Genetic Algorithm (CS+BRKGA). The BFS-SSP was able to generate good initial solutions and could match the best known solutions in 52.90% of the considered instances in negligible running times. Both ILS and CS+BRKGA were able to achieve the optimal solutions in a small running time for many of the instances; however, those instances were considered to be trivial. For more complex instances, the gap between the two methods became apparent. The ILS was able to establish new best solutions for three different groups of instances and proved to be more robust than the current state-of-the-art CS+BRKGA.

In a general manner, the proposed ILS consistently outperformed the state-of-the-art method for solving the SSP. Although the running time considerably increased on the largest instances, those times are still acceptable in practice, while completely justified by the solution quality. We believe the good performance of the proposed methods is due to *i*) the perspective that the problem was addressed by the constructive heuristic, and *ii*) the tailored initial solution and local search methods, which perform effective operations that increase solution quality.

Future works will focus on experimenting with different neighborhood structures and speeding up the proposed local search, for example by the use of techniques such as parallelism and distributed computing, which would allow a greater exploration of the search space. This work may also be extended to consider the SSP on identical parallel machines, which adds the decision on assigning jobs to machines, the dimension of job processing times, and the objective of minimizing the makespan. Another interesting extension of the SSP is to consider flexible machines in a flow shop environment, where on the same sequence, m machines process each of n jobs with tooling constraints, sequence-dependent setup times and processing times, also with the objective of minimizing the makespan.

Acknowledgements

This research was supported by the National Counsel of Technological and Scientific Development (Conselho Nacional de Desenvolvimento Científico e Tecnológico, CNPq) and the Minas Gerais State Agency for Research and Development (Fundação de Amparo à Pesquisa do Estado de Minas Gerais, FAPEMIG).

References

Adjiashvili, D., Bosio, S., Zemmer, K., 2015. Minimizing the number of switch instances on a flexible machine in polynomial time. Oper. Res. Lett. 43 (3), 317–322. doi:10.1016/j.orl.2015.04.001.

Al-Fawzan, M.A., Al-Sultan, K.S., 2003. A tabu search based algorithm for minimizing the number of tool switches on a flexible machine. Comput. Ind. Eng. 44 (1), 35–47. doi:10.1016/S0360-8352(02)00183-3.

Amaya, J.E., Cotta, C., Fernández, A.J., 2008. A memetic algorithm for the tool switching problem. In: Blesa, M.J., Blum, C., Cotta, C., Fernández, A.J., Gallardo, J.E., Roli, A., Sampels, M. (Eds.), Hybrid Metaheuristics. In: Lecture Notes in Computer Science, 5296. Springer, pp. 190–202.

Amaya, J.E., Cotta, C., Fernández-Leiva, A.J., 2011. Memetic cooperative models for the tool switching problem. Memet. Comput. 3 (3), 199–216. doi:10.1007/s12293-011-0059-6.

Amaya, J.E., Cotta, C., Fernández-Leiva, A.J., 2013. Cross entropy-based memetic algorithms: an application study over the tool switching problem. Int. J. Comput. Intell. Syst. 6 (3), 559–584. doi:10.1080/18756891.2013.792542.

Avci, S., Akturk, M.S., 1996. Tool magazine arrangement and operations sequencing on cnc machines. Comput. Oper. Res. 23 (11), 1069–1081.

Bard, J.F., 1988. A heuristic for minimizing the number of tool switches on a flexible machine. IIE Trans. 20 (4), 382–391. doi:10.1080/07408178808966195.

Beezão, A.C., Cordeau, J.-F., Laporte, G., Yanasse, H.H., 2017. Scheduling identical parallel machines with tooling constraints. Eur. J. Oper. Res. 257 (3), 834–844. doi:10.1016/j.ejor.2016.08.008.

Burger, A.P., Jacobs, C.G., van Vuuren, J.H., Visagie, S.E., 2015. Scheduling multicolour print jobs with sequence-dependent setup times. J. Sched. 18 (2), 131–145. doi:10.1007/s10951-014-0400-2.

Carvalho, M.A.M., Soma, N.Y., 2015. A breadth-first search applied to the minimization of the open stacks. J. Oper. Res. Soc. 66 (6), 936–946. doi:10.1057/jors.2014.60.

Catanzaro, D., Gouveia, L., Labbé, M., 2015. Improved integer linear programming formulations for the job sequencing and tool switching problem. Eur. J. Oper. Res. 244 (3), 766–777. doi:10.1016/j.ejor.2015.02.018.

Chaves, A.A., Lorena, L.A.N., Senne, E.L.F., Resende, M.G.C., 2016. Hybrid method with CS and BRKGA applied to the minimization of tool switches problem. Comput. Oper. Res. 67, 174–183. doi:10.1016/j.cor.2015.10.009.

Chaves, A.A., Yanasse, H.H., Senne, E.L.F., 2012. Uma nova heurística para o problema de minimização de trocas de ferramentas. Gest. Prod. 19 (1), 17–30. doi:10.1590/S0104-530X2012000100002.

Crama, Y., 1997. Combinatorial optimization models for production scheduling in automated manufacturing systems. Eur. J. Oper. Res. 99 (1), 136–153.

Crama, Y., Kolen, A.W.J., Oerlemans, A.G., Spieksma, F.C.R., 1994. Minimizing the number of tool switches on a flexible machine. Int. J. Flex. Manuf. Syst. 6 (1), 33–54. doi:10.1007/BF01324874.

Crama, Y., Moonen, L.S., Spieksma, F.C., Talloen, E., 2007. The tool switching problem revisited. Eur. J. Oper. Res. 182 (2), 952–957. doi:10.1016/j.ejor.2006.07.028.

Denizel, M., 2003. Minimization of the number of tool magazine setups on automated machines: a lagrangean decomposition approach. Oper. Res. 51 (2), 309–320.

Djellab, H., Djellab, K., Gourgand, M., 2000. A new heuristic based on a hypergraph representation for the tool switching problem. Int. J. Prod. Econ. 64 (1), 165–176.

Fathi, Y., Barnette, K.W., 2002. Heuristic procedures for the parallel machine problem with tool switches. Int. J. Prod. Res. 40 (1), 151–164. doi:10.1080/00207540110076115.

Furrer, M., Mütze, T., 2017. An algorithmic framework for tool switching problems with multiple objectives. Eur. J. Oper. Res. 259 (3), 1003–1016.

Gendreau, M., Hertz, A., Laporte, G., 1992. New insertion and postoptimization procedures for the traveling salesman problem. Oper. Res. 40 (6), 1086–1094.

Ghiani, G., Grieco, A., Guerriero, E., 2007. An exact solution to the tlp problem in an nc machine. Robot. Comput. Integr. Manuf. 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 55 (4), 379–385. doi:10.1002/net.20341.

Ghrayeb, O.A., Phojanamongkolkij, N., Finch, P.R., 2003. A mathematical model and heuristic procedure to schedule printed circuit packs on sequencers. Int. J. Prod. Res. 41 (16), 3849–3860.

Gray, A.E., Seidmann, A., Stecke, K.E., 1993. A synthesis of decision models for tool management in automated manufacturing. Manag. Sci. 39 (5), 549–567.

Hertz, A., Laporte, G., Mittaz, M., Stecke, K.E., 1998. Heuristics for minimizing tool switches when scheduling part types on a flexible machine. IIE Trans. 30 (8), 689–694. doi:10.1080/07408179808966514.

Hertz, A., Widmer, M., 1996. An improved tabu search approach for solving the job shop scheduling problem with tooling constraints. Discret. Appl. Math. 65 (1–3), 319–345.

Konak, A., Kulturel-Konak, S., 2007. An ant colony optimization approach to the minimum tool switching instant problem in flexible manufacturing system. In: Proceedings of IEEE Symposium on Computational Intelligence in Scheduling, SCIS'07. IEEE, pp. 43–48.

Konak, A., Kulturel-Konak, S., Azizoğlu, M., 2008. Minimizing the number of tool switching instants in flexible manufacturing systems. Int. J. Prod. Econ. 116 (2), 298–307.

Laporte, G., Salazar-Gonzalez, J.J., Semet, F., 2004. Exact algorithms for the job sequencing and tool switching problem. IIE Trans. 36 (1), 37–45. doi:10.1080/07408170490257871.

Lourenço, H.R., Martin, O.C., Stützle, T., 2003. Iterated local search. In: Glover, F., Kochenberger, G.A. (Eds.), Handbook of Metaheuristics. Springer, Boston, MA, US, pp. 320–353. doi:10.1007/0-306-48056-5_11.

López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L.P., Birattari, M., Stützle, T., 2016. The irace package: iterated racing for automatic algorithm configuration. Oper. Res. Perspect. 3, 43–58. doi:10.1016/j.orp.2016.09.002.

Marvizadeh, S.Z., Choobineh, F.F., 2013. Reducing the number of setups for CNC punch presses. Omega 41 (2), 226–235. doi:10.1016/j.omega.2012.06.001.

Matzliach, B., 1998. The online tool switching problem with non-uniform tool size. Int. J. Prod. Res. 36 (12), 3407–3420. doi:10.1080/002075498192120.

McGeoch, L.A., Sleator, D.D., 1991. A strongly competitive randomized paging algorithm. Algorithmica 6 (1), 816–825. doi:10.1007/BF01759073.

Privault, C., Finke, G., 1995. Modelling a tool switching problem on a single NC-machine. J. Intell. Manuf. 6 (2), 87–94.

Privault, C., Finke, G., 2000. K-server problems with bulk requests: an application to tool switching in manufacturing. Ann. Oper. Res. 96 (1–4), 255–269.

Raduly-Baka, C., Nevalainen, O.S., 2015. The modular tool switching problem. Eur. J. Oper. Res. 242 (1), 100–106. doi:10.1016/j.ejor.2014.09.052.

Rey, D., Neuhäuser, M., 2011. Wilcoxon-signed-rank test. In: Lovric, M. (Ed.), International Encyclopedia of Statistical Science. Springer, Berlin, Heidelberg, pp. 1658–1659. doi:10.1007/978-3-642-04898-2_616.

Salonen, K., Raduly-Baka, C., Nevalainen, O.S., 2006. A note on the tool switching problem of a flexible machine. Comput. Ind. Eng. 50 (4), 458–465.

Senne, E. L. F. U., Yanasse, H. H., 2009. Beam search algorithms for minimizing tool switches on a flexible manufacturing system. Proceedings of the 11th Wseas International Conference on Mathematical and Computational Methods In Science and Engineering, MACMESE '09, 68–72.

Shapiro, S.S., Wilk, M.B., 1965. An analysis of variance test for normality. Biometrika 52 (3), 591–611.

Shirazi, R., Frizelle, G.D.M., 2001. Minimizing the number of tool switches on a flexible machine: an empirical study. Int. J. Prod. Res. 39 (15), 3547–3560. doi:10.1080/00207540110060888.

Song, C.-Y., Hwang, H., 2002. Optimal tooling policy for a tool switching problem of a flexible machine with automatic tool transporter. Int. J. Prod. Res. 40 (4), 873–883.

Tang, C.S., Denardo, E.V., 1988. Models arising from a flexible manufacturing machine, part i: minimization of the number of tool switches. Oper. Res. 36 (5), 767–777. doi:10.1287/opre.36.5.767.

Tzur, M., Altman, A., 2004. Minimization of tool switches for a flexible manufacturing machine with slot assignment of different tool sizes. IIE Trans. 36 (2), 95–110.

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. Gest. Prod. 16 (3), 370–381. doi:10.1590/S0104-530X2009000300005.

Zeballos, L., 2010. A constraint programming approach to tool allocation and production scheduling in flexible manufacturing systems. Robot. Comput. Integr. Manuf. 26 (6), 725–743.

Zhou, B.-H., Xi, L.-F., Cao, Y.-S., 2005. A beam-search-based algorithm for the tool switching problem on a flexible machine. Int. J. Adv. Manuf. Technol. 25 (9–10), 876–882. doi:10.1007/s00170-003-1925-2.