Soft Computing manuscript No.
(will be inserted by the editor)

An Efficient Hybrid Evolutionary Algorithm for Scheduling with Setup Times and Weighted Tardiness Minimization

Miguel Ángel González · Inés González-Rodríguez · Camino R. Vela · Ramiro Varela

Received: date / Accepted: date

Abstract We confront the job shop scheduling problem with sequence dependent setup times and weighted tardiness minimization. To solve this problem, we propose a hybrid metaheuristic that combines the intensification capability of tabu search with the diversification capability of a genetic algorithm which plays the role of long term memory for tabu search in the combined approach. We define and analyze a new neighborhood structure for this problem which is embedded in the tabu search algorithm. The efficiency of the proposed algorithm relies on some elements such as neighbors filtering and a proper balance between intensification and diversification of the search. We report results from an experimental study across conventional benchmarks, where we analyze our approach and demonstrate that it compares favorably to the state-of-the-art methods.

A preliminary version of this work was presented at the 14th Conference of the Spanish Association for Artificial Intelligence, CAEPIA2011, held in La Laguna, Spain, in 2011 (González et al (2011)).

Miguel Ángel González
Computing Technologies Group. Department of Computing
Artificial Intelligence Center, University of Oviedo, Spain,
Campus of Viesques, 33271 Gijón
E-mail: mig@uniovi.es

Camino R. Vela
Computing Technologies Group. Department of Computing
Artificial Intelligence Center, University of Oviedo, Spain,
Campus of Viesques, 33271 Gijón
E-mail: crvela@uniovi.es

Ramiro Varela
Computing Technologies Group. Department of Computing
Artificial Intelligence Center, University of Oviedo, Spain,
Campus of Viesques, 33271 Gijón
E-mail: ramiro@uniovi.es

Inés González-Rodríguez
Dept. of Mathematics, Statistics and Computing,
University of Cantabria, Spain,
E-mail: ines.gonzalez@unican.es

1 Introduction

The Job Shop Scheduling Problem (JSP) has been a research topic over the last decades due to the fact that it is a simple model of many real production processes. However, in many environments the production model has to consider additional characteristics or complex constraints. For example, in automobile, printing, semiconductor, chemical or pharmaceutical industries, setup operations such as cleaning up or changing tools are required between two consecutive jobs on the same machine. These setup operations depend both on the outgoing and incoming jobs, so they cannot be considered as being part of any of these jobs. This situation is well modelled with the JSP with Sequence-Dependent Setup Times (SDST-JSP). This problem received increasing attention by researchers, mostly with makespan minimization as objective function. However, this is not always the best measure for the quality of a schedule since in real-world problems there are usually due dates for the jobs and not all of them are equally important. In this case, an objective function such as Total Weighted Tardiness (TWT) is preferred, as this objective is typically associated with customer satisfaction and service level in make-to-order environments.

The JSP with makespan minimization has been intensely studied, so many formal results have been established and given rise to powerful exact or approximate solution methods. Among these, some pioneering approaches such as non-deterministic schedule builders (Giffler and Thompson (1960)), branch and bound algorithms (Carlier and Pinson (1989)), constraint propagation rules (Dorndorf et al (2000)) or local searchers (Van Laarhoven et al (1992), Dell' Amico and Trubian (1993)) deserve special mention. Incorporating sequence-dependent setup times or TWT estimations changes the nature of scheduling problems, so these well-known results and techniques for the JSP with makespan minimization are not directly applicable to the SDST-JSP. However, some of

these techniques have been extended to deal with the SDST-JSP and makespan minimization. For example, in Brucker and Thiele (1996) the authors develop an exact branch and bound algorithm, and in Vela et al (2010) and González et al (2008) the authors take local search structures proposed in Van Laarhoven et al (1992) as a basis for new neighborhood structures. These methods rely on computing a critical path in a feasible schedule and then defining a branching schema or a neighborhood structure which somehow change the processing order of operations in a critical path, and it is not trivial to extend them to TWT. Indeed, functions of the class of TWT require considering not only one but several critical paths in a schedule in order to properly define some branching or neighboring schema and this usually gives rise to extremely large branching factors or neighborhood structures which can make the search inefficient.

In this paper, we tackle the SDST-JSP with TWT minimization (SDST-JSP-TWT) and propose a hybrid approach that combines a Genetic Algorithm (GA) with Tabu Search (TS). Our intention in doing so is to combine the exploration capability of a stochastic and population-based algorithm such as GA with the intensification capability of a powerful local searcher such as TS. In our approach, TS is issued from every chromosome generated by the GA, so the GA plays the role of the long-term memory typically used by TS algorithms. From the point of view of the efficiency of this approach, the neighborhood structure used in TS plays an essential role. It should generate a small number of neighbors having some chance for improving. Bearing this in mind, we have designed a neighborhood structure, termed $\mathscr{N}$, which is the core of the algorithm. This structure is based on computing as many critical paths as the number of jobs of the problem instance. Then, a candidate neighbor is obtained by changing the order of two consecutive operations in one of these paths. In principle, the number of candidate neighbors is very large so we establish some conditions about feasibility and non-improvement as well as TWT estimation of candidates, so many neighbors can be discarded before being evaluated. As a result, we propose an algorithm termed GT$\mathscr{N}$ which is really efficient in solving the SDST-JSP-TWT. Moreover, when this algorithm is particularized for the JSP with TWT minimization, taking null setup times, it is quite competitive with the state-of-the art approaches for this problem.

The rest of the paper is organized as follows. In the next section, we review the literature about job shop scheduling and some solution methods. In Section 3 we formulate the SDST-JSP-TWT and introduce the notation used across the paper. Section 4 describes the main components of the genetic algorithm. In Section 5, we introduce the proposed neighborhood structure and the main components of the TS algorithm. Section 6 reports results from the experimental study. Finally, in Section 7 we summarize the main conclusions and propose some ideas for future work.

2 Literature Review

From the vast literature on JSP and metaheuristics, in this section we shall focus on contributions related to JSP with setup times or TWT minimization, as well as recent trends in hybrid metaheuristics applied to scheduling and related problems.

2.1 Job Shop Problem and Total Weighted Tardiness

The classical JSP with TWT minimization was first considered in Singer and Pinedo (1998), Singer and Pinedo (1999) and Kreipl (2000). The first paper proposes an exact branch and bound algorithm which fixes arcs in a disjunctive graph. The second proposes a shifting bottleneck algorithm and the third one proposes a heuristic, termed large step random walk (LSRW), which alternates between short steps, based on hill-climbing, and large steps that use Metropolis' algorithm. In DeBontridder (2005) a TS algorithm is proposed for a job shop with generalized precedence relations and time lag constraints. In this work, the problem is formulated as a maximal flow problem and in the experimental study the algorithm is evaluated on JSP instances, being competitive with the three algorithms above. Another competitive approach for this problem is the Genetic Local Search (GLS) given in Essafi et al (2008), combining a GA and an iterated local search (ILS) which is based on reversing critical arcs from a disjunctive graph model. The results reported in this paper show that the average solutions obtained by GLS are similar to the solutions obtained by the methods from Singer and Pinedo (1998), Singer and Pinedo (1999) and Kreipl (2000), while the best solutions obtained by GLS in a number of trials are in general better than those obtained by these other methods.

A local search method for optimizing any regular criterion in JSP can be found in Mati et al (2011). It relies on swapping all critical arcs, introducing a new efficient procedure to estimate neighbor values. The algorithm consists of three alternating phases, namely, improving, intermediate and mixed phases. The first ones perform steepest descent searches for and around local optima while the latter introduces diversification to escape from local optima. The resulting algorithm compares favorably to the approaches given in Singer and Pinedo (1998), Kreipl (2000) and Van Hentenryck and Michel (2004), where a local search using constraint programming is proposed. Recently, a hybrid shifting bottleneck-tabu search heuristic (SB-TS) is proposed in Bülbül (2011) which replaces the re-optimization step in the shifting bottleneck algorithm by TS. The authors report

results across conventional instances showing that SB-TS obtains similar results to other methods. However, in the benchmark proposed in Essafi et al (2008), which includes large instances, they do not report experiments on the largest ones, and their results are worse than those obtained by Essafi el al's GLS algorithm in the two subsets of instances with tight due dates while they are better for the third subset.

Sequence dependent setup times have been considered for the JSP, but in most of the cases with makespan minimization. The first approach is the branch and bound algorithm proposed in Brucker and Thiele (1996) which generalizes the algorithm proposed in Brucker et al (1994) to handle setups. A different branch and bound algorithm, which improves the results of the previous one, is proposed in Artigues et al (2004). In this case, a series of decision problems are considered and a lower bound estimation which requires computing exact solutions to traveling salesman problem instances is used to guide the search. In Balas et al (2008), the authors apply the shifting bottleneck heuristic and obtain competitive results. In Vela et al (2010), a memetic algorithm is proposed which outperforms all the previous methods. This memetic algorithm is improved in González et al (2008), where the authors propose some new neighborhood structures that make the local search more efficient.

Other objective functions for SDST-JSP such as maximum lateness (Balas et al (2008), González et al (2012)). In these approaches, some techniques designed for makespan minimization are extended to deal with these objective functions.

2.2 Hybrid metaheuristics

Hybrid metaheuristics are classical methods for solving combinatorial optimization problems, since they allow algorithm designers to combine the characteristics of different search techniques. In particular, they have a long track of success with scheduling problems. For example, the algorithm proposed in Beck et al (2011) is probably the most efficient approach to the JSP with makespan minimization. This algorithm combines the solution-guided search method proposed previously with the $i-$TSAB algorithm proposed in Nowicki and Smutnicki (2005). In addition to some of the algorithms for JSP and its variants reviewed in the previous section, there are other recent approaches to scheduling problems. For example, in Behnamian et al (2011), the authors propose a hybrid metaheuristic to solve the parallel machine scheduling problem with setups, combining an ant colony optimization (ACO) algorithm with simulated annealing (SA) and variable neighborhood search (VNS). Also, in Tavakkoli-Moghaddam et al (2009) the authors tackle the JSP with machine availability and non-anticipatory setup times combining SA with an electromagnetic-like mechanism.

Another recent hybrid approach is given in Jat and Yang (2011), which combines a hybrid GA with TS for solving the Post Enrolment Course Timetabling Problem (PECTP). This hybrid algorithm consist of two phases. The first one combines a guided search GA with some local search.

In the second phase, a TS algorithm is used to improve the quality of the best solution obtained in the first phase. Notice that this approach is different from that presented in this paper, in the sense that we apply TS to every chromosome generated by the GA. TS is also combined with GRASP algorithms in Angel Bello et al (2011a,b), where the authors propose a GRASP algorithm for a single machine scheduling problem with sequence-dependent setup costs and availability constraints which incorporates a TS as an improvement method.

3 Description of the problem

In the job shop scheduling problem, a set of N jobs, $J = \{J_1, \ldots, J_N\}$, are to be processed on a set of M machines or resources, $R = \{R_1, \ldots, R_M\}$ while minimizing some function of completion times of the jobs, subject to the following constraints: (i) the sequence of machines for each job is prescribed, and (ii) each machine can process at most one job at a time. The processing of a job on a machine is called an operation, and its duration is a given constant. A time may be needed to adjust a machine between two consecutive operations, which is called a setup time, and which may or may not be sequence-dependent. Jobs may also have a due date, that is, a time before which jobs should be completed, and a weight, which represents the relevance of the job. The objective here is to obtain a schedule, i.e. a starting time for each one of the operations, such that the weighted cost of the jobs exceeding its due-dates, also known as the weighted tardiness, is minimized.

We denote by:

- Ω the set of operations
- $\alpha(i)$ and $\omega(i)$ the first and the last operation respectively of job J_i
- d_i the due-date of job J_i
- w_i the weight of job J_i
- p_u the processing time of operation u
- s_{uv} the setup time between two consecutive operations u, v requiring the same machine
- t_u the starting time of operation u that needs to be determined

The SDST-JSP has two binary constraints: precedence and capacity. Precedence constraints, defined by the sequential routings of the tasks within a job, translate into linear inequalities of the type: $t_u + p_u \leq t_v$, where v is the next operation to u in the job sequence. Capacity constraints that restrict the use of each resource to only one task at a time

translate into disjunctive constraints of the form: $t_u + p_u + s_{uv} \leq t_v \vee t_v + p_v + s_{vu} \leq t_u$, where u and v are operations requiring the same machine.

The objective is to obtain a feasible schedule such that the weighted tardiness, defined as follows

$$\sum_{i=1,\ldots,N} w_i T_i \tag{1}$$

where T_i is the tardiness of the job i, given by

$$T_i = \max\{C_i - d_i, 0\} \tag{2}$$

where C_i is the completion time of job i.

This problem is denoted by $J|s_{ij}|\sum w_i T_i$ according to the $\alpha|\beta|\gamma$ notation proposed in Graham et al (1979).

3.1 The disjunctive graph model representation

The disjunctive graph is a common representation in scheduling, its exact definition depending on the particular problem. For the $J|s_{ij}|\sum w_i T_i$ problem, we propose that it be represented by a directed graph $G = (V, A \cup E \cup I_1 \cup I_2)$. Each node in set V represents a task of the problem, with the exception of the dummy nodes *start* and end_i $1 \leq i \leq N$, which represent fictitious operations that do not require any machine. Arcs in A are called *conjunctive arcs* and represent precedence constraints while arcs in E are called *disjunctive arcs* and represent capacity constraints. Set E is partitioned into subsets E_i, with $E = \cup_{j=1,\ldots,M} E_j$, where E_j corresponds to resource R_j and includes two directed arcs (v,w) and (w,v) for each pair v, w of operations requiring that resource. Each arc (v,w) in A is weighted with the processing time of the operation at the source node, p_v, and each arc (v,w) of E is weighted with $p_v + s_{vw}$. Set I_1 includes arcs of the form $(start, v)$ for each operation v of the problem, weighted with s_{0v}. Set I_2 includes arcs $(\omega(i), end_i)$, $1 \leq i \leq N$, weighted with $p_{\omega(i)}$.

A feasible schedule S is represented by an acyclic subgraph of G: $G_S = (V, A \cup H \cup J_1 \cup I_2)$, where $H = \cup_{j=1\ldots M} H_j$, H_j being a minimal subset of arcs of E_j defining a processing order for all operations requiring R_j and where J_1 consists of arcs $(start, v_j)$, $j = 1 \ldots M$, v_j being the first operation of H_j. Finding a solution can thus be reduced to discovering compatible orderings H_j, or partial schedules, that translate into a solution graph G_S without cycles. Figure 1 shows a solution to a problem with 3 jobs and 3 machines; dotted arcs belong to H and J_1, while continuous arcs belong to A.

The total weighted tardiness of a schedule S is determined by a set of critical paths in G_S. A critical path is defined as a largest cost path from node *start* to a node end_i $1 \leq i \leq N$. The length of this path is the completion time of the operation end_i and so it determines the contribution of job J_i to the solution cost. Nodes and arcs in a critical path are also termed critical. A critical path may be represented as a sequence of the form $start, B_1, \ldots, B_r, end_i$, $1 \leq i \leq N$, where each B_k, $1 \leq k \leq r$, is a critical block, a maximal subsequence of consecutive operations in the critical path requiring the same machine. The concepts of critical path and critical block are of major importance for job scheduling problems due to the fact that most formal properties and solution methods rely on them. Some of these properties have given rise to a number of neighborhood structures for the classic JSP, which are based on exchanging the order of operations requiring the same machine (Matsuo et al (1988); Van Laarhoven et al (1992); Taillard (1994); Nowicki and Smutnicki (1996); Dell' Amico and Trubian (1993); Balas and Vazacopoulos (1998)). We propose in this paper some new neighborhood structures which also rely on reversing the processing order of operations in a critical block, with the objective of reducing the contribution of a job to the weighted tardiness of the schedule.

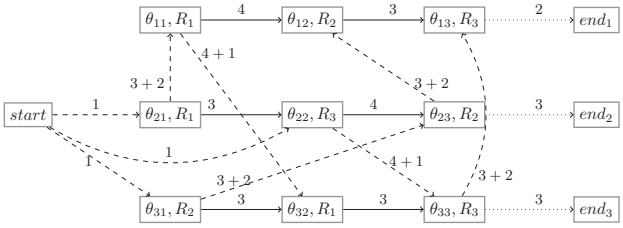


Fig. 1: A feasible schedule to a problem with 3 jobs and 3 machines

In order to simplify expressions, we define the following notation for a feasible schedule. Given a solution graph G_S for the SDST-JSP, the head of an operation v, denoted r_v, is the cost of the longest path from node *start* to node v, i.e., the starting time of v in the schedule represented by G_S. A tail q_v^i, $1 \leq i \leq N$ is the cost of the longest path from node v to node end_i, minus the duration of task in node v. We will take $q_v^i = -\infty$ when no path exist from v to end_i. Notice that here we have defined N tails for each operation, while for makespan minimization only one is just one. Let PJ_v and SJ_v denote respectively the predecessor and successor of v in the job sequence, and PM_v and SM_v the predecessor and successor of v in its machine sequence. We take node *start* to be PJ_v for the first task of every job and PM_v for the first task to be executed in each machine; also end_i is taken as $SJ_{\omega(i)}$. Note that $p_{start} = 0$ and $p_{end_i} = 0$. Thus, the head of every operation v and every dummy node may be computed as follows:

$$r_{start} = 0$$
$$r_v = \max(r_{PJ_v} + p_{PJ_v}, r_{PM_v} + p_{PM_v} + s_{PM_v v})$$
$$r_{end_i} = r_{\omega(i)} + p_{\omega(i)}, 1 \leq i \leq N$$

Similarly, the tail of every operation v and every dummy node are computed as follows:

$$q^i_{end_i} = 0$$
$$q^j_{end_i} = -\infty, j \neq i$$
$$q^j_v = \begin{cases} \max(q^j_{SJ_v} + p_{SJ_v}, q^j_{SM_v} + p_{SM_v} + s_{vSM_v}) & \text{if } SM_v \text{ exists} \\ q^j_{SJ_v} + p_{SJ_v} & \text{otherwise} \end{cases}$$
$$q^j_{start} = \max_{v \in SM_{start}} \{q^j_v + p_v + s_{0v}\}$$

Clearly, heads are computed from node *start* onwards, and tails from nodes end_i backwards. A node v is critical if and only if $r_v + p_v + q^j_v = C_j$ for some job j.

4 Genetic Algorithm for the SDST-JSP

We use here the same GA that was previously used in Vela et al (2010) and González et al (2012) for other variants of the SDST-JSP. In these works, this GA was hybridized with local searches designed specifically to cope with sequence dependent setup times and makespan and maximum lateness minimization respectively, being these algorithms the current state-of-the-art for these problems. Therefore, we have opted to combine here this GA with the TS method designed for the JSP with weighted tardiness minimization. The main characteristics of this GA are the following. In the first step, the initial population is generated and evaluated. Then the genetic algorithm iterates over a number of steps or generations. In each iteration, a new generation is built from the previous one by applying the genetic operators of selection, recombination and replacement. These operators can be implemented in a variety of ways and, in principle, are independent from each other. However, in practice all of them should be chosen considering their effect on the remaining ones in order to get a proper selective pressure and so a successful overall algorithm. The approach taken is the following. In the selection phase all chromosomes are grouped into pairs, and then each one of these pairs is mated to obtain two offspring. Finally, the replacement is carried out as a tournament selection from each pair of parents and their two offspring. This algorithm differs slightly from the classic genetic algorithms in that the selective pressure is introduced in the replacement instead of in the selection phase, and so its evolution strategy is similar to that of the crowding algorithm proposed in Mahfoud (1992). The main difference with this algorithm is that we do not use crowding distances in the replacement phase. In this way, we are less vigilant about the diversity of the population, but at the same time the GA is much less time consuming, especially in the largest instances.

The coding schema is based on permutations with repetition, as proposed in Bierwirth (1995). In this schema a chromosome is a permutation of the set of operations, each one being represented by its job number. In this way a job number appears within a chromosome as many times as the number of its operations. For example, chromosome (2 1 1 3 2 3 1 2 3) actually represents the permutation of operations $(\theta_{21}\,\theta_{11}\,\theta_{12}\,\theta_{31}\,\theta_{22}\,\theta_{32}\;\;\theta_{13}\,\theta_{23}\,\theta_{33})$ and is a valid chromosome for any problem with 3 jobs and 3 machines. This permutation should be understood as expressing partial schedules for each set of operations requiring the same machine. This codification presents a number of interesting characteristics; for example, it is easy to evaluate with different algorithms and allows efficient genetic operators. In Varela et al (2005) this encoding is empirically compared with other permutation based coding schemas and demonstrated to be the best one for the JSP across a set of selected problem instances of common use.

For chromosome mating we have considered the *Job Order Crossover* (JOX) (Bierwirth (1995)). Given two parents, JOX selects a random subset of jobs and copies their genes to the offspring in the same positions as they are in the first parent, then the remaining genes are taken from the second parent so as to maintain their relative ordering. A second offspring is generated inverting the role of the parents.

The operator JOX might swap any two operations requiring the same machine; this is an implicit mutation effect. For this reason, we have not used any explicit mutation operator. Hence, parameter setting in the experimental study is considerably simplified, as crossover probability is set to 1 and mutation probability need not be specified. Of course, for identical parent sequences, the offspring will be identical and consequently the evolution would come to a complete halt if all chromosomes were identical. However, in practice this is not an issue as the algorithm always stops before convergence to such situation. With this setting, we have obtained results quite similar to those obtained with a lower crossover probability and a low probability of applying conventional order based mutation operators.

To build schedules we have used a decoding algorithm which generates active schedules. A schedule is active if no operation can be started earlier without delaying any other operation. In the implementation we used the Serial Schedule Generation Schema (SSGS) proposed in Artigues et al (2005) for the SDST-JSP. SSGS iterates over the operations in the chromosome sequence and assigns each the earliest starting time that satisfies all constraints with respect to the previous scheduled operations. SSGS produces active schedules, provided that the triangular inequality for the setup times holds for all operations requiring the same machine (Artigues et al (2005)): $s_{uw} \leq s_{uv} + s_{vw}$ holds for any u, v and w requiring the same machine. This property is accepted by most researchers in the literature and in fact it does hold for the instances used in our experimental study.

When combined with the the GA, TS is applied to every schedule produced by SSGS. Then, the chromosome is rebuilt from the improved schedule obtained by TS, so its characteristics can be transferred to subsequent offsprings. This effect of the evaluation function is known as Lamarckian evolution.

5 Tabu Search for the Weighted Tardiness minimization in the SDST-JSP

Algorithm 1 shows the tabu search algorithm considered herein. This algorithm is borrowed from González et al (2009), and it is similar to other tabu search algorithms described in the literature (Glover and Laguna (1997); Taillard (1994); Nowicki and Smutnicki (2005)). The first step evaluates the initial solution (i.e. a chromosome generated by the GA, after applying an active schedule builder). Then, it iterates over a number of steps. In each iteration, the neighborhood of the current solution is built and one of the neighbors is selected for the next iteration. The tabu search stops after a number of *maxImproveIter* iterations without improving the current best solution. In order to avoid reevaluating solutions, in addition to tabu tenure, the algorithm uses a cycle checking mechanism.

Algorithm 1 The Tabu Search Algorithm

Require: An initial solution s_0 for a problem instance P
Ensure: A (hopefully improved) solution s_B for instance P
  Set the current solution $s = s_0$ and the best solution $s_B = s$;
  Set *improveIter* = 0, Empty the tabu list;
  while *improveIter* $<$ *maxImproveIter* **do**
    Set *improveIter* = *improveIter*+1;
    Generate neighbors of the current solution s using the neighborhood structure;
    Let $s*$ be the best neighbor either not tabu and not leading to a cycle or satisfying the aspiration criterion. Update the tabu list and the cycle detection structure accordingly and let $s = s*$;
    if $s*$ is better than s_B **then**
      Set $s_B = s*$;
      Set *improveIter* = 0;
    end if
  end while
  return The solution s_B;

5.1 The neighborhood structure

A key component for the success of a TS algorithm is the neighborhood structure used. In this paper, we define and study a new neighborhood structure which is specifically designed for the problem $J|s_{ij}|\sum w_iT_i$. In order to do this, we use a a basis some results and methods given in Vela et al (2010) where a structure for the problem with makespan minimization ($J|s_{ij}|C_{max}$) is proposed. This structure is termed N_1^S and it is based on previous structures given in Matsuo et al (1988) and Van Laarhoven et al (1992) for the standard JSP, which have given rise to some of the most outstanding algorithms for the JSP, such as those proposed in Dell' Amico and Trubian (1993); Nowicki and Smutnicki (2005); Balas and Vazacopoulos (1998); Zhang et al (2008). One of these structures, N_1, was defined in Nowicki and Smutnicki (1996) and considers only moves in the borders of critical blocks, since these are the only arcs that can produce immediate improvement in a single move. N_1^S is an extension of N_1 which considers single moves inside critical blocks as these moves may produce improvements in the problem $J|s_{ij}|C_{max}$ (Vela et al (2010)).

To define neighborhood structures for $J|s_{ij}|\sum w_iT_i$, there is the added difficulty that the cost of a solution can be given by up to N critical paths. For each node end_i, a critical path from $start$ to end_i is considered whenever its cost is greater than the due date of job J_i, d_i. For each of these paths we analyze the possibility of reversing the processing order of some operations to reduce the completion time of the last operation of job J_i. Notice that this move may delay the completion time of another job. In this work, we only consider single moves, i.e. reversing the processing order of two consecutive operations. Even so, the number of neighbors may be exponentially large for many chromosomes, so we use a filtering mechanism based on the results below, which allow the algorithm to discard unfeasible and a number of non-improving neighbors. By doing this we get a neighborhood of reasonable size while augmenting the chance of obtaining improving neighbors.

The first result establishes a sufficient condition for non-improvement when a single arc is reversed in a solution.

Proposition 1 *Let S be a schedule and (v,w) a disjunctive arc which is not in a critical block. Then, if the setup times of the problem instance fulfill the triangular inequality, reversing the arc (v,w) does not produce any improvement even if the resulting schedule S' is feasible.*

Proof If the reversed disjunctive arc does not begin nor end in a critical block, then all critical paths in G_S are paths in $G_{S'}$ as well. Hence the completion time of every job in S' is greater or at least equal than in S.

On the other hand, w can be the first task of a critical block of a critical path which ends in node end_i. Let y be the successor of w in that critical block. Then, in $G_{S'}$ there is a path from node $start$ to node end_i which is identical to the critical path in G_S with the exception that task v is between w and y. In G_S the cost from node w to y is $p_w + s_{wy}$, and in $G_{S'}$ the cost from node w to y is $p_w + s_{wv} + p_v + s_{vy}$. Therefore, provided the setup times fulfill the triangular inequality, that new path in $G_{S'}$ cannot be shorter than the path in G_S.

An analogous reasoning can be applied if v is the last task of a critical block.

Since we are assuming that the triangular inequality for setup times holds, we should only reverse critical arcs in order to obtain improving schedules.

Regarding feasibility, we consider a result from Vela et al (2010) which is also applicable here since it does not depend on the objective function: if there is not an alternative path between operations v and w before reversing the arc (v,w), then the new schedule after reversing (v,w) is feasible. Checking for such alternative path is time consuming; instead, we have opted to use a sufficient (but not necessary) condition as given in the following result.

Proposition 2 *Let S be a schedule and (v,w) an arc in a critical block. A sufficient condition for an alternative path between v and w not to exist is that*

$$r_{PJ_w} < r_{SJ_v} + p_{SJ_v} + min\{s_{kl} | (k,l) \in E, J_k = J_v\} \quad (3)$$

where J_k and J_v denote the jobs of operation k and v respectively.

In consequence, the feasibility of new neighbors may be efficiently verified at the cost of discarding some feasible schedules.

Given these results, we may define the following generic neighborhood structure. Let S be a schedule and let G_S be the associated solution graph. For each tardy job i ($T_i > 0$) let us consider a critical path to node end_i, let Π be a subset of such critical paths and let Γ be the set of critical blocks in Π. Given the results above, we may define the following generic neighborhood structure.

Definition 1 ($\mathscr{N}$) The neighborhood of S, $\mathscr{N}(S)$, consists of all schedules derived from S by reversing one arc (v,w) of a critical block in Γ, provided that the feasibility condition given in Proposition 2 holds.

The generic structure $\mathscr{N}$ defines a family of neighborhood structures, since the set Π can be chosen in different ways. For example, we can choose all critical paths of tardy jobs or just the path with the largest contribution to the weighted tardiness. In the experimental study, we will analyze some possibilities. Here, it is worth to remark that if (v,w) is inside two or more critical paths in Π, the condition from Proposition 2 needs to be evaluated only once. This is also the case for Proposition 3 below.

5.2 Additional non-improving conditions

In Vela et al (2010) the authors define non-improving conditions for certain reversals of critical arcs. These results can be extended for weighted tardiness, taking into account that in this case we need consider several critical paths instead of just one. In particular, we can establish the following results.

Proposition 3 *Let S be a schedule, let B a critical block of this schedule within a critical path to node end_i and let (v,w) be an arc inside B, i.e., PM_v and SM_w belong to B. Then, assuming that the schedule S' obtained from S by reversing arc (v,w) is feasible, the completion time of job J_i in S' is greater than or at least equal to its completion time in S if the following condition holds (where $x = PM_v$ and $y = SM_w$ in the schedule S)*

$$s_{xw} + s_{wv} + s_{vy} \geq s_{xv} + s_{vw} + s_{wy}. \quad (4)$$

Proof In G_S there is a critical path ending in node end_i of the form (b,x,v,w,y,b') where b and b' are sequences of operations. Then in $G_{S'}$ there is a path of the form (b,x,w,v,y,b') from node *start* to node end_i. If previous condition holds, clearly this path is larger or at least equal than the previous critical path in G_S, therefore the completion time of job i is S' is larger or equal than it is in S.

Analogous results can be established if (v,w) is the first arc of the first critical block and if it is the last arc of the last critical block. In the first case, x is taken as *start* while in the second y is taken as end_i.

These two results may be used to avoid the reversal of some critical arcs (v,w). Notice however that, since arc (v,w) can be in two critical paths to nodes end_i and end_j, the conditions above must be evaluated in both contexts in order to definitively discard the move. Therefore, these conditions will be more time consuming than the analogous conditions used in Vela et al (2010) for makespan minimization. On the other hand, they could reduce the effective number of neighbors. In the experimental study, we report some results which clarify the utility of these conditions in the neighborhood structure.

5.3 Weighted Tardiness estimation

Although computing the weighted tardiness of a neighbor only requires to recompute heads (tails) of operations which are after (before) the first (last) operation moved, for the sake of efficiency the selection rule is based on estimations rather than on the actual value of the total weighted tardiness of neighbors. Based on the procedure given for the JSP in Taillard (1994), we propose a new procedure for total weighted tardiness estimation, termed *lpathSWT*, which is shown in Algorithm 2.

Remember that each task t has N tails denoted by $q_t^1 \ldots q_t^N$. For each $i = 1 \ldots N$, *lpathSWT* estimates the cost of the longest path from node *start* to node end_i through the nodes v and w after the move.

The procedure $lpathSWT$ is efficient as the cost of the new path from node *start* to node end_i through nodes v or w can be estimated with complexity of $O(1)$. Moreover, it returns a lower bound of the cost of a neighboring solution:

Algorithm 2 Procedure *lpathSWT*

Require: A critical arc (v,w) inside a solution graph G_S
Ensure: An estimation of the total weighted tardiness of S' obtained from S by reversing (v,w) in G_S
$TotalEst = 0$;
$r'_w = \max(r_{PJ_w} + p_{PJ_w}, r_{PM_w} + p_{PM_w} + s_{PM_w w})$;
$r'_v = \max(r_{PJ_v} + p_{PJ_v}, r'_w + p_w + s_{wv})$;
for $i = 1$ to N **do**
  $q'^i_v = \max(q^i_{SJ_v} + p_{SJ_v}, q^i_{SM_v} + p_{SM_v} + s_{vSM_v})$;
  $q'^i_w = \max(q^i_{SJ_w} + p_{SJ_w}, q'^i_v + p_v + s_{wv})$;
  $PartialEst = \max(r'_w + p_w + q'^i_w, r'_v + p_v + q'^i_v)$;
  $TotalEst = TotalEst + (\max((PartialEst - d_i), 0) * w_i$;
end for
return $TotalEst$;

Proposition 4 *The procedure lpathSWT returns a lower bound of the schedule S' obtained from S by reversing (v,w).*

Proof If follows easily from the fact that r'_w, r'_v, q'^i_w, q'^i_v, $1 \leq i \leq N$, represent the exact values of the heads and tails of operations w and v in S' obtained from S by reversing the single arc (v,w). So, $r'_w + p_w + q'^i_w$ and $r'_v + p_v + q'^i_v$ are the costs of the largest cost paths from node *start* to node end_i through nodes w and v respectively an then they are lower bounds of C_i.

As a drawback, *lpathSWT* is not very accurate. We have empirically assessed this fact on several instances. We have generated three million neighbors for each solution, obtaining exact estimates in 51.37% of the cases. Other works such as Mati et al (2011) or Essafi et al (2008) use a more accurate and sophisticated estimation procedure, where the complexity to estimate a path from node *start* to node end_i is $O(N)$. That method returns between 57% and 76% of exact estimations, depending on the particular instance.

As a tradeoff between accuracy and efficiency, we have opted to use *lpathSWT* and then calculate the exact total weighted tardiness when the neighbor's estimate is less than the total weighted tardiness value of the original schedule. Some preliminary results have shown that the improvement thus achieved compensates for the time taken, as we shall see in the experimental study. We have therefore opted to use our estimation procedure only as a filter, in order to discard many non-improving neighbors in a relatively short runtime.

6 Experimental Study

It is the purpose of this experimental study to analyze GT$\mathcal{N}$ and to establish a comparison of this algorithm with other methods. The bechmarks and performance metrics used are described in Sections 6.1) and (6.2) respectively. In Section 6.3), we start considering a number of configurations in order to evaluate the contribution of each one of the main components to the performance of the whole algorithm. Then, in the following two sections we compare our algorithm with other approaches. As, to our knowledge, there are not results on the SDST-JSP-TWT reported in the literature, in Section 6.4.1 we compare GT$\mathcal{N}$ with a conventional solver. So, in order to compare GT$\mathcal{N}$ with some state-of-the-art method, we experimented across job-shop instances without setup times and compare with a number of the best approaches for this problem. The results of these experiments are reported in Section 6.4.2). Both the solver and GT$\mathcal{N}$ were run on Windows XP on Intel Core 2 Duo at 2.66GHz with 2Gb RAM. GT$\mathcal{N}$ is implemented in C++.

6.1 Test Problems

We consider two sets for each one of the problems, JSP and SDST-JSP. The first JSP benchmark is that proposed in Singer and Pinedo (1998), and it is composed of 22 instances of size 10×10: ABZ5, ABZ6, LA16 to LA24, MT10, ORB1 to ORB10. Instances LA21 to LA24 that are originally of size 15×10 were converted to 10×10 removing the last 5 jobs. This is the most used benchmark in the literature about TWT minimization in the JSP. The weights of the jobs are defined so as the first 20% of the jobs have weight 4 (very important jobs), the next 60% have weight 2 (moderately important jobs) and the remaining jobs have weight 1 (not important jobs). This means that for an instance with 10 jobs $w_1 = w_2 = 4$, $w_3 = \cdots = w_8 = 2$ y $w_9 = w_{10} = 1$. This way of assigning weights is not completely arbitrary, but it is based on observations in real production environments. The due date of job i is taken as:

$$d_i = f * \sum_{j=1}^{M} p_{ij}, \tag{5}$$

where M is number of machines that coincides with the number of tasks per job. f is a parameter that controls the tightness of the due dates, being 1.3, 1.5 and 1.6 the values usually taken in this and other benchmarks. Therefore, there are 66 instances in all.

The second JSP benchmark was proposed in Essafi et al (2008). It is based on the instances proposed in Lawrence (1984) for makespan minimization. The size of these instances varies from 10×5 (the smallest ones) to 30×10 and 15×15 (the largest ones). Essafi et al. adapt these instances assigning due dates and weights with the same procedure as the Singer and Pinedo benchmark. The instances LA16 to LA20 are the same as those in Singer and Pinedo benchmark, but the instances LA21 to LA24 are not the same, as in this case they are not reduced to size 10×10.

For the SDST-JSP we consider the BT set with 15 instances proposed in Brucker and Thiele (1996). These instances are commonly used as benchmark for makespan minimization. We adapt here these instances for TWT defining

due-dates and job weights in the same way as for previous JSP instances. BT instances are divided in three groups depending on its size: small instances, t2-ps01 to t2-ps05, are 10×5, medium instances, t2-ps06 to t2-ps10, are 15×5, and large instances, t2-ps11 to t2-ps15, are 20×5. These instances verify the triangular inequality for setup times.

In order to experiment with bigger and harder instances than those of the BT set, we also consider the benchmark proposed in Vela et al (2010). This benchmark is interesting because the instances are derived from the set of 10 instances selected in Applegate and Cook (1991) as hard to solve in the classical JSP with makespan minimization. Their sizes are 15×10 for the smallest ones (LA21, LA24 y LA25), 20×10 for the LA27 and LA29, 15×15 for LA38 and LA40, and 20×15 for the largest ones (ABZ7, ABZ8 and ABZ9). These instances are extended to the SDST-JSP using the same criteria as BT instances to generate setups, so they verify the triangular inequality as well. We define the due dates and weights in the same way as before. Each instance is identified by the instance name followed by *sdst*.

6.2 Performance metrics

One of the aims of the experiments is to assess how close the algorithm comes to producing an optimal schedule, measured as the gap w.r.t. the best-know solution (BKS). As source of BKS, we shall use the best values reported so far in the literature together with the values obtained in our experiments, as these often yields better solutions than those from the literature. For all methods, we have used the same BKS to calculate gaps, so we report some values that differ from those given in Bülbül (2011), where the authors use the solutions reported in Singer and Pinedo (1998) to calculate the gaps.

In the same way as is Bülbül (2011), we report total gap and average gap values. The first ones, labeled "Total GAP" in the tables, are computed as the error in percent of the the sum of the obtained TWT for all instances w.r.t. the corresponding sum of the best known TWT. The average gap, "Avg. GAP", is computed as the error in percent averaged for all instances. As it is done in Bülbül (2011), an instance with zero optimal value has to be excluded from the computation of the average gap.

In order to obtain statistically significant results the algorithms were run 30 times for each instance. An exception was done when comparing our method against other state-of-the-art methods in the classical JSP without setup times, where we run the algorithm 10 times as it was done with some of these methods. In any case, we report the best TWT and the average TWT obtained in all runs. To assess the significance of the results, we provide confidence intervals and the p-values from Wilconson tests. Also, the rows labeled as "T(s)" in the tables report the time taken by our algorithm in one single run for the corresponding instance.

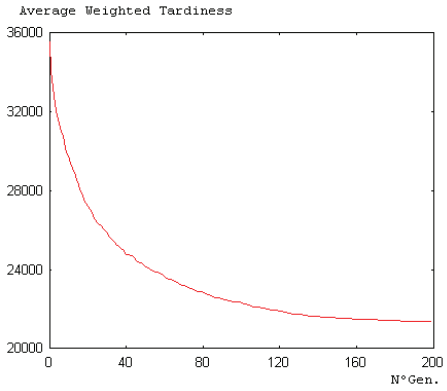


Fig. 2: Evolution of the best solution of GT$\mathscr{N}$ for the instance ABZ7sdst ($f = 1.5$) averaged for 30 runs.

6.3 Analysis of GT$\mathscr{N}$

To analyze GT$\mathscr{N}$ we have established a base configuration. We started fixing the population size at 100 chromosomes, TS is issued from each new chromosome with *maxImproveIter=50* and the estimation of TWT is used as filtering. Then, we have done some preliminary experiments to obtain a number of generations that allows GT$\mathscr{N}$ to converge for all instances of the set. Figure 2 shows the convergence pattern for the ABZ7sdst, one of the largest instances. From these results we have set the number of generations at 200 in the base configuration of GT$\mathscr{N}$. Notice that the number of chromosomes in the population and the stopping criterion of TS will remain fixed in the remaining of this section, when comparing different variants of GT$\mathscr{N}$. We shall use the running time as stopping criterion, so every configuration takes the same running time as our base configuration. Thus, we obtain a fair comparison between variants of the algorithm under equivalent conditions.

As we have mentioned, we use the estimation procedure as a filter so as those neighbors having TWT estimation larger or equal than that of the current schedule are discarded. The remaining ones are actually evaluated and the best of these is the selected neighbor. If the estimation for all neighbors is worse than that of the current one, then the neighbor with the best estimation is selected and evaluated. However, it is usual that estimation procedures are used as a selection criterion by themselves, and in that case the neighbor with the best estimation is selected and only this one is evaluated. Moreover, we could consider a third possibility that consist in evaluating all neighbors without previous estimation. To evaluate these three options, termed filtering,

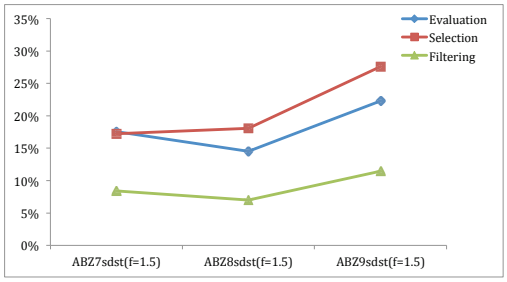


Fig. 3: Average relative errors obtained by GT$\mathcal{N}$ on the ABZ instances with three different selection schemas: filter, selection and evaluation. The details of these methods are given in the text.

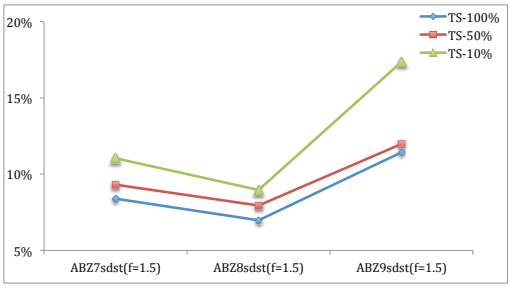


Fig. 4: Average relative errors obtained by applying TS to all or only a portion of the population of GA for the three largest instances.

selection and evaluation respectively, we have done some experiments. The results from the ABZ instances, given in Figure 3, shows that the filtering option is the best one and so it is taken in the remaining experiments.

Applying TS to every new chromosome is computationally expensive and at the same time it may have an effect on the diversity of the population. For this reason, it is common to apply TS to a portion of the new chromosomes instead. In order to analyze this possibility, we have done some experiments with a variant of GT$\mathcal{N}$ where TS is issued with a given probability. Figure 4 shows the results of these experiments for the ABZ instances, in which we have considered three different probabilities: 0.1, 0.5 and 1.0. These results show that applying TS with probability 1.0 is the best option.

In order to demonstrate that the hybrid algorithm is more efficient than any of the two metaheuristics by itself, we have experimented with GA and TS alone. In this case GA was run with a population of 1000 chromosomes as it hardly can converge any more after a few hundred generations with a population of 100 chromosomes, and TS was iterated along 4,5 millions steps to obtain the same run time. The results for the ABZ instances are shown in Figure 5 where we can see that TS is much better than GA. However, it is the synergy

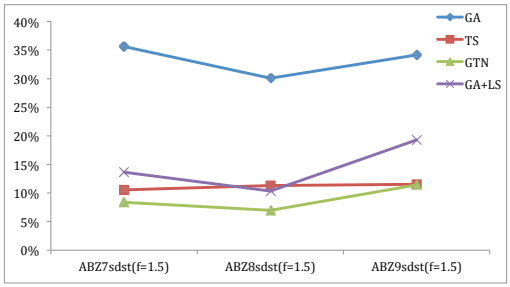


Fig. 5: Average relative errors obtained by applying GA and TS alone, the combination of both, and GA with hill-climbing, for the three largest instances

gained from their use in combination what allows GT$\mathcal{N}$ for exploiting the complementary strengths of global search of GA and local optimum avoidance of TS to obtain a really efficient algorithm. We have also experimented with a combination of GA and a simple local search based on hill-climbing instead of TS. The results, reported in Figure 5 as well (GA+LS), show that this combination is good but it is not better than the combination of GA with TS.

In Section 5.2 we have defined additional conditions to discard some non-improving neighbors. We have carried out some experiments to assess these conditions. Even though their use allow the algorithm to evaluate about 25% more solutions taking the same time as when these conditions are not exploited, the overall results were not better. It seems that if these conditions are not used, TS gets a better balance between the number of improving and non-improving neighbors, thus avoiding getting stuck in local optima and reaching eventually better solutions. For this reason, we have not considered these conditions in the remaining of our experimental study.

We have also analyzed the influence of the number of critical paths in the set Π used to obtain the neighbors. We have considered two options: select all critical paths and only the critical path that contributes the most to the TWT of the schedule. Each one of these methods was the best for some instances and the worst for others, and in average there were not significant differences among them, so we have finally opted to choose randomly one of these options in each run of TS. We have observed that this strategy has also been good for the population diversity.

6.4 Comparison of GT$\mathcal{N}$ against other algorithms

The purpose of this section is to compare GT$\mathcal{N}$ with other current approaches. For the reasons given at the beginning of Section 6, we compare GT$\mathcal{N}$ with a number of state-of-the-art methods across JSP instances without setup times.To our knowledge, there exist no other approaches in the liter-

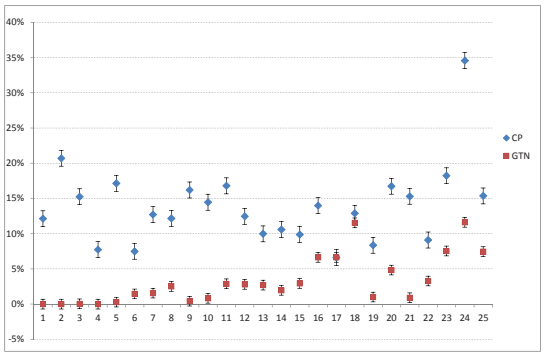
ature to minimizing total weighted tardiness for the SDST-JSP. In the absence of algorithms specifically designed for this problem, we compare GT$\mathcal{N}$ with a constraint programming algorithm run on IBM ILOG CPLEX CP Optimizer tool (CP in the following. This is a commercial solver embedding powerful constraint propagation techniques and a self-adapting large neighborhood search method dedicated to scheduling (Philippe Laborie (2009)) and it is often used to compare with other approaches to scheduling problems. For example, in Gacias et al (2010) the authors confront a parallel machine scheduling problem with precedence constraints and setup times by means of a branch-and bound procedure combined with a climbing discrepancy search algorithm. The results of this algorithm are compared with those from CP Optimizer and in some cases this solver achieves the best results. This solver is expected to be very efficient for a variety of scheduling problems as it is pointed in IBM (2009).

6.4.1 Results from SDST-JSP instances

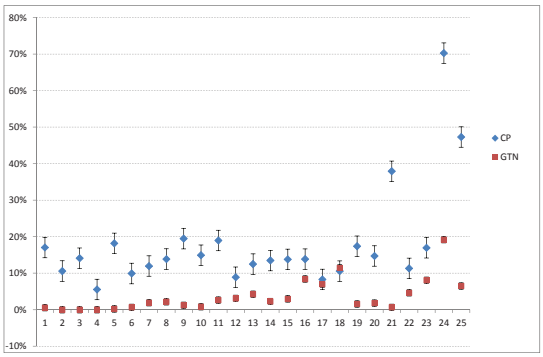
As before, GT$\mathcal{N}$ was parameterized as /100/200/50/. Both GT$\mathcal{N}$ and CP were run 30 times for each instance and CP was given a run time about 20% larger than the time taken by GT$\mathcal{N}$ in each instance. The option *Extended* of CP was set for parameter *NoOverlapInferenceLevel* as the results in this case were slightly better.

Table 1 summarizes the results from these experiments. On average, the total and average gaps obtained by GT$\mathcal{N}$ are about 10% lower than those obtained by CP. The differences are larger for $f = 1.5$ and $f = 1.6$ than they are for $f = 1.3$ due to for those values GT$\mathcal{N}$ is much better than CP for two of the largest instances. To better illustrate the comparison between these two methods, in Figures 6 (a), (b) and (c) we show the average gaps together with 95% confidence intervals for each method, calculated with the typical error. We can see that the average GAPs are really large if we are aware of the different scales in the three figures. Notice that the upper bound of the intervals for GT$\mathcal{N}$ are much smaller than the lower bound of the intervals for CP, with only six exceptions in all the 75 instances.

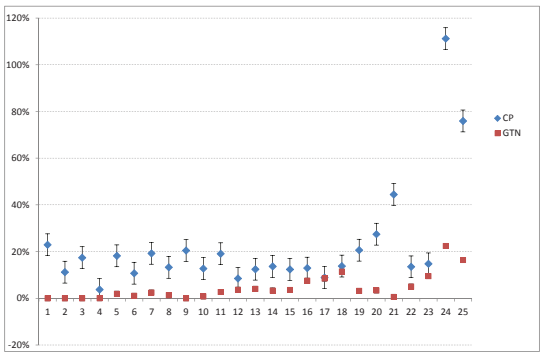
To enhance the conclusions of the experimental comparison between GT$\mathcal{N}$ and CP, we have conducted some statistical analysis. Following Garcia et al (2009), as it is a multiple-problem analysis, we have used non-parametric statistical tests, in fact we have used paired Wilcoxon tests samples. Notice that we consider the average value obtained in the 30 runs as the solution given by the corresponding method for the instance as a way to eliminate the random effect of the method. We have used as alternative hypothesis that CP values are greater than GT$\mathcal{N}$ values. The p-value was 3.431e-14 when all 75 instances was taken as sample and it was 7.381e-06, 5.96e-08 and 2.98e-08 respectively



(a) $f = 1.3$



(b) $f = 1.5$



(c) $f = 1.6$

Fig. 6: Average GAP and 95% confidence intervals for GT$\mathcal{N}$ and CP in SDST-JSP instances

when each of the three families of 25 instances was taken as sample, so the null hypothesis is rejected at a high level of significance. Then it is clear that GT$\mathcal{N}$ outperforms CP in this benchmark. Detailed results for this benchmark, including deviations and best values, can be downloaded from http://www.di.uniovi.es/tc/spanish/repository.htm.

6.4.2 Results from JSP instances without setup times

We start considering the benchmark proposed by Singer and Pinedo to compare GT$\mathcal{N}$ with the hybrid genetic algorithm (GLS) proposed in Essafi et al (2008), the general local search

Table 1: Summary of results from GT$\mathscr{N}$ and CP across SDST-JSP-TWT instances

#Inst	Inst.	$f=1.3$ BKS	$f=1.3$ CP	$f=1.3$ GT$\mathscr{N}$	$f=1.5$ BKS	$f=1.5$ CP	$f=1.5$ GT$\mathscr{N}$	$f=1.6$ BKS	$f=1.6$ CP	$f=1.6$ GT$\mathscr{N}$	T(s)
1	t2-ps01	4454	4994	**4454**	3342	3911	**3361**	2852	3506	**2852**	47
2	t2-ps02	3432	4143	**3432**	2674	2957	**2674**	2301	2558	**2301**	48
3	t2-ps03	3999	4609	**4001**	3120	3560	**3120**	2677	3143	**2677**	51
4	t2-ps04	3732	4021	**3732**	2890	3050	**2890**	2538	2632	**2539**	48
5	t2-ps05	3795	4445	**3806**	2989	3532	**2996**	2570	3038	**2620**	45
6	t2-ps06	9799	10533	**9941**	8186	8997	**8238**	7359	8148	**7436**	121
7	t2-ps07	9362	10552	**9508**	7927	8875	**8079**	7248	8642	**7425**	122
8	t2-ps08	9660	10834	**9902**	8182	9317	**8360**	7524	8521	**7624**	120
9	t2-ps09	9956	11569	**9998**	8116	9697	**8215**	7317	8813	**7317**	124
10	t2-ps10	10483	11999	**10569**	8673	9968	**8745**	7850	8848	**7914**	116
11	t2-ps11	22406	26169	**23052**	20285	24132	**20816**	19239	22909	**19764**	229
12	t2-ps12	22526	25331	**23158**	20487	22309	**21119**	19399	21039	**20106**	241
13	t2-ps13	23394	25729	**24026**	20935	23548	**21821**	19817	22279	**20618**	244
14	t2-ps14	24928	27569	**25416**	22543	25580	**23051**	21190	24079	**21892**	250
15	t2-ps15	24703	27144	**25427**	22368	25450	**23028**	21290	23919	**22049**	237
16	ABZ7sdst	22943	26150	**24467**	19697	22425	**21351**	18350	20721	**19729**	751
17	ABZ8sdst	22763	**24270**	24274	19437	21050	**20793**	17667	19236	**19159**	810
18	ABZ9sdst	21747	24550	**24257**	18778	**20764**	20925	17256	19633	**19235**	787
19	LA21sdst	8741	9471	**8829**	5868	6888	**5960**	4514	5444	**4657**	218
20	LA24sdst	8501	9923	**8912**	5941	6815	**6049**	4674	5956	**4831**	224
21	LA25sdst	8648	9971	**8726**	5435	7495	**5471**	4105	5930	**4129**	222
22	LA27sdst	26976	29431	**27861**	22348	24878	**23384**	20047	22751	**21051**	478
23	LA29sdst	25039	29603	**26928**	20833	24364	**22513**	19185	22014	**21009**	483
24	LA38sdst	7317	9846	**8167**	3344	5693	**3985**	1856	3920	**2272**	334
25	LA40sdst	7750	8940	**8327**	4026	5930	**4287**	2191	3855	**2550**	344
Total GAP(%)			12,89	4,07		14,82	4,44		15,52	4,88	
Avg. GAP(%)			13,87	3,27		16,85	3,56		22,36	4,51	

method (MDL) proposed in Mati et al (2011), and the hybrid shifting bottleneck-tabu search (SB-TS) proposed in Bülbül (2011). GLS is implemented in C++ and was run in a PC with a 2.8 GHz processor and 512 MB RAM and MDL was run in a Pentium with a 2.6 GHz processor, in both cases giving maximum runtime of 18 seconds per run. SB-TS was run in a PC with 2.4 GHz processor and 3.25 Gb RAM; however a Excel/VB environment was used, so the running times may not be comparable with those from the other methods. The target machine used for GT$\mathscr{N}$ is similar to that used for GLS, so we have given GT$\mathscr{N}$ 18 seconds per run.

Table 2 shows the results from these experiments together with the results from the other methods. Remember that all algorithms were run 10 times for each instance, with the exception of SB-TS that was run only once. GT$\mathscr{N}$ was the only algorithm reaching the BKS in at least one run for all 66 instances. Globally, GT$\mathscr{N}$ obtains the BKS in 517 of the 660 runs (78.3%), GLS in 443 (67.1%), MDL in 458 (69.4%) and SB-TS in 43 of the 66 runs (65.2%). Regarding total and average gaps obtained with GT$\mathscr{N}$, they are similar to the average values of the other methods for $f = 1.3$, a little bit better for $f = 1.5$ and clearly better for $f = 1.6$. The results for the ORB instances are summarized in Figure 7.

Paired Wilcoxon tests comparing GT$\mathscr{N}$ versus GLS, MDL and SB-TS give p-values 0.01159, 0.01174 and 0.4054 respectively. So, there are not difference between GT$\mathscr{N}$ and SB-TS. In this case, the results from tests restricted to each family ($f-$value) show that there are difference for $f = 1.6$ only.

We now consider the benchmark proposed in Essafi et al (2008). Table 3 shows the results from GT$\mathscr{N}$ together with the results of GLS reported in Essafi et al (2008) and the results of SB-TS reported in Bülbül (2011). GLS was run for 200 generations disregarding the size of the instances, but the authors do not give further details about the time taken in their experiments, so the comparison may not be entirely accurate. Also, SB-TS was run only once and the authors report results neither from the largest instances, LA31 to LA40, nor for the time taken for the remaining instances.

Compared to GLS, GT$\mathscr{N}$ obtains better average TWT in 51 instances (42.5%), the same value in 31 instances (25.8%) and it is worse in 38 instances (31.7%). With respect to SB-

Table 2: Summary of results from SB-TS, GLS, and GT$\mathcal{N}$ across Singer and Pinedo's benchmark

	$f = 1.3$					$f = 1.5$					$f = 1.6$				
Inst.	BKS	SB-TS	GLS	MDL	GT$\mathcal{N}$	BKS	SB-TS	GLS	MDL	GT$\mathcal{N}$	BKS	SB-TS	GLS	MDL	GT$\mathcal{N}$
ABZ5	1403	1462	*	1414	1412	69	*	*	*	*	0	*	*	*	*
ABZ6	436	*	*	*	*	0	*	*	*	*	0	*	*	*	*
LA16	1169	*	1175	*	*	166	*	*	*	*	0	*	*	*	*
LA17	899	*	*	*	*	260	*	*	*	*	65	*	*	*	*
LA18	929	*	933	934	*	34	*	*	*	*	0	*	*	*	*
LA19	948	955	949	*	998	21	23	*	*	*	0	*	*	*	*
LA20	805	*	*	*	834	0	1	*	*	0	0	*	*	*	*
LA21	463	*	*	*	*	0	*	*	*	*	0	*	*	*	*
LA22	1064	1084	1087	**1077**	1079	196	*	*	*	*	0	*	*	*	*
LA23	835	877	**865**	**865**	870	2	*	*	*	*	0	*	*	*	*
LA24	835	*	*	*	*	82	*	86	86	88	0	*	*	*	*
MT10	1363	*	1372	*	1383	394	*	*	*	*	141	155	162	152	**145**
ORB1	2568	2630	2651	2639	**2578**	1098	1202	1159	1247	*	566	619	688	653	**592**
ORB2	1408	*	1444	1426	1426	292	*	*	*	*	44	52	*	*	*
ORB3	2111	**2115**	2170	2158	2160	918	**928**	943	961	939	422	461	514	463	**434**
ORB4	1623	1652	1643	1690	**1632**	358	*	394	435	*	66	*	78	68	*
ORB5	1593	*	1659	1775	1615	405	*	*	415	428	163	181	181	**176**	**176**
ORB6	1790	*	*	1793	1854	426	*	440	437	430	28	31	*	*	*
ORB7	590	616	592	*	*	50	*	55	*	*	0	*	*	*	*
ORB8	2429	**2453**	2522	2523	2477	1023	*	1059	1036	1033	621	672	669	643	**639**
ORB9	1316	*	*	*	*	297	*	311	299	302	66	*	83	80	*
ORB10	1679	1801	**1718**	1774	1731	346	**424**	400	436	430	76	78	142	117	**82**
Total GAP(%)		1,40	1,67	2,25	1,51		3,03	3,87	6,20	2,37		8,33	17,54	10,23	3,42
Avg. GAP(%)		1,31	1,27	1,71	1,41		2,24	3,18	4,09	2,27		7,21	18,89	11,15	2,57
# Opt		10	20	20	22		22	18	17	22		15	18	20	22

***** The value is that of the optimal solution

TS, GT$\mathcal{N}$ is better in 50 instances (55.6%), equal in 28 instances (31.1%) and worse in 12 instances (13.3%). In these experiments, GT$\mathcal{N}$ has the best average gap for the three values of f.

Figure 8 shows the average gaps (averaged for instances with the same size) for the three methods. As we can see GT$\mathcal{N}$ is quite competitive with the other methods in all groups of instances, with the exception of the 30×10 instances where GLS is the best algorithm.

We have done paired Wilcoxon tests for the instances solved by the three methods. These tests return p-vales 2.642e-06 and 2.384e-08 when comparing GT$\mathcal{N}$ with GLS and SB-TS respectively. So, they confirm that GT$\mathcal{N}$ performs better than both GLS and SB-TS in this set.

In conclusion, the results in this second benchmark show that GT$\mathcal{N}$ is competitive with two state-of-the-art methods such as GLS and SB-TS, even though the comparison may not be entirely accurate for the reasons commented above. However, considering all the experiments across JSP instances, we may consider GT$\mathcal{N}$ to be at the same level of other state-of-the-art algorithms for the JSP.

7 Conclusions

We have considered the job shop scheduling problem with sequence-dependent setup times, where the objective is to minimize the total weighted tardiness. We have described a disjunctive graph representation for this problem which serves as a basis for defining a generic neighborhood structure $\mathcal{N}$. This generic structure has then been used in a tabu search algorithm, which is embedded in a genetic algorithm framework. We have also defined a method for estimating the total weighted tardiness of the neighbors and proved that it is efficient but not very accurate. We have empirically demonstrated that in our approach it is much more effective to use this estimation procedure as a filter than as a selection criterion, in an opposite way to what is usually seen in the literature. In the experimental study we have also shown that some non-improving conditions for our neighborhood structure do not yield overall better results. Also, it is better to choose randomly in each local search between considering every critical path or only the most important one than applying any of the two alternatives. We have also considered the possibility of applying the tabu search only to a percentage of the population, but the best option is to apply it

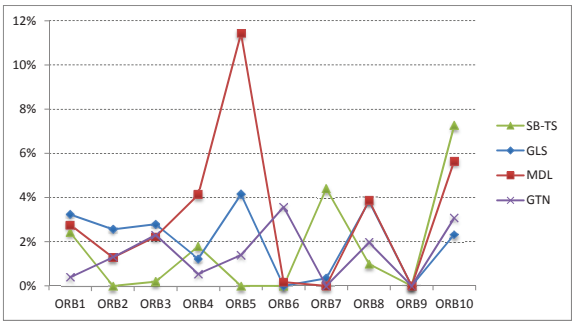
Table 3: Summary of results from SB-TS, GLS, and GT$\mathscr{N}$ across Essafi's benchmark

Inst.	Size	$f = 1.3$ SB-TS	$f = 1.3$ GLS	$f = 1.3$ GT$\mathscr{N}$	$f = 1.5$ SB-TS	$f = 1.5$ GLS	$f = 1.5$ GT$\mathscr{N}$	$f = 1.6$ SB-TS	$f = 1.6$ GLS	$f = 1.6$ GT$\mathscr{N}$	T(s)
LA01	10×5	2299	2299	2299	1616	**1610**	**1610**	1230	1230	1230	44
LA02		1762	1762	1762	1028	1028	1028	695	695	695	43
LA03		1951	1951	1951	1280	1280	1280	1024	1024	1024	46
LA04		1917	1917	1917	1277	1277	1277	1068	**1029**	**1029**	43
LA05		1878	1878	1878	1205	1205	1205	877	877	877	43
LA06	15×5	6008	**5827**	5964	4821	**4658**	4685	4180	4130	**4122**	116
LA07		5961	**5801**	5808	4624	4548	**4470**	**3843**	3988	**3843**	112
LA08		5560	**5482**	5533	4423	4094	**4079**	3584	**3400**	3421	118
LA09		6116	5648	**5608**	4618	**4421**	4424	4040	3835	**3815**	116
LA10		6734	6621	**6618**	5304	**5148**	5162	4728	4533	**4513**	101
LA11	20×5	12792	12341	**12200**	10682	10332	**10187**	9679	9399	**9301**	247
LA12		11238	10683	**10607**	9494	**9084**	9105	8663	8302	**8258**	245
LA13		12533	11889	**11608**	10158	9846	**9678**	9244	8916	**8845**	237
LA14		13681	13225	**13175**	12024	**11382**	11412	11292	10594	**10564**	223
LA15		12964	12428	**12364**	10464	10455	**10377**	9675	9392	**9365**	237
LA16	10×10	1169	1169	1169	166	166	166	0	0	0	66
LA17		899	899	899	260	260	260	65	65	65	67
LA18		929	929	929	34	34	34	0	0	0	62
LA19		955	**948**	949	23	**21**	**21**	0	0	0	64
LA20		**805**	**805**	838	0	0	0	0	0	0	53
LA21	15×10	3841	**3771**	3857	1740	**1692**	1760	**890**	949	915	201
LA22		4453	4471	**4317**	**2099**	2273	2264	**1364**	1450	1404	217
LA23		4103	**3955**	3957	1731	1683	**1510**	1010	977	**946**	192
LA24		3770	3831	**3697**	1849	**1618**	1621	**693**	773	749	195
LA25		3724	3569	**3410**	1499	1497	**1439**	929	922	**893**	197
LA26	20×10	10562	9748	**9416**	6328	**6106**	6324	5236	5125	**4890**	441
LA27		**9827**	9860	10316	6252	**6142**	6303	4441	4590	**4396**	489
LA28		10198	9757	**9578**	**6096**	6254	**6096**	**4403**	4594	4545	464
LA29		9792	9397	**9354**	6265	6392	**6027**	5049	4706	**4569**	468
LA30		9297	**8968**	9060	**5273**	5496	5539	**3821**	4131	4211	429
LA31	30×10	-	**28999**	30936	-	**23417**	24720	-	**21141**	21827	1899
LA32		-	**32004**	34228	-	**25766**	27364	-	**22918**	24038	1931
LA33		-	**27216**	28249	-	**21767**	22067	-	**18801**	19417	1870
LA34		-	**29131**	31139	-	**23341**	23974	-	**20548**	21589	1905
LA35		-	**30979**	31750	-	**24654**	25841	-	**21991**	22988	1887
LA36	15×15	-	3187	**3132**	-	866	**589**	-	192	**151**	278
LA37		-	2880	**2580**	-	589	**415**	-	0	0	239
LA38		-	2287	**2276**	-	438	**416**	-	0	0	263
LA39		-	1861	**1820**	-	9	**7**	-	0	0	228
LA40		-	2305	**2207**	-	**79**	84	-	0	0	258
Total GAP(%)		6,90	2,86	4,92	6,25	4,18	5,86	6,42	4,97	6,65	
Avg GAP(%)		5,01	3,84	3,80	5,35	7,04	4,99	4,57	5,51	4,23	

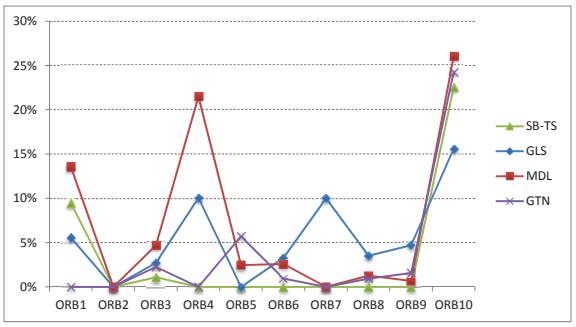
to each and every chromosome. Finally, we have shown the synergy between both metaheuristics, since the hybrid algorithm achieves better results than any of the metaheuristics on its own.

Additionally, we have compared our approach against state-of-the-art algorithms, both for classical JSP and SDST-JSP benchmarks. In the SDST-JSP we have used the 15 instances of the BT-set proposed in Brucker and Thiele (1996) and the set of instances proposed in Vela et al (2010). We have defined due dates and job weights for them and have compared our approach to an implementation on the IBM ILOG CPLEX CP Optimizer tool, showing that GT$\mathscr{N}$ obtains clearly better results in most cases. In particular, the best known solutions have been established by GT$\mathscr{N}$ for all instances, while CP has reached these solutions for none of them.

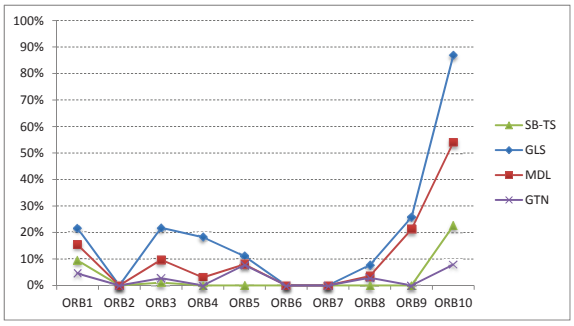
For the JSP we have considered the set of instances proposed in Singer and Pinedo (1998), comparing GT$\mathscr{N}$ to three methods: the hybrid genetic algorithm from Essafi et al (2008), the local search method from Mati et al (2011) and



(a) $f = 1.3$

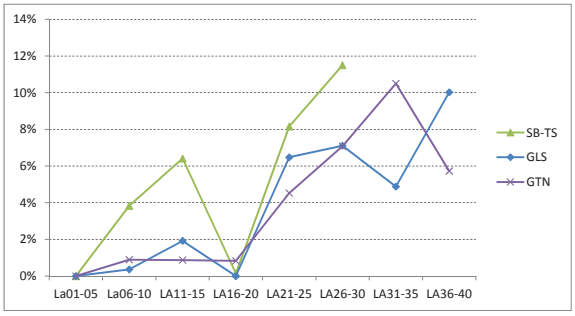


(b) $f = 1.5$

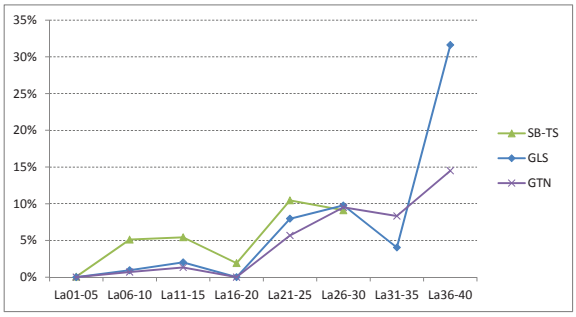


(c) $f = 1.6$

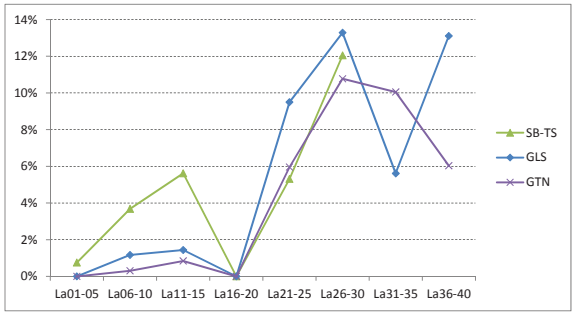
Fig. 7: Average GAP for SB-TS, GLS, MDL and GT$\mathcal{N}$ from the ORB instances of Singer & Pinedo's set for the JSP



(a) $f = 1.3$



(b) $f = 1.5$



(c) $f = 1.6$

Fig. 8: Average GAP for SB-TS, GLS and GT$\mathcal{N}$ from Essafi et al.'s instances for the JSP

the shifting bottleneck-tabu search hybrid from Bülbül (2011). We have also used for the comparison a set of instances proposed in Essafi et al (2008). In general GT$\mathcal{N}$ compares favorably to the other methods, so we can conclude that the proposed approach is competitive with these state-of-the-art methods.

As future work we plan to consider other scheduling problems which are closer to real-life problems. For example we plan to add uncertainty or robustness considerations to this problem. We shall also consider other metaheuristics like scatter search with path relinking to solve the same and other similar scheduling problems.

Acknowledgments

This research has been supported by the Spanish Government under research grants FEDER TIN2010-20976-C02-02 and MTM2010-16051.

References

Angel Bello F, Alvarez A, Pacheco J, Martinez I (2011a) A heuristic approach for a scheduling problem with periodic maintenance and sequence-dependent setup times. Computers and Mathematics with Applications 61(4):797–808

Angel Bello F, Alvarez A, Pacheco J, Martinez I (2011b) A single machine scheduling problem with availability constraints and sequence-dependent setup costs. Applied Mathematical Modelling 35(4):2041–2050

Applegate D, Cook W (1991) A computational study of the job-shop scheduling problem. ORSA Journal of Computing 3:149–156

Artigues C, Belmokhtar S, Feillet D (2004) A new exact solution algorithm for the job shop problem with sequence-dependent setup times. In: Proceedings of CPAIOR 2004, Springer, LNCS, vol 3011, pp 37–49

Artigues C, Lopez P, Ayache P (2005) Schedule generation schemes for the job shop problem with sequence-dependent setup times: Dominance properties and computational analysis. Annals of Operations Research 138:21–52

Balas E, Vazacopoulos A (1998) Guided local search with shifting bottleneck fo job shop scheduling. Management Science 44(2):262–275

Balas E, Simonetti N, Vazacopoulos A (2008) Job shop scheduling with setup times, deadlines and precedence constraints. Journal of Scheduling 11:253–262

Beck JC, Feng T, Watson JP (2011) Combining constraint programming and local search for job-shop scheduling. Informs Journal on Computing 23:1–14, DOI 10.1287/ijoc.1100.0388

Behnamian J, Zandieh M, Fatemi Ghomi S (2011) Bi-objective parallel machines scheduling with sequence-dependent setup times using hybrid metaheuristics and weighted min-max technique. Soft Computing 15:1313–1331

Bierwirth C (1995) A generalized permutation approach to jobshop scheduling with genetic algorithms. OR Spectrum 17:87–92

Brucker P, Thiele O (1996) A branch and bound method for the general-job shop problem with sequence-dependent setup times. Operations Research Spektrum 18:145–161

Brucker P, Jurisch B, Sievers B (1994) A branch and bound algorithm for the job-shop scheduling problem. Discrete Applied Mathematics 49:107–127

Bülbül K (2011) A hybrid shifting bottleneck-tabu serach heuristic for the job shop total weighted tardiness problem. Computers & Operations Research 38:967–783

Carlier J, Pinson E (1989) An algorithm for solving the job-shop problem. Management Science 35(2):164–176

DeBontridder K (2005) Minimizing total weighted tardiness in a generalized job shop. Journal of Scheduling 8:479–496

Dell' Amico M, Trubian M (1993) Applying tabu search to the job-shop scheduling problem. Annals of Operational Research 41:231–252

Dorndorf U, Pesch E, Phan-Huy T (2000) Constraint propagation techniques for the disjunctive scheduling problem. Artificial Intelligence 122:189–240

Essafi I, Mati Y, Dauzère-Pérès S (2008) A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem. Computers and Operations Research 35:2599–2616

Gacias B, Artigues C, Lopez P (2010) Parallel machine scheduling with precedence constraints and setup times. Computers and Operations Research 37:2141–2151

Garcia S, Molina D, Lozano M, Herrera F (2009) A study on the use of non-parametric tests for analyzing the evolutionary algorithms' Behaviour: a case study on the CEC'2005 special session on real parameter optimization. Journal of Heuristics 15:617–644

Giffler B, Thompson GL (1960) Algorithms for solving production scheduling problems. Operations Research 8:487–503

Glover F, Laguna M (1997) Tabu Search. Kluwer Academic Publishers

González MA, Vela CR, Varela R (2008) A new hybrid genetic algorithm for the job shop scheduling problem with setup times. In: Proceedings of ICAPS-2008, AAAI Press, Sidney, pp 116–123

González MA, Vela CR, Varela R (2009) Genetic algorithm combined with tabu search for the job shop scheduling problem with setup times. In: Proceedings of IWINAC 2009, LNCS-5601, Springer, pp 265–274

González MA, Vela CR, Varela R (2011) Weighted tardiness minimization in job shops with setup times by hybrid genetic algorithm. In: Proceedings of CAEPIA 2011, Springer, La Laguna (Spain), LNCS, pp 363–372

González MA, Vela C, González-Rodríguez I, Varela R (2012) Lateness minimization with tabu search for job shop scheduling problem with sequence dependent setup times. Journal of Intelligent Manufacturing pp 1 – 14

Graham R, Lawler E, Lenstra J, Rinnooy Kan A (1979) Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of Discrete Mathematics 4:287–326

IBM (2009) Modeling with IBM ILOG CP Optimizer - practical scheduling examples

Jat SN, Yang S (2011) A hybrid genetic algorithm and tabu search approach for post enrolment course timetabling. Journal of Scheduling 14:617–637, DOI DOI 10.1007/s10951-010-0202-0

Kreipl S (2000) A large step random walk for minimizing total weighted tardiness in a job shop. Journal of Scheduling 3:125–138

Lawrence S (1984) Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (supplement). Tech. rep., Graduate School of Industrial Administration, Carnegie Mellon University

Mahfoud SW (1992) Crowding and preselection revisited. In: Parallel Problem Solving From Nature, Elsevier, pp 27–36

Mati Y, Dauzere-Peres S, Lahlou C (2011) A general approach for optimizing regular criteria in the job-shop scheduling problem. European Journal of Operational Research 212:33–42

Matsuo H, Suh C, Sullivan R (1988) A controlled search simulated annealing method for the general jobshop scheduling problem. Working paper 03-44-88, Graduate School of Business, University of Texas

Nowicki E, Smutnicki C (1996) A fast taboo search algorithm for the job shop scheduling problem. Management Science 42:797–813

Nowicki E, Smutnicki C (2005) An advanced tabu search algorithm for the job shop problem. Journal of Scheduling 8:145–159

Philippe Laborie P (2009) Ibm ilog cp optimizer for detailed scheduling illustrated on three problems. In: Proceedings of CPAIOR09, pp 148–162

Singer M, Pinedo M (1998) A computational study of branch and bound techniques for minimizing the total weighted tardiness in job shops. IIE Transactions 30:109–118

Singer M, Pinedo M (1999) A shifting bottleneck heuristic for minimizing the total weighted tardiness in a job shop. Naval Research Logistics 46(1):1–17

Taillard ED (1994) Parallel taboo search techniques for the job shop scheduling problem. ORSA Journal on Computing 6(2):108–117

Tavakkoli-Moghaddam R, Khalili M, Naderi B (2009) A hybridization of simulated annealing and electromagnetic-like mechanism for job shop problems with machine availability and sequence-dependent setup times to minimize total weighted tardiness. Soft Computing 13:995–1006

Van Hentenryck P, Michel L (2004) Scheduling abstractions for local search. Lecture Notes in Computer Science 3011:319–334

Van Laarhoven P, Aarts E, Lenstra K (1992) Job shop scheduling by simulated annealing. Operations Research 40:113–125

Varela R, Serrano D, Sierra M (2005) New codification schemas for scheduling with genetic algorithms. Proceedings of IWINAC 2005 Lecture Notes in Computer Science 3562:11–20

Vela CR, Varela R, González MA (2010) Local search and genetic algorithm for the job shop scheduling problem with sequence dependent setup times. Journal of Heuristics 16:139–165

Zhang CY, Li P, Rao Y, Guan Z (2008) A very fast TS/SA algorithm for the job shop scheduling problem. Computers and Operations Research 35:282–294