Research Article GU J Sci 35 (1): 113-130 (2022) DOI: 10.35378/gujs.681151

Gazi University

Journal of Science

http://dergipark.gov.tr/gujs

Minimization of Number of Tool Switching Instants in Automated Manufacturing Systems

Burak GOKGUR[1,*], Selin OZPEYNIRCI[2]

[1]*Sabancı Business School, Sabancı University, Orta Mahalle, Tuzla 34956, İstanbul, Turkey*
[2]*Department of Industrial Engineering, İzmir University of Economics, İzmir, Turkey*

Highlights

- We study the problem of minimizing tool switching instants in automated manufacturing systems.
- We provide a mathematical programming model and two constraint programming models for the problem.
- The proposed solution approaches are promising in terms of solution quality.

Article Info

Received: 29 Jan 2020
Accepted: 28 Feb 2021

Keywords

Tool switching instants
Job grouping problem
Mathematical model
Heuristic algorithm
Constraint programming

Abstract

This study addresses the problem of minimizing tool switching instants in automated manufacturing systems. There exist a single machine and a group of jobs to be processed on it. Each job requires a set of tools, and due to limited tool magazine capacity, and because it is not possible to load all available tools on the machine, tools must be switched. The ultimate goal, in this framework, is to minimize the total number of tool switching instants. We provide a mathematical programming model and two constraint programming models for the problem. Because the problem is proven to be NP-hard, we develop two heuristic approaches, and compare their performance with methods described in the literature. Our analysis indicates that our constraint programming models perform relatively well in solution quality and execution time in small-sized problem instances. The performance of our greedy approach shows potential, reaching the optimal solution in 82.5% of instances. We also statistically demonstrate that the search algorithm enhances the quality of the solution obtained by the greedy heuristic, particularly in large sets. Hence, the solution approach, i.e., the greedy heuristic and the search algorithm proposed in this study is able to quickly reach near-optimal solutions, showing that the method is appropriate for manufacturing settings requiring sudden adjustments.

1. INTRODUCTION

Advances in information technologies and data analytics led to the emergence of smart factories, which include highly flexible and automated manufacturing operations by incorporating real-time data into their system. Despite the developments of newly designed systems, such as re-configurable manufacturing system (RMS), companies still need to adapt automated manufacturing systems to deal with large product variety and complex system requirements.

In automated manufacturing systems, each operation demands a set of various tools to be loaded on a machine. Each numerically controlled machine has its own tool magazine but because of economic restrictions, the capacity of these is less than the total number of tool slots needed in all operations. Thus, it is not possible to avoid tool switching between the processing of different jobs. Ineffective management of tool operations may result in underutilized resources, due to the increase in machine idle time (Shirazi and Frizelle [1]). Calmels [2] discusses the impact of the roles of tool operations in automated manufacturing systems.

Tool switching problem is one of the key tenets of the automated manufacturing system, and its importance has been also recognized in academia. Crama [3] mentions tool switching problems in their study on problems faced in automated manufacturing systems and solution approaches. Tang and Denardo [4]

*Corresponding author, e-mail: burak.gokgur@sabanciuniv.edu

assume that the necessary time for tool change is constant and minimize the number of job groups to be formed. They develop lower and upper bounds on the optimal solution and use these for branch and bound technique. Song and Hwang [5] aim to minimize the number of movements of tool carrier and develop an optimal strategy for a given job sequence. Denizel [6] aims to minimize the required number of slots and uses Lagrangean relaxation method to develop lower and upper bounds for a branch and bound algorithm. Konak et al. [7] develop two tabu search heuristics with the aim of minimizing tool switching instants on a single machine. Konak and Kulturel-Konak [8] develop an ant colony solution algorithm for the same problem. Adjiashvili et al. [9] develop a polynomial time algorithm to minimize the number of tool switching instances in a system incorporating a flexible machine. Baykasoğlu and Ozsoydan [10- 12] study the problem of minimizing non-machining times (i.e., tool switching and indexing times) in automatic machining centers. Their novel methodology employs shortest path algorithm and two metaheuristics to determine index positions where duplication of cutting tools is allowed [10]. In a later study, they propose a simulated annealing approach to minimize tool switching and indexing times [11]. They also consider the problem under dynamic operating conditions and propose a simulated annealing algorithm with the objective of minimizing the makespan [12]. Chaves et al. [13] develop a hybrid heuristic approach for the problem of minimizing the number of tool switches. Furrer and Mütze [14] develop a branch-and-bound based algorithm which aims to minimize simultaneously the number of tool switches and machine stops over time. Marvizadeh and Choobineh [15] develop three heuristics for the problem of minimizing number of setups and compare the solution performance of the three heuristics by conducting a computational experiment. Paiva and Carvalho [16] propose an iterated local search metaheuristic based on a new graph representation for the job sequencing and tool switching problem. Dang et al. [17] focus on the scheduling problem of a group of jobs on identical parallel machines, where each job needs a different set of tools to be processed, and they develop a mathematical model for the problem. They also propose a matheuristic technique that combines a genetic algorithm and an integer linear programming model in order to solve industry-sized problems at scale. They observe that the matheuristic surpasses the mathematical model in solution quality. Atta et al. [18] develop a harmony search algorithm for the tool indexing problem, and report that the proposed search algorithm exhibits a promising performance. For the job sequencing and tool switching problem, da Silva et al. [19] propose a new mathematical model in the multicommodity flow framework and observe that it performs better in both execution time and the optimality gap.

Several studies in the literature consider practical job grouping problems. Smed et al. [20] develop an integer programming model and heuristic approach for job grouping problem in a production line, significantly improving the system. Yuan et al. [21] study planning of container terminals to minimize the makespan. Jans and Desrosiers [22] propose a new formulation for the job grouping problem to eliminate the symmetry between the identical machines. They also develop different symmetry breaking constraints, such as variable reduction and lexicographic ordering constraints, and conclude that the proposed formulations outperform the existing approaches in the literature. Dadashi et al. [23] consider a new version of tool switching problem, with tool life that may be faced in practice, and propose a genetic algorithm approach.

In this study, we provide a mathematical formulation and two constraint programming models which yield good solution quality for the problem. Because the problem of interest is proven to be NP-Hard by Konak et al. [7], we develop an easily-implementable heuristic algorithm with the goal of obtaining good solutions in large problem sets. Then, we compare the proposed algorithm with Konak et al's [7] approach. For small sized problem instances, the proposed heuristic approach produces optimal or near-optimal solutions in very short computational times, and for a majority of large instances, our algorithm performs better than that of Konak et al. [7].

The outline of the paper is as follows: In Section 2, we provide the definition of the problem setting and the mathematical model. In Section 3, constraint programming approach is discussed, and two models developed for minimizing tool switching instants are presented. Sections 4 and 5 give the heuristic algorithm and search algorithm that aims to improve the heuristic solution, respectively. In Sections 2-5, the computational experiments on the corresponding method are reported. We conclude the paper with future research directions in Section 6.

2. PROBLEM DEFINITION AND MATHEMATICAL MODEL

In this section, we first visit the problem environment and explain the notation used. We then present the mathematical model for job grouping problem. Suppose there are n jobs to be processed on a single machine. There exists t types of tools and the tools in set $l(i)$ must be installed on the machine in order to process job i. We assume that every tool holds one slot, and that the number of tools demanded by each job cannot be greater than the tool magazine capacity. Finally, the tool magazine of the machine can hold c tools.

Indices:
i: Job index, i=1,2,...,n
k: Tool type index, k=1,2,...,t
g: Group index, g=1,2,...,s

Parameters:
$l(i)$: Set of tools required by job i
c: Tool magazine capacity

Decision variables
X_{ig}: 1, if job i is assigned to group g; 0, otherwise
Y_{kg}: 1, if tool type k is assigned to group g; 0, otherwise
Z_g: 1, if group g is created; 0, otherwise

The mathematical model (JG) is given below:

$$\text{(JG) } Minimize \sum_{g=1}^{s} Z_g \tag{1}$$

s.t.

$$\sum_{g=1}^{s} X_{ig} = 1, \ \forall i \tag{2}$$

$$Z_g \geq X_{ig}, \ \forall i, g \tag{3}$$

$$X_{ig} \leq Y_{kg}, \forall i, g, k \in l(i) \tag{4}$$

$$\sum_{k=1}^{t} Y_{kg} \leq c, \ \forall g \tag{5}$$

$$X_{ig} \in \{0,1\}, \ \forall i, g \tag{6}$$

$$Y_{kg} \in \{0,1\}, \ \forall k, g \tag{7}$$

$$Z_g \in \{0,1\}, \ \forall g. \tag{8}$$

The objective (1) is to minimize the number of groups so that the total number of tool switching instants is minimized. Constraint set (2) ensures that every job is assigned to exactly one group. Constraint set (3) guarantees that a job can be assigned to a group only if that group is created. Also, a job can be assigned to a group only if required tools are assigned to that group (4). Constraint set (5) prevents the number of tools required by a group from exceeding the tool magazine capacity. Finally, (6-8) are the set constraints. Next, we provide an illustrative example to elaborate on the problem's dynamics.

2.1. An Illustrative Example

To elucidate the problem environment, this section provides an illustrative example, followed by the presentation of the optimal solution.

Consider an automated manufacturing system in which there are ten jobs to be processed on a single machine with tool magazine capacity of three tool slots. Suppose that eight different tool types are required to complete the processing of all ten jobs. Tables 1 and 2 present tool requirements of each job and the available number of tool copies, respectively.

Table 1. *Set of tools requirement by each job*

Job	1	2	3	4	5	6	7	8	9	10
Tools	5,6	1	1	6,8	6,8	3,4	1	1,4	8	7

Table 2. *Number of tool copies*

Tool	1	2	3	4	5	6	7	8
Number of copies	1	2	2	1	2	2	2	1

The optimal number of job groups equals 3; that is, ten jobs are clustered into three different groups based on the tool types required by each to be processed. The optimal grouping structure of the operations is presented in Table 3.

Table 3. *The optimal grouping structure of the illustrative example*

Job	1	2	3	4	5	6	7	8	9	10
Group	1	1	1	2	2	3	1	3	2	2

Table 4. *The optimal tool allocation of the illustrative example*

Group	1	2	3
Set of tools	1,5,6	6,7,8	1,3,4

The optimal solution, presented in Table 3, shows that the number of different tool types assigned to each group does not exceed the machine's tool magazine capacity. The corresponding tool allocation is presented in Table 4. For instance, tool types 1, 5 and 6 are assigned to Group 1. The jobs assigned to Group 2 (3) require tool types 6, 7 and 8 (1, 3 and 4) to be processed. Also, in Table 3, we can easily discern that each job is assigned to exactly one group.

In the next section, we provide a numerical study that demonstrates the performance of the mathematical model. Then, we present our proposed solution approaches, i.e., constraint programming and heuristic models, to obtain near-optimal solutions in short computational times.

2.2. Computational Experiments on Mathematical Model

We generate several problem instances of different sizes to apply the solution approaches as given in Table 5. $|l(i)|$ values are drawn from discrete uniform distribution and the tools in set $l(i)$ are generated randomly.

Table 5. *Problem parameters for small instances*

Set	n	t	$\|l(i)\|$	c
1	10	8	U[1,2]	3
2	10	16	U[1,4]	6
3	15	8	U[1,2]	3

Model is coded using Visual C++ and solved using CPLEX 11.2 on a Pentium 4, 3 GHz, 1.96 GB RAM computer. 10 instances are generated with each parameter combination and the solutions are given in Table 6.

Table 6. *Optimal number of job groups and solution times*

Problem Set	Number of job groups	Solution time (sec.)	Problem Set	Number of job groups	Solution time (sec.)
1-1	3	6.74	3-1	4	7.31
1-2	3	4.93	3-2	3	6.73
1-3	3	5.02	3-3	3	7.45
1-4	3	4.89	3-4	4	9.35
1-5	3	5.06	3-5	4	6.02

1-6	2	4.45	3-6	3	7.48
1-7	4	5.17	3-7	3	4.77
1-8	3	4.58	3-8	4	6.82
1-9	3	4.81	3-9	4	6.27
1-10	2	4.19	3-10	3	6.62
2-1	3	23.62	4-1	5	217.93
2-2	3	5.31	4-2	4	9.54
2-3	4	5.11	4-3	5	131.03
2-4	4	11.15	4-4	4	20.00
2-5	3	5.37	4-5	3	7.10
2-6	3	4.77	4-6	4	45.23
2-7	3	5.87	4-7	4	32.58
2-8	3	4.83	4-8	5	11.63
2-9	3	5.99	4-9	5	73.47
2-10	4	4.64	4-10	4	29.06

As seen in Table 6, for small sized problems, the mathematical model reaches the optimal solutions in a very short time, but the solution time increases with problem size.

3. CONSTRAINT PROGRAMMING APPROACH

Constraint programming approach is a subfield of artificial intelligence and converts the characteristics of a combinatorial problem to the properties of constraint satisfaction problem. Constraint programming approach is often preferred, especially for modeling and solving scheduling problems because efficient constraint-based scheduling algorithms can quickly reach the optimal solution and prove the optimality. Constraint programming algorithms and modeling techniques are also used for problems other than scheduling. However, in these problems, although the optimal solution can be found quickly, the model cannot prove its optimality. In this study, the constraint programming model uses IBM ILOG CP Optimizer 2.3 software, in which definitions are given for many constraint programming algorithms, and resource constraints required for modeling.

In the literature of tool management and operation scheduling, there are several studies on constraint programming approach for solving problems, such as scheduling of flexible manufacturing systems with machine and tool limitations (Zeballos [24], Zeballos et al. [25]), scheduling with machine eligibility constraints (Edis and Ozkarahan [26]), integrated view of scheduling problems in a flexible manufacturing setting (Novas and Henning [27]), and parallel machine scheduling with tool loading (Gökgür et al. [28]). In the subsequent section, we present a detailed view of the constraint programming models developed for the job grouping problem.

3.1. Constraint Programming Models

In the formulation of the mathematical model provided in Section 2, we observe that it is possible to compactly formulate the problem of the assignment of jobs to groups, along with the required set of tools. For this aim, we provide two different constraint programming models.

One way to model the assignment of jobs and the corresponding set of required tools to the same group is changing the structure of the variable X, as shown in constraint set (13), whose domain is the indices of the available group numbers. $X_i = g$ if and only if job i is assigned to group g. Then, logical constraints (11) state that if job i is assigned to group g its required set of tools must be loaded to the same group. The ultimate goal of the constraint programming model JG-CP1 is to understand the impact of eliminating redundancies in the assignment of jobs to groups. The developed constraint programming model (JG-CP1) is given below:

$$\text{(JG-CP1) } \textit{Minimize } \sum_{g=1}^{s} Z_g \tag{9}$$

s.t.

$$Z_{X_i} = 1, \ \forall i \quad (10)$$

$$(X_i = g) \rightarrow (Y_{kg} = 1), \ \forall i, g, k | k \in l(i) \quad (11)$$

$$\sum_{k=1}^{t} Y_{kg} \leq c, \ \forall g \quad (12)$$

$$X_i \in \{1, \dots, s\}, \ \forall i \quad (13)$$

$$Y_{kg} \in \{0,1\}, \ \forall k, g \quad (14)$$

$$Z_g \in \{0,1\}, \ \forall g. \quad (15)$$

The objective function (9) minimizes the total number of groups. Constraint (10) ensures that a group is opened when a job is assigned to it. Constraint (11) forces a job and its required set of tools to be present on the same group. In constraint (11), we use a logical type of constraint to ensure that both a job and its required tool set are assigned to the same group. Constraint (12) states that number of tools assigned to a group cannot be greater than its capacity. Constraints (13), (14) and (15) are the set constraints.

The observation made from the problem is that group numbers do not impact the solution, i.e., groups are indistinguishable. In the mathematical model provided in Section 2, we differentiate between the group numbers, bringing additional complexity to the problem. Therefore, eliminating the symmetry among group numbers may accelerate the search and improve the solution quality.

There are two different types of symmetry in constraint programming: solution symmetry and problem symmetry. Solution symmetry can be defined as the permutation of pairs of variable and value which keeps the set of solutions; problem symmetry, on the other hand, is a permutation of pairs of variable and value which maintains the set of constraints. The special cases of these two types of symmetry are called variable and value symmetry. Variable symmetry indicates that the variables in the variable set are freely replaced, and the value symmetry is the free replacement of values. Different methods have been developed to break symmetries in problems; three of these are reformulation, static symmetry breaking, and dynamic symmetry breaking (Gökgür et al. [28]). In this study, we lexicographically order the columns using the global *lex* constraint to avoid redundancies in the presence of row/column symmetry. Hence, we incorporate a new constraint set, i.e., constraint set (22), into the second constraint programming model to lexicographically order the group numbers in grouping the jobs. The second constraint programming model with global *lex* constraint, (JG-CP2), is given below:

(JG-CP2) *Minimize* $\sum_{g=1}^{s} Z_g$ (16)

s.t.

$$Z_g \leq 1, \ \forall g \quad (17)$$

$$Z_g \geq X_{ig}, \ \forall g, i \quad (18)$$

$$\sum_{g=1}^{s} X_{ig} = 1, \ \forall i \quad (19)$$

$$X_{ig} \leq Y_{kg}, \ \forall i, g, k | k \in l(i) \quad (20)$$

$$\sum_{k=1}^{t} Y_{kg} \leq c, \ \forall g \quad (21)$$

$$lex\left(\sum_{g=1}^{s-1} X_{ig}, \sum_{g=1}^{s-1} X_{ig+1}\right), \forall i \quad (22)$$

$$X_{ig} \in \{0,1\}, \ \forall i, g \quad (23)$$

$$Y_{kg} \in \{0,1\}, \ \forall k, g \quad (24)$$

$$Z_g \in \{0,1\}, \ \forall g. \quad (25)$$

The objective function (16) minimizes the total number of groups. Constraints (17) and (18) state that a group is not opened unless a job is assigned to the group. Constraint (19) ensures that a job can only be assigned to exactly one group. Constraint (20) ensures that a job and its required tool set to be processed are assigned to the same group. Constraint (21) is a tool magazine capacity constraint which states that number of tools assigned to a group cannot be greater than its capacity. In constraint (22), we lexicographically order group indexes using *lex* global constraint (readers interested in propagation algorithms for lexicographic ordering are referred to Frisch et al. [29]). Constraints (23), (24) and (25) are the set constraints.

We now turn our attention to the illustrative example we presented in Section 2.1 to elaborate on the dynamics of the constraint programming models we introduced in this section, namely JG-CP1 and JG-CP2. Tables 7-8 and 9-10 demonstrate the optimal solutions achieved by the constraint programming models JG-CP1 and JG-CP2, respectively.

Table 7*. Results of the illustrative example achieved by JG-CP1*

Job	1	2	3	4	5	6	7	8	9	10
Group	1	1	1	2	2	3	1	3	2	2

Table 8*. The optimal tool allocation of the illustrative example*

Group	1	2	3
Set of tools	1,5,6	6,7,8	1,3,4

Table 9*. Results of the illustrative example achieved by JG-CP2*

Job	1	2	3	4	5	6	7	8	9	10
Group	1	1	1	2	2	3	1	3	2	2

Table 10*. The optimal tool allocation of the illustrative example*

Group	1	2	3
Set of tools	1,5,6	6,7,8	1,3,4

In the first constraint programming model, i.e., JG-CP1, Equations (10), (11), and (12) mainly preserve the feasibility of the solution produced by the model. Constraint (10) ensures that an additional group is not created unless a job remains unassigned to any of the existing groups. Table 7 shows that the optimal number of groups equals three, and each job is assigned to one of these three groups, based on the set of different tool types they require. Besides, constraint (12) provides that the number of different tool types assigned to each group does not exceed the machine's tool magazine capacity, which is three in this example. When we observe the set of different tool types assigned to each group from Table 8, we can see that constraint (11) ensures that a job and its required set of different tool types are present in the same group.

In the JG-CP2 model, Equations (17), (18), (19), (20), and (21) ensure the feasibility of the solution to be produced. On the other hand, constraint (22) prevents the model from exploring the equivalent regions of the feasible search space. Equations (17) and (18) ensure that a group is created if a job is assigned to that group. Table 9 indicates that there are only three groups that all ten jobs are assigned to. Table 9 demonstrates that each job is assigned to exactly one group, which is satisfied by Constraint (19). Also, we can observe from Table 10 that constraints (20) and (21) guarantee, respectively, that each job and its required set of tool types are assigned to the same group, and that the number of different tool types assigned to each group is less than or equal to the machine's tool magazine capacity.

3.2. Computational Experiments on Constraint Programming Models

In this subsection, results of computational experiments designed to compare two constraint programming models are reported. Small sized problem instances given in Table 5 are used. A time limit of 1 hour is set, and the execution of the algorithm is terminated if optimal solution is not found in this duration. Also, the restart option is chosen as the search type for each CP model. Both models reach the optimal solution in all instances within the specified time limit. Table 11 gives the average computation time in seconds for the two CP models.

Table 11*. Average solution times (in seconds) of constraint programming models*

Set	JG-CP1	JG-CP2
1	3.49	3.03
2	5.47	4.38

3	15.28	18.62
4	1063.00	1309.00

According to Table 11, there is no significant difference between the performances of the two CP models. Although each constraint programming model was developed with different techniques, both perform reasonably well in small sized problem instances and find the optimal solutions in all instances within the specified time limit. However, it can be readily noticed from Table 11 that the solution time increases as the problem size increases in complexity in terms of the number of jobs and of different tool types.

Hence, it is apparent that the computation overhead may be extended with an increase in the number of jobs and tool types, including toolkits, i.e., symmetry breaking constraints aimed at eliminating symmetries in the problem (Flener et al. [30]). Therefore, a need emerges for a solution approach which yields promising solutions within reasonable execution times, especially in large-sized problem instances.

4. HEURISTIC APPROACH

This section presents a greedy approach that starts with computing the number of common tools required by each pair of jobs. The pair with the highest number of common tools is then selected to form a group, if the total number of tools does not exceed the tool magazine capacity. The job with the largest number of common tools with these jobs is added to the group, if feasible. After the tool magazine is full, any jobs that do not require additional tools are added to the group. A new group is created when no more jobs can be added to the existing group. The procedure continues until all jobs are assigned to a group.

In this section, we first explain the steps of the heuristic algorithm and then discuss its performance.

4.1. Steps of the Heuristic Approach

The parameters used in the heuristic algorithm are defined below:

D: Set of jobs that have not been assigned to a group yet

U_{ij}: Number of common tools of jobs i and j in set D

w_g: Set of tools required by the jobs in group g

A_g: Set of jobs assigned to group g

The steps of heuristic algorithm are explained below:

Step 0. Set $D = \{1, 2, ..., n\}$, $A_g = \emptyset$ and $w_g = \emptyset$ for all g.

Step 1. Find U_{ij} values for all (i, j) pairs in D such that $U_{ij} = |l(i) \cap l(j)|$, $\forall i, j \in D$. Set $g = 1$.

Step 2. Find jobs $i' \in D$ and $j' \in D$ such that $Max\{U_{ij}\} = U_{i'j'}$.

2.a. If all $U_{i'j'}$ values are equal to 0, find a job $p \in D$ that requires the maximum number of tools. Remove job p from set D and add to set A_g. Set $w_g = l(p)$ and go to Step 3.

2.b. If $|l(i') \cap l(j')| > c$, set $U_{i'j'} = 0$ and repeat Step 2.

2.c. If $|l(i') \cap l(j')| \leq c$, assign jobs i' and j' to group g. Remove them from set D and add to set A_g. Set $w_g = l(i') \cup l(j')$.

If $|w_g| < c$, go to Step 3. If $|w_g| = c$, go to Step 5.

Step 3. Find a job $k \in D$ such that the number of common tools of job k with the jobs in set A_g is maximum, i.e., $Max_{i \in D}\{|l(i) \cap w(g)|\} = |l(k) \cap w(g)|$.

3.a. If $|l(k) \cup w(g)| = 0$, go to Step 4.

3.b. If $|l(k) \cup w(g)| > c$, repeat Step 3 without job k.

3.c. If $|w_g \cup l(k)| \leq c$, assign job k to group g. Remove it from set D and add to set A_g. Set $w_g = w_g \cup l(k)$.

If $|w_g| < c$, repeat Step 3. If $|w_g| = c$, go to Step 5.

Step 4. Find the unused tool magazine capacity for group g, $c - |w_g|$. Find a job $q \in D$ such that $|l(q)| \leq c - |w_g|$. Remove job q from set D and add to set A_g. Set $w_g = w_g \cup l(q)$. Repeat Step 3. If no such q exists, go to Step 6.

Step 5. Find a job $h \in D$ such that $l(h) \subseteq w_g$. Assign job h to group g. Remove it from set D and add to set A_g. Continue until no more jobs can be assigned to group g. Go to Step 6.

Step 6. If $D = \emptyset$, stop. Otherwise, set $g = g + 1$ and go to Step 2.

Step 1 requires computation for each pair of jobs that can be shown to be $O(n^2)$. Step 2 requires finding the largest U_{ij} value, and in the worst case, all pairs may have to be evaluated, which corresponds to $O(n^2)$. This step has to be repeated for every pair, due to the condition indicated in Step 2b. Hence, the computational complexity of this step can be $O(n^4)$ in the worst case. Other steps do not require higher level of computations. As a result, the heuristic runs in $O(n^4)$.

4.2. Computational Experiments on the Heuristic Approach

In order to measure the performance of the heuristic approach, we run experiments on the problem sets given in Table 5 and problem sets developed by Konak et al. [7] given in Table 12. They generated 30 instances for each parameter combination. The algorithm is coded using Visual C++ and solved on an AMD Phenom(tm) II P820, 1.80 GHz, 4 GB RAM computer.

Table 12. *Problem parameters of Konak et al. [7]*

Problem Set	*n*	*t*	*c*
L-I	40	20	15
L-II	50	25	20
L-III	60	30	25
VL-I	120	30	20
VL-II	120	60	20
VL-III	150	30	20
VL-IV	150	60	20
VL-V	180	30	20
VL-VI	180	60	20
VL-VII	210	30	20
VL-VIII	210	60	20

Table 13 gives the number of groups found by the heuristic approach and mathematical model for small problem sets. All instances are solved in less than 1 second. The heuristic approach reaches the optimal solution in 33 of 40 instances (82.5%). In the remaining 7 (17.5%), the heuristic approach forms an additional group. Although it is a myopic approach, the performance on small problem sets is quite satisfactory.

We also apply two sample *t*-test to the differences in means of the number of groups in the optimal solution (μ_O) and heuristic solution (μ_H). For this aim, the following hypothesis is designed:

H_0: $\mu_O - \mu_H = 0$

H_1: $\mu_O - \mu_H \neq 0$

with a 1–α=0.99 confidence. The value of *t* statistic is obtained by the following equation:

$$t_0 = \frac{\bar{x}_O - \bar{x}_H}{S_p\sqrt{\frac{1}{n_O}+\frac{1}{n_H}}}$$

where $\bar{x}_O$ and $\bar{x}_H$ are the sample means for the number of groups found by the mathematical model and heuristic approach, respectively. These parameters are calculated as $\bar{x}_O = 3.5$ and $\bar{x}_H = 3.7$. There are $n_O = n_H = 40$ instances (observations) for small sized problems. s_p^2 is calculated as $s_p^2 = \frac{(n_O-1)s_O^2+(n_H-1)s_H^2}{n_0+n_H-2}$ where the sample variances are calculated as $s_O^2 = 0.564103$ for the optimal solution and $s_H^2 = 0.676923$ for the heuristic solution. Hence, the value of test statistic is $t_0 = -1.135454$. Since $t_{\alpha/2,n_O+n_H-2} = t_{0.005,78} = 2.64034$, and $-2.64034 < t_0 < 2.64034$, the null hypothesis cannot be rejected.

Table 13. *The number of groups found by the mathematical model and the heuristic approach for small problem sets*

Problem Set	Optimal Solution	Heuristic Solution	Problem Set	Optimal Solution	Heuristic Solution
1-1	3	3	3-1	4	4
1-2	3	3	3-2	3	3
1-3	3	3	3-3	3	3
1-4	3	3	3-4	4	4
1-5	3	3	3-5	4	4
1-6	2	3	3-6	3	4
1-7	4	4	3-7	3	3
1-8	3	3	3-8	4	5
1-9	3	3	3-9	4	4
1-10	2	3	3-10	3	3
2-1	3	3	4-1	5	6
2-2	3	3	4-2	4	4
2-3	4	4	4-3	5	5
2-4	4	4	4-4	4	4
2-5	3	3	4-5	3	4
2-6	3	3	4-6	4	4
2-7	3	3	4-7	4	5
2-8	3	3	4-8	5	5
2-9	3	3	4-9	5	5
2-10	4	4	4-10	4	5

The results of the computational experiments conducted to measure the performance of heuristic algorithm on large problem sets generated by Konak et al. [7] are reported in Table 14. The first columns provide the results found by Konak et al. [7] for sets L-I, L-II and L-III. The optimal solution or the upper bound with the gap in parentheses found by the IP model are given for L-I. For L-II and L-III, best solution found after 10 replications of tabu search are given. The second columns report the solutions found by our heuristic approach. As seen in the table, heuristic algorithm gives poor results in large problem sets, which is expected due to its myopic nature.

We employ two sample *t*-test to the differences in means of the number of groups in the tabu search solution (μ_T) and heuristic solution (μ_H) for large problem sets, with the aim of testing the following hypothesis:

H_0: $\mu_T - \mu_H = 0$

H_1: $\mu_T - \mu_H \neq 0$

with a 1–α=0.99 confidence. The required parameters are calculated as $\bar{x}_T = 9.348837$, $\bar{x}_H = 11.48837$, $s_T^2 = 12.18276$ and $s_H^2 = 17.73516$. We consider 86 instances (observations) where both algorithms

return a solution. Hence, the value of test statistic is $t_0 = -3.62746$. Since $t_{\alpha/2,n_T+n_H-2} = t_{0.005,174} = 2.640379$, and $t_0 < -2.640379$, we reject the null hypothesis.

Table 14. *The number of groups found by the heuristic algorithm for large problem sets*

	L-I		L-II		L-III	
Problem Set	Optimal / Upper Bound	Heuristic Solution	Tabu Search	Heuristic Solution	Tabu Search	Heuristic Solution
1	9(%44)	10	9	13	10	14
2	8(%35)	12	16	17	9	10
3	3	4	9	11	8	13
4	9(%44)	10	2	2	5	6
5	6	7	11	12	13	16
6	7(%26)	8	12	15	12	15
7	-	11	8	10	14	16
8	7(%28)	9	8	11	10	14
9	8(%39)	10	10	12	12	15
10	9(%42)	10	13	17	10	13
11	10(%50)	11	8	12	14	18
12	10(%59	10	10	12	8	10
13	8(%50)	11	4	5	12	20
14	-	12	17	19	13	16
15	12(%58)	13	9	15	11	13
16	4	5	7	8	9	13
17	5	6	13	14	16	18
18	11(%58)	12	11	14	4	5
19	6	8	14	17	15	16
20	8(%37)	9	10	11	8	12
21	3	3	8	10	12	16
22	9(%35)	9	4	4	12	17
23	-	14	9	12	7	9
24	6	7	10	13	3	4
25	9(%53)	12	13	16	17	18
26	9(%44)	11	7	8	5	7
27	10(%50)	12	14	16	8	11
28	-	15	5	6	11	13
29	9(%44)	11	4	5	19	23
30	8(%37)	8	11	12	8	10

We also carried out experiments on very large problem sets (VL-I)-(VL-VIII) generated by Konak et al. [7]. Konak et al. [7] report the average number of job groups found by two tabu search algorithms TSBE2

and TSLE. Table 15 gives these results and the number of job groups found by our heuristic algorithm. The optimal solutions of these problem sets are not found.

Table 15. *The average number of groups found by the heuristic algorithm for very large problem sets*

Problem Set	TSBE2	TSLE	Heuristic Solution
VL-I	32.4	28.3	35.07
VL-II	61.8	55.8	52.53
VL-III	37.3	34.6	41.17
VL-IV	80.9	71.4	64.90
VL-V	43.3	40.6	49.86
VL-VI	99.9	85.7	77.80
VL-VII	53.5	46.4	56.33
VL-VIII	118.6	100.2	89.90

The solution times of the heuristic algorithm are less than 1 second, even for very large problems, hence not reported. Average number of job groups found by our heuristic are less than that of both tabu search algorithms in 4 out of 8 problem sets, and larger in other sets.

Clearly, the heuristic algorithm performs better on very large problem sets compared to large sets. One possible reason is that, for large sets, the heuristic solutions are compared with the best solution found by tabu search experiments, but for very large sets, with average performance of tabu search algorithms. Another possible reason is that, with increasing problem size, the deterioration of tabu search performance is faster than heuristic algorithm.

5. SEARCH ALGORITHM

This section gives the definition of the search algorithm developed to improve the solutions of heuristic algorithm. The search algorithm takes the initial solution found by heuristic algorithm and searches for better solutions in the neighborhood. A neighbor solution is found by reassigning a job from its group to another group together with the required tools. Throughout the algorithm, only feasible solutions are considered, i.e. a job is reassigned only if the tool magazine capacity is not exceeded. If we can find a solution among the neighbors of the initial solution with a smaller number of groups, this is set as the current solution, and we start searching for a better solution in its neighborhood. Otherwise, we search the neighborhoods of the neighbors. Thus, we obtain a tree structure. To limit the tree size, we allow searching on a maximum of three levels; and if the solution cannot be improved, we consider one of the neighbors as the current solution and build a new tree. The maximum of three levels is determined by analyzing the solutions of the heuristic in a preliminary study. The search algorithm is terminated on reaching the upper limit for computation time or the solutions analyzed. Figure 1 shows the tree structure of the search algorithm. In this section, we first explain the steps of the search algorithm and then discuss its performance.

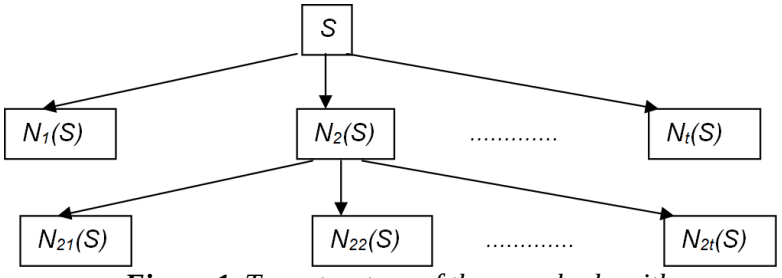

Figure 1. *Tree structure of the search algorithm*

5.1. Steps of the Search Algorithm

The steps of the search algorithm are described below. We use the same notation given in Section 4.

Step 1. Find an initial feasible solution using the heuristic given in Section 4, and denote this solution by S. Set i=1 and l=1 where i and l correspond to job and search level indices, respectively.

Step 2. If l = 3, terminate the algorithm. If l < 3, remove job i from its current group and place it in another group with its required tools.

2.a. Suppose $i \in A_g$, then $A_g \rightarrow A_g \backslash \{i\}$, $w_g \rightarrow w_g \backslash \{k | k \in l(i), k \notin l(q), \forall q \in A_g$ and $g \neq i\}$.

2.b. Choose one of the groups randomly and denote it by b such that $b \neq g$, if $|w_b \cup l(i)| \leq c$, then $A_b \rightarrow A_b \cup \{i\}$, $w_b \rightarrow w_b \cup l(i)$ and keep this solution as $N_1(S)$. Otherwise, choose another b.

2.c. If $N_1(S)$ includes less number of groups than S, then set $S = N_1(S)$ and go to Step 2. If there are same number of groups in $N_1(S)$ and S, apply Step 2.a for another group b. If there is no reduction in the number of groups, repeat this step for all groups and keep the feasible solutions found as $N_2(S)$, $N_3(S)$...

2.d. If $i = n$, go to Step 3. If $i < n$, set $i = i + 1$ and repeat Step 2.

Step 3. If l = 3, terminate the algorithm. If l < 3, set $l = l + 1$. If there is no reduction in the number of groups in the neighborhood of S, find the neighbors of neighbors of S and repeat Step 2. Obtain a tree structure as in Figure 1. When the algorithm finds a solution with less number of groups, keep it as S and go to Step 2. If there is no reduction in the number of groups in the second level of the tree, go to Step 4.

Step 4. If l = 3, terminate the algorithm. If l < 3, set $l = l$ +1. Generate the third level of the tree. Search the neighborhood of solutions $N_{11}(S)$, $N_{12}(S)$... When the algorithm finds a solution with less number of groups, keep it as S and go to Step 2. If there is no reduction in the number of groups in the third level of the tree, select randomly one of the solutions found so far, keep it as S and go to Step 2.

In Step 2, we try to assign each job to another group, which requires a computation of $O(n^2)$ in the worst case. We evaluate all neighbors for at most 3 levels, which leads us to a computational complexity of $O((n^2)^3) = O(n^6)$.

5.2. Computational Experiments on the Search Algorithm

We perform a computational experiment on problem instances given in Section 4.2 to evaluate the performance of the search algorithm. In small problem instances, the search algorithm could not improve the initial solutions found by the heuristic approach. In these problem sets, heuristic approach performs very well, and finds the optimal solution of most instances. Table 16 reports the solutions of Konak et al. [7] and the search algorithm for large instances. Solution times are not reported since the algorithm terminates in less than 1 second.

Table 16. *The number of groups found by the search algorithm for large problem sets*

	L-I		L-II		L-III	
Problem Set	Optimal / Upper Bound	Search Algorithm	Tabu Search	Search Algorithm	Tabu Search	Search Algorithm
1	9(%44)	10	9	11	10	14
2	8(%35)	11	16	17	9	10
3	3	3	9	11	8	10
4	9(%44)	10	2	2	5	6
5	6	7	11	12	13	15

6	7(%26)	8		12	14	12	15
7	-	11		8	10	14	16
8	7(%28)	9		8	10	10	12
9	8(%39)	10		10	12	12	14
10	9(%42)	9		13	15	10	12
11	10(%50)	11		8	10	14	17
12	10(%59)	10		10	11	8	10
13	8(%50)	10		4	5	12	19
14	-	11		17	18	13	14
15	12(%58)	13		9	13	11	13
16	4	5		7	7	9	12
17	5	6		13	13	16	17
18	11(%58)	10		11	12	4	5
19	6	7		14	17	15	15
20	8(%37)	8		10	11	8	11
21	3	3		8	10	12	14
22	9(%35)	9		4	4	12	15
23	-	14		9	10	7	8
24	6	7		10	12	3	4
25	9(%53)	11		13	16	17	17
26	9(%44)	11		7	7	5	7
27	10(%50)	11		14	15	8	10
28	-	14		5	6	11	12
29	9(%44)	10		4	5	19	23
30	8(%37)	8		11	12	8	10

Table 17. *The average number of groups found by the search algorithm for very large problem sets*

Problem Set	TSBE2	TSLE	Search Algorithm	CPU Time (sec)
VL-I	32.4	28.3	34.60	2.70
VL-II	61.8	55.8	52.50	2.47
VL-III	37.3	34.6	40.60	5.35
VL-IV	80.9	71.4	64.70	5.13
VL-V	43.3	40.6	49.86	-
VL-VI	99.9	85.7	77.80	-
VL-VII	53.5	46.4	56.33	-
VL-VIII	118.6	100.2	89.90	-

Table 16 shows that the search algorithm improves the initial solution for large problem sets. We apply two sample *t*-test to the differences in means of the number of groups in the tabu search solution (μ_T) and

heuristic solution (μ_H) for large problem sets given in Table 16, with the aim of testing the following hypothesis for the improved solutions:

H_0: $\mu_T - \mu_H = 0$

H_1: $\mu_T - \mu_H \neq 0$

with a $1-\alpha=0.99$ confidence. The sample mean and variance for tabu search results remain the same, and the required parameters for the improved heuristic solutions are calculated as $\bar{x}_H = 10.81395$ and $s_H^2 = 15.68263$. Hence, the value of test statistic is $t_0 = -2.57388$. Since $t_{\alpha/2,n_T+n_H-2} = t_{0.005,174} = 2.640379$, and $-2.640379 < t_0 < 2.640379$, we cannot reject the null hypothesis. The result of the *t*-test shows that the search algorithm significantly increases the quality of the heuristic solutions.

The average solutions of Konak et al. [7] and the search algorithm for very large problems are given in Table 17. The search algorithm slightly improves the initial solution for large and very large problem sets. In the first four sets of very large problems, the search algorithm provides improvements in very short times, but in the final four, an improved solution cannot be obtained due to the search tree's increasing size. Therefore, we give the initial solution in the table for these sets. The CPU times of the initial solutions are all less than 1 second, hence not reported. When we compare the search algorithm with the tabu search algorithm developed by Konak et al. [7], we can say that none of the algorithms dominate in terms of solution quality. In half of the problem sets, our algorithm gives better solutions than Konak et al's. [7].

Figure 2 from Konak et al. [7] shows the average solution times of tabu search algorithms for very large problem sets. Comparing the algorithm in terms of solution times, we observe that our algorithm is much faster than both tabu search approaches.

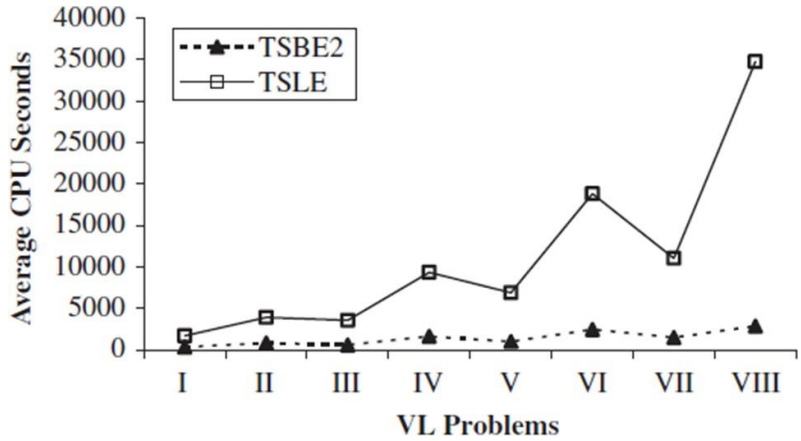

Figure 2. *Average solution times of tabu search algorithms for very large problem sets (Konak et al. [7])*

6. CONCLUSION

We study the problem of minimizing the number of tool switching instants in an automated manufacturing system. Each job demands a set of tools to be processed. We assume that the assignment of jobs to machines is known. In addition to the job processing times, it may be necessary to switch the tools between jobs since the tool magazine capacity is too small to hold all necessary tools. We aim to reduce the total time to complete all jobs by minimizing the total tool switching times. Since the tool switching times are constant, regardless of the type and number of tools, the problem can be considered as a job grouping problem, with the objective of minimizing the number of groups.

We develop a mathematical programming model and two constraint programming models. These models can find the optimal solution for small sized problems but perform less well for large sized instances since the problem is NP-hard. Therefore, we develop a heuristic approach to obtain feasible solutions in reasonable times. We then construct a search algorithm to improve the heuristic solution.

We compare the performance of the heuristic algorithm proposed in this study with that of Konak et al.'s

[7] tabu search algorithm. The computational tests show that the methods are equivalent in terms of solution quality; however, our heuristic approach produces good quality solutions in a shorter time. This study provides the following managerial insights on the efficient planning of tool management:

- We propose one mathematical programming model and two constraint programming models which pave the way for decision makers to increase the efficiency of a manufacturing system.
- The solution approach proposed by the study provides high quality (optimal or near optimal) solutions in a very short time, which indicates the applicability of the approach to manufacturing systems requiring frequent adaptation to rapid changes (or adjustments).
- Operating efficiency of a manufacturing system can be increased through the proposed solution approach. This implicitly augments utilization level of resources and, in turn, increases the system's manufacturing capacity.

The performance of constraint programming models may be improved by developing problem-specific constraint programming solution approaches or hybrid solution approaches. Also, the proposed approaches can be adapted and applied to the extension of this problem, in which the time needed for switching tools depends on their type and number.

ACKNOWLEDGEMENTS

This work is supported by The Scientific and Technological Research Council of Turkey (TÜBİTAK-3501 programme), Grant No: 110M492.

CONFLICTS OF INTEREST

No conflict of interest was declared by the authors.

REFERENCES

[1] Shirazi, R., Frizelle, G. D. M., "Minimizing the number of tool switches on a flexible machine: An empirical study", International Journal of Production Research, 39(15): 3547–3560, (2001).

[2] Calmels, D., "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, (2019).

[3] Crama, Y., "Combinatorial optimization models for production scheduling in automated manufacturing systems", European Journal of Operational Research, 99: 136–153, (1997).

[4] Tang, C. S., Denardo, E. V., "Models arising from a flexible manufacturing machine, Part II: minimization of the number of switching instants", Operations Research, 36(5): 778–784, (1988).

[5] Song, C. Y., Hwang, H., "Optimal tooling policy for a tool switching problem of a flexible machine with an automatic tool transporter", International Journal of Production Research, 40: 873–883, (2002).

[6] Denizel, M., "Minimization of the number of tool magazine setups on automated machines: a lagrangean decomposition approach", Operations Research, 51(2): 309–320, (2003).

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

[8] Konak, A., Kulturel-Konak, S., "An ant colony optimization approach to the minimum tool switching instant problem in flexible manufacturing system", Proceedings of IEEE Symposium on Computational Intelligence in Scheduling, 43–48, (2007).

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

[10] Baykasoğlu, A., Ozsoydan, F. B., "An improved approach for determination of index positions on CNC magazines with cutting tool duplications by integrating shortest path algorithm", International Journal of Production Research, 54(3): 742–760, (2016).

[11] Baykasoğlu, A., Ozsoydan, F. B., "Minimizing tool switching and indexing times with tool duplications in automatic machines", The International Journal of Advanced Manufacturing Technology, 89: 1775–1789, (2017).

[12] Baykasoğlu, A., Ozsoydan, F. B., "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, (2018).

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

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

[15] Marvizadeh, S. Z., Choobineh, F.F., "Reducing the number of setups for CNC punch presses", Omega, 41(2): 226–235, (2013).

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

[17] Dang, Q.-V.., van Diessen, T., Martagan, T., Adan, I., "A matheuristic for parallel machine scheduling with tool replacements", European Journal of Operational Research, https://doi.org/10.1016/j.ejor.2020.09.050.

[18] Atta, S., Sinha Mahapatra, P. R., Mukhopadhyay, A., "Solving tool indexing problem using harmony search algorithm with harmony refinement", Soft Computing, 23: 7407-7423, (2019)

[19] da Silva, T. T., Chaves, A. A., Yanasse, H. Y., "A new multicommodity flow model for the job sequencing and tool switching problem", International Journal of Production Research, https://doi.org/10.1080/00207543.2020.1748906.

[20] Smed, J., Johnsson, M., Puranen, M., Leipälä, T., Nevalainen, O., "Job grouping in surface mounted component printing", Robotics and Computer-Integrated Manufacturing, 15: 39–49, (1999).

[21] Yuan, S., Skinner, B.T., Huang, S., Liu, D. K., Dissanayake, G., Lau, H., Pagac, D., "A job grouping approach for planning container transfers at automated seaport container terminals", Advanced Engineering Informatics, 25: 413–426, (2011).

[22] Jans, R., Desrosiers, J., "Efficient symmetry breaking formulations for the job grouping problem", Computers & Operations Research, 40(4): 1132–1142, (2013).

[23] Dadashi, H., Moslemi, S., Mirzazadeh, A., "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): 52–58, (2016).

[24] Zeballos, L. J., "A constraint programming approach to tool allocation and production scheduling in flexible manufacturing systems", Robotics and Computer-Integrated Manufacturing, 26(6): 725–743, (2010).

[25] Zeballos, L. J., Quiroga, O. D., Henning, G. P., "A constraint programming model for the scheduling of flexible manufacturing systems with machine and tool limitations", Engineering Applications of Artificial Intelligence, 23(2): 229–248, (2010).

[26] Edis, E. B., Ozkarahan, I., "A combined integer/constraint programming approach to a resource-constrained parallel machine scheduling problem with machine eligibility restrictions", Engineering Optimization, 43(2): 135–157, (2011).

[27] Novas, J. M., Henning, G. P., "Integrated scheduling of resource-constrained flexible manufacturing systems using constraint programming", Expert Systems with Applications, 41(5): 2286–2299, (2014).

[28] Gökgür, B., Hnich, B., Özpeynirci, S., "Parallel machine scheduling with tool loading: a constraint programming approach", International Journal of Production Research, 56(16): 5541-5557, (2018).

[29] Frisch, A. M., Hnich, B., Kiziltan, Z., Miguel, I., Walsh, T., "Propagation algorithms for lexicographic ordering constraints", Artificial Intelligence, 170(10): 803–834, (2006).

[30] Flener, P., Frisch, A. M., Hnich, B., Kiziltan, Z., Miguel, I., Pearson, J., Walsh, T., "Breaking row and column symmetries in matrix models", International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science, 2470: 462–476, (2002).