Management Science and Information Technology

MSIT

2024, Volume 1, Issue 1, pp. 42-56
ISSN-Print: Pending
ISSN-Online: Pending
https://msit.refconf.com

Solving a New Multi-Objective Model for a Tool Switching Problem in Flexible Manufacturing Systems by a Genetic Algorithm

Hamid Dadashi [a], Zohreh Molamohamadi [a*] and Abolfazl Mirzazadeh [a]

[a] *Department of Industrial Engineering, Kharazmi University, Tehran, Iran*

Abstract

This paper deals with the Tool Switching Problem (ToSP), a famous problem in operations research. The simple ToSP includes finding a sequence of products and tool loading on a machine with the objective of minimizing the total number of tool switches. This paper presents a new multi-objective model for the ToSP, in which unlike the previous studies, the multi job tools have been considered (i.e., each tool can perform several tasks or jobs). This new model determines a product sequence and tool assigning for each stage that optimizes three objectives, namely 1) minimizing the total number of tool switches, 2) minimizing the overuse of tools per stage, and 3) balancing the tool usage. It is known that the ToSP is an NP-hard one, which is so difficult to be optimally solved in a reasonable computational time for large size problems. Therefore, a meta-heuristic, based on genetic algorithm (GA), is proposed in this study to solve such a hard problem. Moreover, a new tool loading algorithm is used to help the proposed GA related to the machine loading. Finally, the related results and conclusion are presented and discussed. The results of the numerical examples represented that the obtained results by GA are 3.5%. far from the optimum solutions found by the Branch-and-bound method.

Keywords: Tool switches; Flexible machine; Multi-objective optimization; Tool loading; Genetic algorithm.

1. Introduction

Nowadays, the manufacturing industry is increasingly requesting flexible manufacturing systems (FMSs) than conventional inflexible production systems. This is because FMSs are self-adjusting to produce various products and/or alter the product generation's order. Basically, an FMS includes a machine with several slots for loading different tools. Each slot supports only one tool, and every product running on that machine requires a specific set of tasks. Products are processed sequentially, so, each time a product is processed, the correct tools must be loaded on the machine. As the accessible slots are finite, it may eventually be necessary to remove a tool from the magazine and replace it with another tool. On this basis, the tool management is a demanding task which directly affects the efficiency of FMSs. Minimizing different factors such as the total time, the risk of tool breakdowns, and tool switching are important in FMSs (Pasha et al., 2024).

Although the order of tools in the magazine is mostly unrelated, the need to change a tool depends to the order, in which the product is performed. The simple Tool Switching Problem (ToSP) consists to the finding an appropriate product sequence and an associated sequence of tool switches for minimizing the number of tool loading/unloading operations in the magazine. Clearly, this problem is especially interesting when the needed time for change a tool is a significant part of the processing time of all the products (and hence the tool switching policy will significantly affect

*Corresponding author email address: zmmohamadi@gmail.com
DOI: 10.22034/ISS.2024.7784.1009

the system performance). Different examples of the problem can be found in the diverse areas, such as electronic industry, metalworking industry, computer memory management, aeronautics and generally, in the manufacturing companies (Bard, 1988; Belady, 1966; Privault & Finke, 1995; Shirazi & Frizelle, 2001; Tang & Denardo, 1988). In addition, the ToSP has a number of variants (Błazewicz & Finke, 1994; Jun et al., 1999; Kashyap & Khator, 1990).

In this model we tackle with multi job tools and the tools can do several tasks or jobs. This model determines a product sequence and tool assigning for each stage dedicating to one product with the following objectives:

1. minimizing the total number of tool switches;
2. minimizing the overuse of tools per stage;
3. balancing tool usage.

Note that overuse of the specific tool means that this tool has been used more than one time per stage, which it was collaborated on Section 3.1.2.

The rest of the paper is organized as follows: In section 2, the previous studies have been reviewed. The mathematical model of the proposed ToSP is presented in section 3 and the solution method is discussed in section 4. Section 5 verifies the developed model by solving different numerical examples and section 6 concludes the study.

2. Literature Review

The first study in the ToSP can be traced back to the early 60's (Belady, 1966). Later, the uniform ToSP has been tackled by a number of various techniques. In the late 80's, some studies have been contributed particularly to solve the problem (ElMaraghy, 1985; Kiran & Krason, 1988). (Tang & Denardo, 1988) developed an Integer Linear Programming (ILP) formulation of the problem, which followed later by (Bard, 1988) who described a non-linear integer programming formulation with a dual-based relaxation heuristic method. Researchers have also applied heuristic-based constructive methods to solve the problem. For instance, (Djellab et al., 2000) dealt with the ToSP by a hypergraph representation and suggested a particular heuristic method to minimize the number of gaps in edge-projection. They used the hypergraph to represent the relation between the products and the necessary tools. Moreover, (Hertz et al., 1998) described three constructive methods, called FI, GENI and GENIUS, where at each step the product supposed to be inserted in the current tour and the best position in the tour are selected. In addition, they considered the nearest neighbor (NN) and 2-opt search methods.

Some studies have applied the exact methods for solving the problem. For example, (Laporte et al., 2004) proposed two exact algorithms, branch-and-bound approach and linear programming-based branch-and-cut algorithm. The latter is based on a new ILP formulation which has a better linear relaxation than the one proposed earlier by (Tang & Denardo, 1988). It should not be neglected that since the ToSP is NP-hard for $c>2$, where c is the number of the slots on the machine's magazine (Crama et al., 1994; Oerlemans, 1992), these exact methods are limited. This limitation is also underlined in (Laporte et al., 2004), where they reported that their proposed algorithm showed a very low success ratio for instances with more than 10 products. Studies have also considered the clustering and grouping methods. (Salonen et al., 2006) explored the uniform ToSP of the printed circuit boards (PCBs) and developed an algorithm that prevents being stuck in local minimum points. To avoid identical groupings, the hierarchical grouping technique is applied. The metaheuristic methods have been also applied in recent studies. Several tabu search methods have been used in the literature (Al-Fawzan & Al-Sultan, 2003; Amaya et al., 2008; Hertz & Widmer, 1996; Konak et al., 2008; Salonen et al., 2006). A different attractive approach, named beam search algorithm, has been developed by (Zhou et al., 2005), which is especially efficient and practical in comparison to the previous techniques. It is because the search width and the evaluation functions can be changed to adjust the performance of the algorithm. (Solimanpur & Rastgordani, 2012) proposed an ant colony optimization algorithm to minimize the tool switching and indexing times in automatic machining centers. They compare the performance of the proposed algorithm with a heuristic approach, named multiple start greedy (MSG), through nine sample tests and concluded that the solutions of the ant colony algorithm is promising. (Amaya et al., 2012) applied memetic algorithm to solve ToSP and showed its effectiveness as a search paradigm. (Dadashi et al., 2016) proposed a new version of the ToSP in FMSs, in which a tool life is assumed for every versatile tool. They applied GA to solve the Np-hard problem and find the minimum total part type tardiness and tool purchasing cost. (Paiva & Carvalho, 2017) proposed an Iterated Local Search (ILS) method for the Job Sequencing and Tool Switching Problem (SSP). They further represented the competitiveness of the suggested approach through computational experiments.

(Amouzgar et al., 2021) proposed a mathematical formulation to find the optimum answers of a multi-objective tool-indexing problem and developed a modified genetic algorithm to obtain feasible solutions. (Wang et al., 2022) studied operation scheduling, tool scheduling, and restrained resources in a parallel machine scheduling problem to benefit the real industry. To obtain the optimal solutions, they proposed a Tabu-Genetic Algorithm that can find the solutions of large size problems within reasonable times and outperforms Tabu search algorithm and genetic algorithm in selecting local and global optima. (Rifai et al., 2022)explored job sequencing and tool switching problem with sequence-dependent setup, and developed a two-stage heuristic procedure to solve the problem. To find the near-optimal job sequence, an adaptive large neighborhood search (ALNS) is applied in the first stage, and proposed a combination of the Keep Tool Needed Soonest (KTNS) policy and simulated annealing (SA) for the tooling sub-problem. The efficacy and robustness of their proposed method is demonstrated by carrying out comprehensive computational experiments. (Darányi et al., 2023) presented a heuristic algorithm for tool assignment task of CNC machines with the objective of minimizing the tools changeovers. They dealt with this problem as a multi-objective hierarchical clustering problem, in which the similarity of the tool demands was the basis of grouping the products. Comparing the results of their proposed algorithm with the optimal approach depicted minor differences, while the running time of the heuristic approach was considerably lower. (Tri Windras Mara et al., 2023) proposed two integer linear programming models for SSP and executed experimental tests to compare the sequence-dependent SSP with the uniform SSP, and the results showed the effectiveness of multicommodity flow formulation. (Iori et al., 2024) addressed four types of ToSP, proposed their mathematical formulations, and solved them by dedicated arc flow models.

3. Problem Formulation

In this study, *n* products, that should be processed on a single FMS machine, are considered. The products and machine are available at the beginning of the process. It is supposed that each product needs a specific set of works and each work can be done by a specific set of tools. There are p different works and m available tools. Tools are multi jobs and each tool can do specific works. Let si_i denotes the set of works required by product i and sl_l denotes the set of tools, each of which can do work l. The tool magazine has c tools slots and each tool occupies exactly one slot of the tool magazine. The tool magazine of the machine can accommodate any combination of the tools. We assume that the number of tools required to process each product is not larger than the magazine capacity c. There are n stages corresponding to each product. The tool switches are done at the first of each stage and switches among the processing of product are avoided. The following notations have been used in this model.

Let $N_k = \{1, \dots, k\}$. Also, we denote product with $i \in N_n$, tool with $k \in N_m$, work with $l \in N_p$ and stage with $j \in N_n$. The decision variables are given below.

$$x_{i,j} = \begin{cases} 1 & \text{if product } i \text{ is scheduled to stage } j \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

$$y_{i,j} = \begin{cases} 1 & \text{if tool } k \text{ is assigned to stage } j \\ 0 & \text{otherwis} \end{cases} \quad (2)$$

$$z_{i,k,l} = \begin{cases} 1 & \text{if product } i \text{ uses tool k for doing work } l \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

3.1. Objectives of the problem

As mentioned before, the proposed model in this research involves determining a product sequence and tool assigning for each stage (i.e., determining which tools should be employed for each work of the product) with the following objectives:

1. Minimizing the total number of tool switching.
2. Minimizing the overuse of tools per stage.
3. Balancing the tool usage.

3.1.1. The total number of tool switching

We denote the total number of tool switching, as an objective denoted by *U*. This objective is computed by:

$$U = \sum_k \sum_j y_{k,j}(1 - y_{k,j-1}) \quad (4)$$

where $y_{k,0} = 0$ for all k.

3.1.2. The overuse of tools per stage

This objective is denoted by W and calculates the total number of the tool usage that is more than one time per stage (e.g., if a number of usage for tool k for four stages are 1, 3, 2, and 1 respectively, the overuse of tool k will be $t_k = (1-1) + (3-1) + (2-1) + (1-1) = 3$. To formulate W, the number of tool usage at stage *j* for tool *k* ($c_{k,j}$) is firstly calculated as follows:

$$c_{k,j} = \sum_i x_{i,j} \sum_l z_{i,k,l} \quad (5)$$

$$W = \sum_k \sum_j y_{k,j}\left(c_{k,j} - 1\right) \quad (6)$$

3.1.3. Tool balancing

This objective aims to adjust and balance the tools usage to prevent tool's failures, which is caused by overuse of the tool during processing of the products. This objective is shown by V. Let a_k be the number of tool usage for tool k and determine its value as follows:

$$a_k = \sum_i \sum_l z_{i,k,l} \quad (7)$$

where, $\bar{a}$ is the mean of tool usage and is defined by:

$$\bar{a} = \frac{\sum_k a_k}{k} \quad (8)$$

Now, V will be:

$$V = \sum_k |a_k - \bar{a}| \quad (9)$$

3.2. Constraints definition

The constraints of the presented problem will be as follows:

$$\sum_j x_{i,j} = 1; \quad \text{for all } i \quad (10)$$

$$\sum_i x_{i,j} = 1; \quad \text{for all } j \quad (11)$$

$$x_{i,j}|si_i| \leq \sum_{i \in si_i} \sum_{k \in sl_l} y_{k,j} z_{i,k,l}; \quad \text{for all } i, j \quad (12)$$

$$\sum_i \sum_k z_{i,k,l} = |si_i|; \quad \text{for all } i \quad (13)$$

$$\sum_{k \in sl_l} z_{i,k,l} = 1; \quad \text{for all } i, (l \in si_i) \quad (14)$$

$$\sum_k y_{k,j} \leq c; \quad for\ all\ j \quad (15)$$

$$x_{i,j}, y_{k,j}, z_{i,k,l} \in \{0, 1\}$$

By considering Eq. 10, product *i* should be scheduled to one stage. Eq. 11 ensures that every stage should be occupied by one product. Eqs. 12 to 14 ensure that a product can be scheduled on specific stage only if all of its required works are satisfied by the existing tools on magazine ($|si_i|$ is cardinality of si_i). Constraint 15 states that tool usage should not be greater than magazine capacity c.

3.3. Multi-objective optimization method

To optimize the objectives, the sum of objectives are considered to transform the multi-objective problem to a single objective optimization as follows (Wierzbicki, 1982, 1986, 1999):

$$z = \min\left[aa_1\left(\frac{U - U^{mid}}{U^{nad} - U^*}\right) + aa_2\left(\frac{V - V^{mid}}{V^{nad} - V^*}\right) + aa_3\left(\frac{W - W^{mid}}{W^{nad} - W^*}\right)\right] \quad (16)$$

The weighted Lp-metric method with p=1 is used in this research to solve the multi-objective problem, where aa_1, aa_2, and aa_3 are weights for U, V, and W, respectively. U^*, V^*, and W^* are the best values that can be achieved by solving a single objective problem with the objectives U, V, and W, respectively, subject to the problem constraints. Also, U^{nad}, V^{nad} and W^{nad} are the weakest (nadir) values of U, V, and W, respectively. U^{mid}, V^{mid}, and W^{mid} can be defined by:

$$U^{mid} = \frac{U^{nad} - U^*}{2} \tag{17}$$

$$V^{mid} = \frac{V^{nad} - V^*}{2} \tag{18}$$

$$W^{mid} = \frac{W^{nad} - W^*}{2} \tag{19}$$

The reason for using this method is that in the applied tool loading's algorithms (described in Section 4.1), the weight of objectives should be determined firstly; thus, the decision maker (DM) is taken apart in the solution process. In other words, we use one of posteriori methods (i.e., weighted L_p-metric method) (Wierzbicki, 1982) to solve the problem.

4. Solving the Problem

The ToSP can be divided into three sub-problems (Tzur & Altman, 2004). The first sub-problem is machine loading and includes determining the sequence of the products. The second sub-problem is tool loading and includes determining which tool has to switch (if a switch is needed) before processing a product. The third sub-problem is slot loading that consists of deciding where to place each tool. As the uniform ToSP is considered in this paper, only two sub-problems have to be taken into account (i.e., machine loading and tool loading). In the simple ToSP (Amaya et al., 2008), the tool loading sub-problem can be optimally solved if the sequence of products is known by following a specific tool switching policy that guarantees to obtain the optimal number of tool switches for a given products sequence. It is solved by the Keep Tool Needed Soonest (KTNS) method (Bard, 1988; Belady, 1966; Tang & Denardo, 1988); therefore, the meta-heuristic effort is concentrated on the machine loading stage (Amaya et al., 2008). In our presented model for tool loading problem, a heuristic algorithm is introduced.

4.1. Tool loading

In the context of the uniform ToSP, the cost of switching a tool is assumed to be a constant (for all tools). The introduced algorithm in this study, gets value Z that is a near-optimal solution for the given product sequence. As mentioned above, we assume that the tools are multi jobs; but, if otherwise, then our tool loading algorithm will turn to the KTNS algorithm. Unlike the KTNS-procedure, our tool loading algorithm does not guarantee to produce an optimal solution. However, as shown in Section 5, it obtains a good and reasonable solution that is near to the optimum point.

4.1.1. Tool loading algorithm

The steps of the proposed tool loading algorithm are as follow:

1. Get the product sequence (sop)
2. For all products (stages) with respect to the product sequence, do as follows (i):
   - 2.1. Sort the works of product i (si_i) according to the number of tools that can do work l, in an ascending sort order and name this set swo (i.e., works will be sorted in an ascending order with respect to the value of cardinality of its sl_l)
   - 2.2. For all members (works) in swo , do the following steps.
     - 2.2.1. For work $l \in si_i$ (or $l \in swo$), score to all of the tools in set sl_l as follows and name its set $score1$ (the member of set '$score1$' is score of the corresponding tool on set swo):
       - 2.2.1.1. Being on machine on the previous stage (i.e., product) or selected for the previous works of the current stage (i.e., if it is true, then the corresponding tool's score will be one; otherwise, it is zero).

2.2.1.2. Power of being used on sub-sequent stages (see algorithm's comments).

2.2.1.3. Sum values of Steps 2.2.1.1 and 2.2.1.2.

2.2.2. Normalize $score1$.

2.2.3. For work $l \in si_i$ (or $l \in swo$), score to all of the tools in set sl_l as follows and name its set as $score2$:

Calculate the number of tool usage on the previous product, and the number of times of corresponding tool is used at the current stage (product) for the previous work on set swo and sum all of them. It is a score of the corresponding tool on set $score2$.

2.2.4. Normalize $score2$.

2.2.5. For work $l \in si_i$ (or $l \in swo$), score to all of tools in set sl_l as follows and name its set as $score3$:

The number of times that the corresponding tool is used at the current product (for the previous works on set swo).

2.2.6. Normalize $score3$.

2.2.7. Calculate set '$total\ score$' (it is explained in Section 4.1.2).

2.2.8. Define the member with maximum value on set '$total\ score$' and select corresponding tool on set swo for the corresponding work.

2.3. Define a number of slots that do not schedule for tools until now (these will be places of unused tools at the current stage) and name it as R.

2.4. If we are at the final stage, go to Step 2.5; otherwise, go to Step 2.6.

2.5. Consider the tools that existed on a machine at the previous stage and is not selected for the current stage yet, select R tools among these tools and go to the final step.

2.6. Consider the tools that are not selected for the current stage and score them as like as Steps 2.2.1 to 2.2.7, except Steps 2.2.5 and 2.2.6 (i.e., we eliminate $score3$ at Step 2.2.7) and select R tools with the best score.

3. End.

4.1.2. Algorithm's comments

In this section, we introduce some definitions for performing the tool loading algorithm:

sp: It is a product-work matrix, whose arrays are binary if a specific product needs specific work for its processing, then its corresponding array will be true (i.e., one).

sj: It is a work-tool matrix, arrays are binary if a specific tool can do specific work then the corresponding array will be true.

$Spsj$: It is a product-tool matrix and its arrays denote the rate of the application of the corresponding tool for the related product. It is defined by $Spsj = sp \times sj$.

$Spsjs$: It is a matrix like $Spsj$; but its row's dimension is arranged with respect to the product sequence.

Note 1: At Step 2.2.1.2 for defining the value of the 'power of being used on subsequent stages' for a specific tool, we consider the corresponding column of matrix *Spsjs* and pick up arrays of subsequent products, in which this set is named as *E*. We assume that *g* indicates the number of subsequent products, then *E* is a dimensional matrix with $1 \times g$ dimension. Now, we can use matrix *E* to define the 'power of being used on subsequent stages' that is calculated as follows:

Consider a matrix with $g \times 1$ dimension and name it as 'norm', the members of this matrix will be at a descending order, more difference between two subsequent members means more emphasis on the nearest subsequent products. At last for defining the value of the 'power of being used', we multiply matrix *E* to the matrix norm. For instance, if we have three products for subsequent stages and assume *E*= [2 3 5] and $norm = [s^2 \quad s^1 \quad s^0]^T$, the value of the

'power of being used' will be (s^2×2)+(s×1)+(s×5). Note that s is a normalized coefficient. To define value s in our experiment results, we find that if this value is greater than 2×c, the result will be better, i.e., among products that have connection with a specific tool (corresponding array on *Spsjs* is not zero), the nearest consider more than others, this section of algorithm is like KTNS algorithm in a simple ToSP problem.

Note 2: To normalize score1, score2, score3, the following step should be followed.

Get the maximum and minimum values of set's members and name them as max and min, respectively.

For its entire arrays of score1, do as follows and replace the result with the previous value of the array (assume that Q is a specific array):

$$\frac{Q-min}{max-min} \tag{20}$$

And for score2 and score3, do as follows:

$$\frac{Q-min}{min-max} \tag{21}$$

Finally, we multiply score1, score 2, score 3 to their corresponding normalized coefficients, and name it as 'wscore1, wscore2, wscore3', respectively. Normalized scores with coefficients are as follows:

$$wscore1 = \frac{aa_1}{U^{nad}-U^*} \times (score1) \tag{22}$$

$$wscore2 = \frac{aa_2}{W^{nad}-W^*} \times (score2) \tag{23}$$

$$wscore3 = \frac{aa_3}{V^{nad}-V^*} \times (score3) \tag{24}$$

Then, the total score of the corresponding tool is obtained by:

total score =wscore1+wscore2+wscore3

4.1.3. The machine loading

In this section, we consider the product's sequence and look for the best sequence of the product to minimize our objective (Z). As mentioned before, this problem is NP-hard for c>2 and therefore, we use meta-heuristic algorithm for solving the machine loading problem. Note that we do not intend to compare different meta-heuristic algorithms with each other. We want to show a method to solve our proposed ToSP problem; therefore, it can be possible that other ways (i.e., other meta-heuristic algorithms and parameter's values, such as the crossover or mutation rate) exist to obtain a better result.

In this section, we consider the genetic algorithm to solve the product's sequence (i.e., machine loading) problem. For recombination, a crossover scheme, called Alternating Position Crossover (APX), is used that consists of selection of genes alternating of each parents (Larranaga et al., 1999). For the mutation operator, we consider the block neighborhood that is proposed for the ToSP in (Al-Fawzan & Al-Sultan, 2003). It is based on swapping the whole segments of contiguous positions. This mutation operator is Random Block Insertion (RBI) and works as follows:

1. A random block length $b_l \in N_{n/2}$ is uniformly selected.

2. The starting point of the block $b_s \in N_{n-2b_l}$ is subsequently selected at random.

3. Finally, an insertion point bi is selected, such that $b_s + b_l \leq b_i \leq n - b_l$ and the segments

$(b_s, b_s + b_l)$ and $(b_i, b_i + b_l)$ are swapped.

5. Experimental Results

This section aims to verify the performance of the proposed model by sixteen numerical examples which have various sizes. Branch-and-bound (B&B) method is used for finding the optimal values of small-sized examples under Lingo 8.0 software on a PC with two Intel® CoreTM2 T9300@ 2.5 GHz processors and 2 GB RAM. However, for large-

sized examples, it is not possible to find the optimal solution in a reasonable CPU time. Moreover, all of the examples have been solved by the proposed genetic algorithm and the results are compared with the optimal solutions in terms of the objective function values (i.e., *U*, *V*, *W* and *Z*; however, we emphasis on *Z*) and CPU time. Every example is solved 20 times by GA and the best objective function value of *Z* and mean CPU time are reported. For large-sized problems, the run time of Lingo is limited to three hours and the best solution obtained is reported. This limitation is considered according to the quality of the obtained solutions by the proposed GA. However, a feasible solution cannot be found for any large examples after three hours. We consider eight small and eight large-sized examples for our experimental results. Note that to obtain *Z* according to Eq. 16, we assume that aa_1, aa_2, and aa_3 are equal to one; in other words, the objectives do not have any preference to each other.

5.1. Small-sized examples

As mentioned before, we consider eight small-sized instances, the relation between the product and work (sp), and the relation between work and tool (sj) for small-sized examples are as follows:

Examples 1 & 2:

$$sp = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix} \quad sj = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

Examples 3 & 4:

$$sp = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \quad sj = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

Example 5:

$$sp = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{bmatrix} \quad sj = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

Examples 6 to 8:

$$sp = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \quad sj = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

The population size of our proposed GA is 60 and the number of generations for small-sized examples are shown in Table 1.

Table 1. Number of generations for small-sized instances

Example1	Example2	Example3	Example4	Example5	Example6	Example7	Example8
30	30	60	60	100	200	300	500

We solve small-sized problems with both B&B method and GA. Table 2 shows the related results. The average of the relative gap between the B&B and GA in term of *Z* is computed about 3.5%. We also compare the CPU times of the B&B and GA as represented in Fig. 1. The exponential trend of the B&B's CPU time when the size of instances increases is visible in this figure.

Table 2. Comparison between B&B and GA runs for small-sized instances

No.	Problem Information														
	Product	Work	Tool	*c*	*U*	*V*	*W*	*Z*	CPU time (Sec.)	*U*	*V*	*W*	*Z*	Mean of CPU time	Gap (%)
1	4	5	6	3	1	6	0	0.6077	15	0	9	0	0.65	0.95	6.5
2	4	5	6	4	0	6	0	0.3248	19	0	6	0	0.3248	1.04	0
3	5	4	4	3	1	1.5	0	0.641	57	1	3	0	0.7221	1.8	12.5
4	5	4	4	4	0	1.5	0	0	55	0	1.5	0	0	2.04	0
5	6	5	5	3	3	0	0	0.753	586	3	0	0	0.753	24.5	0
6	6	8	10	3	5	11.2	0	0.3476	6480	5	11.2	0	0.3476	56	0
7	6	8	10	4	4	11.2	0	0.2643	6710	4	11.2	0	0.2643	92	0
8	6	8	10	5	3	11.2	0	0.2201	7450	4	11.2	0	0.2403	161	9.1

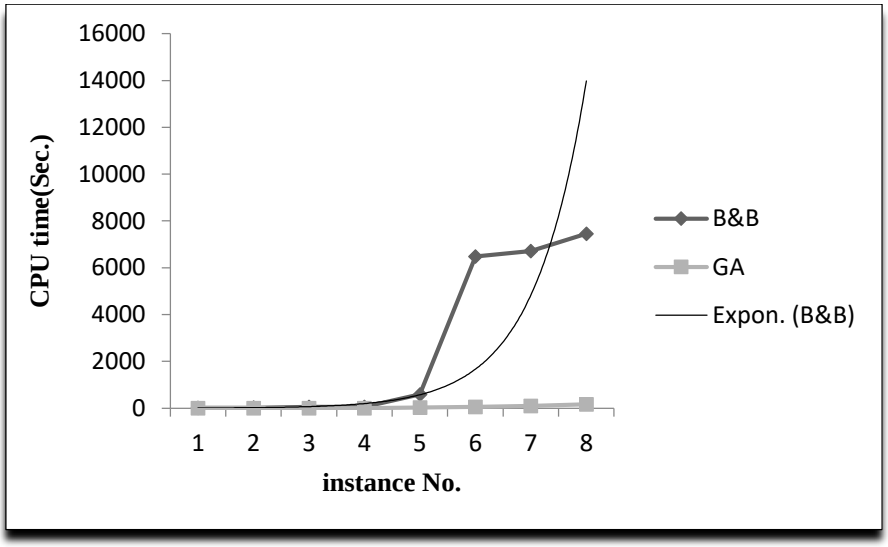

Fig. 1. Comparison between the B&B and GA in terms of CPU times

5.2. Large-sized examples

We consider eight instances for large-sized examples solved by both the B&B and GA. To run our proposed GA, the population size is 60 and the number of generations is 1000 for all 8 instances. The relation between the product and work (*sp*), and relation between the work and tool (*sj*) for large-sized examples are given bellow.

Examples 9 and 10:

$$sp = \begin{bmatrix} 1&0&0&1&0&0&0&1&0&0 \\ 1&0&1&0&0&1&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0&1 \\ 0&0&0&0&1&1&0&1&0&0 \\ 0&0&0&0&0&0&0&1&0&1 \\ 0&0&0&1&0&1&0&0&0&1 \\ 1&0&0&0&0&0&1&0&1&0 \\ 1&0&1&0&0&0&0&1&0&0 \end{bmatrix} \quad sj = \begin{bmatrix} 1&0&0&1&0&0&0&1&0&0 \\ 0&0&1&0&1&0&0&0&0&0 \\ 0&1&0&0&0&0&0&1&0&0 \\ 1&0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&1&0&1&0&1 \\ 0&1&0&1&0&0&1&0&0&0 \\ 0&1&0&0&1&0&1&0&0&0 \\ 1&0&0&0&0&0&0&0&1&0 \\ 1&0&0&0&1&1&0&0&0&1 \\ 0&0&0&0&0&0&1&1&0&1 \end{bmatrix}$$

Examples 11 and 12:

$$sp = \begin{bmatrix} 1&0&0&1&0&0&0&1&0&0 \\ 1&0&1&0&0&1&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0&1 \\ 0&0&0&0&1&1&0&1&0&0 \\ 0&0&0&0&0&0&0&1&0&1 \\ 0&0&0&1&0&1&0&0&0&1 \\ 1&0&0&0&0&0&1&0&1&0 \\ 1&0&1&0&0&0&0&1&0&0 \\ 1&1&0&0&0&0&0&0&0&1 \\ 0&0&0&0&1&1&0&1&0&0 \end{bmatrix} \quad sj = \begin{bmatrix} 1&0&0&1&0&0&0&1&0&0 \\ 0&0&1&0&1&0&0&0&0&0 \\ 0&1&0&0&0&0&0&1&0&0 \\ 1&0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&1&0&1&0&1 \\ 0&1&0&1&0&0&1&0&0&0 \\ 0&1&0&0&1&0&1&0&0&0 \\ 1&0&0&0&0&0&0&0&1&0 \\ 1&0&0&0&1&1&0&0&0&1 \\ 0&0&0&0&0&0&1&1&0&1 \end{bmatrix}$$

Examples 13and 14:

$$sp = \begin{bmatrix} 1&0&0&1&0&0&0&1&0&0 \\ 1&0&1&0&0&1&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0&1 \\ 0&0&0&0&1&1&0&1&0&0 \\ 0&0&0&0&0&0&0&1&0&1 \\ 0&0&0&1&0&1&0&0&0&1 \\ 1&0&0&0&0&0&1&0&1&0 \\ 1&0&1&0&0&0&0&1&0&0 \\ 1&1&0&0&0&0&0&0&0&1 \\ 0&0&0&0&1&1&0&1&0&0 \\ 0&0&0&0&1&1&1&0&0&0 \\ 0&1&0&0&0&0&1&0&0&1 \end{bmatrix} \quad sj = \begin{bmatrix} 1&0&0&1&0&0&0&1&0&0 \\ 0&0&1&0&1&0&0&0&0&0 \\ 0&1&0&0&0&0&0&1&0&0 \\ 1&0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&1&0&1&0&1 \\ 0&1&0&1&0&0&1&0&0&0 \\ 0&1&0&0&1&0&1&0&0&0 \\ 1&0&0&0&0&0&0&0&1&0 \\ 1&0&0&0&1&1&0&0&0&1 \\ 0&0&0&0&0&0&1&1&0&1 \end{bmatrix}$$

Example 15:

$$sp = \begin{bmatrix} 1&1&1&0&1&0&1&0&0&1&0&0&1&0&0&0 \\ 1&0&0&1&0&0&0&0&1&0&1&0&0&0&0&1 \\ 0&1&0&1&0&0&0&1&1&0&1&0&1&0&0&1 \\ 0&1&0&1&0&1&0&0&1&0&0&1&0&0&0&0 \\ 1&0&1&0&1&0&1&0&1&0&0&1&0&0&1&0 \\ 0&0&1&0&0&1&0&0&0&1&0&1&0&1&0&0 \\ 1&0&0&0&1&1&1&0&0&0&0&0&1&1&1&0 \\ 1&1&1&1&1&0&1&0&0&0&0&1&1&0&0&0 \\ 0&0&1&0&0&0&1&0&0&1&0&0&0&0&1&0 \\ 0&0&0&0&0&0&1&1&1&0&0&0&0&0&0&1 \\ 1&0&1&0&0&1&0&0&1&0&0&1&0&0&0&0 \\ 0&0&0&1&0&1&0&0&1&0&0&0&0&0&1&0 \\ 1&0&0&0&1&0&1&0&0&1&0&1&0&0&0&1 \\ 1&1&0&0&0&0&0&1&1&1&1&0&0&0&0&0 \\ 1&0&1&1&0&0&1&0&0&0&0&1&0&0&0&0 \\ 0&1&0&0&0&1&0&0&1&0&1&0&0&0&1&0 \end{bmatrix} \quad sj = \begin{bmatrix} 0&0&0&1&1&0&0&0&0&0&1&0&0&0&0&0 \\ 0&0&1&0&0&0&1&1&1&0&0&0&1&0&0&1 \\ 1&0&1&0&0&0&0&0&0&0&0&0&0&1&0&1 \\ 1&1&0&0&0&1&0&0&0&0&0&1&0&0&0&1 \\ 0&0&0&0&0&0&0&1&1&1&1&1&0&0&0&0 \\ 0&0&0&0&0&1&0&1&0&1&0&0&0&0&0&1 \\ 1&0&0&1&1&0&0&0&0&0&0&0&0&0&1&0 \\ 1&0&0&0&0&1&0&0&0&0&0&0&0&0&1&0 \\ 0&1&0&0&0&0&1&0&0&0&1&0&0&1&0&1 \\ 0&0&0&0&1&0&0&0&0&0&1&0&1&0&0&0 \\ 0&0&0&1&1&0&0&0&0&0&0&1&0&0&1&0 \\ 1&0&0&0&0&1&0&0&0&0&0&0&1&0&0&1 \\ 1&0&0&1&1&1&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1&0&0&0&0&0&1&0&0 \\ 1&0&0&0&0&1&0&0&0&1&0&0&1&0&0&0 \\ 0&0&1&0&0&1&0&0&0&1&0&0&1&0&0&1 \end{bmatrix}$$

Example 16:

$$sp = \begin{bmatrix}
0&1&1&0&1&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&1&0&0&0&0&1&0&1&0&0&0&0&1&0&0&0&1&0&0&0&1&0\\
0&1&0&0&0&0&0&1&1&0&1&0&1&0&0&1&0&0&0&0&0&0&1&0&1\\
0&1&0&0&0&1&0&0&1&0&0&1&0&0&0&0&0&0&0&0&0&1&0&0&1\\
0&0&1&0&1&0&0&0&1&0&0&1&0&0&1&0&1&1&1&1&0&0&0&&\\
0&0&1&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0&1&1&1&1&0&0\\
1&0&0&0&1&1&1&0&0&0&0&0&1&1&1&0&0&0&1&0&0&0&0&1&0\\
1&1&1&1&1&0&1&0&0&0&0&1&1&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&1&0&0&0&1&0&0&1&0&0&0&0&1&0&0&1&0&1&0&0&0&0&0\\
0&0&0&0&0&0&1&1&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&1&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&1&0&0&1&0&1&0&0&0&1&1&1&1&1&0&0&0&0&0\\
1&1&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&1&0&0&0&1&0&0&0&0&1&0&0&0&1&0&0&1&0&0&1&0&1&0&1\\
0&0&0&1&0&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1&1&1\\
0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0&0&1&0&0&1&0&0&1&0\\
0&1&0&0&0&1&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&1&1\\
0&1&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&1
\end{bmatrix}$$

$$sj = \begin{bmatrix}
1&0&1&0&0&0&0&0&0&0&0&0&1&0&0&1&0&0&0&0&0&0&0&1&0\\
1&1&0&0&0&1&0&0&0&0&0&1&0&0&0&1&0&0&1&0&0&0&0&0&0\\
0&0&0&1&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1&1\\
1&0&0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1&0\\
0&0&0&1&1&1&0&0&0&0&0&0&0&0&0&1&0&0&1&0&0&1&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0&0&1&0&0&1&0&0&0&0&0&0&0&1\\
0&0&0&0&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&1&0&0&0&0&0\\
0&0&1&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&1&0&1&0&0&0&0&1&0&0&0&0&0&0&1&0&0&0\\
0&0&0&1&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&1&1&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&1&0&0&0&1&0&0&1&0&1&0&0&1&0&0&1&0&0&1\\
0&0&0&0&1&0&0&0&0&0&1&0&1&0&0&0&0&1&0&1&1&1&0&0&0\\
0&0&0&1&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1&1\\
1&0&0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1&0\\
0&0&0&1&1&1&0&0&0&0&0&0&0&0&0&1&0&0&1&0&0&1&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0&0&1&0&0&1&0&0&0&0&0&0&0&1\\
1&1&1&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&1&1&0&0&0&0&0&0&0&0&0&1&0&0&1&0&0&1&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0&0&1&0&0&1&0&0&0&0&0&0&0&1\\
0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0\\
0&0&1&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&1&0&0&0&1&1&1&0&0&0&0&0&0&0&0&0&0
\end{bmatrix}$$

We also solve large-sized problems with both the B&B and GA. Table 3 shows the related results. According to this table, the B&B method obtains the feasible space within three hours in four of eight instances, and the results of this instances are weaker than the proposed GA in this research. In this case, as shown in Figure 2, the CPU time of the GA demonstrates a polynomial behavior when the size of instances increases. Furthermore, Figure 3 represents a typical convergence of the GA during 2000 successive generations related to a single run.

Table 3. Comparison between B&B and GA runs for large-sized instances

No.	Problem Information													
	Product	Work	Tool	*c*	*U*	*V*	*W*	*Z*	CPU time (Sec.)	*U*	*V*	*W*	*Z*	Mean of CPU time
1	8	10	10	5	5	9.6	0	0.5	10800	5	8	0	0.43	399
2	8	10	10	6	4	6.8	0	0.4107	10800	3	8.4	0	0.3036	401
3	10	10	10	5	11	5.2	0	0.8627	10800	9	5.2	0	0.4615	654
4	10	10	10	6	6	8.8	1	0.8768	10800	3	8.8	0	0.3961	640
5	12	10	10	5	-	-	-	-	10800	7	6	0	0.4471	655
6	12	10	10	6	-	-	-	-	10800	4	7.2	0	0.3059	660
7	16	16	16	10	-	-	-	-	10800	14	10.25	0	0.25	2215
8	20	25	25	10	-	-	-	-	10800	48	32	0	1.18	5800

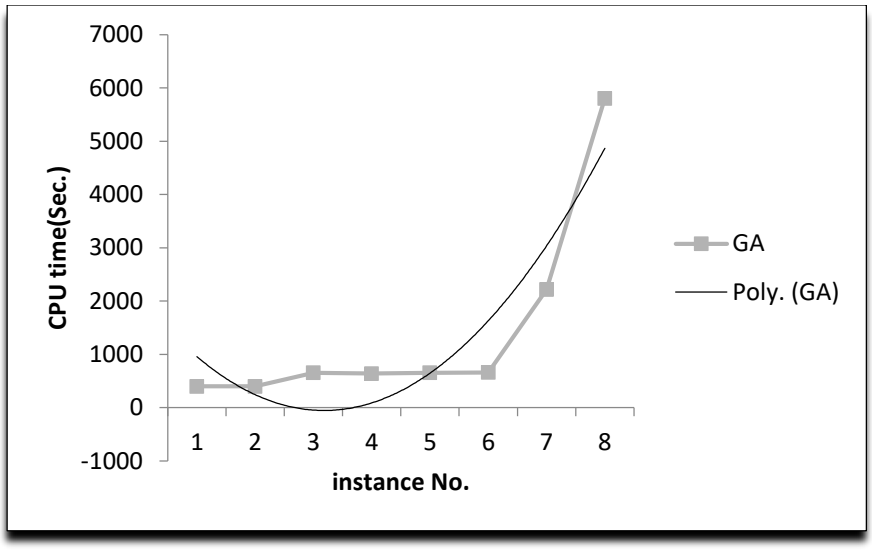

Figure 2. Polynomial trend of the GA's CPU time for large-sized problems

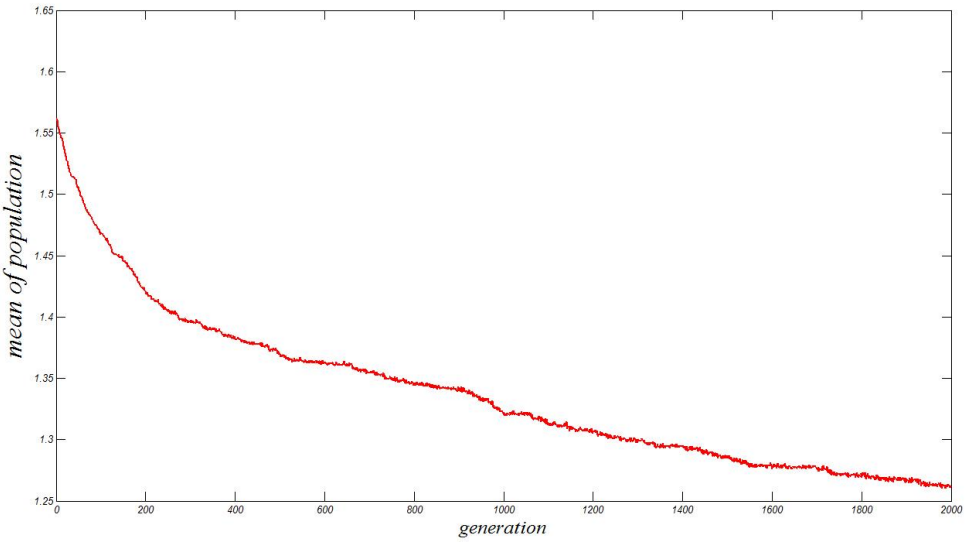

Figure 3. Typical convergence of GA during 2000 successive generations related to a single run

6. Conclusion

This paper considers the Tool Switching Problem that is a famous problem in the field of operations research. The simple ToSP deals with determining the product sequence and tool loading on a machine with the objective of minimizing the total number of tool switches. This study introduces a new multi-objective ToSP model. Unlike the previous studies, we have the multi job tools (i.e., each tool can perform several tasks or jobs). The new ToSP model has determined the product sequence and tool assigning for each stage with the following objectives:

1. Minimizing the total number of tool switches.
2. Minimizing the overuse of tools per stage.
3. Tool balancing usage.

The ToSP is an NP-hard problem, which means that large-sized problems will be solved hardly; therefore, the meta-heuristic based on GA is proposed in this paper. We have first presented the new mathematical formulation of the

ToSP, and then introduced a new tool loading algorithm to help the meta-heuristic algorithm related to the machine loading. Furthermore, we have solved the presented model for various instances by the use of the branch-and-bound and genetic algorithm methods. The obtained results have been compared, in which a relative gap between solutions reported by GA and optimum solutions found by B&B in terms of the objective function value (Z) has been about 3.5%. Future research can focus on proposing other metaheuristic algorithms, such as hybrid ones, and compare the results and the CPU time with the applied approach in this research. Moreover, the model can be solved for real data to verify the performance of the model in industry.

References

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. *Computers and Industrial Engineering*, *44*(1), 35–47. https://doi.org/10.1016/S0360-8352(02)00183-3

Amaya, J. E., Cotta, C., & Fernández, A. J. (2008). A memetic algorithm for the tool switching problem. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 5296 LNCS*, 190–202. https://doi.org/10.1007/978-3-540-88439-2_14

Amaya, J. E., Cotta, C., & Fernández-Leiva, A. J. (2012). Solving the tool switching problem with memetic algorithms. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing: AIEDAM*, *26*(2), 221–235. https://doi.org/10.1017/S089006041100014X

Amouzgar, K., Nourmohammadi, A., & Ng, A. H. C. (2021). Multi-objective optimisation of tool indexing problem: a mathematical model and a modified genetic algorithm. *International Journal of Production Research*, *59*(12), 3572–3590. https://doi.org/10.1080/00207543.2021.1897174

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

Belady, L. A. (1966). A study of replacement algorithms for a virtual-storage computer. *IBM Systems Journal*, *5*(2), 78–101. https://doi.org/10.1147/sj.52.0078

Błazewicz, J., & Finke, G. (1994). Scheduling with resource management in manufacturing systems. *European Journal of Operational Research*, *76*(1), 1–14. https://doi.org/10.1016/0377-2217(94)90002-7

Crama, Y., Oerlemans, A. G., & Spieksma, F. C. R. (1994). Minimizing the number of tool switches on a flexible machine. *The International Journal of Flexible Manufacturing Systems*, *6*, 165–195. https://doi.org/10.1007/978-3-662-00459-3_8

Dadashi, H., Moslemi, S., & Mirzazadeh, A. (2016). Optimization of a New Tool Switching Problem in Flexible Manufacturing Systems with a Tool Life by a Genetic Algorithm. *International Journal of Industrial and Manufacturing Systems Engineering*, *1*(3), 52–58. https://doi.org/10.11648/j.ijimse.20160103.12

Darányi, A., Czvetkó, T., Kummer, A., Ruppert, T., & Abonyi, J. (2023). Multi-objective hierarchical clustering for tool assignment. *CIRP Journal of Manufacturing Science and Technology*, *42*, 47–54. https://doi.org/10.1016/j.cirpj.2023.02.002

Djellab, H., Djellab, K., & Gourgand, M. (2000). New heuristic based on a hypergraph representation for the tool switching problem. *International Journal of Production Economics*, *64*(1), 165–176. https://doi.org/10.1016/S0925-5273(99)00055-9

ElMaraghy, H. A. (1985). Automated tool management in flexible manufacturing. *Journal of Manufacturing Systems*, *4*(1), 1–13. https://doi.org/10.1016/0278-6125(85)90003-2

Hertz, A., Laporte, G., Mittaz, M., & Stecke, K. E. (1998). Heuristics for minimizing tool switches when scheduling part types on a flexible machine. *IIE Transactions (Institute of Industrial Engineers)*, *30*(8), 689–694. https://doi.org/10.1080/07408179808966514

Hertz, A., & Widmer, M. (1996). An improved tabu search approach for solving the job shop scheduling problem with tooling constraints. *Discrete Applied Mathematics*, *65*(1–3), 319–345. https://doi.org/10.1016/0166-218X(95)00040-X

Iori, M., Locatelli, A., Locatelli, M., & Salazar-González, J. J. (2024). Tool switching problems with tool order constraints. *Discrete Applied Mathematics*, *347*(April), 249–262. https://doi.org/10.1016/j.dam.2023.12.031

Jun, H. B., Kim, Y. D., & Suh, H. W. (1999). Heuristics for a tool provisioning problem in a flexible manufacturing system with an automatic tool transporter. *IEEE Transactions on Robotics and Automation*, *15*(3), 488–496. https://doi.org/10.1109/70.768181

Kashyap, A. S., & Khator, S. K. (1990). Modeling of a Tool Shared Flexible Manufa Cturing System. *Proceedings of the 1994 Winter Simulation Conference*, *1249*(Oct), 395–403.

Kiran, A., & Krason, R. (1988). Automated tooling in a flexible manufacturing system. *Industrial Engineering*.

Konak, A., Kulturel-Konak, S., & Azizoğlu, M. (2008). Minimizing the number of tool switching instants in Flexible Manufacturing Systems. *International Journal of Production Economics*, *116*(2), 298–307. https://doi.org/10.1016/j.ijpe.2008.09.001

Laporte, G., Salazar-González, J. J., & Semet, F. (2004). Exact algorithms for the job sequencing and tool switching problem. *IIE Transactions (Institute of Industrial Engineers)*, *36*(1), 37–45. https://doi.org/10.1080/07408170490257871

Larranaga, P., Kuijpers, C. M. H., Murga, R. H., Inza, I., & Dizdarevic, S. (1999). Costs of secondary parasitism in the facultative hyperparasitoid Pachycrepoideus dubius: Does host size matter? *Artificial Intelligence Review*, *13*, 129–170. https://doi.org/10.1023/A

Oerlemans, A. (1992). *Production planning for flexible manufacturing systems*. University of Limburg, Maastricht, Limburg, Netherlands.

Paiva, G. S., & Carvalho, M. A. M. (2017). Improved heuristic algorithms for the Job Sequencing and Tool Switching Problem. *Computers and Operations Research*, *88*, 208–219. https://doi.org/10.1016/j.cor.2017.07.013

Pasha, N., Amoozad Mahdiraji, H., Razavi Hajiagha, S. H., Garza-Reyes, J. A., & Joshi, R. (2024). A multi-objective flexible manufacturing system design optimization using a hybrid response surface methodology. *Operations Management Research*, *17*(1), 135–151. https://doi.org/10.1007/s12063-023-00412-w

Privault, C., & Finke, G. (1995). Modelling a tool switching problem on a single NC-machine. *Journal of Intelligent Manufacturing*, *6*(2), 87–94. https://doi.org/10.1007/BF00123680

Rifai, A. P., Mara, S. T. W., & Norcahyo, R. (2022). A two-stage heuristic for the sequence-dependent job sequencing and tool switching problem. *Computers & Industrial Engineering*, *163*, 107813. https://doi.org/10.1016/j.cie.2021.107813

Salonen, K., Raduly-Baka, C., & Nevalainen, O. S. (2006). A note on the tool switching problem of a flexible machine. *Computers and Industrial Engineering*, *50*(4), 458–465. https://doi.org/10.1016/j.cie.2004.11.002

Shirazi, R., & Frizelle, G. D. M. (2001). Minimizing the number of tool switches on a flexible machine: An empirical study. *International Journal of Production Research*, *39*(15), 3547–3560. https://doi.org/10.1080/00207540110060888

Solimanpur, M., & Rastgordani, R. (2012). Minimising tool switching and indexing times by ant colony optimisation in automatic machining centres. *International Journal of Operational Research*, *13*(4), 465–479. https://doi.org/10.1504/IJOR.2012.046228

Tang, C. S., & Denardo, E. V. (1988). Models arising from a flexible manufacturing machine, part II: minimization of the number of switching instants. *Operations Research*, *36*(5), 778–784. https://doi.org/10.1287/opre.36.5.778

Tri Windras Mara, S., Sutoyo, E., Norcahyo, R., & Pratama Rifai, A. (2023). The job sequencing and tool switching problem with sequence-dependent set-up time. *Journal of King Saud University - Engineering Sciences*, *35*(1), 53–61. https://doi.org/10.1016/j.jksues.2021.02.015

Tzur, M., & Altman, A. (2004). Minimization of tool switches for a flexible manufacturing machine with slot assignment of different tool sizes. *IIE Transactions (Institute of Industrial Engineers)*, *36*(2), 95–110. https://doi.org/10.1080/07408170490245351

Wang, S., Zou, H., & Wang, S. (2022). A Tabu-GA-based parallel machine scheduling with restrained tool resources. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, *236*(1–2), 39–50. https://doi.org/10.1177/0954405420928691

Wierzbicki, A. P. (1982). A mathematical basis for satisficing decision making. *Mathematical Modelling*, *3*(5), 391–405. https://doi.org/10.1016/0270-0255(82)90038-0

Wierzbicki, A. P. (1986). On the Completeness and Constructiveness of Parametric Characterizations to Vector Optimization Problems. *OR Spectrum*, *8*, 73–87.

Wierzbicki, A. P. (1999). Reference point approaches. In *Multicriteria Decision Making: Advances in MCDM Models, Algorithms, Theory, and Applications by T. Gal, T.J. Stewart, T. Hanne (Eds.)* (p. Chapter 9).

Zhou, B. H., Xi, L. F., & Cao, Y. S. (2005). A beam-search-based algorithm for the tool switching problem on a flexible machine. *International Journal of Advanced Manufacturing Technology*, *25*(9–10), 876–882. https://doi.org/10.1007/s00170-003-1925-2