Applied Soft Computing 65 (2018) 214–229

Contents lists available at ScienceDirect

Applied Soft Computing

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

A hybrid method of 2-TSP and novel learning-based GA for job sequencing and tool switching problem

Ehsan Ahmadi[a,*], Boris Goldengorin[a,b], Gürsel A. Süer[a], Hadi Mosadegh[c]

[a] Department of Industrial and Systems Engineering, Russ College of Engineering and Technology, Ohio University, Athens, OH 45701, USA
[b] Mathematics and Statistical Sciences Department, College of Liberal Arts and Sciences, University of Colorado Denver, Denver, CO 80217, USA
[c] Department of Industrial Engineering, Amirkabir University of Technology, Tehran, Iran

ARTICLE INFO

Article history:
Received 4 June 2017
Received in revised form
19 December 2017
Accepted 30 December 2017
Available online 3 January 2018

Keywords:
Combinatorial optimization
Job scheduling
Tool switches
Genetic algorithm
Q-learning
Reinforcement learning

ABSTRACT

One of the well-known problems in single machine scheduling context is the Job Sequencing and Tool Switching Problem (SSP). The SSP is optimally sequencing a finite set of jobs and loading restricted subset of tools to a magazine with the aim of minimizing the total number of tool switches. It has been proved in the literature that the SSP can be reduced to the Job Sequencing Problem (JSeP). In the JSeP, the number of tool switches from the currently processed job to the next job depends on the sequencing of all predecessors. In this paper, the JSeP is modeled as a Traveling Salesman Problem of Second Order (2-TSP). We call the induced JSeP by 2-TSP as the Job Sequencing Problem of Second Order (2-JSeP) with a different objective function formulation from JSeP and prove that 2-JSeP is *NP*-hard. Then the Assignment Problem of Second Order (2-AP) and Karp-Steele patching heuristic are incorporated to solve 2-JSeP. The obtained solution, however, does not guarantee the optimal sequence and are used to seed a Dynamic Q-learning-based Genetic Algorithm (DQGA) to improve the solution quality. Q-learning, which is a kind of reinforcement learning method, is used to learn from the experience of selecting the order of mutation and crossover operators in each generation of the genetic algorithm. The computational results on 320 benchmark instances show that the proposed DQGA is comparable to the state-of-the-art methods in the literature. The DQGA even outperforms the existing methods for some instances, as could improve the reported "best-known solutions" in notably less time. Finally, through the statistical analysis, the performance of DQGA is compared with those of non-learning genetic algorithms.

1. Introduction and background

The Flexible Manufacturing Systems (FMS) containing Computer Numerical Control (CNC) machines are able to process various kinds of jobs when the tools required for processing the jobs are available in the machine's magazine. The magazine has C slots for tools and each tool occupies one slot. C is also known as the *capacity* of the magazine and no job requires more than C tools. If the tools required by a job are not on the machine's magazine, a tool switch, i.e. removing one of the existing tool and then inserting a new one, is necessary [1]. Usually, tool switching is time consuming and therefore, finding a sequence of the jobs minimizing the total number of tool switches is an important issue within the FMS.

In this paper, we investigate how to find the sequence of jobs with the objective of minimizing the total number of tool switches. This problem is known as the *Job Sequencing and Tool Switching Problem* (SSP) [1] and is applicable to different areas such as computer memory allocation [2,3], metalworking industry [4] chemical manufacturing plant, insurance company, pharmaceutical packaging [5] and mailroom insert planning [6,7]. Another important application of SSP is in the electronics industry [8–11], where different types of printed circuit boards (PCBs) need to be processed on a component assembly machine. The different PCBs can be translated to jobs and the different electronic components, which are required to be loaded into the feeder of the machine, can be translated to tools. In addition, Shirazi and Frizelle [12] investigated the real-world application of known methods of solving SSP in seven different companies in aerospace, CNC producers, and tooling industries.

Tang and Denardo [1] discussed that the SSP can be represented by the following two sub-problems.

1. The *Job Sequencing Problem* (JSeP), which is a scheduling problem to identify an optimal sequence of jobs to be processed on the machine.

* Corresponding author.
E-mail address: ea162814@ohio.edu (E. Ahmadi).

https://doi.org/10.1016/j.asoc.2017.12.045
1568-4946/© 2018 Elsevier B.V. All rights reserved.

2. The *Tool Replacement Problem* (TRP), indicates uploading the required tools to support the processing of the provided sequence of jobs (see JSeP), with the aim of minimizing the total number of tool switches.

Tang and Denardo [1] showed that given a sequence of jobs, TRP can be solved by applying a greedy algorithm called *Keep Tool Needed Soonest* (*KTNS*) in polynomial time. Therefore, SSP can be reduced to JSeP, which is combinatorial, and in fact the JSeP becomes the main problem to be addressed.

Crama et al. [13] proved that the JSeP is *NP*-hard. Several studies have considered JSeP and modeled the problem with exact algorithms. Laporte et al. [14] have reported the optimality of solved JSeP for some benchmark instances up to 25 jobs with respect to their Integer Linear Programming (ILP) models. Yanasse et al. [15] proposed an enumeration scheme based on partial orders that was able to improve the lower bound for the SSP instances. Catanzaro et al. [16] have investigated 3 other ILP formulations and report some improvements of the lower bounds compared to Laporte et al. [14] counterparts. Furrer and Mütze [6] proposed a branch and bound algorithm to deal with multiple objectives of the JSeP including minimization of the number of tool switches and the number of machine stops over time.

A stream of literature for solving JSeP is dedicated to developing heuristic and meta-heuristic techniques. Bard [17] formulated JSeP as a nonlinear integer program and solved with a dual-based relaxation heuristic to find a good local solution for instances containing at most 25 jobs. Al-Fawzan and Al-Sultan [18] proposed a variety of tabu search algorithms and made a comparison to identify a better algorithm over some random test instances. Denizel [19] developed a Lagrangean decomposition approach for JSeP and represented its superior performance up to 30 jobs against that of Tang and Denardo [20]. Konak et al. [21] proposed two tabu search methods to deal with the JSeP. Their methods are evaluated using the instances of Tang and Denardo [20], Denizel [19] and randomly generated instances up to 210 jobs. Their methods were able to find high-quality solutions in reasonable time.

Amaya et al. [4] proposed a memetic algorithm coupled with a genetic algorithm and a hill climbing procedure. Their computational results showed that the combined algorithm is superior to the individual ones. In another study, Amaya et al. [22] evaluated the combination of the memetic algorithm with three local search techniques, hill climbing, tabu search, and simulated annealing. They concluded that memetic algorithm works better with hill climbing than other methods.

Crama et al. [13] proposed several heuristics based on traveling salesman problem (TSP) and then assessed their computational performances. Hertz et al. [23] also presented a heuristic based on TSP with a different definition of "distance" between tools. They have shown that their heuristic is superior to the method proposed by Crama et al. [13]. Chaves et al. [24] also proposed a hybrid heuristic based on the biased random key genetic algorithm and the clustering search. They have conducted experiments over the benchmark instances of Yanasse et al. [15], Crama et al. [13], and reported better solution quality for some instances. Paiva et al. [25] proposed an Iterated Local Search (ILS) meta-heuristic and their results show the superiority of their method over Crama et al. [13],Yanasse et al. [15] and Catanzaro et al. [16] counterparts.

Ghiani [26] formulated JSeP as a TSP in a particular case when the number of tools needed to process each job equals to the capacity of the magazine. The authors showed that for a given sequence of N jobs $(1, 2, \ldots, i-1, i, j, j+1, \ldots, N)$, the total number of tool switches between job i and job j depends on the predecessor of job i, i.e. $(1, 2, \ldots, i-1)$.

In this paper, we formulate the JSeP as Traveling Salesman Problem of Second Order (2-TSP) introduced by Jäger and Molitor [27]. In the case of JSeP, the 2-TSP measures the number of tool switches by considering the current job and its predecessor while the TSP reduction just considers the current job to be switched to the next one. In other words, given the sequence (i, j, k), the 2-TSP measure the number of tool switches required in transition from job j to job k by considering both job i and job j, not only job j. Therefore, in comparison with the TSP, the 2-TSP provides an improved definition of "distance", i.e. number of tool switches, between two successive jobs. We call the induced JSeP by 2-TSP as Job Sequencing Problem of Second Order (2-JSeP) with a different objective function formulation from JSeP and prove that 2-JSeP is *NP*-hard. By means of computational study, we also show that finding a solution to the corresponding 2-JSeP will be a better approximation to the unknown optimal solution of the JSeP compared to its counterpart solution returned by the TSP reduction of JSeP. The approximating solution within the proposed 2-JSeP model will be found by Assignment Problem of Second Order (2-AP) proposed by Jäger and Molitor [27] and Karp-Steele patching heuristic [28]. The solution found is utilized to supply a proposed dynamic Q-learning-based genetic algorithm (DQGA) to solve JSeP.

In a genetic algorithm (GA), selecting type of operators and their sequence has always been a concern. Starkweather et al. [29] extensively compared six types of crossover genetic operators on TSP and concluded that adding a mutation operator might change the performance of an algorithm. In a classical GA, the crossover operator is applied before executing the mutation operator (CM). However, a modified GA can be implemented with reverse order of operators, i.e. first mutation is applied and then crossover (MC). In this paper, a Q-learning procedure, which is a kind of reinforcement learning method, is utilized to learn from the experience of selecting the order of mutation and crossover operators in each generation of the GA. The results of the proposed DQGA are compared with non-learning algorithms of DGA-CM and DGA-MC, in addition to the state-of-the-art methods in the literature.

The rest of the paper is organized as follows. In Section 2, we first discuss more details of the JSeP specifications and then formulate the 2-JSeP. In Section 3, a heuristic method is presented to quickly solve 2-JSeP. The proposed DQGA seeded by the 2-JSeP is also presented in this section. In Section 4, we define all necessary parameters for our DQGA. In Section 5, the results of the computational experiments with 320 benchmark instances of Crama et al. [13] and Catanzaro et al. [16], are reported and compared with the state-of-the-art method proposed by Paiva et al. [25]. The statistical analysis to identify the dominant algorithm between DQGA, DQGA-CM and DQGA-MC is also covered in Section 5. Finally, conclusions and possible future research directions related to this study are covered in Section 6.

2. Problem formulation

In this section, we first describe the JSeP in more depth and then formulate the 2-JSeP.

2.1. Job sequencing problem

JSeP consists of a set of jobs $J = \left\{1, 2, \ldots, n\right\}$ which require to be processed on a single machine. There is a set of tools $T = \left\{1, 2, \ldots, m\right\}$ available for the machine and each job j requires a specific subset $T_j \subseteq T$ of tools be loaded on the machine before being processed. The magazine capacity is C, i.e., the number of available slots, and each tool occupies one slot. It is assumed that $C < m$, otherwise the problem is trivial [30]. The JSeP is the problem of finding an optimal sequence of jobs minimizing the total number of tool switches. An example of the JSeP borrowed from Tang and Denardo

Table 1
An example of the JSeP [1] with magazine capacity of 4 slots.

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

Table 2
Tool-Job matrix.

	Job									
Tool	1	2	3	4	5	6	7	8	9	10
1	1	1	0	0	0	0	1	0	0	1
2	0	0	1	0	0	0	0	0	0	1
3	0	1	0	0	0	1	0	1	0	0
4	1	0	0	0	0	0	0	0	0	1
5	0	1	0	0	0	0	1	1	1	0
6	0	0	1	0	1	0	0	0	0	0
7	0	0	1	1	0	0	1	0	1	0
8	1	0	1	0	0	0	0	1	0	0
9	1	0	0	0	0	0	1	0	0	0

Table 3
Example of JSeP after applying dominance rule.

	Job					
Tool	1	2	3	7	8	10
1	1	1	0‘	1	0	1
2	0	0	1	0	0	1
3	0	1	0	0	1	0
4	1	0	0	0	0	1
5	0	1	0	1	1	0
6	0	0	1	0	0	0
7	0	0	1	1	0	0
8	1	0	1	0	1	0
9	1	0	0	1	0	0

[1] is shown in Table 1 with the magazine capacity of 4 slots and a set of 9 tools.

Table 1 gives the required set of tools in columns corresponding to jobs. For example, job 1 requires tools 1, 4, 8 and 9 on the machine. Another representation of the JSeP can be presented as shown in Table 2. The corresponding matrix is called Tool-Job incidence matrix $A = \left[a_{ij}\right]$, where $a_{ij} = 1$ if tool i is needed to process job j, and 0 otherwise.

In the JSeP, some of the jobs might be dominated by others. In Table 2, for example, job 9 is dominated by job 7 since its tool set (5 and 7) is the subset of the tool set required by job 7 (1, 5, 7, and 9). Tang and Denardo [1] applied their *dominance rule* to remove the dominated jobs without any impact on the final optimal schedule. In fact, the removed jobs can be added right after the dominant job in the final solution without any increase in the number of the tool switches. Table 3 shows the Tool-Job matrix where four dominated jobs 4, 5, 6 and 9 were eliminated through the dominance rule by jobs 3, 2 and 7.

Table 4 shows the global optimal solution to this example which is calculated by complete enumeration of 6! possible permutations of all 6 jobs. In Table 4 the symbols $\oplus$ and $\otimes$ indicate the insertion and deletion of the corresponding tools with the purpose to form the required set of tools for the currently processed job. The tools which are shown by a preceding arrow indicate that the corresponding tools are preserved in the given sequence of jobs. For example, starting from job 7, tools 1, 5, 7, and 9 are needed to be loaded on the machine which induce the total tool switches equal to 4. The next job 8 needs the tools 3, 5, and 8 only. Hence, two tools among 1, 7, and 9 should be removed and tools 3 and 8 should be loaded. According to the *KTNS* rule, tool 1 is preserved because it

Table 4
The solution of the example shown in Table 2. Symbols $\oplus$ and $\otimes$ indicate the tools that need to be inserted and removed, respectively. Symbol $\rightarrow$ shows the tool that is required for processing the job, but had been already loaded on the machine for previous jobs. So it should not be counted as the number of tool switches.

	Job											
Tool		7		8		2		1		10		3
1	$\oplus$	1	$\rightarrow$	1		1		1		1		0
2		0		0		0		0	$\oplus$	1	$\otimes$	1
3		0	$\oplus$	1		1	$\otimes$	0		0		0
4		0		0		0	$\oplus$	1		1	$\otimes$	0
5	$\oplus$	1		1		1	$\otimes$	0		0		0
6		0		0		0		0		0	$\oplus$	1
7	$\oplus$	1	$\otimes$	0		0		0		0	$\oplus$	1
8		0	$\oplus$	1		1		1	$\rightarrow$	1		1
9	$\oplus$	1	$\otimes$	0	$\rightarrow$	0	$\oplus$	1	$\otimes$	0		0
Total tool switches	4		6		6		8		9		11	

will be used for job 2. So, tools 7 and 9 would be removed and tools 3 and 8 are loaded and consequently, the total tool switches become 6. Likewise, the other jobs are processed and finally, the total number of tool switches equals 11.

Now the jobs removed through dominance rule can be added to obtain the final sequence. As described above, jobs 4 and 5 are inserted after job 3, job 6 is inserted after job 2 and job 9 is inserted after job 7. An alternative optimal sequence will be {7, 9, 8, 2, 6, 1, 10, 3, 4, 5} with the total tool switches of 11.

2.2. Job sequencing problem of second order

Jäger and Molitor [27] introduced a new combinatorial optimization problem called Traveling Salesman Problem of Second Order (2-TSP). We applied this concept to formulate the 2-JSeP as provided by the following Lemma.

Lemma 1. *2-JSeP is NP-hard in the strong sense.*

Proof. It has been proven by Jäger and Molitor [27] that 2-TSP is *NP*-hard. We try to show that 2-TSP is not more difficult than 2-JSeP which means 2-TSP is reducible to 2-JSeP in a polynomial time complexity. For this purpose, let $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{A})$ be a complete directed graph where a 2-TSP of size $(n+1)$ is defined with $n \geq 2$. Consider $\boldsymbol{V} = \{v_0, v_1, \cdots, v_n\}$ as the set of vertices, and $\boldsymbol{A} = \left\{\left(v_i, v_j\right)\right\}$, with $i, j = 0,\ 1,\ \cdots,\ n$ and $i \neq j$, as the set of arcs with a distance function $d : |\boldsymbol{V}| \times |\boldsymbol{V}| \rightarrow \mathbb{Z}^{+} \cup \{\infty\}$ such that $0 \leq d_{ij} < \infty$, where $i \neq j$, and $d_{ij} = \infty$ otherwise. In our instance of 2-TSP, it is assumed that $d_{0j} = d_{j0} = 0$ for $j = 1,\ 2,\ \cdots n$. Now, consider the cost function $C : |\boldsymbol{V}| \times |\boldsymbol{V}| \times |\boldsymbol{V}| \rightarrow \mathbb{Z}^{+} \cup \{\infty\}$ such that, $C_{ijk} = \infty$ where $i = j$ or $i = k$ or $j = k$, and $1 \leq C_{ijk} = d_{ij} + d_{jk} < \infty$ otherwise. The problem is finding a Hamiltonian tour such as $\left\{v_{\sigma[0]}, v_{\sigma[1]}, \cdots, v_{\sigma[n]}\right\}$ with minimizing the second order travel cost

$$TC = \sum_{j=0}^{n-2} C_{\sigma[j],\sigma[j+1],\sigma[j+2]} + C_{\sigma[n-1],\sigma[n],\sigma[0]} + C_{\sigma[n],\sigma[0],\sigma[1]},$$

where each vertex is visited exactly once, and solution $\sigma[\cdot]$ returns index of the vertex in our tour.

The reduction is performed here by constructing an instance of 2-JSeP based on the constructed instance of 2-TSP. without losing the generality, assume the instances of 2-JSeP when the number of tools needed to process each job equals the capacity of the magazine ($|T_j| = C$). Let $\boldsymbol{J} = \left\{j_0, j_1, \cdots, j_n\right\}$ denote the set of jobs such that $|\boldsymbol{J}| = |\boldsymbol{V}|$. Also, let N_{ij} be the number of tool switches required between jobs i and j, where $i, j = 0,\ 1,\ \cdots n$. Assign all distance parameters into number of required tool switches as $N_{ij} \longleftarrow d_{ij}$. In this case, j_0 represents a dummy node which is dominated by other jobs. The reduction procedure is performed in the polynomial complexity order of $O\left(n^2\right)$. Here, we define $q_{ijk} = N_{ij} + N_{jk}$ as the number of tool switches from job j to job k considering that job i was the predecessor with $i, j, k = 0, 1, \cdots n$. Our instance construction of 2-JSeP is completed by minimizing the second order total number of tool switches of job sequence $\left\{j_{\Sigma[0]}, j_{\Sigma[1]}, \cdots, j_{\Sigma[n]}\right\}$ as

$$Q = \sum_{j=0}^{n-2} q_{\Sigma[j],\Sigma[j+1],\Sigma[j+2]} + q_{\Sigma[n-1],\Sigma[n],\Sigma[0]} + q_{\Sigma[n],\Sigma[0],\Sigma[1]},$$

where solution $\Sigma[\cdot]$ returns index of the job in the sequence. Now, let $\left\{j_{\Sigma^*[1]}, j_{\Sigma^*[2]}, \cdots, j_{\Sigma^*[n]}\right\}$ be the optimal solution of the constructed 2-JSeP. It is obvious that the travel cost of optimal Hamiltonian tour in the 2-TSP instance can be derived in terms of the total tool switches of optimal sequence in the 2-JSeP instance as $TC^* = \sum_{j=0}^{n-2} C_{\Sigma^*[j],\Sigma^*[j+1],\Sigma^*[j+2]} + C_{\Sigma^*[n-1],\Sigma^*[n],\Sigma^*[0]} + C_{\Sigma^*[n],\Sigma^*[0],\Sigma^*[1]}$.

Finally, it would be sufficient to transform Σ^* to σ^*. Therefore, the equivalent optimal Hamiltonian tour is derived by translating sequence of jobs into a tour of visiting vertices as $\sigma^*[j] \longleftarrow \Sigma^*[j]$ which is implemented in $O(n)$ complexity, and hence the proof is complete. □

The Proof of Lemma 1 provides detailed formulation of 2-JSeP based on 2-TSP and vice versa. Therefore, we can utilize algorithms developed for 2-TSP to solve 2-JSeP efficiently. Then the obtained solution can be used as a good initial seed in GA to deal with JSeP.

3. Solution method

In this section, we present a solution approach to solve 2-JSeP, followed by describing the basic concept of DQGA and its implementation.

3.1. Assignment-Patching heuristic

Jäger and Molitor [27] proposed an assignment-patching heuristic to solve 2-TSP. We have adopted this heuristic to tackle our 2-JSeP as follows.

3.1.1. Assignment problem of second order

Assignment Problem of Second Order (2-AP) is a natural extension of the Assignment Problem (AP). In the AP, we have n workers and n jobs and each worker performs one job and each job is performed by one worker. Now we wish to find an assignment solution such that the sum of the costs of the individual assignments of workers to the jobs is minimized. On the other hand, 2-AP is a problem of finding a two-to-one-mapping $f: J \times J \rightarrow J$ so that the cost $\sum_{i=1}^{N} C[i, f(i), f(f(i))]$ is minimum. 2-AP can be formulated as follows:

Objective function:

$$Z = \min \sum_{i \in V} \sum_{\substack{j \in V \\ j \neq i}} \sum_{\substack{k \in V \\ k \neq i \neq j}} C_{ijk} . X_{ijk} \tag{1}$$

Subject to:

$$\sum_{\substack{j \in V \\ j \neq i}} \sum_{\substack{k \in V \\ k \neq i \neq j}} X_{ijk} = 1 \quad \forall i \in V \tag{2}$$

$$\sum_{\substack{i \in V \\ i \neq j}} \sum_{\substack{k \in V \\ k \neq i \neq j}} X_{ijk} = 1 \quad \forall j \in V \tag{3}$$

$$\sum_{\substack{i \in V \\ i \neq k}} \sum_{\substack{j \in V \\ j \neq i \neq k}} X_{ijk} = 1 \quad \forall k \in V \tag{4}$$

$$\sum_{\substack{k \in V \\ k \neq i \\ k \neq j}} X_{ijk} = \sum_{\substack{k \in V \\ k \neq i \\ k \neq j}} X_{kij} \quad \forall i \in V, j \in V, j \neq i \tag{5}$$

$$X_{ijk} \in \{0, 1\} \tag{6}$$

In this model, X_{ijk} is the binary variable which is 1 if pair (i, j), precedes job k and is 0 otherwise. C_{ijk} is the number of tool switches from job j to job k given the predecessor is job i.

Equations (2), (3) and (4) mean that each vertex appears exactly once as the first, second or third vertex in a path of three vertices of a tour. Equation (5) specifies that if there is a path from (i, j) to k, then there is another path from k to (i, j). Solving this 2-AP leads to generate unknown number (Y) of subtours which are locally optimal. Now, in order to obtain a Hamiltonian tour, the Karp-Steele patching algorithm [28] is used. This patching algorithm is described in the next section.

3.1.2. Patching algorithm

Given the solution of 2-AP containing Y cycles, which is a good lower bound for optimum 2-JSeP, patching algorithm tries to generate a Hamiltonian tour. Karp and Steele [28] suggested that in each step, two cycles containing the most vertex should be patched in such a way that it has the minimum patching cost. Assume that Y_1 and Y_2 are two cycles that are going to be patched. For each i and j in Y_1 and k and l in Y_2 where i and l succeed j and k respectively, two arcs (i, k) and (j, l) are inserted and two arcs (i, j) and (k, l) are deleted with the patching cost of $(C_{i,k} + C_{j,l} - C_{i,j} - C_{k,l})$. In fact, patching cost shows the amount of objective function which is sacrificed through patching operation.

Jäger and Molitor [27] extended the patching concept to 2-TSP. Without losing the general idea, we adopted the patching algorithm for 2-TSP to our 2-JSeP in the following way: for $Y_1 = (V_1, W_1)$ and $Y_2 = (V_2, W_2)$ consider vertices $\{i, j, k, l\} \in V_1$ and $\{n, p, q, t\} \in V_2$, arcs $(i, j, k), (j, k, l) \in W_1$ and $(n, p, q), (p, q, t) \in W_2$, then patching is finding a tour with the minimum of following term out of the $M = V_1 . V_2$ total number of possible patches.

$$(TS_{i,j,q} + TS_{j,q,t} + TS_{n,p,k} + TS_{p,k,l}) - (TS_{i,j,k} + TS_{j,k,l} + TS_{n,p,q} + TS_{p,q,t}) \tag{7}$$

where $TS_{a,b,c}$ with $a \in V_1 \cup V_2, b \in V_1 \cup V_2, c \in V_1 \cup V_2$ and $a \neq b \neq c$ is the number of tool switches from job b to job c with the predecessor of job a. In order to illustrate more details of the patching algorithm, consider an example of 10 jobs and 10 tools with given tools-jobs matrix as shown in Table 5 (capacity = 4). $TS_{a,b,c}$ is calculated and presented in Table A1 of Appendix A.

Table 5
Tools-Job Matrix for JSeP.

Jobs		1	2	3	4	5	6	7	8	9	10
Tools	1	0	0	1	1	0	1	0	0	1	1
	2	0	1	0	0	0	0	0	0	0	1
	3	0	1	0	0	0	1	0	0	0	0
	4	0	0	0	0	0	0	0	0	1	1
	5	0	0	0	1	1	0	1	1	1	0
	6	1	0	1	0	0	1	1	0	0	0
	7	1	0	0	0	0	0	0	1	0	0
	8	0	0	0	0	1	1	1	1	1	0
	9	0	0	1	1	0	0	1	0	0	0
	10	0	0	0	0	1	0	0	0	0	1

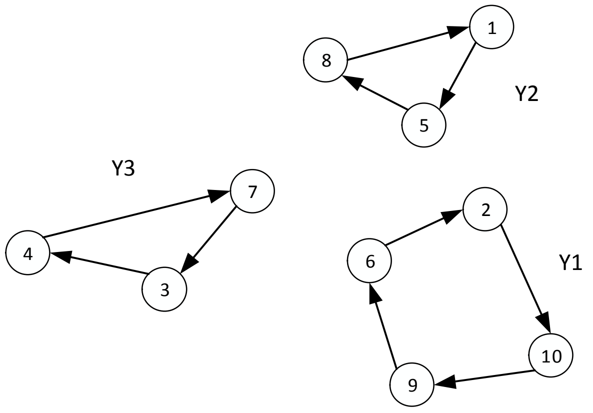


Fig. 1. 2-AP solution of 2-TSP for 10 jobs.

Solving the 2-AP of this example yielded 3 cycles as shown in Fig. 1. Patching Operation of *Y* cycles consist of *Y* – 1 operation of deletion and insertion of two vertices simultaneously.

In our instance, since Y*1* is the largest cycle, it should be selected for the first patching operation. *Y2* and *Y3* can be selected arbitrarily. Table 6 shows 12 possible patching for *Y1* and *Y2*. In particular, the first raw of Table 6 indicates that two arcs (6,2,10) and (2,10,9) from *Y1* and two arcs (1,5,8) and (5,8,1) from *Y2* should be deleted and then four arcs (6,2,8), (2,8,1), (1,5,10) and (5,10,9) are inserted to obtain a tour with 5 unit patching cost. Patching number 9 has the minimum cost and therefore, it is selected for the second patching operation (see Fig. 2). This procedure continues until all cycles are patched and a Hamiltonian tour shown in Fig. 3 is obtained.

Finally, the Hamiltonian tour which is generated by the patching algorithm, must be translated to the sequence of the jobs. Therefore, we numerate all possibilities of the sequences, which equals to *n* for *n* jobs. Considering the number of tool switches, these sequences are locally optimal, and can lead to a good starting point for our proposed GA described in the next section.

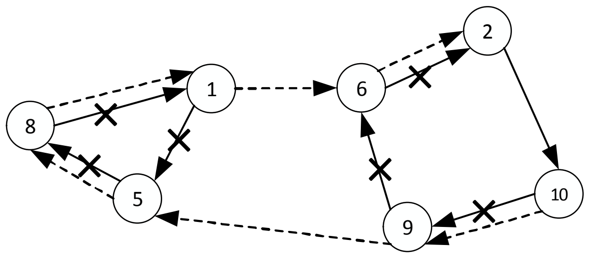


Fig. 2. Best patching for Y1 and Y2 in the first patching operation with the patching cost of 0 unit. Numbers in circles stand for the jobs. Arcs with cross sign are deleted and arcs shown as dash lines are inserted.

3.2. Dynamic Q-learning based genetic algorithm

In order to solve JSeP, we have utilized the GA, which is seeded by the solutions obtained from solving 2-JSeP. GA is an evolutionary algorithm based on stochastic search, which is originally proposed by Holland [31] to solve intractable optimization problems. It is a nature-inspired technique based on the Darwin's "survival of the fittest" that starts with a set of generated potential solutions, called the population. GA is extensively applied to solve scheduling problems (See for example [32–34]) and basically attempts to combine individuals via crossover operators and keep diversity in each population via mutation operators. An advantage of GA is its capability to perform global search [35]. In a GA, decision variables need to be coded as an array, i.e., chromosome, and each chromosome has an associated fitness, which is calculated by use of the objective function. Chromosomes with better fitness value have more chance to survive to the next generation and consequently, the algorithm converges to an optimum or near optimum solution.

3.2.1. Chromosome representation

Chromosome representation is an important issue for the GA in terms of computational time and also generating a feasible solution. In the current study, a permutation-based chromosome represen-

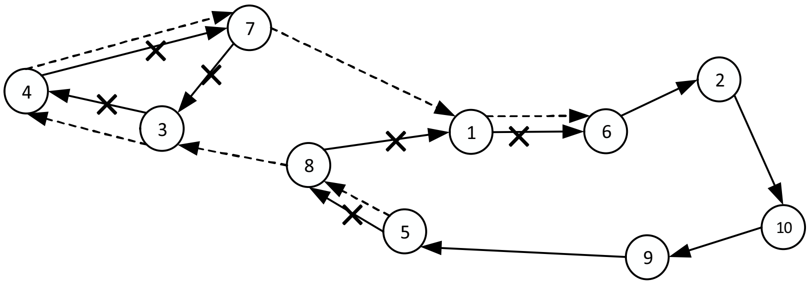


Fig. 3. A possible patching in the second patching operation. Numbers in circles stand for the jobs. Arcs with cross sign are deleted and arcs shown as dash lines are inserted.

Table 6
Calculation of first patching operation.

Possible Patch #	Cycle1				Cycle2				Deleted Arcs	Inserted Arcs	Patching Cost
	i	*j*	*k*	*l*	*n*	*p*	*q*	*t*			
1	6	2	10	9	1	5	8	1	(6,2,10), (2,10,9), (1,5,8), (5,8,1)	(6,2,8), (2,8,1), (1,5,10), (5,10,9)	(3+4+6+5)-(3+5+3+2)=5
2	6	2	10	9	5	8	1	5	(6,2,10), (2,10,9), (5,8,1), (8,1,5)	(6,2,1), (2,1,5), (5,8,10), (8,10,9)	(2+5+4+6)-(3+5+2+2)=5
3	6	2	10	9	8	1	5	8	(6,2,10), (2,10,9), (8,1,5), (1,5,8)	(6,2,5), (2,5,8), (8,1,10), (1,10,9)	(3+4+5+6)-(3+5+2+3)=5
4	2	10	9	6	1	5	8	1	(2,10,9), (10,9,6), (1,5,8), (5,8,1)	(2,10,8), (10,8,1), (1,5,9), (5,9,6)	(6+4+5+4)-(5+4+3+2)=5
5	2	10	9	6	5	8	1	5	(2,10,9), (10,9,6), (5,8,1), (8,1,5)	(2,10,1), (10,1,5), (5,8,9), (8,9,6)	(5+4+3+4)-(5+4+2+2)=3
6	2	10	9	6	8	1	5	8	(2,10,9), (10,9,6), (8,1,5), (1,5,8)	(2,10,5), (10,5,8), (8,1,9), (1,9,6)	(5+3+3+6)-(5+4+2+3)=3
7	10	9	6	2	1	5	8	1	(10,9,6), (9,6,2), (1,5,8), (5,8,1)	(10,9,8), (9,8,1), (1,5,6), (5,6,2)	(3+2+5+4)-(4+3+3+2)=2
8	10	9	6	2	5	8	1	5	(10,9,6), (9,6,2), (5,8,1), (8,1,5)	(10,9,1), (9,1,5), (5,8,6), (8,6,2)	(4+3+4+4)-(4+3+2+2)=4
9	10	9	6	2	8	1	5	8	(10,9,6), (9,6,2), (8,1,5), (1,5,8)	(10,9,5), (9,5,8), (8,1,6), (1,6,2)	(3+2+3+4)-(4+3+2+3)=0
10	9	6	2	10	1	5	8	1	(9,6,2), (6,2,10), (1,5,8), (5,8,1)	(9,6,8), (6,8,1), (1,5,2), (5,2,10)	(4+2+5+4)-(3+3+3+2)=4
11	9	6	2	10	5	8	1	5	(9,6,2), (6,2,10), (5,8,1), (8,1,5)	(9,6,1), (6,1,5), (5,8,2), (8,2,10)	(3+3+3+5)-(3+3+2+2)=4
12	9	6	2	10	8	1	5	8	(9,6,2), (6,2,10), (8,1,5), (1,5,8)	(9,6,5), (6,5,8), (8,1,2), (1,2,10)	(4+3+3+5)-(3+3+2+3)=4

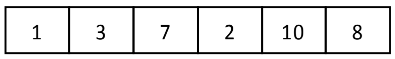


Fig. 4. Chromosome representation. Each number represents a non-dominated job.

tation, which is common in sequencing problems, has been used. It covers whole solution space and any permutation of jobs can correspond to a feasible solution. Every chromosome constitutes n (number of non-dominated jobs) genes and each gene holds a number which is a label of a job. The sequence of the jobs is the label of the genes from the first one to the nth one. Fig. 4 shows an example of chromosome representation for the example given in Table 3.

3.2.2. Initialization

Seeding is a technique which is used by researchers to increase initial population quality and guide the GA towards the optimum solution [36]. Bentley [37] shows that for TSP problems, the quality of initial population would affect the running time and the final solution quality. Therefore, in this paper, seeds are generated in two steps by incorporating the solutions obtained from the heuristic described in Section 3.1. The first step consists of all possible sequences which can be obtained from the Hamiltonian tour. This step contains n sequences for n jobs. In the second step, ($nPop$-n) chromosomes are generated by randomly selecting subtours (cycles) and then converting them into the sequence. One should note that the subtours do not contain identical jobs and any random selection will generate a feasible solution. These subtours are locally optimal and therefore, lead to obtaining a high-quality population as the starting point for GA.

3.2.3. Selection strategy

Selection strategy is a procedure that selects individuals for reproduction by moving them into the mating pool. In this paper, the roulette-wheel technique is chosen to select parents based on their selection probability as given by the equations (8), (9) and (10).

$$P_{ind} = \frac{F_{ind}}{F_{tot}} \quad ind = 1, 2, \ldots, nPop \tag{8}$$

$$F_{ind} = \frac{1}{TS_{ind}} \tag{9}$$

$$F_{tot} = \sum_{ind=1}^{nPop} F_{ind} \tag{10}$$

Where, P_{ind} is the probability of choosing the indth individual; $nPop$ is the population size; F_{ind} is the indth individual fitness; F_{tot} is the total fitness of all individuals in the current generation; and TS_{ind} is the number of tool switches for the indth individual. Is should be noted that given a sequence of the jobs, TS_{ind} is calculated through the *KTNS* rule.

3.2.4. Genetic operators

The performance of a GA to achieve better schedule depends on the performance of the genetic operators that are used [38]. Therefore, using appropriate operators is fundamental to extend any GA. Furthermore, performing genetic operators may produce an infeasible schedule and, consequently, need to perform a repair mechanism that could be time consuming. Thus, it is more practical to design the operators that maintain feasibility of the schedule and avoid the repair mechanisms [34]. In this paper, we have utilized a partially-mapped crossover operator (PMX) and a swap mutation operator. The procedure of each operator is described in the following sections.

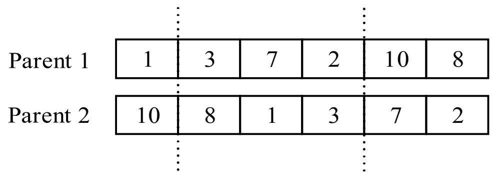


Fig. 5. Random cut points.

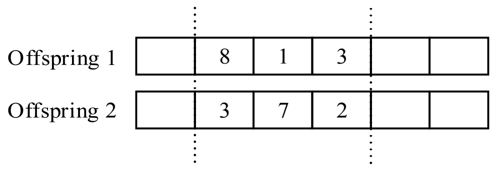


Fig. 6. Exchange mapping sections.

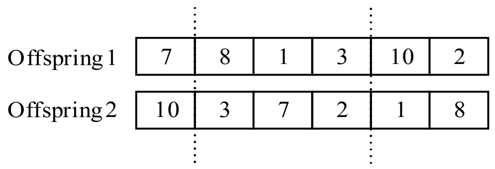


Fig. 7. Final offspring of crossover operator.

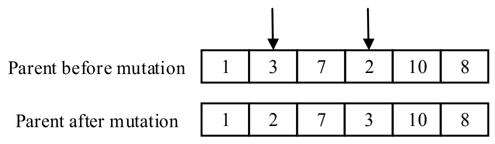


Fig. 8. Final offspring of mutation operator.

3.2.4.1. Partially-Mapped crossover operator (PMX).

To generate offspring, two chromosomes would be selected as the parents. Then two random cut points are selected as represented in Fig. 5 by dash line.

The genes between two cut points are called mapping sections. The mapping section of the first parent is copied into the second offspring, as shown in Fig. 6, and the mapping section of the second parent is copied into the first offspring. The mapping gens for this example are: $3 \leftrightarrow 8 \leftrightarrow 2$ and $7 \leftrightarrow 1$.

The empty genes of offspring 1 and offspring 2 would be filled up by parent 1 and parent 2, respectively. For example, the first gene of the offspring 1 would be 1, the same as the first gene of parent 1. However, 1 already exists in offspring 1. So, due to mapping of $7 \leftrightarrow 1$, we put 7 in the first gene of offspring 1. The fifth gene in offspring 1 would be 10 which is directly taken from the fifth gene of parent 1. Finally, the last gene in offspring 1 is 2, which is chosen from mapping $3 \leftrightarrow 8$ and $2 \leftrightarrow 3$. Fig. 7 shows the final offspring.

3.2.4.2. Swap mutation operator.

This operator randomly selects two genes in a chromosome and interchanges the values. For example, consider the parent shown in Fig. 8. Suppose that genes two and four are selected randomly, then these two genes are exchanged and the corresponding parent is depicted in Fig. 8.

3.2.5. Local search

To present the local search developed in this paper, let us introduce the concept of 0-block and 1-block defined by Crama et al. [13]. They defined 0-block and 1-block as a maximal subset of consecutive 0 and 1, respectively, in a given row of the Tool-Job matrix. They further discussed that by ignoring the *KTNS* algorithm, each 1-block in the Tool-Job matrix induces a tool switch. Therefore, the number of 1-block in the Tool-Job matrix is an overestimation of the tool switches.

Paiva et al. [25] proposed a local search method to reduce the number of 1-block. We have developed a local search by inspiring from their work, embedded in the GA, in the following way: given a set Ω of the best solutions in a generation of the GA, the local search attempts to group 1-blocks in each row of Tool-Job matrix. We first identified the rows with two or more 1-blocks and then for each selected row, pairs of 1-blocks are grouped in either forward or backward direction. Finally, all obtained non-worsen solutions in terms of tool switches, compared to the best solution of the population, are added to the current population for further evaluation to be selected for the next generation.

Let us recall the Tool-Job matrix presented in Table 5. Given a sequence $E = [3, 1, 2, 10, 6, 5, 4, 9, 7, 8]$ with 15 tool switches, after permuting of columns according to the E, the first row of the Tool-Job matrix would be [1001101100], which contains three 1-blocks [3], [10, 6]and [4, 9]. For each pair of 1-blocks $\{i, j\}$, in the forward direction, block i is moved to before and after of block j. Conversely, in the backward direction, block j is moved to before and after of block i. The decision of forward and backward movement is made randomly. Let us consider a forward movement for the first two 1-blocks, i.e. $\{[3], [10, 6]\}$, which would result in two sequences $[1, 2, 3, 10, 6, 5, 4, 9, 7, 8]$ and $[1, 2, 10, 6, 3, 5, 4, 9, 7, 8]$ with 15 and 14 tool switches, respectively. Since both movements resulted in non-worse solutions, they are added to the current population. The pseudo-code for the proposed local search is presented in Algorithm 1.

```
Algorithm 1: Local Search
 Input: A set Ω of the best solutions in the current population
 for $E \in \Omega$ do
      In Tool-Job matrix permute columns according to $E$
      for each row of Tool-Job matrix do
          for each pair of 1-blockes $i$ and $j$ do
              if $rand() \geq 0.5$
                 Forward movement: move 1-block $i$ before and after 1-block $j$ and obtain
                 local optimum $E_{LO}$ and $E'_{LO}$
              else
                 Backward movement: move 1-block $j$ before and after 1-block $i$ and obtain
                 local optimum $E_{LO}$ and $E'_{LO}$
              end if
                Compare $E_{LO}$ and $E'_{LO}$ with the best solution of the current population and
                add non-worsen ones to the population $\Omega'$
          end for
      end for
 end for
 Output: $\Omega'$
```

The local search is executed once the GA could not improve the best-obtained solution for a pre-specified number of generations and is applied on α percentage of the best solutions within the current population.

3.2.6. Similarity function for chromosome removal

To avoid early convergence to local optima, in every generation we have calculated a *similarity rate* between each pair of chromosomes (including parents and off-springs) and then if similarity rate is above a threshold θ, one of them, i.e. the one with higher number of tool switches, is removed from the population. This will guarantee that no duplicate chromosomes are in the population. To calculate the similarity rate between two sequences of E_i and E_j, we first define an array $X = [e_h]$, where $e_h = 1$ if position h in E_i and E_j is occupied by the same job and $e_h = 0$ otherwise. Then similarity rate defined as the summation of two or more consecutive 1 in the X array, divided by the number of jobs. To illustrate the above procedure, consider two sequences of $E_1 = [1, 6, 2, 10, 9, 5, 8, 3, 4, 7]$ and $E_2 = [1, 10, 8, 6, 9, 5, 2, 3, 4, 7]$ that result in an array of $X = [1, 0, 0, 0, 1, 1, 0, 1, 1, 1]$, which finally ends up with a similarity rate of $\left(\frac{5}{10}\right) = 0.5$.

3.2.7. Q-learning algorithm

In the context of machine learning, learning techniques can be categorized into four groups: (1) *supervised learning*, which takes place based on a labeled dataset and frequently used for classification purpose [39]; (2) *unsupervised learning*, which is exploring an unlabeled dataset to identify similar instances and cluster then into particular class [40]; (3) *semi-supervised learning*, which is a process of learning with both labeled and unlabeled datasets [40]; (4) *reinforcement learning*, which is a learning technique derived from supervised technique and dynamic programming to simulate the process of human learning. The learners are able to learn from their own experience through the feedback that they have received from the environment. The feedback is in the form of a reward or penalty, which is earned as a consequence of their action [41].

The Q-learning algorithm is a kind of reinforcement learning method and is basically proposed to obtain a near-optimal policy for Markov decision process problems [42,43]. Due to the stochastic nature of the GA, it can be treated as a stochastic process and a control mechanism, such as Q-learning, can be incorporated to guide algorithm towards the global optimum solution.

Q-learning algorithm consists of a set of states in the system and a set of actions allowed in each state. The system is initially in a state S_0 and tacking action a can cause a transition to another state. The quality of the taken action is then rewarded, and the value of reward will be used as a base to calculate Q-values associated to each tuple state-action, which are sorted in a look-up table. The look-up table is a two-dimensional table that its rows represent states and its columns stand for actions. In a given state, the action corresponding to the highest Q-value is called *greedy* action and the other actions are called *non-greedy* actions [44]. The algorithm should be able to exploit the prior knowledge, i.e. greedy action, to earn reward and also should explore non-greedy actions for better future decision. This mechanism is implemented by a parameter called ε-*greedy*.

With this consideration, we designed a Q-learning algorithm, to help with the decision of selecting the sequence of the genetic operators in each generation. Let us define S as a set of states and $A(i)$ as a set of actions in state i as follows:

$$S = \begin{cases} 1 & 0 \leq R(a) \leq \eta_1 \\ 2 & \eta_1 + 1 \leq R(a) \leq \eta_2 \\ 3 & \eta_2 + 1 \leq R(a) \leq \eta_3 \\ 4 & \eta_3 + 1 \leq R(a) \leq \eta_4 \\ 5 & \eta_4 + 1 \leq R(a) \leq nPop \end{cases} \tag{11}$$

$$A(i) = \begin{cases} CM & \text{first apply crossover then mutation} \\ MC & \text{first apply mutation then crossover} \end{cases} \tag{12}$$

where $R(a)$ denotes the number of chromosomes within the population that are replaced by executing action a. Clearly, $R(a)$ is a stochastic process that can take integer values in the interval of $[0, nPop]$. $Q(i, a)$ is the Q-value associated with state i and action a, which is updated according to the expression (13).

$$Q(i, a) \longleftarrow \left(1 - \beta\right) Q(i, a) + \beta \left[r(i, a, j) + \lambda \max_{b \varepsilon A(j)} Q(j, b) \right] \tag{13}$$

where $r(i, a, j)$ is the immediate reward received in transition from state i to state j after taking action a. The parameters $\beta \in [0, 1]$ is learning rate and $\lambda \in [0, 1]$ is discount factor. For more details of Q-learning algorithm, readers are referred to [44].

In this paper, the immediate reward is composed of two measures. The first measure accounts for the ability of selected action for generating off-springs with higher quality in terms of the objective function. It implies that a set of off-springs with closer cost, i.e.

mean tool switches, to the current population is preferred over others. Therefore, we have defined a *cost distance* measure as presented in (14).

$$CD(a) = \frac{1}{\frac{\sum_{ind \in Pop_a} TS_{ind}}{|Pop_a|} - \frac{\sum_{ind \in Pop} TS_{ind}}{|Pop|} + 1} \tag{14}$$

where Pop_a is the population of off-springs obtained by applying action a, i.e. either CM or MC operators, on the current population of Pop. The TS_{ind} is the number of tool switches (objective function) for the individual int. Theoretically, $CD(a)$ might become a negative number in the case when $\frac{\sum_{ind \in Pop_a} TS_{ind}}{|Pop_a|} < \left(\frac{\sum_{ind \in Pop} TS_{ind}}{|Pop|} + 1\right)$. However, in practice, our observation shows that the operators employed in this paper always generate off-springs with a mean of tool switches higher than the mean of tool switches of the current population. Therefore, the value of $CD(a)$ lies in the interval of $(0, 1]$.

The second measure, defined by [45], is a normalized *diversity* measure, ranging from 0 to 1, which accounts for the number of chromosomes with unique objective function values in the population of off-springs obtained from executing of action a. This measure is calculated as shown in equation (15).

$$DV(a) = \frac{|U| - 1}{|Pop| - 1} \tag{15}$$

where $U \subseteq Pop_a$, and $\forall TS_{ind} \in U$ and $TS_{ind'} \in U$, $TS_{ind} \neq TS_{ind'}$ if $ind \neq ind'$.

Both of the two above-mentioned measures ranging from 0 to 1 and the higher values are more desired. Therefore, they can linearly be combined to formulate immediate reward function as $r(i, a, j) = CD(a) + DV(a)$.

3.2.8. Restarting-phase

In order to enforce GA to dynamically and effectively search the solution space, we have introduced a procedure called *restarting-phase*. While running the algorithm, when population converges into a local optimum and remains in this situation for a number of consecutive non-improving generations, restarting-phase occurs. Restarting-phase is designed to enhance population diversity once we do not expect to exit from this local optimum solution. In the restarting-phase, we keep a set of best-obtained solution and then generate population based on the procedure described in Section 3.1.

3.2.9. DQGA flowchart

The flowchart of the proposed DQGA, has been illustrated in Fig. 9. The algorithm starts with the initial solution generated by the heuristic described in Section 3.1. In each generation, Q-learning method is applied to make a decision about selecting an appropriate operator sequence, i.e. CM or MC. Before evaluating chromosomes based on their objective function values to be selected for the next generation, the pair of chromosomes are evaluated with respect to their similarity rate and some of them are removed from the population. So that the chromosomes selected for the next generation are diverse enough to avoid early convergence. Furthermore, a local-search and restarting-phase mechanism are also incorporated to escape out of local optima. The algorithm terminates after reaching the maximum number of generations.

4. Experimental design

In this section, parameter tuning is described first and then we have presented the benchmark instances that are used for evaluation of our algorithm.

4.1. Parameter setting

The mathematical model for 2-AP was solved in CPLEX version 12.6 and all other algorithms are coded and executed in MATLAB R2017a on an Intel® Core™ i7 CPU @3.40 GHz computer with 16 GB of RAM. We have conducted multiple experiments to figure out the best parameter values for the DQGA. The parameters are described as follows. *nPop* considered to be 100 individuals, crossover and mutation rates are 0.9 and 0.7, respectively. Local search is performed after 15 generations of consecutive non-improving best-obtained solution with $\alpha = 20\%$. In the last 50 generations the local search is performed every 10 generations with $\alpha = 50\%$. The number of consecutive non-improving generations, once algorithm immaturely converged, to apply restarting-phase is 20. The parameter settings of Q-learning algorithm are listed as follows: the system is initially is state 1, all initial Q-values equals to 0, $\eta_1 = 10$, $\eta_2 = 20$, $\eta_3 = 35$, $\eta_4 = 50$, $\beta = 0.15$, $\gamma = 0.75$, and $\varepsilon = 0.35$. For similarity rate, θ is considered to be 0.8. The stopping criterion is determined based on the number of generations, which is 350, and 10 replications are considered to run for each benchmark instance.

4.2. Benchmark problems

In order to test the performance of the proposed algorithms, 32 datasets each of which consist of 10 instances (total 320 instances), proposed by Catanzaro et al. [16] and Crama et al. [13], have been selected. These datasets were also used in Chaves et al. [24] and Paiva et al. [25] studies. Small instances have 10 and 15 jobs, medium instances 30 jobs and large instances 40 jobs. Table 7 shows the configuration of each dataset. We will compare our computational results with the state-of-the-art results reported by Paiva et al. [25] since their algorithm has superior performance to all previous studies that used Catanzaro et al. [16] and Crama et al. [13] datasets.

5. Experimental results

Numerical results are divided into three categories: correlation of objective functions, performance of algorithms and statistical analysis.

5.1. Correlation of objective functions

In order to show the appropriateness of formulating the JSeP as a 2-TSP, which we called it 2-JSeP, rather than formulating as a TSP, we compared the correlation between 2-JSeP and JSeP versus TSP and JSeP. To do so, we generated 1000 random sequences for one instance in each dataset and evaluated the sequences by three objective functions of JSeP, 2-JSeP and TSP. As it can be seen in Table 8, there are higher correlations between the objective functions of JSeP and 2-JSeP than JSeP and TSP in all datasets. The correlation coefficient itself between JSeP and 2-JSeP is relatively high, on average 0.69 and 0.70 for Catanzaro et al. [16] and Crama et al. [13] datasets, respectively. It means finding a better solution to 2-JSeP, which is used as an initial population for genetic algorithms, would be a better solution to JSeP as well.

Fig. 10 also illustrates scatter plots of 1000 randomly generated sample sequences from instance1 in *datB1* for JSeP against 2-JSeP and TSP.

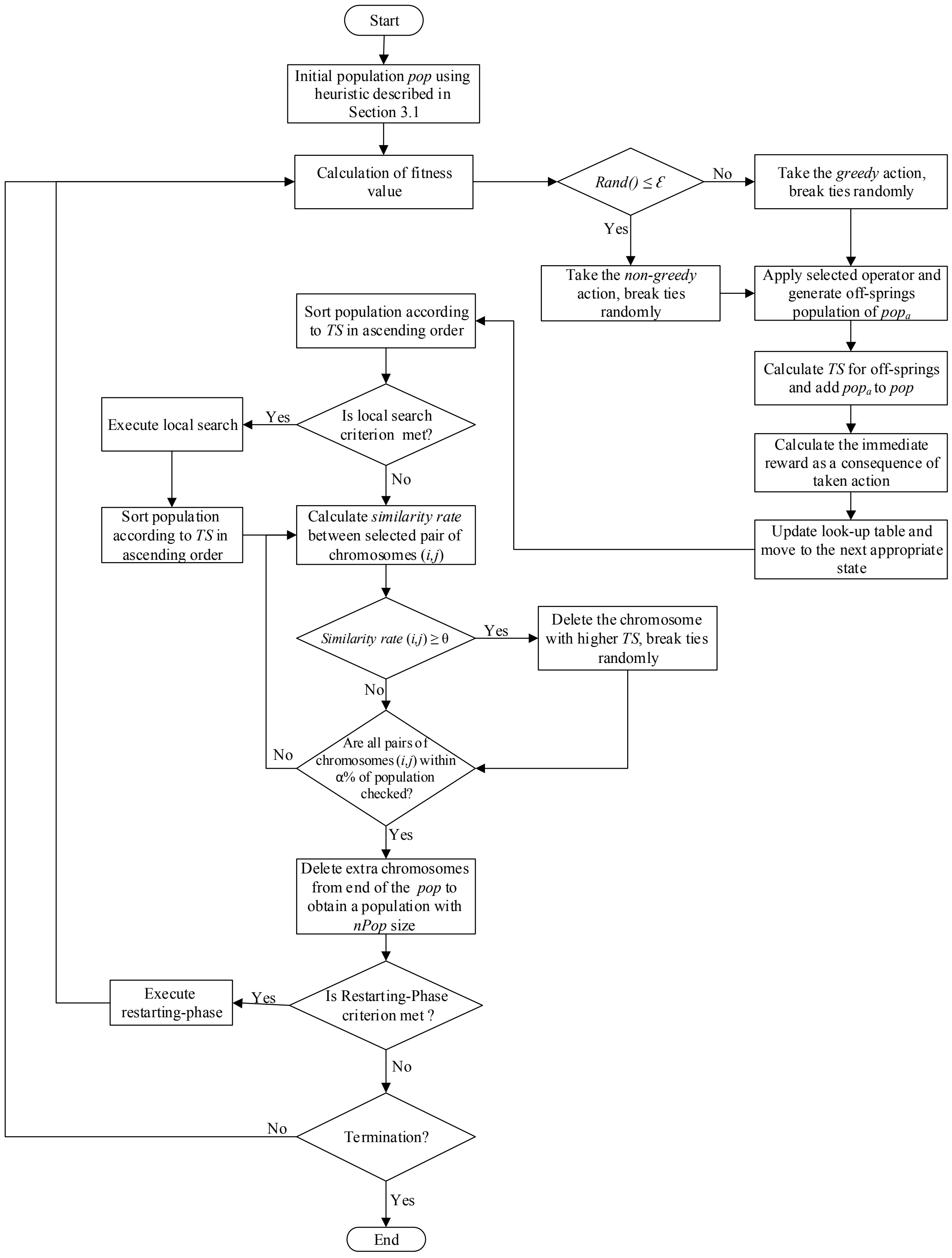

Start
Initial population pop using heuristic described in Section 3.1
Calculation of fitness value
Rand() ≤ ℰ
No
Yes
Take the greedy action, break ties randomly
Take the non-greedy action, break ties randomly
Apply selected operator and generate off-springs population of pop_a
Calculate TS for off-springs and add pop_a to pop
Calculate the immediate reward as a consequence of taken action
Update look-up table and move to the next appropriate state
Sort population according to TS in ascending order
Is local search criterion met?
Execute local search
Sort population according to TS in ascending order
Calculate similarity rate between selected pair of chromosomes (i,j)
Similarity rate (i,j) ≥ θ
Delete the chromosome with higher TS, break ties randomly
Are all pairs of chromosomes (i,j) within α% of population checked?
Delete extra chromosomes from end of the pop to obtain a population with nPop size
Is Restarting-Phase criterion met ?
Execute restarting-phase
Termination?
End

Fig. 9. Flowchart of the proposed DQGA seeded by solutions of 2-JSeP.

5.2. Algorithms' performance examination

In order to evaluate computational results of the proposed algorithms, i.e. DGA-CM, DGA-MC and DQGA, we have used the *gap percentage* and *CPU times* as two performance measures. The gap percentage is calculated as follows:

$$GAP = \left(\frac{Sol_{Alg} - Best}{Best} \times 100\right)\% \qquad (16)$$

Table 7
Benchmark problems.

Size	Dataset		#Instances	# Jobs	# Tools	Capacity
	Catanzaro et al. [16]	Crama et al. [13]				
Small	*datA1*	*G1C1*	10	10	10	4
	datA2	*G1C2*	10	10	10	5
	datA3	*G1C3*	10	10	10	6
	datA4	*G1C4*	10	10	10	7
	datB1	*G2C1*	10	15	20	6
	datB2	*G2C2*	10	15	20	8
	datB3	*G2C3*	10	15	20	10
	datB4	*G2C4*	10	15	20	12
Medium	*datC1*	*G3C1*	10	30	40	15
	datC2	*G3C2*	10	30	40	17
	datC3	*G3C3*	10	30	40	20
	datC4	*G3C4*	10	30	40	25
large	*datD1*	*G4C1*	10	40	60	20
	datD2	*G4C2*	10	40	60	22
	datD3	*G4C3*	10	40	60	25
	datD4	*G4C4*	10	40	60	30

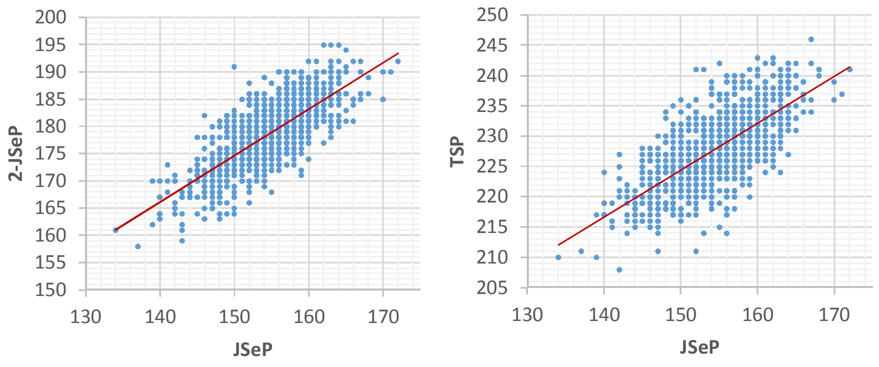


Fig. 10. Scatter plots of 1000 sample sequences generated from instance1 in *datB1*.

Table 8
correlation between objective functions.

Catanzaro et al. [16]			Crama et al. [13] datasets		
Dataset name	JSeP, 2-JSeP	JSeP, TSP	Dataset name	JSeP, 2-JSeP	JSeP, TSP
datA1	0.83	0.72	*G1C1*	0.84	0.76
datA2	0.67	0.56	*G1C2*	0.78	0.62
datA3	0.62	0.47	*G1C3*	0.76	0.59
datA4	0.52	0.40	*G1C4*	0.63	0.48
datB1	0.76	0.59	*G2C1*	0.62	0.52
datB2	0.62	0.51	*G2C2*	0.76	0.65
datB3	0.58	0.44	*G2C3*	0.54	0.42
datB4	0.53	0.33	*G2C4*	0.63	0.52
datC1	0.79	0.70	*G3C1*	0.74	0.65
datC2	0.76	0.63	*G3C2*	0.73	0.65
datC3	0.70	0.51	*G3C3*	0.67	0.53
datC4	0.62	0.42	*G3C4*	0.57	0.43
datD1	0.82	0.72	*G4C1*	0.75	0.67
datD2	0.76	0.66	*G4C2*	0.72	0.59
datD3	0.76	0.61	*G4C3*	0.73	0.56
datD4	0.69	0.52	*G4C4*	0.67	0.53

Where Sol_{Alg} indicates the solution obtained by each algorithm and *Best* corresponds to the best-known solutions found by our proposed algorithms. One should notice that we have set a fixed termination condition of 350 generations, but we recorded the time of the best-obtained solution by algorithms, not necessarily the time of 350 generations. In running DGA-CM, DGA-MC algorithms, all strategies and parameters are kept the same as DQGA, except that they did not have the privilege of Q-learning.

Fig. 11 shows the convergence plot of instance 5 in *datC3* with the effect of the *restarting-phase*. In iterations 150 and 212, after applying the restarting-phase, which resulted in enhanced diversity, better solutions have been found.

Fig. 12 depicts how the designed Q-learning algorithm converges to the optimum action within each state. The convergence here means that the greedy action, i.e. the action corresponding to the highest Q-value in a given state, will not be changed by further iteration. In the early generations (see Fig. 12a) that algorithm has not been trained, the greedy action might frequently change. As the algorithm further evolves (see Fig. 12b), the greedy action in each state emerges with the higher Q-values. However, in some states, e.g. state 1, two actions of MC and CM are still competing with each other. Toward end of the run (see Fig. 12c), the algorithm is converged to the optimal action, which are shown with boxes corresponding to the largest Q-value in each state.

The results of our algorithms for Catanzaro et al. [16] and Crama et al. [13] datasets are summarized in Tables 9 and 10, respectively, and compared with those reported in [25]. The tables show the average solutions (*AS*) of 10 instances and over 10 replications within each dataset. They also show the average best solutions (*ABS*) and computational time in seconds (*T*). The bold values in Tables 9 and 10 indicate that the algorithm could reach or improve the average best-known solution. We have also reported the results of the 2-JSeP method, which has been used to seed genetic algorithms with the initial population.

As it can be seen in Table 9, considering the *AS* and *ABS* criteria, the results of our three algorithms for *datA* datasets are identical with those of ILS. For *datB* datasets, all three algorithms could reach the average best-known solutions. The small datasets of *datA* and *datB* contains trivial instances that all algorithms could reach to *ABS* without significant computational burden. In *datC* datasets, DQGA

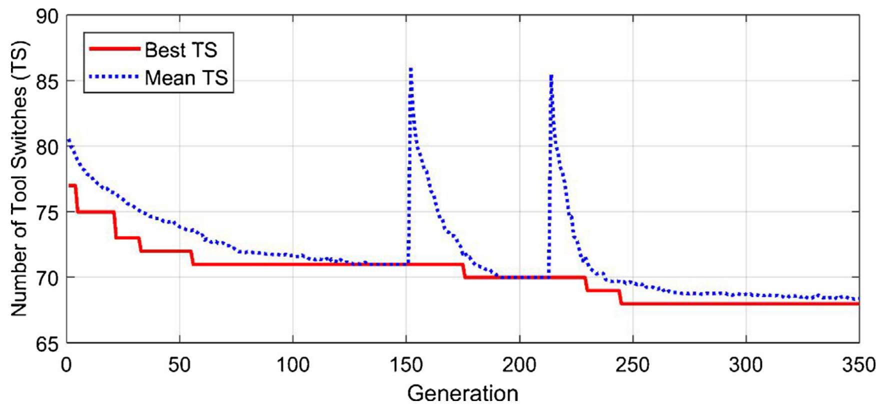
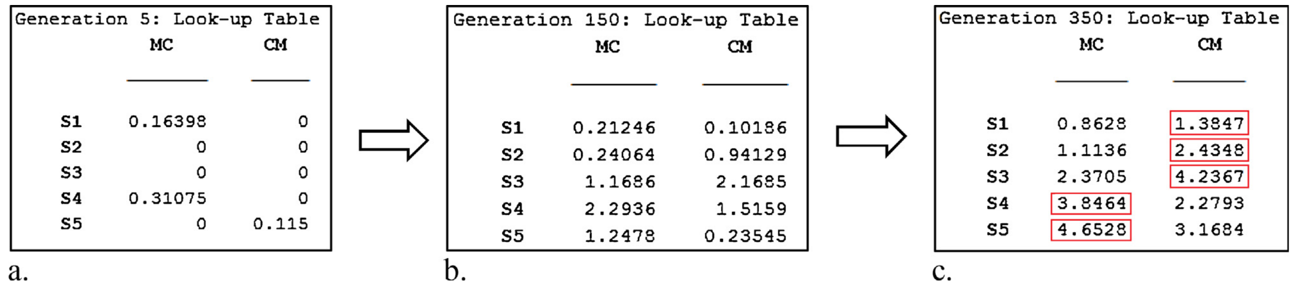


Fig. 11. Convergence plot for instance 5 in dataset *datC3* with showing the impact of the *restarting-phase.*


Generation 5: Look-up Table

	MC	CM
S1	0.16398	0
S2	0	0
S3	0	0
S4	0.31075	0
S5	0	0.115

a.

Generation 150: Look-up Table

	MC	CM
S1	0.21246	0.10186
S2	0.24064	0.94129
S3	1.1686	2.1685
S4	2.2936	1.5159
S5	1.2478	0.23545

b.

Generation 350: Look-up Table

	MC	CM
S1	0.8628	$\boxed{1.3847}$
S2	1.1136	$\boxed{2.4348}$
S3	2.3705	$\boxed{4.2367}$
S4	$\boxed{3.8464}$	2.2793
S5	$\boxed{4.6528}$	3.1684

c.

Fig. 12. Convergence of Q-values represented within look-up tables in generation 5 (a), generation 150 (b) and generation 350 (c) for instance 5 in dataset *datC3.*

Table 9
Computational results of our three algorithms for Catanzaro et al. [16] datasets in comparison with the results in [25].

Dataset Name	ILS [25]			Proposed Methods										
				2-JSeP		DGA-CM			DGA-MC			DQGA		
	AS	*ABS*	*T*	*ABS*	*T*	*AS*	*ABS*	*T*	*AS*	*ABS*	*T*	*AS*	*ABS*	*T*
datA1	**12.50**	**12.50**	0.09	12.80	0.13	**12.50**	**12.50**	0.15	**12.50**	**12.50**	0.16	**12.50**	**12.50**	0.17
datA2	**10.80**	**10.80**	0.09	11.50	0.12	**10.80**	**10.80**	0.16	**10.80**	**10.80**	0.17	**10.80**	**10.80**	0.17
datA3	**10.10**	**10.10**	0.05	10.30	0.13	**10.10**	**10.10**	0.19	**10.10**	**10.10**	0.18	**10.10**	**10.10**	0.19
datA4	**10.00**	**10.00**	0.04	**10.00**	0.17	**10.00**	**10.00**	0.23	**10.00**	**10.00**	0.20	**10.00**	**10.00**	0.22
datB1	**26.50**	**26.50**	1.09	29.20	0.60	26.74	**26.50**	3.04	26.92	**26.50**	3.57	26.72	**26.50**	1.62
datB2	**21.70**	**21.70**	1.38	23.40	0.59	21.76	**21.70**	2.26	21.92	**21.70**	1.68	21.74	**21.70**	2.12
datB3	**19.70**	**19.70**	1.12	20.80	0.61	19.76	**19.70**	1.83	19.74	**19.70**	1.34	19.76	**19.70**	1.39
datB4	**19.20**	**19.20**	0.71	19.30	0.62	**19.20**	**19.20**	0.93	**19.20**	**19.20**	0.32	**19.20**	**19.20**	0.79
datC1	99.09	98.90	99.47	110.07	7.35	100.06	99.10	172.39	100.40	99.20	135.52	99.82	**98.80**	145.29
datC2	82.82	**82.50**	137.08	94.70	9.00	83.86	83.00	204.79	84.08	82.80	174.37	83.74	**82.50**	196.09
datC3	66.78	**66.60**	172.59	77.50	7.51	68.10	67.00	225.68	68.28	67.10	201.83	67.82	**66.60**	220.32
datC4	51.47	51.30	137.79	58.60	7.78	52.36	51.50	151.10	52.24	51.40	139.40	52.10	**51.20**	144.89
datD1	198.36	**198.00**	488.66	216.30	32.63	200.12	198.50	372.79	199.86	198.4	290.29	200.16	**198.00**	213.43
datD2	174.05	173.60	706.13	194.90	36.54	176.30	174.40	409.71	176.36	173.90	380.32	175.68	**173.50**	485.47
datD3	146.68	**146.20**	1069.98	166.10	38.15	149.08	146.60	779.72	149.70	147.00	709.51	148.80	**146.20**	749.28
datD4	115.42	**115.20**	1518.80	135.20	48.08	118.26	116.30	901.38	119.14	116.50	978.28	117.22	**115.20**	1138.28

shows better performance in which could improve the *ABS* within *datC1* and *datC4*. The DQGA algorithm matches the ILS results in *datD1*, *datD3* and *datD4*, but outperforms the ILS in *datD2*. It has also been shown in Table 9 that the DQGA algorithm requires remarkably less computational time for datasets *datD* than ILS.

Regarding the Crama et al. [13] datasets, it can be observed in Table 10 that our three algorithms were able to reach to the *ABS* in small group datasets of *G1* and *G2*. Again, because of trivial instances of these groups best known solutions were obtained in an average time of 0.71 s. For the medium and large datasets, i.e. groups *G3* and *G4*, that more computational effort is required, DQGA performs either equally or better than DGA-CM and DGA-CM. Within these groups, DQGA were able to find new *ABS* for *G3C1*, *G3C2*, *G3C4*, *G4C2* and *G4C3*, compared to the ILS [25].

In addition, with respect to the computational time, although the ILS [25] was coded in different software than DQGA, both ILS and DQGA were run in computers with similar architecture of RAM and processor. The average computational time that DQGA needs to solve large instances of group *datD* in Catanzaro et al. [16] datasets is 647 s, while this time for ILS [25] is 946 s. A relatively similar time gap can also be observed in Crama et al. [13] datasets, as DQGA takes on average 825 s to solve large instances of group *G4*, which is comparable to the 902 s solved by ILS [25].

The 2-JSeP heuristic could find high-quality solutions for both studied datasets. In Catanzaro et al. [16] datasets, 2-JSeP found solutions with 9.05% gap, on average, with the best-known solutions in an average computational time of 11.87 s. The same behavior is also observed in Crama et al. [13] datasets, as 2-JSeP could find solutions

Table 10
Computational results of our three algorithms for Crama et al. [13] datasets in comparison with the results in [25].

Dataset Name	ILS [25]			Proposed Methods											
				2-JSeP		DGA-CM			DGA-MC			DQGA			
	AS	*ABS*	*T*	*ABS*	*T*	*AS*	*ABS*	*T*	*AS*	*ABS*	*T*	*AS*	*ABS*	*T*	
G1C1	13.11	**13.10**	0.08	13.80	0.10	**13.10**	**13.10**	0.21	**13.10**	**13.10**	0.20	**13.10**	**13.10**	0.20	
G1C2	**11.20**	**11.20**	0.08	11.40	0.11	**11.20**	**11.20**	0.22	**11.20**	**11.20**	0.23	**11.20**	**11.20**	0.23	
G1C3	**10.30**	**10.30**	0.06	**10.30**	0.11	**10.30**	**10.30**	0.25	**10.30**	**10.30**	0.26	**10.30**	**10.30**	0.25	
G1C4	**10.10**	**10.10**	0.04	**10.10**	0.09	**10.10**	**10.10**	0.22	**10.10**	**10.10**	0.21	**10.10**	**10.10**	0.22	
G2C1	26.63	**26.60**	0.84	28.80	0.56	26.72	**26.60**	1.96	26.74	**26.60**	2.10	26.70	**26.60**	1.99	
G2C2	**21.70**	**21.70**	1.09	23.20	0.59	21.72	**21.70**	1.24	**21.70**	**21.70**	1.28	**21.70**	**21.70**	1.36	
G2C3	**20.10**	**20.10**	0.87	20.70	0.61	**20.10**	**20.10**	1.08	**20.10**	**20.10**	0.65	**20.10**	**20.10**	0.94	
G2C4	**19.60**	**19.60**	0.60	**19.60**	0.57	**19.60**	**19.60**	0.67	**19.60**	**19.60**	0.37	**19.60**	**19.60**	0.56	
G3C1	111.63	106.40	86.81	117.50	6.88	107.56	106.50	187.97	107.72	**106.30**	191.15	106.88	**106.30**	162.02	
G3C2	88.59	88.30	132.01	99.60	7.71	89.72	88.20	196.20	89.26	**88.10**	259.08	89.08	88.20	223.16	
G3C3	70.71	**70.40**	173.47	80.50	8.39	71.82	70.60	194.82	71.56	70.50	213.21	71.36	**70.40**	197.00	
G3C4	53.14	52.90	146.81	62.20	8.56	54.24	53.10	198.13	54.08	53.00	171.08	53.64	**52.70**	166.61	
G4C1	199.00	**198.40**	441.66	219.00	45.64	200.96	198.60	592.19	201.8	199.4	562.10	200.92	**198.40**	526.47	
G4C2	174.04	173.50	665.13	196.20	45.75	175.72	**173.30**	708.21	176.16	**173.30**	722.12	176.00	**173.30**	695.52	
G4C3	146.52	146.00	1016.37	166.10	47.38	149.36	**145.90**	829.78	148.92	146.10	812.04	148.84	**145.90**	859.65	
G4C4	114.18	**113.90**	1484.97	133.40	48.04	117.40	114.60	1280.20	117.12	114.50	1133.46	116.30	**113.90**	1219.28	

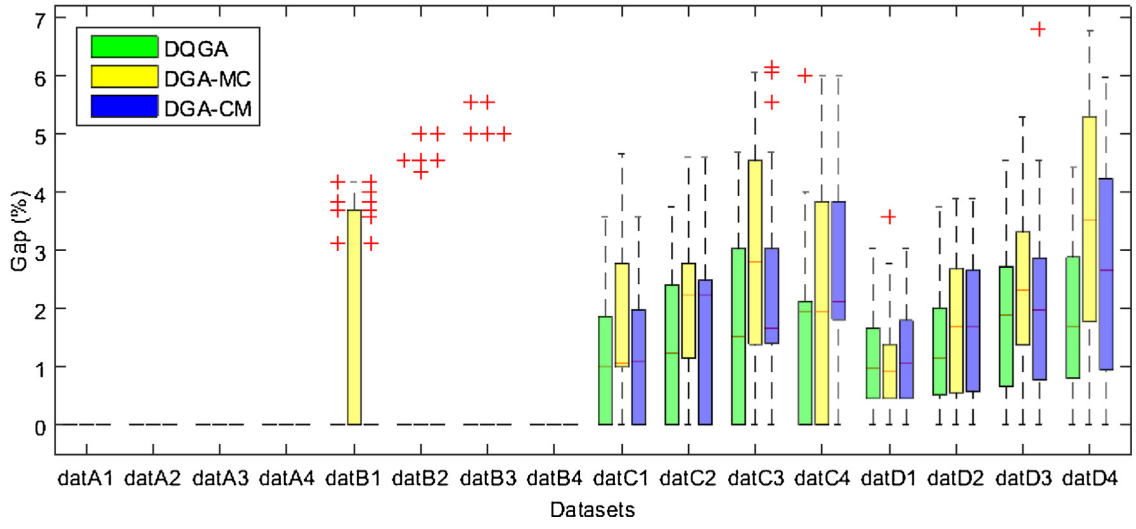


Fig. 13. Box and whisker plots of Gap% criterion for all datasets of Catanzaro et al. [16].

with an average of 8.50% gap with the best-known solution in an average 13.81 s. In the next set of paragraphs, we more elaborate on the performance of DQGA against those non-learning algorithms of DGA-CM and DGA-MC.

In order to conduct a comparison between our three proposed algorithms within each dataset, we have visualized the computational results using box and whisker plots, as illustrated in Figs. 13 and 14 for Catanzaro et al. [16] datasets. Top and bottom of each box represent the third and the first quartile, respectively; and the horizontal line stands for second quartile (median) of the data. Outliers are also plotted as cross points. In particular, Fig. 13 represents the gap percentage that we discussed in equation (16). The DQGA algorithm has less variability than two other algorithms of DGA-CM and DGA-MC, as it has a maximum gap of 4.5% (see the green box for *datC3* dataset). Considering the computational time, although the DQGA algorithm has higher average computational time in *datD2, datD3* and *datD4* datasets than two other algorithms (see Fig. 14), it can be justified by its higher solutions quality found in these datasets.

The graphical representation of percentage gap and computational time for each algorithm across all datasets of Crama et al. [13] is depicted in Figs. 15 and 16. These results show that the DQGA performs either better than or equal to two other algorithms. However, this statement is only driven from the graphical representations of the results over 10 independent replications. Due to the randomness properties of the algorithms, we have conducted paired sample *t*-test in the following section to investigate their performances from a statistical viewpoint.

5.3. Statistical analysis

In order to compare the quality of the solutions between three DQGA, DGA-CM and DGA-MC algorithms, paired sample *t*-test is carried out at the significance level of 0.05, which means there would be difference between performances of the algorithms in statistical sense if null hypothesis is rejected with *p*-value smaller than 0.05. The null hypothesis, i.e., a zero mean difference between paired observations, and alternative hypothesis are as follows:

$$\begin{aligned} H_0 &: \mu_A^C - \mu_B^C = 0 \\ H_1 &: \mu_A^C - \mu_B^C \neq 0 \end{aligned} \quad (17)$$

Where μ_A^C stands for the average of the criterion *C* (time or percentage gap) in the algorithm *A*, i.e. DQGA, and μ_B^C stands for the average of the criterion *C* in the algorithm *B*, i.e. DGA-CM or DGA-

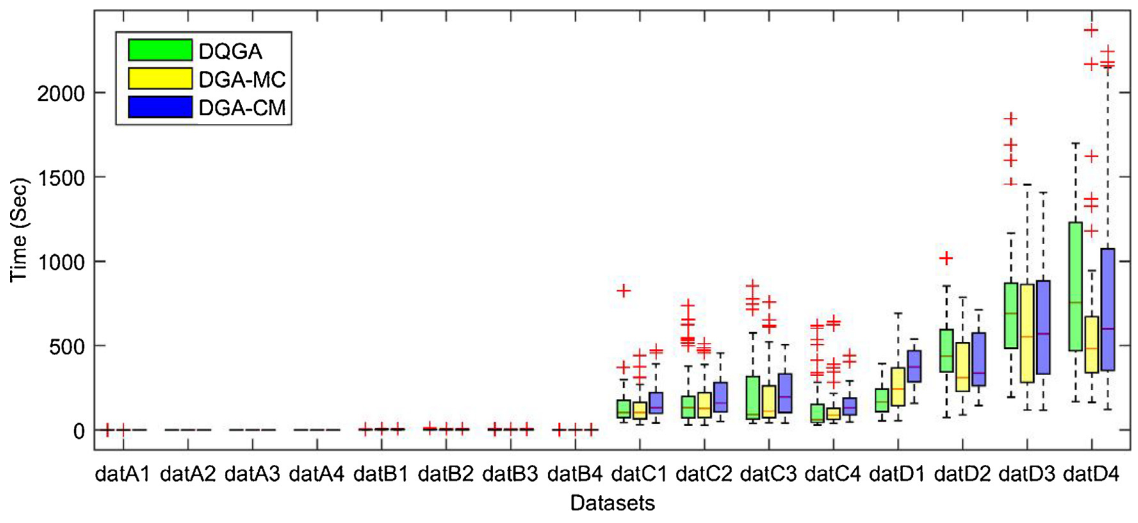


Fig. 14. Box and whisker plots of Time criterion for all datasets of Catanzaro et al. [16].

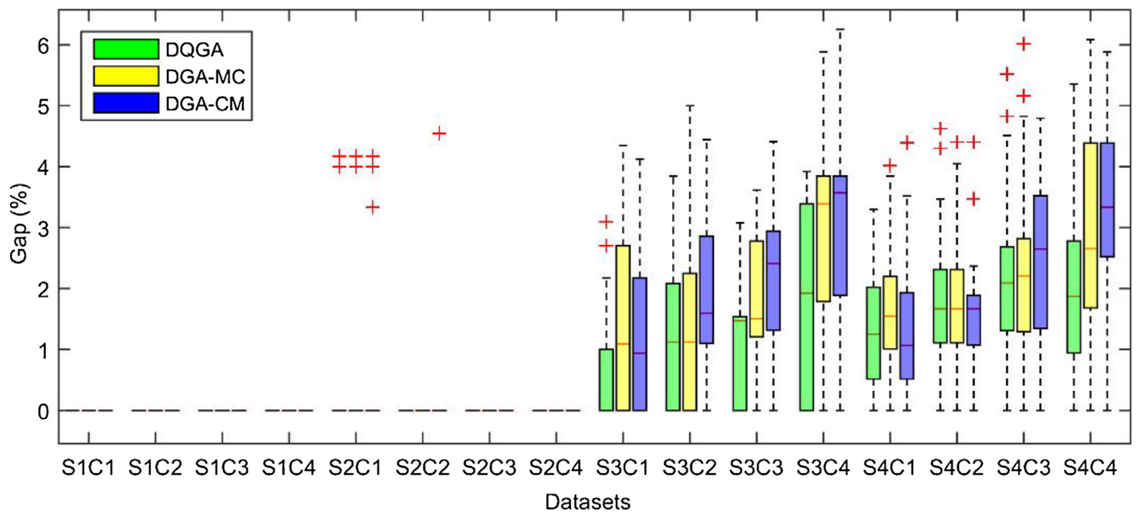


Fig. 15. Box and whisker plots of Gap% criterion for all datasets of Crama et al. [13].

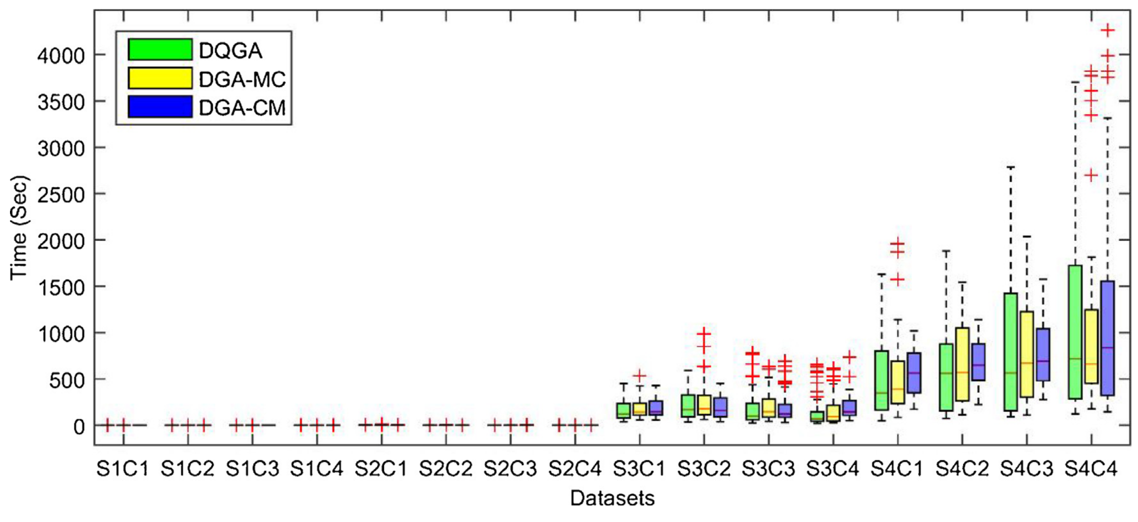


Fig. 16. Box and whisker plots of Time criterion for all datasets of Crama et al. [13].

Table 11
Statistical result of the paired sample *t*-test for two criteria.

Criterion *C*	Algorithm *A*	Algorithm *B*	Catanzaro et al. [16] datasets		Crama et al. [13] datasets	
			p-value	Result	*p*-value	Result
Gap (%)	DQGA	DGA-CM	0.000	H_0 Is rejected, DQGA performs better	0.000	H_0 Is rejected, DQGA performs better
	DQGA	DGA-MC	0.000	H_0 Is rejected, DQGA performs better	0.000	H_0 Is rejected, DQGA performs better
T (Sec)	DQGA	DGA-CM	0.892	Fail to reject H_0, There is no significant difference	0.388	Fail to reject H_0, There is no significant difference
	DQGA	DGA-MC	0.011	H_0 Is rejected, DGA-MC performs better	0.798	Fail to reject H_0, There is no significant difference

MC. The test is run over the results of all instances for each of two databases and the outputs are reported in Table 11.

Regarding the percentage gaps, it is statistically proven that DQGA outperforms DGA-CM and DGA-MC in both Catanzaro et al. [16] and Crama et al. [13] datasets. With respect to the computational time (*T*), no significant differences are observed in the performance of DQGA and DGA-CM in both datasets, as statistical test failed to reject the null hypothesis. DGA-MC has shown that requires less computational effort than DQGA in Catanzaro et al. [16] dataset. However, DQGA found solutions with higher quality.

6. Conclusions and future work

In this study, we have addressed the Job Sequencing and Tool Switching Problem (SSP) by hybrid methods of Traveling Salesman Problem of Second Order (2-TSP) and a Dynamic Q-learning-based Genetic Algorithm (DQGA). It has been shown that SSP is equivalent to the Job Sequencing Problem (JSeP). In the JSeP, the number of tool switches required to process a job depends upon all of its predecessors. By utilizing 2-TSP, one predecessor of the current job is taken into account in transition to the next job, which comparing to the TSP provides better distance value between two consecutive nodes. We called the induced JSeP by 2-TSP as Job Sequencing Problem of Second Order (2-JSeP) and proved that 2-JSeP is *NP*-hard. The 2-JSeP has been solved using a heuristic and the obtained solutions have been considered as an initial population for the DQGA.

In designing a genetic algorithm, the order of executing genetic operators, i.e. first crossover then mutation (CM) or first mutation then crossover (MC), might change the ability of the algorithm in searching solution space. This motivated us to propose a method that enables the genetic algorithm to learn from the experience of selecting the order of mutation and crossover operators in each generation. Therefore, a Q-learning procedure, which is a kind of reinforcement learning method, is designed to help with the decision of selecting operator's sequence. In addition, the proposed genetic algorithm is equipped with a local search, removal of similar chromosomes in each generation, as well as a mechanism to dynamically explore the solution space. These three strategies were selected to escape from local optima.

The results of the proposed DQGA are compared with the state-of-the-art method in the literature, as well as with those of non-learning DGA-CM and DGA-MC algorithms. The computational results on 320 benchmark instances show that the proposed DQGA is not only competitive with the current state-of-the-art methods, but also could improve the reported best-known solutions for some instances in notably less time. Moreover, the 2-JSeP was able to generate high quality initial solutions for genetic algorithms, with an average of 9.05% and 8.50% gap with best-known solutions for two studied datasets and in an average time of 11.87 and 13.81 s, respectively. Finally, through the statistical analysis, the performance of DQGA is compared with those of non-learning GAs.

One of the future research directions will be utilizing *k*-JSeP approximation model with *k* as close as possible to the capacity in the JSeP. In addition, more jobs and different algorithms may be considered and compared to our proposed model. It would be also interesting to see the performance of the other meta-heuristic methods against the proposed DQGA. Moreover, the proposed approach is quite general, which is applicable to the scheduling problems with sequence-dependent setup time. So, adopting described techniques to solve this class of problems would be an interesting future research study.

Appendix A

Table A1
Number of tool twitches for the sequence of (a, b, c). For example, in the first raw the number of tool switches equals 4 for the job sequences $(1, 2, 3)$.

To job *c*		1	2	3	4	5	6	7	8	9	10
From job (*a*,*b*)	**(1,2)**	–	–	4	5	5	4	5	4	6	5
	(1,3)	–	4	–	3	5	4	4	4	5	5
	(1,4)	–	5	3	–	5	5	4	4	5	6
	(1,5)	–	5	5	5	–	5	4	3	5	6
	(1,6)	–	4	4	5	5	–	5	5	5	6
	(1,7)	–	5	4	4	4	5	–	4	5	7
	(1,8)	–	4	4	4	3	4	3	–	4	6
	(1,9)	–	6	6	5	5	6	6	5	–	6
	(1,10)	–	5	6	6	6	7	8	7	6	–
	(2,1)	–	–	4	5	5	4	5	4	6	5
	(2,3)	4	–	–	4	6	4	5	6	6	5
	(2,4)	5	–	4	–	5	5	5	5	5	5
	(2,5)	5	–	6	5	–	5	5	4	5	5
	(2,6)	4	–	4	5	5	–	5	5	5	6
	(2,7)	5	–	5	5	5	6	–	5	6	8
	(2,8)	4	–	6	5	4	5	5	–	5	6
	(2,9)	6	–	6	5	5	6	6	5	–	6
	(2,10)	5	–	5	5	5	6	7	6	5	–
	(3,1)	–	3	–	2	4	3	3	3	4	4
	(3,2)	3	–	–	3	5	3	4	5	5	4
	(3,4)	2	3	–	–	3	3	2	3	3	4
	(3,5)	4	5	–	4	–	5	4	4	4	5
	(3,6)	3	3	–	4	4	–	4	4	4	5
	(3,7)	3	4	–	3	3	4	–	3	4	6
	(3,8)	3	5	–	4	4	5	4	–	4	6
	(3,9)	5	5	–	4	4	5	5	4	–	5
	(3,10)	5	4	–	5	5	6	7	6	5	–
	(4,1)	–	4	2	–	4	4	3	3	4	5
	(4,2)	4	–	3	–	4	4	4	4	4	4
	(4,3)	2	3	–	–	3	3	2	3	3	4
	(4,5)	4	4	4	–	–	4	3	3	3	4
	(4,6)	4	4	4	–	5	–	5	5	5	6
	(4,7)	3	4	3	–	3	4	–	3	4	6
	(4,8)	3	4	4	–	3	4	3	–	3	5
	(4,9)	4	4	4	–	3	4	4	3	–	4
	(4,10)	5	4	5	–	5	6	7	6	5	–
	(5,1)	–	4	4	4	–	4	3	2	4	5
	(5,2)	4	–	5	4	–	4	4	3	4	4
	(5,3)	4	5	–	3	–	4	4	5	5	5
	(5,4)	4	4	3	–	–	4	3	3	3	4
	(5,6)	4	4	4	5	–	–	5	5	5	6
	(5,7)	3	4	3	3	–	4	–	3	4	6
	(5,8)	2	3	4	3	–	4	3	–	3	4
	(5,9)	4	4	4	3	–	4	4	3	–	4
	(5,10)	5	4	5	5	–	6	7	6	5	–
From job (*a*,*b*)	**(6,1)**	–	2	2	3	3	–	3	2	3	4
	(6,2)	2	–	2	3	3	–	3	3	3	3
	(6,3)	2	2	–	2	3	–	2	3	3	4
	(6,4)	3	3	2	–	3	–	3	3	3	5
	(6,5)	3	3	4	3	–	–	3	3	3	4
	(6,7)	3	4	3	3	3	–	–	3	4	6
	(6,8)	2	3	4	3	3	–	3	–	3	5
	(6,9)	4	4	4	3	3	–	4	3	–	4
	(6,10)	5	4	5	5	5	–	7	6	5	–
	(7,1)	–	3	2	2	2	3	–	1	3	5
	(7,2)	3	–	3	3	3	3	–	3	4	5
	(7,3)	2	3	–	1	3	2	–	3	3	4
	(7,4)	2	3	1	–	2	3	–	2	2	4
	(7,5)	2	3	3	2	–	3	–	2	3	4
	(7,6)	3	3	3	4	4	–	–	4	4	5
	(7,8)	1	3	3	2	2	3	–	–	3	5
	(7,9)	4	4	4	3	3	4	–	3	–	4
	(7,10)	6	5	6	6	6	7	–	7	6	–
	(8,1)	–	3	3	3	2	3	2	–	3	5
	(8,2)	3	–	5	4	3	4	4	–	4	5
	(8,3)	3	5	–	3	5	4	4	–	5	6
	(8,4)	3	4	3	–	3	4	3	–	3	5
	(8,5)	2	3	4	3	–	4	3	–	3	4
	(8,6)	4	4	4	5	5	–	5	–	5	6
	(8,7)	3	4	3	3	3	4	–	–	4	6
	(8,9)	4	4	4	3	3	4	4	–	–	4
	(8,10)	6	5	6	6	6	7	8	–	6	–
	(9,1)	–	4	3	3	3	3	3	2	–	4
	(9,2)	4	–	4	3	3	3	4	3	–	3
	(9,3)	3	4	–	2	4	3	3	4	–	4
	(9,4)	3	3	2	–	2	3	2	2	–	3
	(9,5)	3	3	3	2	–	3	3	2	–	3
	(9,6)	3	3	3	4	4	–	4	4	–	5
	(9,7)	3	4	3	3	3	4	–	3	–	6
	(9,8)	2	3	3	2	2	3	3	–	–	4
	(9,10)	4	3	4	4	4	5	6	5	–	–
	(10,1)	–	3	3	4	4	4	5	4	4	–
	(10,2)	3	–	3	3	3	3	5	4	3	–
	(10,3)	3	3	–	3	4	4	4	5	4	–
	(10,4)	4	3	3	–	3	5	4	4	3	–
	(10,5)	4	3	4	3	–	4	4	3	3	–
	(10,6)	4	4	4	5	5	–	5	5	5	–
	(10,7)	5	6	5	5	5	6	–	5	6	–
	(10,8)	4	4	5	4	3	5	5	–	4	–
	(10,9)	4	4	4	3	3	4	4	3	–	–

References

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

[2] G. Ghiani, A. Grieco, E. Guerriero, An exact solution to the TLP problem in an NC machine, Robot. Comput. Integr. Manuf. 23 (2007) 645–649, http://dx.doi.org/10.1016/j.rcim.2007.02.011.

[3] C. Privault, G. Finke, K-Server problems with bulk requests: an application to tool switching in manufacturing, Ann. Oper. Res. 96 (2000) 255–269, http://dx.doi.org/10.1023/A:1018939132489.

[4] J.E. Amaya, C. Cotta, A.J. Fernández, A memetic algorithm for the tool switching problem, Hybrid Metaheuristics (2008) 190–202, http://dx.doi.org/10.1007/978-3-540-88439-2_14.

[5] T. Mütze, Scheduling with few changes, Eur. J. Oper. Res. 236 (2014) 37–50, http://dx.doi.org/10.1016/j.ejor.2013.11.011.

[6] M. Furrer, T. Mütze, An algorithmic framework for tool switching problems with multiple objectives, Eur. J. Oper. Res. 259 (2017) 1003–1016, http://dx.doi.org/10.1016/j.ejor.2016.11.034.

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

[8] M. Tzur, A. Altman, Minimization of tool switches for a flexible manufacturing machine with slot assignment of different tool sizes, IIE Trans. (Inst. Ind. Eng.) 36 (2004) 95–110, http://dx.doi.org/10.1080/07408170490245351.

[9] O.A. Ghrayeb, N. Phojanamongkolkij, P.R. Finch, A mathematical model and heuristic procedure to schedule printed circuit packs on sequencers, Int. J. Prod. Res. 41 (2003) 3849–3860, http://dx.doi.org/10.1080/0020754031000118071.

[10] M. Hirvikorpi, K. Salonen, T. Knuutila, O.S. Nevalainen, The general two-level storage management problem: a reconsideration of the KTNS-rule, Eur. J. Oper. Res. 171 (2006) 189–207, http://dx.doi.org/10.1016/j.ejor.2004.08.031.

[11] P.J.M. Van Laarhoven, W.H.M. Zijm, Production preparation and numerical control in PCB assembly, Int. J. Flex. Manuf. Syst. 5 (1993) 187–207, http://dx.doi.org/10.1007/BF01328741.

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

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

[14] G. Laporte, J.J. Salazar-Gonzáles, F. Semet, Exact algorithms for the job sequencing and tool switching problem, IIE Trans. 36 (2004) 37–45, http://dx.doi.org/10.1080/07408170490257871.

[15] H.H. Yanasse, R. de C.M. Rodrigues, E.L.F. Senne, Um algoritmo enumerativo baseado em ordenamento parcial para resolução do problema de minimização de trocas de ferramentas, Gest. Prod. 16 (2009) 370–381, http://dx.doi.org/10.1590/S0104-530x2009000300005.

[16] D. Catanzaro, L. Gouveia, M. Labbé, Improved integer linear programming formulations for the job Sequencing and tool Switching Problem, Eur. J. Oper. Res. 244 (2015) 766–777, http://dx.doi.org/10.1016/j.ejor.2015.02.018.

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

[18] M.A. Al-Fawzan, K.S. Al-Sultan, A tabu search based algorithm for minimizing the number of tool switches on a flexible machine, Comput. Ind. Eng. 44 (2002) 35–47, http://dx.doi.org/10.1016/S0360-8352(02)00183-3.

[19] M. Denizel, Minimization of the number of tool magazine setups on automated machines: a lagrangean decomposition approach, Oper. Res. 51 (2003) 309–320, http://dx.doi.org/10.1287/opre.51.2.309.12784.

[20] C.S. Tang, E.V. Denardo, Models arising from a flexible manufacturing machine, part II: minimization of the number of switching instants, Oper. Res. 36 (1988) 778–784, http://dx.doi.org/10.1287/opre.36.5.778.

[21] A. Konak, S. Kulturel-Konak, M. Azizoğlu, Minimizing the number of tool switching instants in Flexible Manufacturing Systems, Int. J. Prod. Econ. 116 (2008) 298–307, http://dx.doi.org/10.1016/j.ijpe.2008.09.001.

[22] J.E. Amaya, C. Cotta, A.J. Fernández-Leiva, Solving the tool switching problem with memetic algorithms, Artif. Intell. Eng. Des. Anal. Manuf. 26 (2012) 221–235, http://dx.doi.org/10.1017/S089006041100014X.

[23] A. Hertz, G. Laporte, M. Mittaz, K.E. Stecke, Heuristics for minimizing tool switches when scheduling part types on a flexible machine, IIE Trans. 30 (1998) 689–694, http://dx.doi.org/10.1023/A:1026434104330.

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

[25] G.S. Paiva, M.A.M. Carvalho, Improved heuristic algorithms for the job sequencing and tool switching problem, Comput. Oper. Res. 88 (2017) 208–219, http://dx.doi.org/10.1016/j.cor.2017.07.013.

[26] G. Ghiani, A. Grieco, E. Guerriero, Solving the job sequencing and tool switching problem as a nonlinear least cost Hamiltonian cycle problem, Networks 55 (2010) 379–385, http://dx.doi.org/10.1002/net.20341.

[27] G. Jäger, P. Molitor, Algorithms and experimental study for the traveling salesman problem of second order, in: Comb. Optim. Appl, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 211–224, http://dx.doi.org/10.1007/978-3-540-85097-7_20.

[28] R.M. Karp, J.M. Steele, Probabilistic analysis of heuristics, in: D.B.S.E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan (Eds.), Travel. Salesm. Probl., John Wiley & Sons, Chichester, 1985, pp. 561–573.

[29] T. Starkweather, S. Mcdaniel, K. Mathias, W. Darrell, A comparison of genetic sequencing operators, Proc. Fourth Int. Conf. Genet. Algorithms 1991 (2016) 69–76, 10.1.1.18.4329.

[30] J.E. Amaya, C. Cotta, A.J. Fern, A.J.F. Leiva, Hybrid cooperation models for the tool switching problem, Nat. Inspired Coop. Strateg. Optim. (NICSO 2010) 3 (2010) 39–52, http://dx.doi.org/10.1007/978-3-642-12538-6_4.

[31] J.H. Holland, Adaptation in Natural and Artificial Systems, 1975.

[32] O. Engin, G. Ceran, M.K. Yilmaz, An efficient genetic algorithm for hybrid flow shop scheduling with multiprocessor task problems, Appl. Soft Comput. 11 (2011) 3056–3065, http://dx.doi.org/10.1016/j.asoc.2010.12.006.

[33] J.S. Neufeld, J.N.D. Gupta, U. Buscher, A comprehensive review of flowshop group scheduling literature, Comput. Oper. Res. 70 (2016) 56–74, http://dx.doi.org/10.1016/j.cor.2015.12.006.

[34] E. Ahmadi, M. Zandieh, M. Farrokh, S.M. Emami, A multi objective optimization approach for flexible job shop scheduling problem under random machine breakdown by evolutionary algorithms, Comput. Oper. Res. 73 (2016) 56–66, http://dx.doi.org/10.1016/j.cor.2016.03.009.

[35] Z.-J. Lee, S.-F. Su, C.-C. Chuang, K.-H. Liu, Genetic algorithm with ant colony optimization (GA-ACO) for multiple sequence alignment, Appl. Soft Comput. 8 (2008) 55–78, http://dx.doi.org/10.1016/j.asoc.2006.10.012.

[36] S. Mirshekarian, G.A. Süer, Experimental study of seeding in genetic algorithms with non-binary genetic representation, J. Intell. Manuf. (2016) 1–10, http://dx.doi.org/10.1007/s10845-016-1204-3.

[37] J.L. Bentley, Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, 1990.

[38] M. Gen, R. Cheng, Genetic Algorithms and Engineering Optimization, Wiley, 2000.

[39] J. Han, J. Pei, M. Kamber, Data Mining: Concepts and Techniques, Elsevier, 2011.

[40] A. Albalate, W. Minker, Semi-Supervised and Unervised Machine Learning: Novel Strategies, John Wiley & Sons, 2013.

[41] P. Kulkarni, Reinforcement and Systemic Machine Learning for Decision Making, John Wiley & Sons, 2012, http://dx.doi.org/10.1002/9781118266502.

[42] B. Doğan, T. Ölmez, A novel state space representation for the solution of 2D-HP protein folding problem using reinforcement learning methods, Appl. Soft Comput. 26 (2015) 213–223, http://dx.doi.org/10.1016/j.asoc.2014.09.047.

[43] C.J.C.H. Watkins, P. Dayan, Q-learning, Mach. Learn. 8 (1992) 279–292, http://dx.doi.org/10.1007/BF00992698.

[44] A. Gosavi, Simulation-Based Optimization Parametric Optimization Techniques and Reinforcement Learning, Springer, US, 2015, http://dx.doi.org/10.1007/978-1-4899-7491-4.

[45] K.Q. Zhu, Z. Liu, in: J.-F. Boulicaut, F. Esposito, F. Giannotti, D. Pedreschi (Eds.), Population Diversity in Permutation-Based Genetic Algorithm, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 537–547, http://dx.doi.org/10.1007/978-3-540-30115-8_49.