A Memetic Algorithm for the Tool Switching Problem

Jhon Edgar Amaya[1], Carlos Cotta[2], and Antonio J. Fernández[2]

[1] Universidad Nacional Experimental del Táchira (UNET)
Laboratorio de Computación de Alto Rendimiento (LCAR), San Cristóbal, Venezuela
jedgar@unet.edu.ve

[2] Dept. Lenguajes y Ciencias de la Computación, ETSI Informática,
University of Málaga, Campus de Teatinos, 29071 - Málaga, Spain.
{ccottap,afdez}@lcc.uma.es

Abstract. This paper deals with the Tool Switching Problem (ToSP), a well-known problem in operations research. The ToSP involves determining a job sequence and the tools to be loaded on a machine with the goal of minimizing the total number of tool switches. This problem has been tackled by a number of algorithmic approaches in recent years. Here, we propose a memetic algorithm that combines a problem-specific permutational genetic algorithm with a hill-climbing procedure. It is shown that this combined approach outperforms each of the individual algorithms, as well as an ad-hoc beam search heuristic defined in the literature for this problem.

1 Introduction

For some time now, the manufacturing industry is more and more often demanding flexible manufacturing systems (FMSs) as an alternative to traditional rigid production systems. This increasing interest is motivated by the fact that FMSs have the ability for self-adjustment to generate different products and/or change the order of product generation, i.e., they incorporate versatility and efficiency in mass production [1]. Basically, a FMS consists of a single machine that has several slots into which different tools can be loaded. Each slot just admits one tool, and each job executed on that machine requires a particular set of tools to be done. Jobs are sequentially executed, and therefore each time a job is to be processed, the corresponding tools must be loaded in the machine magazine. Since the number of available slots is limited, it may be required at some point to perform a tool switch, i.e., removing a tool from the magazine and inserting another one in its place. In this context, tool management is a challenging task that directly influences the efficiency of flexible manufacturing systems.

Although the order of tools in the magazine is often irrelevant, the need of performing a tool switching does depend on the order in which the jobs are executed. The Tool Switching Problem (ToSP) consists of finding an appropriate job sequence in which jobs will be executed, and an associated sequence of tool switches that minimizes the number of tool loading/unloading operations in the magazine. Clearly, this problem is specially interesting when the time needed to change a tool is a significant part of the processing time of all the jobs (and hence the tool switching policy will significantly affect the performance of the system). Different examples of the problem can be found in diverse areas such as electronic industry, metalworking industry, computer memory management, aeronautics, and manufacturing companies in general [1–5]. The ToSP has also a number of variants; see for example [6–8].

Despite the ToSP has been tackled via different optimization techniques (including exact and metaheuristics methods – see next section), to the best of our knowledge, no population-based algorithm (let alone a hybrid population-local approach) has been applied to its resolution. This paper gives a first step in this direction and demonstrates empirically that hybrid evolutionary techniques are effective solving strategies for this problem.

2 The Tool Switching Problem

Before getting to the algorithmic approaches considered for tackling the ToSP, let us describe more formally the problem, and previous related work in the literature.

2.1 Problem Formulation

As mentioned before, the ToSP involves scheduling a number of jobs on a single machine such that the resulting number of tool switches required is kept to a minimum. This can be formalized as follows: let a ToSP instance be represented by a 4-tuple, $I = (C, n, m, A)$ where

– C denotes the magazine capacity (i.e., number of available slots),
– n is the number of jobs to be processed,
– m is the total number of tools required to process all jobs (it is assumed that $C < m$; otherwise the problem is trivial).
– A is a $m \times n$ Boolean matrix termed the *incident matrix*. This matrix defines the tool requirements to execute each job, i.e., A_{ij} =`TRUE` if, and only if, tool i is required to execute job j.

The solution to such an instance is a sequence $J_1, \cdots, J_n$ determining the order in which the jobs are executed, and a sequence $T_1, \cdots, T_n$ of tool configurations ($T_i \subset \{1, \cdots, m\}$) determining which tools are loaded in the magazine at a certain time.

Let $\mathbb{N}_k = \{1, \cdots, k\}$ henceforth. An integer linear programming (ILP) formulation for the ToSP is shown below, using two sets of zero-one decision variables:

– $x_{jk} = 1$ if job $j \in \mathbb{N}_n$ is assigned to position $k \in \mathbb{N}_n$ in the sequence, and 0 otherwise – see Eqs. (2) and (3),
– $y_{ik} = 1$ if tool $i \in \mathbb{N}_m$ is in the magazine at time $k \in \mathbb{N}_n$, and 0 otherwise – see Eq. (4).

Processing each job requires a particular collection of tools loaded in the magazine. It is assumed that no job requires a number of tools higher than the magazine capacity, i.e., $\sum_{i=1}^{m} \delta_{A_{ij},\texttt{TRUE}} \leqslant C$ for all j, where δ_{ij} is Kronecker's delta. Tool requirements are reflected in Eq. (5). Following [1], we assume the initial condition $y_{i0} = 1$ for all $i \in \mathbb{N}_m$. This initial condition amounts to the fact that the initial loading of the magazine is not considered as part of the cost of the solution (in fact, no actual switching is required for this initial load). The objective function $F(\cdot)$ counts the number of switches that have to be done for a particular job sequence – see Eq. (1).

$$\min \; F(y) = \sum_{j=1}^{n} \sum_{i=1}^{m} y_{ij}(1 - y_{i,j-1}) \tag{1}$$

$$\forall j \in \mathbb{N}_n : \; \sum_{k=1}^{n} x_{jk} = 1 \tag{2}$$

$$\forall j \in \mathbb{N}_n : \; \sum_{k=1}^{n} x_{kj} = 1 \tag{3}$$

$$\forall k \in \mathbb{N}_n : \; \sum_{i=1}^{m} y_{ik} \leqslant C \tag{4}$$

$$\forall j, k \in \mathbb{N}_n \; \forall i \in \mathbb{N}_m : \; A_{ij} x_{jk} \leqslant y_{ik} \tag{5}$$

$$\forall j, k \in \mathbb{N}_n \; \forall i \in \mathbb{N}_m : \; x_{jk}, y_{ij} \in \{0, 1\} \tag{6}$$

It must be noted that the general definition above can be augmented if additional constraints are posed on tools or on the magazine. For example, it might be the case that different tools require slots of different sizes (or more than one slot). This is the so-called non-uniform ToSP [9]. Be as it may, we will consider in the following the uniform ToSP as previously defined.

2.2 Related Work

References to the ToSP can be found in the literature as early as in the 60's [2]. Since then, the uniform ToSP has been tackled via many different techniques. The late 80's contributed specially to solve the problem [10, 11]. Tang and Denardo [3] proposed an ILP formulation of the problem, and later Bard [1] described a non-linear integer programming formulation with a dual-based relaxation heuristic.

Heuristics-based constructive methods have also been applied to the problem. For instance Djellab *at al.* [12] tackled ToSP via a hypergraph representation and proposed a particular heuristic oriented towards minimizing the number of (total weighted) gaps in edge-projection where a projection is basically a permutation satisfying some specific constraints; the hypergraph is used to represent the relation among jobs and the needed tools. Also, Hertz *et al.* [13] described three constructive methods (i.e., FI, GENI and GENIUS) in which at each step both a job to be inserted in current tour and the best position in the tour are selected. Additionally, nearest neighbor (NN) and 2-opt search were also considered.

Exact methods have been also applied to the problem. For instance, Laporte *et al.* [14] propose two exact algorithms: a branch-and-bound approach and a a linear programming-based branch-and-cut algorithm. Precisely this last one is based on a new ILP formulation having a better linear relaxation than that proposed previously by Tang and Denardo [3]. It must be noted that these exact methods are inherently limited, since Oerlemans [15] and Crama *et al.* [16] proved formally that the ToSP is NP-hard for $C > 2$. This limitation was already highlighted in [14], where Laporte *et al.* reported that their algorithm was capable of managing instances with 9 jobs but it presented a very low success ratio for instances over 10 jobs.

Clustering/grouping methods have also been attended. For instance, Salonen et al. [17] attacked the uniform ToSP of printed circuit boards (PCBs) and described an algorithm that iterated the process of first determining a good (or even optimal) grouping of the PCBs for further sequencing them. A hierarchical job grouping technique, based on the Jaccard similarity coefficient as clustering criterion, is additionally employed to avoid identical groupings.

The use of metaheuristics has been also recently considered. So, several tabu search approaches [17–19] have been used in the literature. A different, and very interesting, approach has been described by Zhou *et al.* [20] that proposed a beam search algorithm. This method was demonstrated to be specially efficient and practical compared to other techniques previously presented. The reason provided to justify this efficiency was that the performance of the algorithm can be adjusted by changing the search width and the evaluation functions. We will return later to this approach since, due to its proved efficiency, it has been included in our experimental comparison.

In any case, to the best of our knowledge, no population-based algorithm has been proposed so far to solve this problem.

3 Solving the ToSP

The ToSP can be divided into three subproblems [21]: the first subproblem is *machine loading* and consists of determining the sequence of jobs; the second subproblem is *tool loading*, consisting of determining which tool to switch (if a switch is needed) before processing a job; finally, the third subproblem is *slot loading*, and consists of deciding where (i.e., in which slot) to place each tool. We are considering the uniform ToSP, and therefore only two subproblems have to be taken into account: machine loading and tool loading.

As it will be shown in next subsection, the tool loading subproblem can be optimally solved if the sequence of jobs is known by following a specific tool switching policy (described in Section 3.1) that guarantees to obtain the optimal number of tool switches for a given job sequence. Therefore, the metaheuristic effort is concentrated on the machine loading stage. For this purpose, we will consider the use of memetic algorithms (MAs). As already mentioned, the beam search heuristic defined by Zhou *et al.* [20] is used for comparison purposes in the experimental section (see Section 5) and will be also described for the sake of completeness in Section 3.2.

3.1 The KTNS Method for Tool Loading

In the context of the uniform ToSP, the cost of switching a tool is considered a constant (the same for all tools). Under this assumption, if the job sequence is fixed, the optimal tool switching policy can be determined in polynomial time using a greedy procedure termed *Keep Tool Needed Soonest* (KTNS) [1, 3][3]. The functioning of this procedure is as follows:

1. At any instant, insert all the tools that are required for the current job.
2. If one or more tools are to be inserted and there are no vacant slots on the magazine, keep the tools that are needed soonest. Let $J = \langle J_1, \cdots, J_n \rangle$ be the job sequence, and let $T_i \subset \mathbb{N}_m$ be the tool configuration at time i. Let $\Xi_{ik}(J) = \min\left\{j \mid (j > k) \wedge A_{i,J_j}\right\}$, that is, the next instant after time k at which tool i will be needed again given sequence J. If a tool has to be removed, the tool i^* maximizing $\Xi_{ik}(J)$ is chosen, i.e., remove the tools whose next usage is farther in time.

The KTNS policy states that when tool changes are necessary, the tools required for an upcoming job should be kept in the magazine. As a side remark, the tool loading problem is NP-hard in the non-uniform ToSP, even if the job sequence is known and unit loading/unloading costs are assumed [9].

3.2 A Beam Search Heuristic

The beam search algorithm defined by Zhou *et al.* [20] is a powerful approach to tackle the ToSP. Beam search is a derivative of branch-and-bound that uses a breadth-first traversal of the search tree, and incorporates a heuristic choice to keep at each level only the best (according to some *quality* measure) β nodes (the so-called *beam width*). This sacrifices completeness, but provides a very effective heuristic search approach.

The best β nodes are selected by one-step priority evaluation functions which estimate the cost of expanding the current solution. Note that nodes in the beam represent partial solutions (i.e., sequences of λ jobs $\langle J_1, \cdots, J_\lambda \rangle$ with $\lambda < n$; if $\lambda = n$ they actually represent solutions). For each node in the current level, a decision about which job will be added to the partial sequence is done. Let $\tau_j = \{i \mid A_{ij} = \texttt{TRUE}\}$, i.e., the set of tools required by job j. Two simple functions are used to ensure the quality of the solutions obtained:

$$h_1(J, k) = \#(\tau_{J_k} \cap \tau_{J_{k+1}}) \tag{7}$$

$$h_2(J, k) = \#(\tau_{J_k} \cup \tau_{J_{k+1}}) \tag{8}$$

[3] As Błażewicz and Finke [7] point out, the KTNS property was already known to Belady[2].

where $\#S$ is the cardinality of set S. Thus, $h_1(J, k)$ returns the number of common tools needed to process job J_k and candidate job J_{k+1} to be added to the partial job sequence. As to $h_2(J, k)$, it computes the total number of tools required to process job J_k and the candidate job J_{k+1}. These functions are used to select the beam nodes in each level, trying to maximize h_1 and using h_2 (to be minimized) to break ties.

4 A Memetic Approach to the ToSP

According to the previous discussion, the role of the MA is to determine the best job sequence, such that the total number of switches is minimized. Therefore, a permutational encoding arises as the natural way to represent solutions. Next sections will be devoted to describe our evolutionary algorithm, the neighborhood structures defined on the permutational encoding, and how these are used within the evolutionary algorithm to produce a memetic algorithm.

4.1 A Population-based Attack to the ToSP

We have considered a steady-state genetic algorithm (GA) to evolve promising job sequences: a single solution is generated in each generation, and inserted in the population replacing the worst individual. Selection is done by binary tournament. For recombination we initially explored two schemes: the well-known order crossover (OX), and a crossover scheme named *Alternating Position* (APX) that consists in select genes alternating of each parents [22]. Preliminary experiments that we executed showed that the employment of APX provided better results, in terms of solution quality, than using OX so that we elected APX as the crossover operator.

For the purposes of mutation we have considered the *block* neighborhood. This neighborhood is proposed for the ToSP in [19] and is based on swapping whole segments of contiguous positions. The resulting mutation operator is called Random Block Insertion (RBI) and works as follows:

1. A block length $b_l \in \mathbb{N}_{n/2}$ is uniformly selected at random.
2. The starting point of the block $b_s \in \mathbb{N}_{n-2b_l}$ is subsequently selected at random.
3. Finally, an insertion point b_i is selected, such that $b_s + b_l \leqslant b_i \leqslant n - b_l$, and the segments $\langle b_s, b_s + b_l\rangle$ and $\langle b_i, b_i + b_l\rangle$ are swapped.

Obviously, if the block length $b_l = 1$ then the operation reduces to a simple position swap, but this is not typically the case when performing mutation.

4.2 Local Search

A specific local search approach considered in this work is based on the well-known all-pairs neighborhood, i.e., two permutations are neighbors if they just differ in two positions of the sequence. A steepest-ascent hill climbing (HC) approach is defined on the basis of this neighborhood structure: the neighborhood $\mathcal{N}(x)$ of the current solution x is partially traversed, and the best solution found is taken as the new current solution, provided it is better than the current one (otherwise, it is considered that there is a stagnation).

Note that the exploration of the whole neighborhood is not executed as this process becomes more and more costly as the number of jobs increases e.g., for 50 jobs, the number of neighbors for a given candidate is 1225. For this reason only a set $\mathcal{N}_x \subset \mathcal{N}(x)$, with $\#\mathcal{N}_x = \alpha n$, is explored in each step of the HC method. The selection of candidates in $\mathcal{N}_x$ is done randomly.

4.3 The Memetic Proposal

On the basis of the previously described GA, we have defined a memetic algorithm (MA) by endowing the GA with a local search scheme. To be precise, we have used the all-pairs hill climbing algorithm defined in Section 4.2 just after the mutation stage. This local search is performed for a number of $maxEval$ evaluations, or until it stagnates. It must be also noted that the local search is always performed on every new individual generated.

In all our proposals (i.e., HC, GA and MA) the fitness of the candidate is obtained by the value returned after applying the KTNS method to the candidate. The objective is thus minimizing this value.

5 Experimental results

The experiments have been performed using four different algorithms: the beam search (BS) presented in [20] and described in Section 3.2, and the three algorithms proposed in preceding section, that is to say, a steady-state permutational GA, an steepest-ascent hill climbing (HC) search and a memetic algorithm. In the case of BS, five different equally-spaced values between 1 and 5 were considered for the beam width. As to the HC, the value $\alpha = 4$ was chosen for exploration of the neighborhood. Also, when HC is working alone, each time that local search is considered stagnated this is reactivated from a randomly selected candidate; when used inside the MA as an improvement operator, the HC is executed until reaching a stagnation or a maximum number of evaluations (i.e., exactly 1000; obviously there is no reactivation from a random point). As to the GA (and subsequently to the MA), an elitist generational model replacing the worst individual of the population ($popsize = 30$, $p_X = 1.0$, $p_M = 1/n$ where n is the number of jobs i.e., number of genes per individual) with binary tournament selection has been utilized; alternating position crossover (APX) is used, and mutation is done by applying the RBI operator. As to the MA, HC was always applied to each offspring generated after the mutation step (i.e., the probability of improvement was 1.0). The election of the parameter values (including the value for α) was done after an extensive phase of experimentation with many different values. The best combinations of the values were finally selected.

As far as we know, no standard data instance exists for this problem (at least publicly available) so that we have arbitrarily selected a wide set of problem instances that were attacked in [1, 13, 19, 20]; more specifically, 16 instances were chosen with values for the number of jobs, number of tools, and machine capacity ranging in [10,50], [9,60] and [4,25] respectively. Table 1 shows the different problem instances chosen for the experimental evaluation where a specific instance with n jobs, m tools and machine capacity C is labeled as $C\zeta_n^m$.

	$4\zeta_{10}^{10}$	$4\zeta_{10}^{9}$	$6\zeta_{10}^{15}$	$6\zeta_{15}^{12}$	$6\zeta_{15}^{20}$	$8\zeta_{20}^{15}$	$8\zeta_{20}^{16}$	$10\zeta_{20}^{20}$	$24\zeta_{20}^{30}$	$24\zeta_{20}^{36}$	$30\zeta_{20}^{40}$	$10\zeta_{30}^{25}$	$15\zeta_{30}^{40}$	$15\zeta_{40}^{30}$	$20\zeta_{40}^{60}$	$25\zeta_{50}^{40}$
Min.	2	2	3	3	3	3	3	4	9	9	11	4	6	6	7	9
Max.	4	4	6	6	6	8	8	10	24	24	30	10	15	15	20	20
Source	[13]	[1]		[1]			[1]	[1]	[1]	[1]						
	[19]	[20]	[20]	[20]	[13]	[19]	[20]	[20]	[20]	[20]	[20]	[19]	[13]	[19]	[13]	[19]

Table 1. Problem Instances considered in the experimental evaluation. The minimum and maximum of tools required for all the jobs is indicated in second and third rows respectively. Fourth row display the bibliography reference from which the problem instance was obtained.

Five different datasets[4] (i.e., incident matrixes or relations among tools and jobs) were generated randomly per instance. Each dataset was generated with the restriction, already imposed in previous works such as [13], that no job is *covered* by any other job in the sense that $\forall i, j \in [1, m] \wedge i \neq j : \tau_i \nsubseteq \tau_j$ where $\tau_k = \{h \mid A_{hk} = \texttt{TRUE}\}$ is defined as before, i.e., the set of tools required to process job k. The reason to enforce this constraint is to avoid the simplification of the problem by preprocessing techniques as done for instance in [1] and [20].

For GA, HC and MA, the algorithms were run 10 times (per instance and dataset) and a maximum of $\varphi n|n - C|$ evaluations[5] per run (with $\varphi > 0$). Preliminary experiments on the value of φ proved that $\varphi = 80$ is an appropriate value that allows to keep an acceptable relation between solution quality and computational cost. Regarding the BS algorithm, because of its deterministic nature, just one execution per dataset (and per value of beam width) was run and the algorithm was allowed to be executed until exhaustion (i.e., until completing the search). All methods were implemented in Java language version 1.5, and all the experiments were carried out on a PC computer Toshiba with Operating System Debian 1.6.x and 1.5 GHz/512 MB RAM. Tables 2 and 3 show the obtained results grouped by problem instance.

Several considerations can be done here. For instance, HC performed better, with respect to the best solution result, in most of the cases than BS; this is specially evident in the lower instances (i.e., those with smaller values of

[4] All datasets are available at *http://www.unet.edu.ve/∼jedgar/ToSP/ToSP.htm*

[5] Observe that the number of evaluations increases with the number of jobs (that is assumed to be related directly with the problem difficulty) and decreases when the magazine capacity increases (that, in certain form, it is also inversely related to the problem difficulty).

		GA	*MA*	HC	1 β	2 β	3 β	4 β	5 β	Evaluations
$4\zeta_{10}^{10}$	Av	8.88	8.68	8.96	10	9.8	9.6	9.6	9.6	4800
	SD	1.518	1.618	1.624	2.098	1.833	2.059	2.059	2.059	
	B	**7**	**7**	**7**	8	8	**7**	**7**	**7**	
$6\zeta_{10}^{15}$	Av	14.1	13.7	14.04	15.2	14.8	14.8	14.8	14.8	3200
	SD	2.012	2.09	2.097	1.47	1.47	1.47	1.47	1.47	
	B	**11**	**11**	**11**	13	13	13	13	13	
$4\zeta_{10}^{9}$	Av	8	7.86	8.04	8.4	8.4	8.4	8.4	8.4	4800
	SD	0.8	0.721	0.894	0.49	0.49	0.49	0.49	0.49	
	B	**7**	**7**	**7**	8	8	8	8	8	
$6\zeta_{15}^{12}$	Av	16.48	15.5	17	18.2	17.6	17.6	17.4	17.4	10800
	SD	2.138	1.982	1.929	0.748	1.02	1.02	1.2	1.2	
	B	13	**12**	14	17	16	16	16	16	
$6\zeta_{15}^{20}$	Av	23.62	22.38	24.08	26.2	25.8	25.2	25.2	25.2	10800
	SD	2.134	1.938	2.115	2.315	2.135	1.6	1.6	1.6	
	B	**20**	**20**	21	23	23	23	23	23	
$8\zeta_{20}^{15}$	Av	23.66	22.36	25.06	27	26	25.6	25.2	25.2	19200
	SD	3.603	3.576	3.652	3.95	4.05	4.271	4.118	4.118	
	B	**17**	**17**	20	21	21	20	20	20	
$8\zeta_{20}^{16}$	Av	28.5	26.66	29.18	29.4	29.4	29.4	29.4	29.4	19200
	SD	2.202	1.986	2.16	1.625	1.625	1.625	1.625	1.625	
	B	24	**23**	25	27	27	27	27	27	
$10\zeta_{20}^{20}$	Av	31.6	29.92	33.1	34.2	33.6	33.4	33.4	33.4	16000
	SD	2.828	2.357	2.147	3.187	2.871	2.8	2.8	2.8	
	B	**26**	**26**	29	30	30	30	30	30	
	mean	26.52	24.9	27.2	33	32.6	32.4	32.4	32.4	6400
$24\zeta_{20}^{30}$	σ	3.061	3.282	3.572	4.427	4.499	4.63	4.63	4.63	
	best	22	**20**	22	28	28	28	28	28	
	mean	48.16	46.54	49.6	54	54	53.8	53.8	53.4	6400
$24\zeta_{20}^{36}$	σ	8.999	8.807	9.481	8.198	8.198	8.328	8.328	7.605	
	best	**35**	36	37	45	45	45	45	45	
	mean	42.82	41.04	44.48	52.2	50.2	50.2	50.2	50.2	16000
$30\zeta_{20}^{40}$	σ	5.183	4.539	5.1	6.242	7.026	7.026	7.026	7.026	
	best	32	**31**	35	42	39	39	39	39	

Table 2. Results for $n \leqslant 20$ of GA, MA, HC, and BS considering several values ($1 \leq i \leq 5$) for the beam width β. Best results are marked in boldface.

n), although in some instances (i.e., $10\zeta_{30}^{25}$, $15\zeta_{40}^{30}$, $20\zeta_{40}^{60}$) BS returned better solutions. However, in average, HC behaves better than BS in the lower instances although it seems evident than BS performs better than HC when the number of jobs increases over 30. Also, it is evident that GA and MA provide the best results and clearly outperform both the HC and BS algorithms.

Focusing on the evolutionary proposals, notice firstly that the permutational GA provides comparable results, in terms of best solutions, to those of MA. In fact, the GA presents roughly the same performance than the MA, but becomes

		GA	MA	HC	1 β	2 β	3 β	4 β	5 β	Evaluations
$10\zeta^{25}_{30}$	mean	69.2	64.92	74.9	73.6	70.8	70.8	70.8	70.6	48000
	σ	3.4	1.573	1.9	1.02	1.47	1.47	1.47	1.497	
	best	**62**	**62**	69	72	68	68	68	68	
$15\zeta^{40}_{30}$	mean	105.08	100.86	111	111.6	110	109.2	107.8	107.8	36000
	σ	13.335	12.9	14.323	15.187	13.55	13.407	13.257	13.257	
	best	**81**	**81**	89	89	89	89	89	89	
$15\zeta^{30}_{40}$	mean	105.28	97.96	111.5	105.2	103.2	102.8	102.8	102.4	80000
	σ	8.775	7.887	8.273	8.518	9.579	9.745	9.745	9.972	
	best	88	**86**	98	96	93	93	93	93	
$20\zeta^{60}_{40}$	mean	220.6	211.88	231.1	221.8	220	218.8	218.6	218.6	64000
	σ	8.825	7.812	9.104	7.026	6.164	5.706	5.817	5.817	
	best	**200**	201	216	215	215	215	215	215	
$25\zeta^{40}_{50}$	mean	161.82	153.36	169	167.2	164	162.8	162.8	161.8	100000
	σ	13.671	13.52	13.485	12.906	12.554	12.335	12.335	12.156	
	best	141	**132**	146	152	147	147	147	147	

Table 3. Results for $n > 20$ of GA, MA, HC, and BS described in [20] considering several values ($1 \leq i \leq 5$) for the beam width β. Best results are marked in boldface.

clearly inferior when the average is considered as the MA always provides better results in this case.

	$4\zeta^{10}_{10}$	$4\zeta^{9}_{10}$	$6\zeta^{15}_{10}$	$6\zeta^{12}_{15}$	$6\zeta^{20}_{15}$	$8\zeta^{15}_{20}$	$8\zeta^{16}_{20}$	$10\zeta^{20}_{20}$	$24\zeta^{30}_{20}$	$24\zeta^{36}_{20}$	$30\zeta^{40}_{20}$	$10\zeta^{25}_{30}$	$15\zeta^{40}_{30}$	$15\zeta^{30}_{40}$	$20\zeta^{60}_{40}$	$25\zeta^{40}_{50}$
GA	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
MA	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

Table 4. Comparison: HC vs. GA/MA. Each cell displays the number of datasets in which HC is considered significantly better than GA/MA according to Wilcoxon ranksum test.

	$4\zeta^{10}_{10}$	$4\zeta^{9}_{10}$	$6\zeta^{15}_{10}$	$6\zeta^{12}_{15}$	$6\zeta^{20}_{15}$	$8\zeta^{15}_{20}$	$8\zeta^{16}_{20}$	$10\zeta^{20}_{20}$	$24\zeta^{30}_{20}$	$24\zeta^{36}_{20}$	$30\zeta^{40}_{20}$	$10\zeta^{25}_{30}$	$15\zeta^{40}_{30}$	$15\zeta^{30}_{40}$	$20\zeta^{60}_{40}$	$25\zeta^{40}_{50}$
HC	0	0	0	0	0	3	0	2	1	2	2	4	5	5	5	4
MA	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 5. Comparison: GA vs. HC/MA. Each cell displays the number of datasets in which GA is considered significantly better than HC/MA according to Wilcoxon ranksum test.

A non-parametrical statistical test –Wilcoxon ranksum– was executed on the results returned by all the executions performed by HC, GA and MA. Then a

	$4\zeta^{10}_{10}$	$4\zeta^{9}_{10}$	$6\zeta^{15}_{10}$	$6\zeta^{12}_{15}$	$6\zeta^{20}_{15}$	$8\zeta^{15}_{20}$	$8\zeta^{16}_{20}$	$10\zeta^{20}_{20}$	$24\zeta^{30}_{20}$	$24\zeta^{36}_{20}$	$30\zeta^{40}_{20}$	$10\zeta^{25}_{30}$	$15\zeta^{40}_{30}$	$15\zeta^{30}_{40}$	$20\zeta^{60}_{40}$	$25\zeta^{40}_{50}$
HC	2	0	0	5	5	5	5	5	3	5	5	5	5	5	5	5
GA	0	0	1	2	4	2	5	4	3	3	1	4	4	5	5	4

Table 6. Comparison: MA vs. HC/GA. Each cell displays the number of datasets in which MA is considered significantly better than HC/GA according to Wilcoxon ranksum test.

comparison between each method with the other two was executed; Tables 4, 5 and 6 show the results of the comparison of HC vs. GA and MA, GA vs. HC and MA, and MA vs. HC and GA respectively. Each cell in the tables indicates the number of times that the corresponding algorithm is significantly better than the other one with respect to the 5 datasets per instance. For example, a 4 appearing in Table 6 for the instance $10\zeta^{25}_{30}$ in the row of GA means that the MA behaves significantly better, according to the statistical test, than the GA in 4 of the 5 datasets that were used to solve the specific problem instance. These results corroborate our preliminar considerations and one can observe that the GA outperforms HC when the number of jobs increases and that the MA outperforms both HC and GA and also behaves in general evidently better than the GA.

6 Conclusions and future work

We have tackled here the tool switching problem (in its uniform version) and have proposed three methods to attack it. Two of the methods are, as far as we know, the first evolutionary approaches to handle the tool switching problem. Combining ideas from the realm of evolutionary computation and hill climbing methods, we have specifically devised an evolutionary proposal that takes the form of a memetic algorithm. An empirical evaluation was executed in order to prove the validity and performance of the proposed techniques. For comparison purposes, we have considered in the experimentation the beam search method described in [20], as this was demonstrated to be specially efficient and practical compared to another techniques previously published.

The experiments demonstrate that the three methods (i.e., a hill climbing search, a genetic algorithm and a memetic algorithm) provide encouraging results and are capable of improving the results obtained by the BS. Focusing on our proposals, the memetic algorithm outperforms both a permutational genetic algorithm and a steepest-ascent hill climbing method. A statistical test demonstrates that the MA is significantly superior to the other two proposals.

We believe that there is room for improvement. For instance, it would be interesting to prove alternative methods to HC such as tabu search or variable neighborhood search. In this case, it would be necessary to obtain a good balance between intensification and exploration in the local search component of the MA; in our proposal we have leaned towards exploration by evaluating only a part of the whole neighborhood space, and have obtained encouraging results, but perhaps a more intensive strategy can also produce valuable results. We also plan to analyze new instances and variants of the problem [6–8].

Acknowledgements

Second and third authors were partially supported by Spanish MCyT under contracts TIN2004-7943-C04-01 and TIN2005-08818-C04-01.

References

1. Bard, J.F.: A heuristic for minimizing the number of tool switches on a flexible machine. IIE Transactions **20**(4) (1988) 382–391
2. Belady, L.: A study of replacement algorithms for virtual storage computers. IBM Systems Journal **5** (1966) 78–101
3. Tang, C.S., Denardo, E.V.: Models arising from a flexible manufacturing machine, part I: minimization of the number of tool switches. Oper. Res. **36**(5) (1988) 767–777
4. Privault, C., Finke, G.: Modelling a tool switching problem on a single nc-machine. Journal of Intelligent Manufacturing **6**(2) (april 1995) 87–94
5. Shirazi, R., Frizelle, G.: Minimizing the number of tool switches on a flexible machine: an empirical study. International Journal of Production Research **39**(15) (2001) 3547–3560
6. Kashyap, A.S., Khator, S.K.: Modeling of a tool shared flexible manufacturing system. In: WSC '94: Proceedings of the 26th conference on Winter simulation, San Diego, CA, USA, Society for Computer Simulation International (1994) 986–993
7. Błażewicz, J., Finke, G.: Scheduling with resource management in manufacturing systems. European Journal of Operational Research **76** (1994) 1–14
8. Hong-Bae, J., kim Yeong-Dae, Suh, S.H.W.: Heuristics for a tool provisioning problem in a flexiblemanufacturing system with an automatic tool transporter. IEEE Transactions on Robotics and Automation **15**(3) (1999) 488–496
9. Crama, Y., Moonen, L.S., Spieksma, F.C., Talloen, E.: The tool switching problem revisited. European Journal of Operational Research **182**(2) (2007) 952–957
10. ElMaraghy, H.: Automated tool management in flexible manufacturing. Journal of Manufacturing Systems **4**(1) (1985) 1–14
11. Kiran, A., Krason, R.: Automated tooling in a flexible manufacturing system. Industrial Engineering **20** (1988) 52–57
12. Djellab, H., Djellab, K., Gourgand, M.: A new heuristic based on a hypergraph representation for the tool switching problem. International Journal of Production Economics **64**(1-3) (March 2000) 165–176
13. Hertz, A., Laporte, G., Mittaz, M., Stecke, K.: Heuristics for minimizing tool switches when scheduling part types on a flexible machine. IIE Transactions **30** (1998) 689–694
14. Laporte, G., Salazar-González, J., Semet, F.: Exact algorithms for the job sequencing and tool switching problem. IIE Transactions **36**(1) (January 2004) 37–45
15. Oerlemans, A.: Production planning for flexible manufacturing systems. Ph.d. dissertation, University of Limburg, Maastricht, Limburg, Netherlands (October 1992)

16. Crama, Y., Kolen, A., Oerlemans, A., Spieksma, F.: Minimizing the number of tool switches on a flexible machine. International Journal of Flexible Manufacturing Systems **6** (1994) 33–54
17. Salonen, K., Raduly-Baka, C., Nevalainen, O.S.: A note on the tool switching problem of a flexible machine. Computers & Industrial Engineering **50**(4) (2006) 458–465
18. Hertz, A., Widmer, M.: An improved tabu search approach for solving the job shop scheduling problem with tooling constraints. Discrete Applied Mathematics **65** (1993) 319–345
19. Al-Fawzan, M.A., Al-Sultan, K.S.: A tabu search based algorithm for minimizing the number of tool switches on a flexible machine. Comput. Ind. Eng. **44**(1) (2003) 35–47
20. Zhou, B.H., Xi1, L.F., Cao, Y.S.: A beam-search-based algorithm for the tool switching problem on a flexible machine. The International Journal of Advanced Manufacturing Technology **25**(9-10) (Mayo 2005) 876–882
21. Tzur, M., Altman, A.: Minimization of tool switches for a flexible manufacturing machine with slot assignment of different tool sizes. IIE Transactions **36**(2) (2004) 95–110
22. Larrañaga, P., Kuijpers, C., Murga, R., Inza, I., Dizdarevic, S.: Genetic algorithms for the travelling salesman problem: A review of representations and operators. Articial Intelligence Review **13** (1999) 129–170