Applied Soft Computing 13 (2013) 462–474

Contents lists available at SciVerse ScienceDirect

Applied Soft Computing

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

A hybrid Differential Evolution—Tabu Search algorithm for the solution of Job-Shop Scheduling Problems

Antonin Ponsich [a,*], Carlos A. Coello Coello [b]

[a] Universidad Autónoma Metropolitana, Unidad Azcapotzalco, Av. San Pablo No. 180, Col. Reynosa Tamaulipas, México, DF 02200, Mexico
[b] CINVESTAV-IPN (Evolutionary Computation Group), Departamento de Computación, Av. IPN No. 2508, Col. San Pedro Zacatenco, México, DF 07300, Mexico

ARTICLE INFO

Article history:
Received 7 April 2011
Received in revised form 2 May 2012
Accepted 2 July 2012
Available online 25 September 2012

Keywords:
Job-Shop Scheduling
Differential Evolution
Tabu Search

ABSTRACT

The Job-Shop Scheduling Problem (JSSP) has drawn considerable interest during the last decades, mainly because of its combinatorial characteristics, which make it very difficult to solve. The good performances attained by local search procedures, and especially Nowicki and Smutnicki's *i*-TSAB algorithm, encouraged researchers to combine such local search engines with global methods. Differential Evolution (DE) is an Evolutionary Algorithm that has been found to be particularly efficient for continuous optimization, but which does not usually perform well when applied to permutation problems. We introduce in this paper the idea of hybridizing DE with Tabu Search (TS) in order to solve the JSSP. A competitive neighborhood is included within the TS with the aim of determining if DE is able to replace the re-start features that constitute the main strengths of *i*-TSAB (i.e., a long-term memory and a path-relinking procedure). The computational experiments reported for more than 100 JSSP instances show that the proposed hybrid DE–TS algorithm is competitive with respect to other state-of-the-art techniques, although, there is still room for improvement if the adequacy between the solution representation modes within DE and TS is properly stressed.

1. Introduction

Scheduling problems, because of their many industrial applications, not only in manufacture but also in the service fields [1], have attracted a raising interest from the Operations Research community. For instance, steel production in small quantities and with different compositions is scheduled in [2] to minimize the lateness with respect to a fixed deadline. A real-world sports league schedule is tackled through the solution of graph coloring problems in [3]. A grid computing application that assigns priorities to tasks submitted by grid users is implemented in order to optimize the available resources [4].

In the manufacturing area, several problems ranging from the single-machine to the Open Shop configurations, combined with specific constraints and objectives, have proved to be NP-hard, i.e., demanding solution times that increase exponentially with the number of considered operations. The Job-Shop Scheduling Problem (JSSP) is, in this framework, one of the most complex examples. A set of jobs, which all consist of several operations, is processed in a certain order on a set of machines. The processing sequence of each job operation on the machines, as well as the associated processing times, are problem data. The objective is, then, to minimize the completion date of the last scheduled operation.

Because solving exactly the JSSP constitutes a challenge, a considerable amount of research has been dedicated to the development of efficient and effective methods to solve it. In the last few decades of the twentieth century, researchers first concentrated their attention on exact optimization techniques. The disjunctive graph representation [5] (further described in the next section), commonly adopted at the time, led to the formulation of mathematical programming models typically tackled through Branch-and-Bound or similar methods. However, the curse of dimensionality associated to the JSSP considerably limits the size of its instances that are solvable using such methods: beyond 10 jobs and 10 machines, exact solution techniques are unable to provide optimal solutions within reasonable computational times.

Thus, further studies have been devoted to the development of heuristic methods, such as SB-I and SB-II [6], which achieve very good results for small, mid-size and even large JSSP instances. Besides, with the development and enhancement of many metaheuristics, the JSSP appeared to be an ideal platform for testing new solution methods developed for combinatorial frameworks. A great quantity of papers have presented the application of Local Search or population-based heuristics to solve the JSSP. Among the most relevant ones, one could quote the Simulated Annealing approach applied in [7] or the Tabu Search algorithm proposed in [8]. Taillard [9] also implements a Tabu Search approach that adopts the

* Corresponding author.
E-mail addresses: aspo@correo.azc.uam.mx (A. Ponsich), ccoello@cs.cinvestav.mx (C.A. Coello Coello).

1568-4946/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.asoc.2012.07.034

disjunctive graph representation proposed in [5] and introduces a neighborhood based on the computation of the critical path in this graph. The development of specific algorithms for computing tight lower bounds, instead of evaluating each generated solution, as well as the subsequent parallelization of the search technique, allowed to significantly improve the best known solutions.

In [10], a Genetic Algorithm (GA) that encodes solutions as a juxtaposition of job permutations (one for each machine) and includes specific genetic operators, is one of the main attempts for using Evolutionary Algorithms, albeit the GA is not able to achieve Taillard's Tabu Search performance levels. Another population-based method, which evolves dispatching rules and uses concepts drawn from the Shifting Bottleneck heuristic, is proposed in [11].

Actually, the solution of the JSSP through Evolutionary Algorithms has been greatly aroused by the fact that it is one of the well-known hardest combinatorial optimization problems. As mentioned in [12], "how to adapt genetic algorithms to the JSSP is very challenging but frustrating" since this class of metaheuristics is not "well suited for fine-tuning of solutions around optima". The issues of encoding (representing a schedule solution within the evolutionary framework) [13] and of hybridization with other heuristic methods [12] therefore constituted a promising research path, as evidenced by the number of works focusing on it [14,15]. Also, the natural ability of GAs to deal with multi-objective problems (because of their population-based working mode) generated a great interest for the extension of the classical JSSP when several performance criteria are simultaneously considered (for instance, and apart from makespan, let us quote flow time, tardiness or earliness related objectives, as well as total weighted or max versions) [16].

The algorithm introduced by Nowicki and Smutnicki in 1996 (TSAB, acronym of Tabu Search Algorithm with Backtrack jumping, [17]) became a breakthrough in the area. A new neighborhood scheme together with a novel long-term memory included in a Tabu Search algorithm outperformed all the prior methods developed to solve the JSSP, and provided new upper bounds for many of the instances for which the global optimum remains unknown. The same authors improved their algorithm (*i*-TSAB, for *iterated*-TSAB) in 2005 [18] by adding an intensification mechanism based on a path-relinking methodology.

More recently, a GRASP technique was adopted in [19] and was significantly improved by a path-relinking technique. A Genetic Algorithm hybridized with a local search procedure (which uses the Nowicki and Smutnicki's neighborhood) was proposed in [20]. The good quality of the results reported by the authors may be attributed to a large extent to the use of the local search, which took up a large part of the total computational time of the algorithm's execution. Similarly, a Particle Swarm Optimization (PSO) technique, also hybridized with Tabu Search, was implemented in [21]. The adaptation of PSO to the JSSP required the implementation of innovative mechanisms to reproduce the cognition and particle's velocity concepts in a permutation space. A memetic algorithm combining an artificial immune system with a local search engine was developed in [22]. Finally, an Ant Colony Optimization (ACO) method associated with a Tabu Search and a Tabu Search-Simulated Annealing hybrid, were proposed, respectively, in [23,24]. These works allowed the identification of new upper bounds for some complex instances of the JSSP.

Among the great diversity of studies, a marked trend is – as would be expected and can be verified in the specialized literature – the clear superiority of local search techniques: clearly, a small move sometimes allows to jump from an unfavorable schedule structure to a much better one. Thus, exploring the neighborhood of the new schedule leads to higher quality solutions. The best results have been, therefore, obtained by Tabu Search based algorithms, more particularly by those proposed by Nowicki and Smutnicki (TSAB [17] and *i*-TSAB [18]). They are based on the use of an efficient neighborhood structure (the *N5* neighborhood) that uses the specific structure of the JSSP in order to consider only moves that are likely to improve the current solution. Using an appropriately designed set of experiments, [25] have highlighted that, in addition to the efficiency of the *N5* neighborhood, the reasons why TSAB and *i*-TSAB perform so good are because of their use of a long-term memory and the balance between exploration and intensification steps. Both of these features are implemented for the case when the local search procedure gets trapped in a local optimum and needs to be reinitialized.

Since the introduction of the novel mutation operator proposed by Storn and Price in 1997 [26], Differential Evolution (DE) has proved its efficacy and efficiency on a wide variety of continuous optimization problems: a variety of (constrained and unconstrained) instances have been successfully solved, for example, in [26,27]. The implementation of this differentiation operator and the resulting algorithmic performances are, of course, made easier by the real parameter encoding. One would however wonder if this mutation scheme can be reproduced in a combinatorial framework. Even though this might involve the implementation of a somewhat cumbersome encoding technique, this approach seems quite promising. This idea was put forward, using the JSSP as a case study, in [28] and constituted, to the best of our knowledge, the first attempt to apply DE to the classical JSSP (note, however, that a multi-objective memetic algorithm based on DE was proposed in [29] for objectives related to tardiness and job completion times). In [28], we showed that DE alone was not able to compete with other algorithms such as (for instance) a simple Genetic Algorithm (GA), GRASP, or Tabu Search. This (rather disappointing) conclusion can be explained by the inadequacy between the permutation representation scheme and the mutation operator, which did not allow the latter to work as well as in the continuous case (particularly, because it removed the self-adaptiveness feature of the differentiation operator). A possible alternative was to use a local search procedure, hybridized with DE, in order to intensify the search in the neighborhood of the promising regions determined by the evolutionary process.

Concerning the choice of the local search heuristic, it has been shown in [25] that the core procedure does not fully explain the quality of Nowicki and Smutnicki's Tabu Search based algorithms, but rather the above-mentioned internal mechanisms: the specific *N5* neighborhood, the long-term memory and the alternated exploration/intensification steps. So, the choice of the local search method does not really constitute a matter of importance. However, for the sake of simplifying our implementation, we chose to adapt a Tabu Search (TS) technique in order to locally improve the best solutions identified by DE. We therefore propose in this paper a DE–TS hybrid algorithm, whose aim is to identify promising regions in the search space through the use of DE. Then, the insertion of local search steps (inside the DE's evolution loop) is used to determine the local optimum within such regions.

The remainder of this paper is organized as follows. Section 2 proposes a detailed definition of the JSSP problem and of the commonly used formalisms to model it. Section 3 describes the implementation of the hybrid DE–TS algorithm. Section 4 presents the computational results obtained on various sets of instances commonly used in the specialized literature. Finally, some conclusions and possible paths for future research are provided in Section 5.

2. Problem statement

The classical JSSP aims at assigning a set of operations to resources, or machines. These resources are typically considered

to be scarce (if there are more machines than jobs, the problem complexity is considerably reduced). Then, the JSSP consists of a finite set M of machines ($|M| = m$), a finite set O of operations and a finite set J of jobs ($|J| = n$). Each job $j \in J$ is characterized by a subset of operations $O_j \subset O$ that must be processed according to a defined sequence and each operation $o_{ij} \in O_j$ must be processed on a specific machine i.

Unlike the flow-shop case, the processing sequence (of the job operations on the machines) differs from one job to another. The number of operations for a job is, in most cases, equal to the machine number, meaning that each job j is processed exactly once on each machine i. Therefore, the total number of operations is $|O| = nm$ and the commonly used nomenclature refers to $n \times m$-instances.

The typical objective of the classical JSSP is then to minimize the completion time of the last operation scheduled, namely the makespan, while respecting the following major constraints:

- one machine cannot simultaneously process more than one job at a time,
- preemption (i.e., the interruption of the treatment of an operation in order to process another operation) is not allowed,
- the processing sequence of the operations belonging to a job must be respected, which imposes that the starting time of any operation has to be higher than the completion of its predecessor in the job sequence.

The solution is an assignment of operations to machines on a precise time period and is called a *schedule*. When an appropriate schedule builder is used (this aspect is developed later in the paper), this solution can be formulated as a multi-permutation, i.e., a set of job permutations associated to each machine. The resulting complexity of the JSSP is therefore $(n!)^m$.

2.1. Schedule classes

As mentioned in the previous section, the sole permutation of jobs on each machine is not sufficient to determine a schedule and compute the associated makespan objective. Actually, for a certain multi-permutation, an infinity of schedules might be built. An important issue therefore concerns the construction of a feasible schedule, which is a schedule that respects the above-mentioned working hypothesis.

Among the feasible schedules possibly built from a given multi-permutation, the active schedules are those for which no operations can be brought forward without delaying another operation. It is well known that optimal schedules belong to the active class [30]. Another relevant schedule class is the non-delayed one, for which no idle time is allowed on a machine if this latter is free and an operation is available for processing. This latter class also constitutes a subset of the active schedules class, but may not contain the optimal solution. For illustrative purposes, Fig. 1, taken from [30], presents an overview of the most relevant schedule classes.

In order to produce active schedules from a job permutation (or a priority list of jobs) for each machine, Giffler & Thompson's algorithm is a commonly used tool [31]. This algorithm works by determining, among the available operations, the one whose treatment may be completed first. This is the critical operation. All the operations whose processing can start on the same machine, before the completion time of the critical operation are the conflicting operations. The operation to schedule is then determined among the conflict set according to the permutation (or priority list) provided by the user, or by any optimization technique.

A modification of Giffler & Thompson's algorithm consists in restricting the time interval defining the set of conflicting

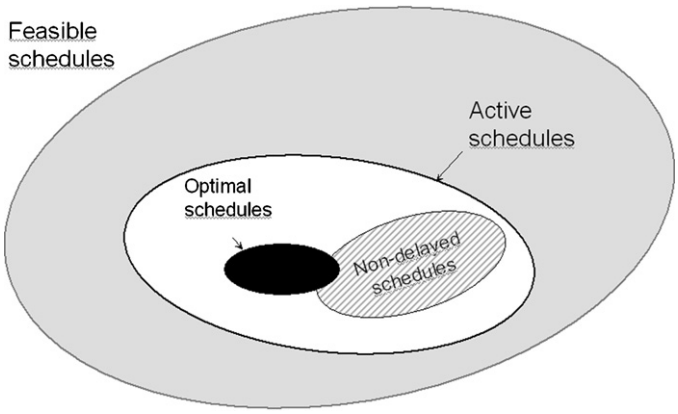

Fig. 1. Illustration of the most relevant schedule classes.

operations, through a parameter δ that basically represents a proportion of the critical operation processing time: $\delta = 1$ corresponds to the original algorithm (i.e., the complete processing time of the critical operation is the interval in which conflicting operations may start) and produces active schedules. When δ is decreased, so does the time interval defining the conflict set, therefore reducing the number of conflicting operations. When $\delta = 0$, the produced schedule is non-delayed (i.e., the earliest available operation is processed first). So, the parameter δ allows to control the search space, which becomes more or less restricted between the non-delayed and the active schedule classes.

Adopting the following notations:

(i, j): the operation of job j that needs to be processed on machine i,
S: the partial schedule that contains scheduled operations,
Ω: the set of schedulable operations,
s_{ij}: the earliest time at which operation $(i, j) \in \Omega$ could be started,
p_{ij}: the processing time of operation (i, j),
c_{ij}: the earliest time at which operation $(i, j) \in \Omega$ could be completed:
$c_{ij} = s_{ij} + p_{ij}$,

the description of the parameterized Giffler & Thompson's algorithm is provided in a straighforward manner:

Step 1. Initialize $S = \varnothing$; Ω is initialized to contain all operations without predecessors.
Step 2. Determine $c^* = \min_{(i,j)\in\Omega}\{c_{ij}\}$. Deduce the associated starting time s^* and the machine m^* on which c^* could be realized.
Step 3. (1) Identify the operations $(i_0, j_0) \in \Omega$ such that (i_0, j_0) requires machine m^*, and $s(i_0, j_0) < s^* + \delta(c^* - s^*)$ (instead of $s(i_0, j_0) < c^*$ in the initial G&T's algorithm).
(2) Choose (i, j), among the operations (i_0, j_0) identified in (1), with the largest priority (or appearing first in the job permutation).
(3) Add (i, j) to S.
(4) Assign s_{ij} as the starting time of (i, j).
Step 4. If a complete schedule has been generated, stop. Else, delete (i, j) from Ω and include its immediate successor in Ω. Then, go to Step 2.

2.2. The disjunctive graph representation scheme

Many formal representations of the JSSP exist but the most traditional model is a disjunctive graph proposed in [5]. This model is detailed here because it will be useful to understand the neighborhood subsequently adopted within the Tabu Search

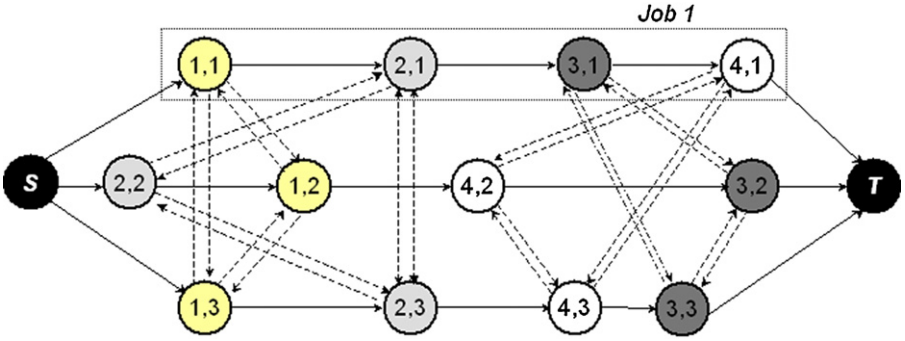

Fig. 2. Disjunctive graph representation of the JSSP.

procedure, which is hybridized with Differential Evolution in this paper.

The disjunctive graph is denoted as $G=(V, A, E)$ where: (i) V is a set of nodes standing for the set of operations O plus two fictitious nodes (source S and sink T); (ii) A is a set of directed – conjunctive – arcs representing the sequence between operations belonging to the same job (or precedence constraints); (iii) E is a set of disjunctive edges, connecting the operations that must be processed in a same machine. All edges (in A and in E) are weighted with a value equal to the processing time of the operation corresponding to their origin node (except for edges from source S, which have no weight).

A job permutation is then directly deduced from the direction of the oriented disjunctive edges. The resulting schedule is feasible whenever the set of arcs determined by $A \cup E$ does not have any cycle. Then, given a consistent orientation of all edges in E, the makespan can be computed as the longest (weighted) path, from the source S to the sink T, called critical path. The objective of the JSSP is thus to find an orientation of E such that the longest path in G is minimized.

Fig. 2 provides an illustration of the disjunctive graph representation for a 3×4-JSSP. Note that the first index of each node indicates the machine on which the operation must be processed, while the second one indicates the job the operation belongs to.

3. Description of the proposed algorithm

As mentioned in the introduction, our early attempt for solving the JSSP with Differential Evolution in [28] was not that satisfactory since the obtained results were not competitive with respect to those generated by state-of-the-art algorithms (and, particularly, with respect to Nowicki and Smutnicki's Tabu Search based algorithms). So, the aim of this study is to improve the results provided by DE by hybridizing it with a local search technique.

For the sake of simplicity, the local search technique adopted is a classical Tabu Search (TS) algorithm, based on the specific neighborhood proposed in [17], which has been found to be particularly efficient for the JSSP. However, our solution technique does not integrate the features that constitute the main strengths of *i*-TSAB (e.g., long-term memory and exploration/intensification procedures, [18]). Our aim is thus to investigate the ability of DE to replace these features in order to identify promising regions, whose local optimum is subsequently located by TS.

The main characteristics of the proposed algorithm are presented in this section, including a short outline of DE and the specific representation schemes used to encode the JSSP, the features of Tabu Search and the *N5* neighborhood, and the description of our hybrid DE–TS algorithm implementation.

3.1. Differential Evolution and adaptation to permutation-based problems

3.1.1. General outline on DE

Differential Evolution (DE) is an Evolutionary Algorithm (EA), originally introduced in [26]. The method was initially proposed for continuous optimization problems. Its basic principle lies on two components: (1) the design of a simple mutation operator that linearly combines three different individuals, and (2) a crossover step that mixes the initial solutions with the mutated ones.

To a certain extent, the implementation of the DE mutation operator can be seen as a self-adaptive methodology, such as the one adopted by the Evolution Strategies [32], since it produces a noise on the involved individuals that gradually decreases when the population converges toward the optimum [33].

Several DE versions exist, depending on the choice of the individuals used to build the mutant individual and depending on the combination of mutated and initial solutions that takes place during the crossover step. The version presented next, which is the most popular one, is called DE/rand/1/bin. The name indicates that the three individuals used for the mutation process are randomly chosen, that there is only one differentiation operation for building the mutant individual, and that the crossover test (assigning either the initial or the mutated individual gene according to a binomial distribution) is performed for each gene independently from what happened to the previous ones. An overview of the other available DE versions is available in [26]. In fact, four other DE versions are presented at the end of this section.

As in most EAs, a selection process is carried out by comparing the trial solutions (obtained by the mutation and crossover steps) to the initial ones. However, a trial vector is not compared with respect to the entire population, but only against its counterpart in the current population. This working mode implies that the new population is necessarily as good as or better than the current generation.

The canonical steps of the algorithm are defined as follows:

Step 1. Random generation of *NP* initial individuals.
Step 2. Evaluation of the objective function f for each individual.
Step 3. For each individual x_i^G in the current population (i being the individual index and G the current generation number), create a mutated solution u_i^G using:

$$u_{ij}^G = x_{1j}^G + F(x_{2j}^G - x_{3j}^G), \quad \forall j \in \{1, \ldots, N\} \tag{1}$$

N is the number of decision variables, F is a scalar ($F \in [0, 1]$) and x_1^G, x_2^G, x_3^G are randomly selected individuals in the current population ($x_1^G \neq x_2^G \neq x_3^G \neq x_i^G$). In Eq. (1), j represents the variable index, so that the operation described by the equation must be performed for each dimension j of the mutated solution u_i^G.

Step 4. Recombination of the initial and mutated individual according to a crossover rate CR ($CR \in [0, 1]$), to create an offspring u_i^{G+1} such as:

$$u_{ij}^{G+1} = \begin{cases} u_{ij}^G & \text{if } rand(j) \le CR \text{ or } j = rnbr(i) \\ x_{ij}^G & \text{if } rand(j) > CR \text{ and } j \ne rnbr(i) \end{cases}, \quad \forall j \in \{1, \ldots, N\}, \forall i \in \{1, \ldots, NP\} \tag{2}$$

$rnbr(i)$ is a random index ($rnbr(i) \in \{1, \ldots, N\}$) generated for each created mutant, which ensures that u_i^{G+1} gets at least one variable from u_i^G. $rand(j)$ is a random number chosen for each decision variable j.

Step 5. Evaluation of the objective function of the offspring u_i^{G+1} and application of a deterministic selection process by comparing the parent and offspring objective values:

$$x_i^{G+1} = \begin{cases} u_i^{G+1} & \text{if } f(u_i^{G+1}) \le f(x_i^G) \\ x_i^G & \text{if } f(u_i^{G+1}) > f(x_i^G) \end{cases}, \quad \forall i \in \{1, \ldots, NP\} \tag{3}$$

Step 6. If the termination criterion is met, output the best solution found so far and its objective value; else, go back to *Step 3*.

The typical termination criterion is to reach a given number of generations NG. F (amplification or scaling factor) and CR (crossover rate) are control parameters that, with NP (population size), must be tuned by the user in an appropriate way. [34] indicates that a fine tuning of these parameters might improve the convergence velocity and the robustness of the search process.

Note, furthermore, that in the computational experiments described in Section 4, four other DE versions, among the most representative ones, were initially considered. These versions, namely DE/best/1/bin, DE/current-to-best/1/bin, DE/best/2/bin and DE/rand/1/exp, are defined in what follows. The two former variants differ from the classical DE/rand/1/bin version in the way individuals involved in the mutation operator are selected. These operating modes are formulated in Eqs. (4) and (5):

$$u_{ij}^G = x_{Best,j}^G + F(x_{1j}^G - x_{2j}^G), \quad \forall j \in \{1, \ldots, N\} \tag{4}$$

$$u_{ij}^G = x_{ij}^G + F(x_{Best,j}^G - x_{ij}^G) + F(x_{1j}^G - x_{2j}^G), \quad \forall j \in \{1, \ldots, N\} \tag{5}$$

Variant DE/rand/2/bin involves two differentiation operators in the mutation step:

$$u_{ij}^G = x_{1j}^G + F(x_{2j}^G - x_{3j}^G + x_{4j}^G - x_{5j}^G), \quad \forall j \in \{1, \ldots, N\} \tag{6}$$

where x_4^G and x_5^G are also randomly selected individuals in the current population (with $x_4^G \ne x_5^G \ne x_1^G \ne x_2^G \ne x_3^G \ne x_i^G$). Finally, the DE/rand/1/exp version is identical to DE/rand/1/bin for the mutation operator. However, the crossover step is performed according to an exponential distribution, involving that genes of the offspring solution are inherited from the mutant individual until a random number (drawn for each new gene) is greater than the crossover rate, CR.

3.1.2. Permutation encoding

When applying mathematical programming techniques to solve the JSSP, the employed variables are, typically, the operation starting times. On the contrary, EAs must directly handle permutations representing job priorities for each machine. In this framework, specific encoding schemes have to be developed. A rather comprehensive overview of encoding techniques is provided in [13] for Genetic Algorithms (GAs). For instance, it is possible to adopt encodings by means of job or operation based lists, preference list-based or priority rule-based representations, etc. But if GAs are able to handle permutations through the definition of appropriate genetic operators, this is not the case for DE, which is based on real numbers encoding. However, some techniques presented in [13] are still applicable when working with real numbers encoding. Such techniques were implemented in [28] and tested on some JSSP instances.

The first method, namely Random Keys, was initially proposed in [35]. A real number, bounded between 0 and 1, is used for each operation. Then, operations corresponding to the same machine are sequenced according to the increasing order of their associated variable. For instance, considering 5 jobs with the following variable vector on machine i: [0.41 0.68 0.02 0.85 0.37], the resulting sequencing order is: [2 4 0 1 3]. For a $n \times m$-JSSP, $n \times m$ variables are needed to represent a complete multi-permutation.

The second tested method is based on the use of dispatching rules. These rules are generally used for simple scheduling problems (for instance, in a single-machine environment, with different kinds of objectives) because they are able to produce optimal solutions without excessive computational cost: the SPT (shortest processing time first), LST (least work remaining first), FCFS (first come first served) or MS (minimum slack first) rules only represent some examples among a wide range of commonly used techniques (we refer the reader to [30] for a more exhaustive description of the commonly used dispatching rules). In [11], Dorndorf and Pesch drew their inspiration from this working mode to introduce the idea of evolving a set of dispatching rules, which will be subsequently used during the schedule building process in order to solve conflicting cases (i.e., when several operations are valid candidates for the processing on a machine). A variable must, in this sense, identify a rule that would settle a conflict in any iteration of the schedule builder algorithm. Since there may be at most $n-1$ conflicts for each machine, this encoding technique needs $(n-1) \times m$ variables for a $n \times m$-JSSP.

The last tested representation method was first presented in [36] with GAs. According to the disjunctive graph, a binary variable x_{ijk} is associated to each disjunctive arc and enforces the priority of two operations j and k on machine i: $x_{ijk} = 0$ if k is scheduled before j and $x_{ijk} = 1$, otherwise. The drawback is that local and global infeasibilities, due to the presence of cycles in the disjunctive graph, can appear. The inconsistent permutations are fixed through harmonization processes that manipulate the variables in order to produce valid priorities. This procedure was slightly modified in [28] for real variables (instead of the binary genes used in the GA). The number of variables is greater than those of the two former cases: for a $n \times m$-JSSP, $n(n-1)m/2$ variables are needed.

The computational experiments carried out in [28] showed that the first and third of the above methods provided the best results. In addition, because of the higher number of variables needed by the binary priorities proposed in [36], the Random Keys technique was found to be the fastest one, for equal numbers of objective function evaluations. Thus, the Random Keys representation scheme will be adopted in our DE implementation.

3.2. Tabu Search and the neighborhood for JSSP

Tabu Search [37] is a neighborhood exploration technique proposed by Fred Glover in 1986. Its working mode is a classical local search process, which visits neighboring solutions by performing simple moves from the current solution. When a better solution is found, the current point is updated and the process is repeated until no improvement is possible. The key-point of the method lies on the use of a tabu list, in which the moves performed in

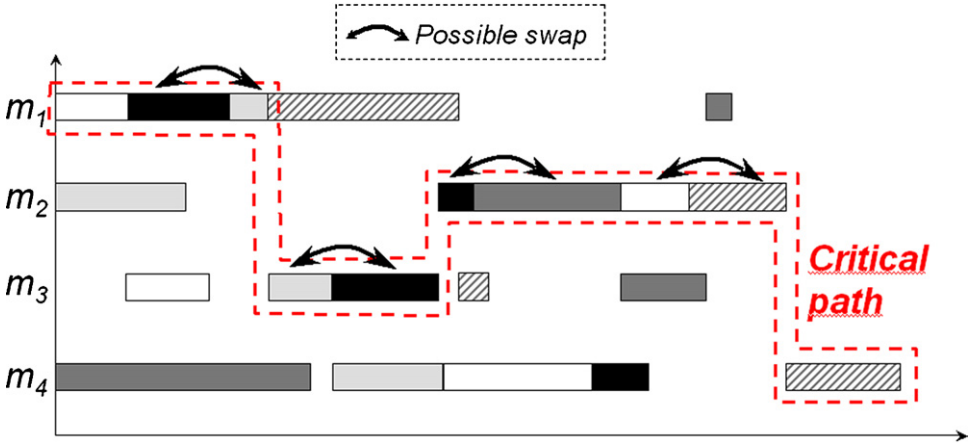

Fig. 3. Illustration of Nowicki and Smutnicki's *N5* neighborhood.

previous iterations are stored in order to avoid cycling moves, such as $j \rightarrow j' \rightarrow j$ (where j and j' are neighboring solutions).

In addition, re-start procedures and intensification/exploration strategies can be aggregated as external loops, in order to visit various regions of the search space and determine their local optimum. These features are not considered in the TS implementation adopted in this study.

The neighborhood choice is essential for an efficient implementation of TS: a vast neighborhood will avoid losing the optimum but will be computationally expensive, while a restricted neighborhood will allow savings in the computational time but will possibly prevent us from reaching the optimum. This balance is, obviously, to be considered in every problem and mostly in the JSSP, where the computational cost might be prohibitive.

The various neighborhoods proposed in the literature exploit the structure of the JSSP: being a list of job permutations, a solution can be easily modified by swapping two consecutive operations on a machine. This neighborhood structure is, however, rather inefficient since the number of possible moves is huge and most of them do not have any impact on the makespan.

Thus, in [9] for instance, an improved neighborhood is suggested, which considers a subset of the above-mentioned one, taking into account that only moves concerning operations belonging to the critical path (i.e., the sequence of consecutive operations that defines the makespan) can modify the objective. Then, the authors propose to swap only consecutive operations on the critical path, leading to a considerable reduction of the neighborhood size without producing a negative impact on the quality of the solution.

Finally, [17] further reduced the previously developed neighborhoods. They introduced the concept of blocks, referring to the sets of consecutive operations belonging to the critical path and processed on the same machine. The authors demonstrated that swapping consecutive operations lying "inside" a block (i.e., that do not involve any of the extreme operations of the block) cannot improve the makespan. Therefore, the resulting *N5* neighborhood only considers swapping the first two or the last two operations on a block (except for the first and last blocks on the critical path). Fig. 3 illustrates this mechanism, which has been adopted in the study presented in this paper.

3.3. The hybridization scheme

Once the internal methods that would constitute the proposed algorithm are defined, the way the hybridization is carried out should be carefully determined. In particular, one needs to particularly focus on some issues, which are explained in detail below.

First, we chose to introduce the TS procedure within the DE generational loop, after the application of recombination and mutation operators. Some individuals from the new generation are then modified by the TS algorithm. Compared with the option of applying TS to the best final solutions provided by DE, the selected strategy enables determining the local optimum of various regions located during the evolutionary search, before convergence is reached, in order to help the algorithm to find the global optimum. Additionally, preliminary experiments had shown that running the TS procedure at each generation was too expensive and that the computational times spent on TS and on DE were significantly unbalanced. Some tests then showed that calling the TS procedure at every 10 generations resulted in a more equitable balance without affecting the solution quality.

Besides, the selection of the individuals on which local search is applied is performed in such a way that only the best solutions can be involved, but also introducing a stochastic aspect. Two parameters should therefore be defined before running the algorithm: the first one, p_{Best}, determines what proportion of the current population represents the best solutions, while the second one, p_{Sel}, defines the number of individuals selected for the TS stage. These individuals are randomly chosen from among the $p_{Best} \times NP$ best solutions of the generation (NP is the population size). In this study, p_{Best} and p_{Sel} were determined through preliminary tests for the minor size instances (in these cases, $0.02 \leq p_{Best} \leq 0.2$ and $0.02 \leq p_{Sel} \leq 0.1$); for huge instances, the computational cost prevented us from carrying out previous sensitivity analysis tests, and thus, we chose: $p_{Best} = 0.1$ and $p_{Sel} = 0.05$.

Finally, the last feature to be defined is the hybrid algorithm working mode regarding the solutions modified by TS: for some instances, computational tests were performed to evaluate the behavior of a Baldwinian strategy (i.e., the makespan of a solution modified by the TS is only used for objective evaluation, but the corresponding variables are not subsequently used in the evolutionary search) and a Lamarckian strategy (i.e., the variables of the solutions modified by the TS are sent back to DE and thus have an impact on the evolutionary search). For more details on Lamarckian and Baldwinian strategies, we refer the reader to [38,39]. The results are not shown here because, surprisingly, no significant difference in terms of solution quality could be highlighted, even though a (very) slight superiority of the Lamarckian algorithm suggested the use of such a strategy.

Note, however, that the representation of a solution differs from DE to TS. As indicated before, a Random Keys encoding scheme was adopted within DE. When the TS procedure is called, the DE solutions are decoded in order to determine the associated job permutation on each machine and the schedule is built by the Giffler &

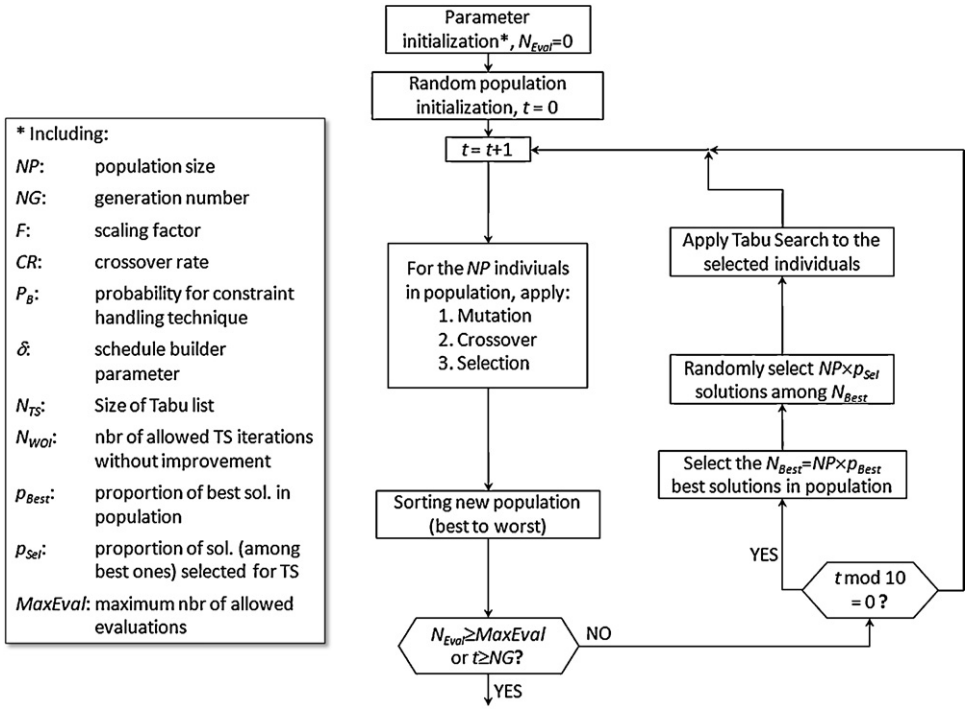

Fig. 4. Flowchart of the global hybrid DE–TS algorithm.

Thompson's algorithm. Then, TS works directly on the schedule (in order to define, at each iteration, the critical path, the corresponding blocks and the possible neighbors). As a consequence, the use of Lamarckism necessarily involves encoding the local optimum found by TS in order to send it back to DE.

No additional features are to be mentioned here since the rest of the implementation is rather classical. Fig. 4 illustrates with a flowchart the global hybridization scheme for the hybrid DE–TS algorithm. Concerning the DE and TS parameters, they are presented in the next section.

4. Computational experiments

4.1. Methodology

This section presents the results obtained with our hybrid DE–TS algorithm on several sets of JSSP instances commonly used in the specialized literature. These examples, drawn from the OR-library [40], were generated randomly and can be classified in terms of size and difficulty. We considered the following benchmarks:

- 10 instances due to [41]: ORB01-ORB10, whose size is 10×10, can be considered as the simplest instances tackled in this study;
- 40 instances due to [42]: LA01-LA40, are divided into eight subsets of five identical size problems, ranging from 10 × 5 to 15 × 15, can be considered as moderately complex instances;
- 4 instances due to [43]: YN1-YN4, whose size is 20×20, can be considered as very complex instances;
- 50 instances due to [44]: TA01-TA50, are divided into five subsets of ten identical size problems, ranging from 15 × 15 to 30 × 20 and from moderate to very high complexity for the last subsets.

Note that the FT and ABZ instances (introduced in [45,6], respectively), which are also commonly tackled benchmarks, are not considered here since they do not constitute challenging problems anymore (most of the recent algorithms solve them to optimality and so does our DE–TS hybrid algorithm).

The solutions provided by our algorithm are compared against some state-of-the-art algorithms. The comparison stands on the basis of similar numbers of objective function evaluations. Note, however, that, depending on the considered solution technique and the specific hardware adopted, the computational time needed for one objective evaluation can vary a lot. Indeed, some methods, instead of computing at each iteration the whole schedule and its associated makespan, make use of cheaper techniques. For instance, [9] develops a procedure that avoids computing the entire configuration and evaluate instead a lower bound of the latter. Similarly, the value of the objective of a current solution is used to deduce, through a very efficient algorithm, the makespan of the visited neighbors in [18].

Furthermore, in some of the solution methods adopted in our comparison, a specific and efficient initialization technique is implemented, such as in TSAB and *i*-TSAB, in which a powerful heuristic (an insertion technique, refined for job-shop problems) directly provides, for some instances, a near-optimal solution. The efficacy of the core solution technique becomes then hard to evaluate with respect to randomly initialized methods. Finally, it is worth recalling that, in our algorithm, the objective evaluation is heavier for DE since it requires to decode the current solution into a multi-permutation and then applies the Giffler & Thompson's algorithm, while TS only manipulates the schedule in order to produce a neighbor. Here, we consider both types of evaluations as identical when counting them. However, and in spite of all of these reasons, we still chose to use a number of objective function evaluations similar to those reported in the literature [25] as our termination criterion.

We acknowledge that this working mode does not guarantee exactly identical operating conditions, even though the above discussion indicates that these cannot be reached. But, we believe that the details provided on the hardware platforms used for the execution of each algorithm (see Section 4.4.) allows the reader to evaluate and compare their relative performance in a fair manner.

We compared the performance of the DE–TS algorithm with respect to those of a combination of recent and older solution techniques, some of them constituting the best approximation

algorithms for the JSSP. Depending on the considered instances, the comparison of our algorithm with the following methods is drawn only when complete results are available.

- **SB, [6].** Shifting Bottleneck procedure.
- **SBGA, [11].** Genetic Algorithm evolving dispatching rules and using a Shifting Bottleneck heuristic.
- **TSAB, [17].**
- **BV, [46].** Guided Local Search with a Shifting Bottleneck heuristic.
- **TSSB, [47].** Tabu Search method guided by the Shifting Bottleneck heuristic.
- **DS, [48].** Parallelized Simulated Annealing.
- **HGA, [20].** Genetic Algorithm hybridized with a Local Search based on the *N5* neighborhood.
- **GRASP, [19].** GRASP technique with a a local search mechanism swapping two consecutive operations on the critical path. Additionally, a path-relinking procedure is introduced in order to intensify the search.
- ***i***-TSAB, **[18].**
- **FL, [49].** Hybrid GRASP procedure that uses B&B during the construction step.
- **SVS, [50].** Tabu Search with an initialization phase based on dispatching rules heuristics and *N5* neighborhood.
- **ACOFT, [23].** Hybrid algorithm combining an Ant Colony Optimization procedure (with a construction mechanism based on the Shifting Bottleneck heuristic) and Tabu Search (similar to Nowicki and Smutnicki's TSAB) to locally improve the obtained solutions.
- **TSSA, [24].** Hybrid algorithm combining Tabu Search and Simulated Annealing (SA). SA is used to construct elite solutions in the promising Big Valley region; TS is subsequently applied for intensification.
- **HA, [51].** Memetic algorithm, based on the extension of a GA by the introduction of priority rules (partial re-ordering, gap reduction, restricted swapping rule) as local search techniques.

4.2. DE and TS working parameters

As previously discussed, both DE and TS have internal parameters that should be appropriately tuned for an efficient work. Concerning DE, preliminary computations are first carried out with the five DE variants presented in Section 3.1.1, on a small subset of instances, in order to determine the best performer. The influence of parameters *F* and *CR* is also evaluated through exhaustive sensitivity analysis. The conclusions of these parameter tuning tests are shown in the next section. Besides, the population size and maximum number of generations are to be set in such a way that the number of objective evaluations does not exceed a fixed value. However, since the termination criterion is the total number of objective function evaluations, the number of generations is generally set to a sufficiently high value such that it is never reached (the search actually stops when attaining the pre-defined number of objective function evaluations). Regarding the population size, number of individuals ranging from 20 for the simplest instances (LA01-LA10) up to 600 (instances YN or TA20–50) were determined after preliminary computations. Moreover, the differentiation mutation is likely to produce variables lying outside their bounds and, thus, a mixed constraint-handling technique is applied according to a given probability P_B (typically set to 0.5): (i) setting the variable value to the violated bound; (ii) using the violated bound as a symmetry center to send the considered variable to the feasible side of the boundary. Due to space limitations, we do not present the computational results of the sensitivity analysis performed to adequately tune the above-mentioned DE's internal parameters.

Regarding TS, besides the above-mentioned neighborhood construction, our implementation is rather classical and uses the parameters suggested in [18]: the size of the tabu list is equal to 8, the allowable number of iterations without improvement (first termination criterion) varies between 2500 and 10,000. The cycle detection checks if some repetition in the visited neighbors is observed twice. In case only forbidden nonprofitable moves are available, the "oldest" move is selected and the tabu list is additionally updated by replications of the "youngest" move (we refer the reader to [17] for more details).

4.3. Preliminary tests

Within this set of computational experiments, the performance level of the five most representative DE versions, presented in Section 3.1.1, is evaluated to identify the best performer. For each tested version, the effect of parameters *F* and *CR* is studied, by running exhaustive sensitivity analysis computations. These preliminary tests are conducted, however, over only a subset of the whole benchmark defined in Section 4.1. The reason is that we cannot afford to perform the huge amount of computational tests that would be required for including all the (more than 100) instances. Thus, although extrapolating a single parameter setting obtained over a subset of all instances might not reach optimal performances, the best DE version, with the most appropriate values of *F* and *CR* will be subsequently maintained for the remainder of the numerical experiments.

The instances used within this first computational round were selected according to two criteria: non-triviality (i.e., no version was able to identify the optimal value without a previous parameter tuning) and size (small or medium size examples were chosen in order to reduce the computational time devoted to these experiments). The list of chosen instances is therefore as follows: ORB05, LA24, LA29, LA37, LA38, LA39, LA40, TA05, TA09. For each selected instance, parameters *F* and *CR* were varied with possible values *F*=0.3, 0.5, 0.7, $rand_{0.3-0.9}$ and *CR*=0.3, 0.5, 0.7, $rand_{0.8-1.0}$. Values denoted as $rand_{0.3-0.9}$ (or $rand_{0.8-1.0}$) indicate that the parameter adopts a randomly generated value, uniformly distributed in the range [0.3,0.9] (or [0.8,1.0]).

For each instance and each tested parameter setting, 20 independent executions were performed. The reported values indicate the Makespan Relative Error of the best makespan f_{Best} found (b-MRE) and of the mean value of the makespan over the 20 executions f_{Mean} (m-MRE), with respect to a reference makespan which is typically a lower bound LB (MRE = 0 indicates that the corresponding instance is solved optimally). The relative error is expressed as a percentage and is computed according to the following formula:

$$\mathrm{MRE} = 100 \times \frac{f_{Best\ or\ Mean} - \mathrm{LB}}{\mathrm{LB}} \tag{7}$$

The results obtained for these computations are presented in Table 1, which shows, for each instance and each DE version, the values of b-MRE and m-MRE, as well as the values of parameters *F* and *CR* that obtained the indicated performance measure.

From the results of the above table, it appears that all five DE versions have similar performance levels. DE/rand/1/bin is slightly better than the others for averaged b-MRE. Regarding averaged m-MRE, four versions obtain almost identical results, DE/current-to-best/1/bin being the best one, while DE/rand/1/exp is left behind. Concerning the tuning of parameters *F* and *CR*, one the one hand, it is quite clear that using a high value of *CR* ($rand_{0.8-1.0}$) is the most suitable choice. On the other hand, for *F*, it seems difficult to establish any pattern that could be applied for all versions and all instances This suggests that $rand_{0.3-0.9}$, which provides varying values in a relatively wide range, might be the most reasonable choice.

Therefore, it might seem acceptable to choose DE/rand/1/bin for the remainder of computational experiments (over the complete

Table 1
Sensitivity analysis results.

Instance	Rand/1/bin		Best/1/bin		Current-to-best/1/bin		Rand/2/bin		Rand/1/exp	
	b-MRE	m-MRE	b-MRE	m-MRE	b-MRE	m-MRE	b-MRE	m-MRE	b-MRE	m-MRE
ORB05	0.000	0.378	0.225	0.293	0.000	0.265	0.225	0.400	0.225	0.490
	$F=0.5$, $CR=rand_{0.8-1.0}$		$F=0.5$, $CR=rand_{0.8-1.0}$		$F=rand_{0.3-0.9}$, $CR=rand_{0.8-1.0}$		$F=0.3$, $CR=rand_{0.8-1.0}$		$F=0.7$, $CR=rand_{0.8-1.0}$	
LA24	0.642	1.294	0.642	1.203	0.428	1.235	0.642	1.235	0.428	1.310
	$F=0.7$, $CR=rand_{0.8-1.0}$		$F=0.3$, $CR=0.3$		$F=rand_{0.3-0.9}$, $CR=rand_{0.8-1.0}$		$F=0.3$, $CR=0.3$		$F=0.5$, $CR=rand_{0.8-1.0}$	
LA29	0.521	1.467	0.694	1.467	0.694	1.450	0.955	1.411	0.955	1.484
	$F=0.7$, $CR=rand_{0.8-1.0}$		$F=rand_{0.3-0.9}$, $CR=rand_{0.8-1.0}$		$F=rand_{0.3-0.9}$, $CR=rand_{0.8-1.0}$		$F=0.7$, $CR=rand_{0.8-1.0}$		$F=0.7$, $CR=rand_{0.8-1.0}$	
LA37	0.716	2.133	0.716	2.090	0.859	1.901	0.716	1.793	0.215	1.986
	$F=rand_{0.3-0.9}$, $CR=rand_{0.8-1.0}$		$F=0.5$, $CR=rand_{0.8-1.0}$		$F=0.5$, $CR=rand_{0.8-1.0}$		$F=0.3$, $CR=rand_{0.8-1.0}$		$F=0.5$, $CR=0.3$	
LA38	0.000	1.417	0.000	1.731	0.502	1.250	0.502	1.459	0.251	1.777
	$F=0.7$, $CR=0.7$		$F=0.5$, $CR=rand_{0.8-1.0}$		$F=0.5$, $CR=0.7$		$F=0.3$, $CR=rand_{0.8-1.0}$		$F=rand_{0.3-0.9}$, $CR=rand_{0.8-1.0}$	
LA39	0.081	1.245	0.000	1.006	0.000	1.095	0.081	1.265	0.000	1.188
	$F=rand_{0.3-0.9}$, $CR=0.7$		$F=rand_{0.3-0.9}$, $CR=rand_{0.8-1.0}$		$F=0.3$, $CR=rand_{0.8-1.0}$		$F=0.3$, $CR=0.3$		$F=0.5$, $CR=0.7$	
LA40	0.164	0.532	0.164	0.646	0.245	0.569	0.245	0.642	0.245	0.749
	$F=0.3$, $CR=rand_{0.8-1.0}$		$F=0.7$, $CR=rand_{0.8-1.0}$		$F=0.7$, $CR=rand_{0.8-1.0}$		$F=0.5$, $CR=0.5$		$F=0.7$, $CR=rand_{0.8-1.0}$	
TA05	0.490	0.788	0.327	0.752	0.490	0.760	0.408	0.772	0.408	0.817
	$F=rand_{0.3-0.9}$, $CR=rand_{0.8-1.0}$		$F=0.3$, $CR=rand_{0.8-1.0}$		$F=rand_{0.3-0.9}$, $CR=0.5$		$F=0.5$, $CR=rand_{0.8-1.0}$		$F=0.7$, $CR=rand_{0.8-1.0}$	
TA09	0.000	0.491	0.000	0.565	0.000	0.679	0.000	0.895	0.000	0.918
	$F=0.3$, $CR=rand_{0.8-1.0}$		$F=0.7$, $CR=rand_{0.8-1.0}$		$F=0.5$, $CR=rand_{0.8-1.0}$		$F=rand_{0.3-0.9}$, $CR=rand_{0.8-1.0}$		$F=0.5$, $CR=rand_{0.8-1.0}$	

problem benchmark). In order to show that, anyway, the choice of the DE version has little impact on the obtained results, the previously indicated DE versions were statistically analyzed. For each instance, a series of Wilcoxon tests was performed, carrying out pairwise comparisons between the results of the 20 replications obtained with DE/rand/1/bin and with those of any of the four other DE versions. The null hypothesis was that both data sets arose from the same population and were identically distributed. The Wilcoxon test, applied with a 95% confidence level, has result 0 if the null hypothesis is not rejected and 1, otherwise. The conclusions from this study are presented in Table 2.

The results of the Wilcoxon tests indicate that, for all instances except LA39, LA40 and TA05, the solutions obtained by all four versions have no significant differences (the tested data sets always look like they arise from the same population). For LA39, it appears that DE/rand/1/bin obtains results significantly different from those of DE/best/1/bin: the numerical values from Table 1 indicate that, indeed, DE/best/1/bin is the only version capable of reaching the best known solution for this test problem (and, thus, having a different behavior from DE/rand/1/bin). For instances LA40 and TA09, DE/rand/1/bin shows results which are significantly different from those of DE/rand/2/bin and DE/rand/1/exp (in both cases, DE/rand/bin performs better than the two other versions). Therefore, apart from these two exceptions, the Wilcoxon test illustrates that all five versions reach comparable performance levels. Finally, this sensitivity analysis study allows us to conclude t hat the DE/rand/1/bin version can be adopted for the complete set of treated instances. For the remainder of our computational experiments, the parameter values $F=rand_{0.3-0.9}$ and $CR=rand_{0.3-0.9}$ were adopted.

Table 2
Wilcoxon tests results.

Instance	Rand/1/bin vs.			
	Best/1/bin	Current-to-best/1/bin	Rand/2/bin	Rand/1/exp
ORB05	0	0	0	0
LA24	0	0	0	0
LA29	0	0	0	0
LA37	0	0	0	0
LA38	0	0	0	0
LA39	1	0	0	0
LA40	0	0	0	1
TA05	0	0	0	0
TA09	0	0	1	1

4.4. Results and discussion

All the experiments were performed on a workstation with a 3.4GHz processor and 1GB of RAM. The computational results obtained with the hybrid DE–TS algorithm are reported in Tables 3–6 and compared with the solutions of the above-mentioned state-of-the-art algorithms.

Concerning the ORB instances, our algorithm is able to identify the optimal solution with a very good repeatability, as indicated by the m-MRE values. Except for TSSA, no method does better than DE–TS. In terms of computational times, the limit was set to 250,000 objective evaluations resulting in less than 7 s for our DE–TS. The experiments reported in [19] with GRASP took place within a parallelized framework, using 7GB of RAM and requiring computational times of at least several minutes. For SVS, no time nor objective evaluation number is mentioned in [50]. The BV algorithm was tested on a SUN Sparc-330 station and the computational times varied between 16 and 285 s. Times between 1 and 14 s were reported for TSSA, in [24], on a personal computer. Finally, note that the high b-MRE results obtained by the FL algorithm might

Table 3
Computational results for the ORB instances [41].

Instance	Size	DE–TS		GRASP[a]	FL[b]	SVS	TSSA[c]		BV[d]
		b-MRE	m-MRE	b-MRE	b-MRE	b-MRE	b-MRE	m-MRE	b-MRE
ORB01	10 × 10	0	0.279	0	8.121	0.472	0	0	0
ORB02	10 × 10	0	0.197	0	3.378	0	0	0.011	0
ORB03	10 × 10	0	1.085	0	9.254	0	0	0.746	0
ORB04	10 × 10	0	0.667	0.597	6.070	0.597	0	0.328	0.796
ORB05	10 × 10	0	0.378	0.225	2.706	0	0	0.180	0.225
ORB06	10 × 10	0	0.926	0.198	3.960	–	0	0	0
ORB07	10 × 10	0	0.202	0	4.282	–	0	0	0
ORB08	10 × 10	0	1.613	0	5.117	–	0	0.389	0
ORB09	10 × 10	0	0.327	0	4.711	–	0	0	0
ORB10	10 × 10	0	0.403	0	4.979	–	0	0	0
Average		0.023	0.610	0.102	5.258	0.214	0	0.166	0.102

[a] GRASP is run during several minutes on a parallel framework with 7 GB od RAM.
[b] FL is run during less than 1 s on a 2.80 GHz hardware.
[c] TSSA is run in times between 1 and 14 s on a personal computer.
[d] BV is run in times between 16 and 285 s on a SUN Sparc-330.

Table 4
Computational results for the LA instances [42].

Instance	Size	DE–TS		GRASP[a]	HGA[b]	TSAB[c]	HA[d]		SBGA	FL[e]	SVS	SB[f]
		b-MRE	m-MRE	b-MRE	b-MRE	b-MRE	b-MRE	m-MRE	b-MRE	b-MRE	b-MRE	b-MRE
LA01–05	10 × 5	0	0.432	0	0	0	0	0.767	0.570	1.211	0	3.093
LA06–10	15 × 5	0	0.066	0	0	0	0	0	0	0	0	0.137
LA11–15	20 × 5	0	0.282	0	0	0	0	0.002	0	0	0	0
LA16–20	10 × 10	0	0.884	0	0.111	0	0.111	1.381	1.114	2.486	0.021	4.200
LA21–25	15 × 10	0.214	0.852	0.760	0.742	0.105	2.143	4.277	1.877	3.607	–	8.253
LA26–30	20 × 10	0.218	0.700	1.584	1.367	0.155	3.175	4.552	1.913	5.417	–	6.701
LA31–35	30 × 10	0	0.158	0	0	0	0	0	–	0	–	0
LA36–40	15 × 15	0.192	1.278	1.457	1.237	0.258	3.229	5.693	4.584	5.739	–	7.103
Average		0.078	0.582	0.475	0.432	0.065	1.082	2.084	1.437	2.308	0.005	3.686

[a] GRASP is run during several hours on a parallel framework with 7 GB od RAM.
[b] HGA is run in times between 1 min and 1 h with a 1.33 GHz processor.
[c] TSAB is run in times between 1 and 490 s on a AT 386DX personal computer.
[d] HA is run during 120 s.
[e] FL is run during less than 3 s on a 2.80 GHz hardware.
[f] SB is run in times between 2 and 60 s.

be explained by very short computational times (less than 1 s on a 2.80 GHz hardware).

On LA instances, the DE–TS performances are, again, very satisfactory and the hybrid algorithm is able to find the optimal solution in 32 of the 40 instances. Nowicki and Smutnicki's TSAB is the algorithm that provides best results. It is also worth noting that the m-MRE for DE–TS is quite comparable to the best results obtained by techniques such as GRASP or HGA (which are still very competitive).

We used a number of objective evaluations varying between 1000 and 10,000 for LA01–20 instances (about 7 s), and between 2.5×10^5 and 1×10^6 for the instances LA21–40 (between 7 and 50 s). For the same instances, GRASP's computational times are bracketted by 1 and several hours. With a 1.33 GHz processor, HGA runs in less than 1 min for simple instances and this time increases up to 1 h for the last examples. HA runs in 120 s (no mention of the hardware characteristics were reported by its authors), FL ran in less than 3 s (probably accounting for, here again, the low quality of the provided solutions) and SB ran in times varying between 2 and 60 s (we did not find any reference to the hardware adopted, which can reasonably be assumed to be much less efficient than nowadays' platforms). No computational time was reported for SBGA.

Table 5
Computational results for the TA instances [44].

Instance	Size	DE–TS		FL[a]	ACOFT[b]	TSSA[c]		*i*-TSAB[d]	TSSB[e]	BV[f]
		b-MRE	m-MRE	b-MRE	b-MRE	b-MRE	m-MRE	b-MRE	b-MRE	b-MRE
TA01–10	15 × 15	0.057	0.471	6.767	0.057	0.008	0.109	0.110	0.450	0.173
TA11–20	20 × 15	3.107	3.738	12.744	2.747	2.371	2.921	2.810	3.473	3.018
TA21–30	20 × 20	5.986	6.590	14.963	5.737	5.435	5.971	5.680	6.500	6.098
TA31–40	30 × 15	1.143	2.331	12.330	0.384	0.551	0.931	0.780	1.921	0.795
TA41–50	30 × 20	6.979	8.075	19.712	4.911	4.139	4.857	4.700	6.043	5.204
Average		3.571	4.354	13.303	2.767	2.501	2.958	2.816	3.677	3.058

[a] FL is run in times between 4 and 63 s on a 2.80 GHz hardware.
[b] ACOFT is run in times between 1400 and 10,000 s.
[c] TSSA is run in times between 10 and 900 s.
[d] *i*-TSAB is run in times between 60 and 1500 s with a 900 MHz processor.
[e] TSSB is run in times between 2200 and 35,000 s on a 133 MHz Pentium.
[f] BV is run in times between 1500 and 16,000 s on a SUN Sparc-330.

Table 6
Computational results for the YN instances [43].

Instance	Size	DE–TS		*i*-TSAB[a]	DS[b]	FL[c]	TSSA[d]	
		b-MRE	m-MRE	b-MRE	b-MRE	b-MRE	b-MRE	m-MRE
YN1	20 × 20	6.265	7.293	6.36	5.674	12.884	4.492	5.355
YN2	20 × 20	4.713	7.011		5.517	13.448	4.253	4.736
YN3	20 × 20	7.976	9.196		7.619	18.571	6.190	6.607
YN4	20 × 20	7.174	8.315		5.978	15.217	5.326	5.717
Average		6.532	7.954	6.36	6.197	15.030	5.065	5.604

[a] *i*-TSAB is run during about 60 s with a 900 MHz processor.
[b] DS is run in times between 17 and 19 h.
[c] FL is run during about 20 s on a 2.80 GHz hardware.
[d] TSSA is run in during about 60 s.

Finally, TSAB's computational times are between 1 and 490 s on a personal computer AT 386DX. Nevertheless, it is worth recalling that for many instances, the time needed by TSAB is equal to 0, therefore indicating that the initialization heuristic identified in such cases the optimal solution: the Tabu Search algorithm is not even used for these examples, and it can be inferred that, for the other instances, the initialization also performed a part of the job (which is difficult to evaluate, since no details on the relative computational times are provided in [17]).

Regarding the fifty Taillard's instances, DE–TS is one of the most efficient algorithms for the small size instances (TA01–30), performing even better than *i*-TSAB for TA01–10, with a quite satisfactory repeatability. For the most complex instances, the quality of DE–TS remains competitive with respect to state-of-the-art algorithms, although algorithms such as BV or even TSSB provide better solutions for TA41–50. The best algorithms on this benchmark are, clearly, TSSA, ACOFT and *i*-TSAB. Note that *i*-TSAB performances are reported here for a number of iterations similar to ours. On average, DE–TS is approximately in a mid-table ranking (however, the solutions that it found are quite close to those of the best techniques).

The chosen numbers of objective evalutions to run DE–TS were equal to 5×10^6 for TA01–10, 20×10^6 for TA11–20 and 50×10^6 for TA21–50. The associated computational times are, respectively, about 100, 500 and 10,000 s. FL was still executed with short runs (between 4 and 63 s); ACOFT's times were bracketted between 1400 and 10,000 s; TSSA's times ranged from 10 s for some of the easiest instances up to 900 s for the hardest ones. For the reported number of iterations, *i*-TSAB's computational times varied between 60 s and 1500 s on a 900 MHz processor. TSSB was run on a 133 MHz Pentium and its execution times range from about 2200–35,000 s (for TA21–30). Finally, BV's times (still on a SUN Sparc-330) were between 1500 and 16,000 s.

It is obviously difficult to compare the reported computational times, because of the hardware differences and for all the other reasons mentioned at the beginning of this section. However, it is worth recalling that the algorithms that obtain the best solutions use problem-specific initialization mechanisms. This must be taken into account for a fair comparison, since other algorithms such as DE–TS depart from a randomly generated set of solutions (which are, of course, of poor quality). In spite of this point, we underline that, on the first instances (TA01–20), DE–TS performs quite well (on TA01–10, DE–TS identifies 7 optimal solutions) and with very efficient computational times.

When dealing with the YN instances, DE–TS is clearly not among the best algorithms. *i*-TSAB and TSSA (except for YN2) are better in terms of solution quality, although the results of DE–TS still remain acceptable. The associated computational time of about 60 s (corresponding to a chosen number of objective evaluations equal to 1×10^6), is in the same order of magnitude as that of *i*-TSAB or TSSA. FL was ran in about 20 s. The stopping criterion of DS was to find a solution within 1% of the Best Known Upper Bound, leading to computational times varying between 17 and 19 h.

Table 7
Confidence intervals.

Instance	95% confidence interval	Interval width
ORB01	[0.042, 0.567]	0.524
ORB02	[0.124, 0.282]	0.158
ORB03	[0.731, 1.418]	0.687
ORB04	[0.567, 0.756]	0.189
ORB05	[0.310, 0.502]	0.192
ORB06	[0.693, 1.153]	0.460
ORB07	[0.088, 0.340]	0.252
ORB08	[1.340, 1.852]	0.512
ORB09	[0.139, 0.535]	0.396
ORB010	[0.254, 0.551]	0.297
YN1	[7.027, 7.559]	0.532
YN2	[6.690, 7.720]	0.580
YN3	[8.958, 9.423]	0.464
YN4	[8.065, 8.565]	0.500
LA01–05	[0.298, 0.582]	0.284
LA11–10	[0.094, 0.509]	0.415
LA16–15	[0.621, 1.148]	0.527
LA21–20	[0.689, 1.000]	0.311
LA26–25	[0.571, 0.840]	0.269
LA31–30	[0.049, 0.322]	0.273
LA36–35	[1.192, 1.614]	0.422
TA01–10	[0.403, 0.605]	0.202
TA11–20	[3.584, 3.893]	0.309
TA21–30	[6.466, 6.730]	0.264
TA31–40	[2.785, 3.288]	0.503
TA41–50	[7.766, 8.242]	0.476

The quality of the corresponding solution needs then to be seen within its appropriate perspective.

A statistical analysis was finally performed in order to determine an estimation of the confidence interval of the mean statistic. From the original sample obtained by the DE–TS algorithm for each tested instance, 10,000 bootstrap re-samples were generated and their respective mean value was computed. This new 10,000 values sample provided a good estimation of the confidence intervals through the bootstrap-percentile technique. In order to be consistent with the results presented in Tables 1–4, the intervals bounds were subsequently normalized with respect to the reference makespan (Best Known Solution or Lower Bound), the same as that used to compute the b-MRE and m-MRE indexes. From the results presented in Table 7, it can be noted that, in all cases, the confidence intervals width is lower than 0.7% of the reference makespan value, confirming the low dispersion of the solutions obtained.

5. Conclusions and future work

As was already mentioned before, the Job-Shop Scheduling Problem has attracted much interest from researchers during the last decades, mainly because of its many industrial applications. In addition, its inherent complexity has made it a challenging problem

for which many (academic) instances remain unsolved (to optimality), in spite of the great efforts devoted to solve them.

Among the proposed solution techniques, neighborhood based heuristics and, particularly, Tabu Search was found to be the most effective one. The aim of this paper was to determine if the association of this Local Search procedure with a global search method was able to attain the same performance levels as the *i*-TSAB algorithm, which includes very competitive re-start features. Differential Evolution, which is known to produce very good performances within the continuous optimization framework, was previously adapted to deal with permutation based problems and is now combined with a simple Tabu Search technique that does not include any re-initialization process.

Exhaustive experiments have been carried out on more than 100 instances drawn from benchmarks commonly used in the specialized literature. The computational results indicate a very good general efficacy of the DE–TS algorithm. For most of the mid size problems adopted, the proposed method was able to identify the optimum solution with a satisfactory repeatability. On more complex instances, a deterioration of the solution quality was observed, although DE–TS remained competitive with respect to other state-of-the-art algorithms. The global efficiency was quite close to that of *i*-TSAB but was not better. This conclusion must however be balanced by the fact that *i*-TSAB (and, actually, most of the best reported approximation algorithms) make use of advanced initialization strategies, which, in some cases, determine sub-optimal solutions (or even, the optimum) before the application of the core heuristic.

The reasons for this final result deterioration might be situated in the representation schemes employed within each method: the Random Keys model adopted for DE must be decoded and translated to a schedule so that Tabu Search can be used. Then, Tabu Search subsequently sends back the modified solutions to the evolutionary loop through a re-encoding (and somehow artificial) mechanism. We think that it is around this point that there is still room for improvement. A key aspect might be to modify the DE's internal procedure (adapting the differentiation operator) for permutation-based problems, such as has been proposed with Geometric Differential Evolution [52].

Additionally, different variables might be used: for instance, the operation starting times could be considered instead of permutations. This would spare the use of a schedule builder, but certainly at the expense of a huge computational space and the need of efficient constraint-handling techniques. Besides, due to the nature of the JSSP, many instances are likely to show one or various local optima. Considering that the selection process in DE is essentially greedy, another option might be the integration of mechanisms that would prevent the algorithm to conclude with premature convergence and help it escaping from local optima. Procedures such as a stochastic selection process should be an interesting option to be explored. Finally, the use of recently published variants of Differential Evolution, such as Self-adaptive DE [53] or Opposition-based DE [54] might be interesting tracks to be explored.

Acknowledgement

The second author acknowledges support from CONACyT project no. 103570.

References

[1] M.L. Pinedo, Planning and scheduling in manufacturing and services Springer Series in Operations Research, Springer, New-York, USA, 2005.
[2] M. Kovavic, B. Sarler, Genetic algorithm-based batch filling scheduling in the steel industry, Materials and Manufacturing Processes 26 (3) (2011) 464–474.
[3] R. Lewis, J. Thompson, On the application of graph colouring techniques in round-robin sports scheduling, Computers and Operations Research 38 (1) (2011) 190–204.
[4] L. Chunlin, L. Lauyan, A system-centric scheduling policy for optimizing objectives of application and resource in grid computing, Computers and Industrial Engineering 57 (3) (2009) 1052–1061.
[5] B. Roy, B. Sussmann, Les problèmes d'ordonnancement avec contraintes disjonctives. Note DS 9 bis, SEMA, Paris, 1964.
[6] J. Adams, E. Balas, D. Zawack, The shifting bottleneck procedure for job shop scheduling, Management Science 34 (1988) 391–401.
[7] P.J.M. Van Laarhoven, E.H.L. Aarts, J.K. Lenstra, Job shop scheduling by simulated annealing, Operations Research 5 (1) (1992) 113–125.
[8] M. Dell'Amico, M. Trubian, Applying Tabu-Search to the Job Shop Scheduling Problem, Politecnico di Milano, Italy, 1991.
[9] E.D. Taillard, Parallel Taboo Search techniques for the job shop scheduling problem, ORSA Journal on Computing 6 (2) (1994) 108–117.
[10] F. Della Croce, R. Tadei, G. Volta, A Genetic algorithm for the job shop problem, Computers and Operations Research 22 (1) (1995) 15–24.
[11] U. Dorndorf, E. Pesch, Evolution based learning in a job shop scheduling environment, Computers and Operations Research 22 (1) (1995) 25–40.
[12] R. Cheng, M. Gen, Y. Tsujimura, A tutorial survey of job-shop scheduling problems using genetic algorithms – Part II: Hybrid genetic search strategies, Computers and Industrial Engineering 36 (2) (1999) 343–364.
[13] R. Cheng, M. Gen, Y. Tsujimura, A tutorial survey of job-shop scheduling problems using genetic algorithms – Part I: Representation, Computers and Industrial Engineering 30 (4) (1996) 983–997.
[14] M. Gen, R. Cheng, Genetic Algorithms and Engineering Design, John Wiley and Sons, New York, USA, 1997.
[15] M. Gen, R. Cheng, Genetic Algorithms and Engineering Optimization, John Wiley and Sons, New York, USA, 2000.
[16] T.P. Baghi, Multiobjective Scheduling by Genetic Algorithms, Kluwer Academic Publishers, Netherlands, 1999.
[17] E. Nowicki, C. Smutnicki, A fast Taboo Search algorithm for the job shop problem, Management Science 42 (6) (1996) 797–813.
[18] E. Nowicki, C. Smutnicki, An advanced Tabu Search algorithm for the job shop problem, Journal of Scheduling 8 (2) (2005) 145–159.
[19] R.M. Aiex, S. Binato, M.G.C. Resende, Parallel GRASP with path-relinking for job shop scheduling, Parallel Computing 29 (4) (2003) 393–430.
[20] J.F. Goncalves, J.J. de Magalhaes Mendes, M.G.C. Resende, A hybrid genetic algorithm for the job shop scheduling problem, European Journal of Operational Research 167 (1) (2005) 77–95.
[21] D.Y. Sha, C.Y. Hsu, A hybrid particle swarm optimization for job shop scheduling problem, Computers and Industrial Engineering 51 (4) (2006) 791–808.
[22] J. Yang, L. Sun, H.P. Lee, Y. Qian, Y.C. Liang, Clonal selection based memetic algorithm for job shop scheduling problems, Journal of Bionic Engineering 5 (2008) 111–119.
[23] K.L. Huang, C.J. Liao, Ant colony optimization combined with Taboo Search for the job shop scheduling problem, Computers and Operations Research 35 (4) (2008) 1030–1046.
[24] C.Y. Zhang, P. Li, Y. Rao, Z. Guan, A very fast TS/SA algorithm for the job shop scheduling problem, Computers and Operations Research 35 (1) (2008) 282–294.
[25] J.P. Watson, A.E. Howe, L.D. Whitley, Deconstructing Nowicki and Smutnicki's i-TSAB Tabu Search algorithm for the job-shop scheduling problem, Computers and Operations Research 33 (9) (2006) 2623–2644.
[26] R. Storn, K.V. Price, Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces, Journal of Global Optimization 11 (1997) 341–359.
[27] E. Mezura-Montes, J. Velázquez-Reyes, C.A. Coello Coello, Modified differential evolution for constrained optimization, in: Proceedings of the 2006 IEEE Congress on Evolutionary Computation (CEC'06), Vancouver, Canada, 2006, pp. 25–32.
[28] A. Ponsich, Ma.G. Castillo Tapia, C.A. Coello Coello, Solving permutation problems with differential evolution: an application to the jobshop scheduling problem, in: Ninth International Conference on Intelligent System Design and Applications (ISDA'09), Pisa, Italy, November–December, 2009.
[29] B. Qian, L. Wang, D.X. Huang, X. Wang, Scheduling multi-objective job shops using a memetic algorithm based on differential evolution, International Journal of Advanced Manufacturing Technology 35 (2008) 1014–1027.
[30] T.E. Morton, D.W. Pentico, in: D.F. Kocaoglu (Ed.), Heuristic Scheduling Systems: With Applications to Production Systems and Project Management, Wiley Series in Engineering & Technology Management, New Jersey, USA, 1993.
[31] B. Giffler, G.L. Thompson, Algorithms for solving production scheduling problems, Operations Research 8 (4) (1960) 487–503.
[32] Hans-Paul Schwefel, Numerical Optimization of Computer Models, Wiley, Chichester, UK, 1981.
[33] N. Chakraborti, A. Kumar, The optimal scheduling of a reversing strip mill: studies using multipopulation genetic algorithms and differential evolution, Materials and Manufacturing Processes 18 (3) (2003) 433–445.
[34] J. Lampinen, I. Zelinka, On stagnation of the differential evolution algorithm, in: P. Osmera (Ed.), Proceedings of MENDEL 2000, 6th International Mendel Conference on Soft Computing, 2000, pp. 76–83.
[35] J. Bean, Genetic algorithms and random keys for sequencing and optimization, ORSA Journal on Computing 6 (1994) 154–160.

[36] R. Nakano, T. Yamada, Conventional genetic algorithm for job shop problems, in: Proceedings of the International Conference on Genetic Algorithms (ICGA'91), 1991, pp. 474–479.
[37] F. Glover, Tabu Search – Part I, ORSA Journal on Computing 1 (3) (1989) 190–206.
[38] G.E. Hinton, S.J. Nowlan, How learning can guide evolution, Complex Systems 1 (1987) 495–502.
[39] D. Whitley, V.S. Gordon, K. Mathias, Lamarckian evolution, the Baldwin effect and function optimization, in: Parallel Problem Solving from Nature - PPSN III, vol. 866 of Lecture Notes in Computer Science, Springer-Verlag, 1994, pp. 6–15.
[40] J.E. Beasley, OR-library: distributing test problems by electronic mail, Journal of the Operational Research Society 41 (11) (1990) 1069–1072, http://mscmga.ms.ic.ac.uk
[41] D. Applegate, W. Cook, A computational study for the job-shop scheduling problem, ORSA Journal on Computing 3 (2) (1991) 149–156.
[42] S. Lawrence, Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (supplement), Technical report, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh (Pennsylvania), USA, 1984.
[43] T. Yamada, R. Nakano, A Genetic algorithm applicable to large-scale job-shop instances, in: Manner, Manderick (Eds.), Parallel instance solving from nature 2, North-Holland, Amsterdam, 1992, pp. 281–290.
[44] E.D. Taillard, Benchmarks for basic scheduling problems, European Journal of Operational Research 64 (2) (1993) 278–285.
[45] H. Fisher, G.L. Thompson, Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules, Industrial Scheduling Edition, Prentice Hall, Englewood Cliffs, NJ, USA, 1963, pp. 225–251.
[46] E. Balas, A. Vazacopoulos, Guided local search with shifting bottleneck for job shop scheduling, Management Science 44 (2) (1998) 262–275.
[47] F. Pezzella, E. Merelli, A Tabu Search method guided by shifting bottleneck for job shop scheduling problem, European Journal of Operational Research 120 (2000) 297–310.
[48] U. Der, K. Steinhöfel, A parallel implementation of a job shop scheduling heuristic, in: PARA'00: Proceedings of the 5th International Workshop on Applied Parallel Computing, New Paradigms for HPC in Industry and Academia, Springer-Verlag, London, UK, 2001, pp. 215–222.
[49] S. Fernandes, H.R. Lourenço, A simple optimised search heuristic for the job-shop scheduling problem, Economics Working Papers 1050, Department of Economics and Business, Universitat Pompeu Fabra, 2007, January.
[50] P.S. Velmurugan, V. Selladurai, A Tabu Search algorithm for job shop scheduling problem with industrial scheduling case study, International Journal of Soft Computing 2 (4) (2007) 531–537.
[51] S.M. Kamrul Hasan, R.A. Sarker, D. Essam, D. Cornforth, Memetic algorithms for solving job-shop scheduling problems, Memetic Computing 1 (1) (2009) 69–83.
[52] A. Moraglio, J. Togelius, Geometric differential evolution, in: GECCO'09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation, ACM, New York (NY), USA, 2009, pp. 1705–1712.
[53] A.K. Qin, P.N. Suganthan, Self-adaptive differential evolution algorithm for numerical optimization, in: IEEE Congress on Evolutionary Computation (CEC'2005), vol. 2, IEEE Press, 2005, pp. 1785–1791.
[54] S. Rahnamayan, H.R. Tizhoosh, Opposition-based differential evolution, IEEE Transactions on Evolutionary Computation 12 (1) (2008) 64–79.