Noname manuscript No.
(will be inserted by the editor)

Solving Job Shop Scheduling with Setup Times through Constraint-based Iterative Sampling: An Experimental Analysis

Angelo Oddi · Riccardo Rasconi · Amedeo Cesta · Stephen F. Smith

Received: date / Accepted: date

Abstract This paper presents a heuristic algorithm for solving a job-shop scheduling problem with sequence dependent setup times and *min/max* separation constraints among the activities (SDST-JSSP/MAX). The algorithm relies on a core constraint-based search procedure, which generates consistent orderings of activities that require the same resource by incrementally imposing precedence constraints on a temporally feasible solution. Key to the effectiveness of the search procedure is a conflict sampling method biased toward selection of most critical conflicts and coupled with a non-deterministic choice heuristic to guide the base conflict resolution process. This constraint-based search is then embedded within a larger iterative-sampling search framework to broaden search space coverage and promote solution optimization. The efficacy of the overall heuristic algorithm is demonstrated empirically both on a set of previously studied job-shop scheduling benchmark problems with sequence dependent setup times and by introducing a new benchmark with setups and generalized precedence constraints.

Keywords random-restart · constraint-based reasoning · job-shop scheduling · setup times · generalized precedence constraints

1 Introduction

This paper describes an approach for solving job-shop scheduling problems with ready times, deadlines, *sequence dependent* setup times and *min/max* separation constraints among activities (SDST-JSSP/MAX). Similar problems are common in a semiconductor manufacturing environment – see Ovacik and Uzsoy (1994, 1997) – and in general, over the last ten years, there has been an increasing interest in solving scheduling

Angelo Oddi · Riccardo Rasconi · Amedeo Cesta
Institute for Cognitive Science and Technology
Italian National Research Council, Rome, Italy
E-mail: name.surname@istc.cnr.it

S.F. Smith
Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA
E-mail: sfs@cs.cmu.edu

problems with setup times (Allahverdi and Soroush 2008; Allahverdi et al 2008). This fact stems mainly from the observation that in many real-word industry or service environments there are tremendous savings when setup times are explicitly considered in scheduling decisions.

The algorithm proposed in this work relies on a *core* constraint-based search procedure, which generates a consistent ordering of activities that require the same resource by incrementally adding precedence constraints between activity pairs belonging to a temporally feasible solution. The algorithm bases its solving capability on a *general* meta-heuristic which, despite its simplicity, has proven to be quite effective, generating competitive results for a wide range of different scheduling problems. Specifically, the algorithm we propose in this work is an extension of the stochastic version of the SP-PCP procedure proposed in Oddi and Smith (1997) suitable for solving scheduling problems with setup times. The resulting procedure is then embedded within a larger *iterative-sampling* search framework, also used in Cesta et al (2002), to broaden search space coverage and promote solution optimization.

The main objective of this work is to investigate the generality of this constraint-based approach and its adaptability to the SDST-JSSP/MAX, to understand the extent to which its traditional effectiveness and efficiency is preserved in this setting. In this regard we show: (1) that the general algorithm can be adapted to the setup times case with minimal effort and change; and (2) that the solving capabilities of our algorithm remain unaltered even in the presence of *min/max* separation constraints. These claims are substantiated by means of an extensive experimental analysis.

Within the current literature, there are several other examples of procedures for solving scheduling problems with setup times that are extensions of counterpart procedures for solving the same (or similar) scheduling problem without setup times. This is the case of the work by Brucker and Thiele (1996), for example, which relies on an earlier solutions introduced in Brucker et al (1994). Another example is the more recent work of Vela et al (2009) and González et al (2009a), which proposes effective heuristic procedures based on genetic algorithms and local search. The local search procedures that are introduced in these works extend a procedure originally proposed by Nowicki and Smutnicki (2005) for the classical job-shop scheduling problem to the setup times case by introducing a neighborhood structure that exhibits similar properties relatively to critical paths in the underlying disjunctive graph formulation of the problem. A third example is the work of Balas et al (2008), which extends the well-know *shifting bottleneck* procedure (Adams et al 1988) to the SDST-JSSP case. All of the procedures just mentioned solve a relaxed version of the full SDST-JSSP/MAX in which there are no *min/max* separation constraints (we will refer to this problem in the following as the SDST-JSSP). Both Balas et al (2008) and González et al (2009a) have produced reference results with their techniques on a previously studied benchmark set of SDST-JSSP problems proposed by Ovacik and Uzsoy (1994). Despite our broader interest in solving the SDST-JSSP/MAX, we use this benchmark problem set as a basis for direct comparison to our solution procedure in the experimental section of this paper.

The procedure we propose is not the only one which relies on the constraint-solving paradigm; another example is described in (Focacci et al 2000), which introduces a more elaborate procedure based on the integration of two different solution models. One key feature of our procedure is its simplicity, in spite of its effectiveness in solving a set of difficult instances of SDST-JSSPs. The advantages of simplicity will be argued as the algorithm and the experimental analysis are presented.

This paper is organized as follows. An introductory section defines the reference SDST-JSSP/MAX problem and its representation. A central section describes the core constraint-based procedure and the overarching iterative sampling search strategy. An experimental section describes the performance of our algorithm the benchmark problem set of well-known SDST-JSSPs and the most interesting results are explained. Finally, a new set of benchmark problems that captures the full complexity of the SDST-JSSP/MAX is introduced. Some conclusions and a discussion of future work end the paper.

2 The Scheduling Problem with Setup Times

In this section we provide a definition of the job-shop scheduling problem with sequence dependent setup times which is extended to include *min/max* separation constraints (SDST-JSSP/MAX). To the best of our knowledge, this generalization of the SDST-JSSP has not been previously considered in the literature. We have selected this rather complex problem as our focus in this paper principally as a means of better showing the effectiveness and versatility of our general constraint-based search procedure. Its effectiveness will be demonstrated by comparing our algorithm's performance with current known best known results on the previously studied SDST-JSSP, which is the relaxed version of the SDST-JSSP/MAX that excludes time windows. Its versatility will be established by first showing the original algorithm's adaptability to the SDST-JSSP, and then its direct applicability to the extended case when time lags are added to the problem.

The SDST-JSSP/MAX entails the synchronization of a set of resources $R = \{r_1, r_2, \ldots, r_m\}$ to perform a set of n activities $A = \{a_1, \ldots, a_n\}$ over time. The set of activities is partitioned into a set of nj jobs $\mathcal{J} = \{J_1, \ldots, J_{nj}\}$. The processing of a job J_k requires the execution of a strict sequence of m activities $a_{ik} \in J_k$ $(i = 1, \ldots, m)$, and the execution of each activity a_{ik} is subject to the following constraints:

- *resource availability* - each activity a_i requires the exclusive use of a single resource r_{a_i} for its entire duration; no *preemption* is allowed and all the activities included in a job J_k require distinct resources.
- *processing time constraints* - each a_i has a fixed processing time p_i such that $e_i - s_i = p_i$, where the variables s_i and e_i represent the start and end time of a_i.
- *separation constraints* - for each pair of successive activities a_{ik} and $a_{(i+1)k}$ in a job J_k, there is a *min/max* separation constraint $l_{ik}^{min} \leq s_{(i+1)k} - e_{ik} \leq l_{ik}^{max}$, $i = 1, \ldots, m-1$.
- *sequence dependent setup times* - for each resource r, the value st_{ij}^r represents the setup time between two generic activities a_i and a_j (a_j is scheduled immediately after a_i) requiring the same resource r, such that $e_i + st_{ij}^r \leq s_j$. As is traditionally assumed in the literature, the setup times st_{ij}^r satisfy the so-called *triangular inequality* (see Brucker and Thiele (1996); Artigues and Feillet (2008)). The triangle inequality states that, for any three activities a_i, a_j, a_k requiring the same resource, the inequality $st_{ij}^r \leq st_{ik}^r + st_{kj}^r$ holds.
- *job release and due dates* - each Job J_k has a release date rd_k, which specifies the earliest time that any activity in J_k can be started, and a due date d_k, which designates the time by which all activities in J_k should be completed. The due date is not a mandatory constraint and can be violated (see below).

Figure 1 shows an example of SDST-JSSP/MAX problem. There are two jobs, which require a set of three resources $R = \{r_1, r_2, r_3\}$. Each job is composed by three activities such that $J_1 = \{a_1, a_2, a_3\}$ and $J_2 = \{a_4, a_5, a_6\}$. Directed edges represent temporal constraints, in particular *min/max* separation constraints are shown as labels $[lb, ub]$, in case there is no label, the edge represents a simple precedence. Finally, undirected and dashed edges link the activities requiring the same resource (shows as edges' labels r_1, r_2, and r_3); ready times and setup times are not shown in the figure.

A feasible *solution* $S = \{S_1, S_2, \ldots, S_n\}$ is an assignment S_i to the activities start-times s_i such that all the above constraints are satisfied. We will consider two objective criteria for optimization: the *maximum lateness* (L_{max}) and the *makespan* (C_{max}). Let C_k be the completion time for the job J_k, the *maximum lateness* is the value $L_{max} = max_{1 \leq k \leq nj}\{C_k - d_k\}$, while the makespan is the value $C_{max} = max_{1 \leq k \leq nj}\{C_k\}$. Depending on the chosen objective, an *optimal* solution S^* is a solution S with the minimum value of L_{max} or C_{max}. Both of these optimization problems are strongly *NP-hard*, because they are extensions of the known problems $1|r_i|L_{max}$ and $J||C_{max}$ Brucker et al (1977) and Sotskov and Shakhlevich (1995), respectively. In addition, the presence of minimum and maximum separation constraints (i.e., time windows) introduces a further element of complexity in searching for a solution.

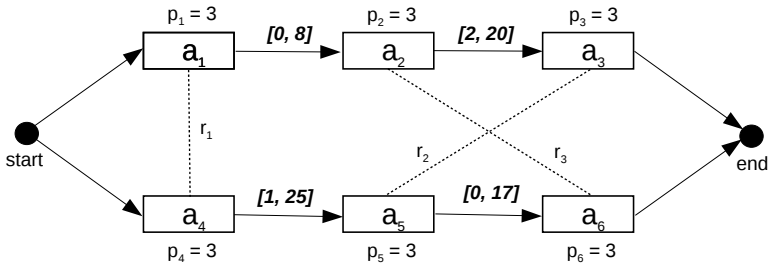

Fig. 1 An example of SDST-JSSP/MAX instance.

3 A CSP Representation

There are different ways to formulate this problem as a *Constraint Satisfaction Problem* (CSP) Montanari (1974). Analogously to Cheng and Smith (1994); Oddi and Smith (1997), we treat the problem as the one of establishing *precedence constraints* between pairs of activities that require the same resource, so as to eliminate all possible conflicts in the resource use. Such representation is close to the idea of *disjunctive graph* initially used for the classical job shop scheduling without setup times and also used in the extended case of setup times in Brucker and Thiele (1996); Balas et al (2008); Vela et al (2009); Artigues and Feillet (2008).

Let $G(A_G, J, X)$ be a graph where the set of vertices A_G contains all the activities of the problem together with two dummy activities, a_0 and a_{n+1}, respectively representing the beginning (reference) and the end (horizon) of the schedule. J is a set

of directed edges (a_i, a_j) representing the precedence constraints among the activities (job precedence constraints) and are weighted with the processing time p_i of the edge's source activity a_i. The set of undirected edges X represents the *disjunctive constraints* among the activities requiring the same resource r. There is an edge for each pair of activities a_i and a_j requiring the same resource r and the related label represents the set of possible ordering between a_i and a_j: $a_i \preceq a_j$ or $a_j \preceq a_i$.

Hence, in CSP terms, a decision variable x_{ijr} is defined for each pair of activities a_i and a_j requiring resource r, which can take one of two values: $a_i \preceq a_j$ or $a_j \preceq a_i$. It is worth noting that in considering either ordering we have to take into account the presence of sequence dependent setup times, which must be included when an activity a_i is executed on the same resource *before* another activity a_j. As we will see in the next sections, if the setup times satisfy the triangle inequality, the previous decisions for x_{ijr} can be represented as the following two temporal constraints: $e_i + st^r_{ij} \leq s_j$ (i.e. $a_i \preceq a_j$) or $e_j + st^r_{ji} \leq s_i$ (i.e. $a_j \preceq a_i$).

To support the search for a consistent assignment to the set of decision variables x_{ijr}, for any SDST-JSSP/MAX we define the directed graph $G_d(V, E)$, called *distance graph*, which is an extended version of the disjunctive graph $G(A_G, J, X)$. The set of nodes V represents time points, where tp_0 is the *origin* time point (the reference point of the problem), while for each activity a_i, s_i and e_i represent its start and end time points respectively. The set of edges E represents all imposed temporal constraints, i.e., precedences, durations and setup times. Given two time points tp_i and tp_j, all the constraints have the form $a \leq tp_j - tp_i \leq b$, and for each constraint specified in the SDST-JSSP/MAX instance there are two weighted edges in the graph $G_d(V, E)$; the first one is directed from tp_i to tp_j with weight b and the second one is directed from tp_j to tp_i with weight $-a$. The graph $G_d(V, E)$ corresponds to a *Simple Temporal Problem* and its consistency can be determined via shortest path computations (see Dechter et al (1991) for more details on the STP). Moreover, any time point tp_i is associated to a given *feasibility interval* $[lb_i, ub_i]$, which determines the current set of feasible values for tp_i. Thus, a search for a solution to a SDST-JSSP/MAX instance *can proceed by repeatedly adding new precedence constraints into* $G_d(V, E)$ and recomputing shortest path lengths to confirm that $G_d(V, E)$ remains consistent. Given a Simple Temporal Problem, the problem is consistent if and only if no closed paths with negative length (i.e., negative cycles) are contained in the graph G_d.

Let $d(tp_i, tp_j)$ $[d(tp_j, tp_i)]$ designate the shortest path length in graph $G_d(V, E)$ from node tp_i to node tp_j [from node tp_j to node tp_i]; then, the constraint $-d(tp_j, tp_i) \leq tp_j - tp_i \leq d(tp_i, tp_j)$ is demonstrated to hold (see Dechter et al (1991)). Hence, the *minimal* allowed distance between tp_j and tp_i is $-d(tp_j, tp_i)$ and the maximal distance is $d(tp_i, tp_j)$. Given that d_{i0} is the length of the shortest path on G_d from the time point tp_i to the origin point tp_0 and d_{0i} is the length of the shortest path from the origin point tp_0 to the time point tp_i, the interval $[lb_i, ub_i]$ of time values associated with the generic time variable tp_i is computed on the graph G_d as the interval $[-d(tp_i, tp_0), d(tp_0, tp_i)]$ (see Dechter et al (1991)). In particular, given a STP, the following two sets of value assignments $S_{lb} = \{-d(tp_1, tp_0), -d(tp_2, tp_0), \ldots, -d(tp_n, tp_0)\}$ and $S_{ub} = \{d(tp_0, tp_1), d(tp_0, tp_2), \ldots, d(tp_0, tp_n)\}$ to the STP variables tp_i are demonstrated to represent feasible solutions called *earliest-time solution* and *latest-time solution*, respectively.

4 A Precedence Constraint Posting Procedure

The proposed procedure for solving instances of SDST-JSSP/MAX is an extension of the SP-PCP scheduling procedure (Shortest Path-based Precedence Constraint Posting) proposed in Oddi and Smith (1997), which utilizes shortest path information in $G_d(V, E)$ for guiding the search process. Similarly to the original SP-PCP procedure, shortest path information is utilized in a twofold fashion to enhance the search process.

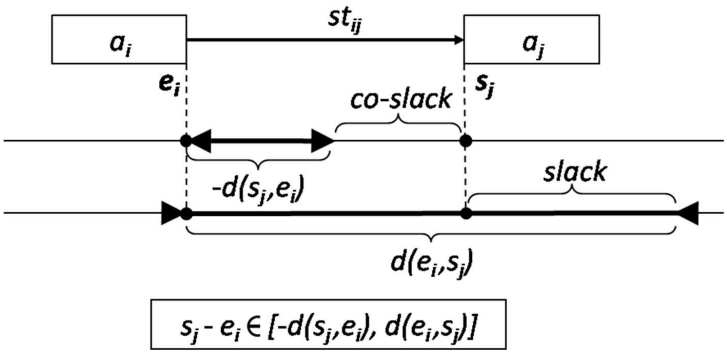

Fig. 2 $slack(e_i, s_j) = d(e_i, s_j) - st^r_{ij}$ Vs. $co\text{-}slack(e_i, s_j) = -d(s_j, e_i) - st^r_{ij}$

The first way of exploiting shortest path information is by introducing new *dominance conditions* (which adapt those presented in Oddi and Smith (1997) to the setup times case), through which problem constraints are *propagated* and unconditional decisions for promoting early pruning of alternatives are identified. The concepts of $slack(e_i, s_j)$ and $co\text{-}slack(e_i, s_j)$ (complementary slack) play a central role in the definition of such new dominance conditions. Given two activities a_i, a_j and the related interval of distances $[-d(s_j, e_i), d(e_i, s_j)]$ [1] and $[-d(s_i, e_j), d(e_j, s_i)]$ [2] on the graph G_d, they are defined as follows (see Figure 2):

- $slack(e_i, s_j) = d(e_i, s_j) - st^r_{ij}$ is the difference between the maximal distance $d(e_i, s_j)$ and the setup time st^r_{ij}. Hence, it provides a measure of the degree of *sequencing flexibility* between a_i and a_j [3] taking into account the setup time constraint $e_i + st^r_{ij} \leq s_j$. If $slack(e_i, s_j) < 0$, then the ordering $a_i \preceq a_j$ is not feasible.
- $co\text{-}slack(e_i, s_j) = -d(s_j, e_i) - st^r_{ij}$ is the difference between the minimum possible distance between a_i and a_j, $-d(s_i, e_j)$, and the setup time st^r_{ij}; if $co\text{-}slack(e_i, s_j) \geq 0$ (in Figure 2 a *negative* co-slack is represented), then there is no need to separate a_i and a_j, as the setup time constraint $e_i + st^r_{ij} \leq s_j$ is already satisfied.

For any pair of activities a_i and a_j that are competing for the same resource r, the new dominance conditions describing the four possible cases of conflict are defined as

[1] Between the end-time e_i of a_i and the start-time s_j of a_j

[2] Between the end-time e_j of a_j and the start-time s_i of a_i

[3] Intuitively, the higher is the degree of *sequencing flexibility*, the larger is the set of feasible assignments to the start-times of a_i and a_j

follows:

1. $slack(e_i, s_j) < 0 \wedge slack(e_j, s_i) < 0$
2. $slack(e_i, s_j) < 0 \wedge slack(e_j, s_i) \geq 0 \wedge co\text{-}slack(e_j, s_i) < 0$
3. $slack(e_i, s_j) \geq 0 \wedge slack(e_j, s_i) < 0 \wedge co\text{-}slack(e_i, s_j) < 0$
4. $slack(e_i, s_j) \geq 0 \wedge slack(e_j, s_i) \geq 0$

Condition 1 represents an *unresolvable conflict*. There is no way to order a_i and a_j taking into account the setup times st^r_{ij} and st^r_{ji}, without inducing a negative cycle in the graph $G_d(V, E)$. When Condition 1 is verified the search has reached an inconsistent state.

Conditions 2, and 3, alternatively, distinguish *uniquely resolvable conflicts*, i.e., there is only one feasible ordering of a_i and a_j, and the decision of which constraint to post is thus unconditional. If Condition 2 is verified, only $a_j \preceq a_i$ leaves $G_d(V, E)$ consistent. It is worth noting that the presence of the condition $co\text{-}slack(e_j, s_i) < 0$ entails that the minimal distance between the end time e_j and the start time s_i is shorter than the minimal required setup time st^r_{ji}; hence, we still need to impose the constraint $e_j + st^r_{ji} \leq s_i$. In other words, the *co-slack* condition avoids the imposition of unnecessary precedence constraints for trivially solved conflicts. Condition 3 works similarly, and entails that only the $a_i \preceq a_j$ ordering is feasible.

Finally, Condition 4 designates a class of *resolvable conflicts*; in this case, both orderings of a_i and a_j remain feasible, and it is therefore necessary to perform a *search decision*.

The second way of exploiting shortest path information is by defining *variable* and *value* ordering heuristics for selecting and resolving conflicts in the set characterized by Condition 4. As stated above, in this context $slack(e_i, s_j)$ and $slack(e_j, s_i)$ provide measures of the degree of *sequencing flexibility* between a_i and a_j. The *variable* ordering heuristic attempts to focus first on the conflict with the least amount of sequencing flexibility (i.e., the conflict that is closest to previous Condition 1). More precisely, the conflict (a_i, a_j) with the overall minimum value of $VarEval(a_i, a_j) = min\{bd_{ij}, bd_{ji}\}$ is always selected for resolution, where the *biased distance* terms are defined as follows:[4]:

$$bd_{ij} = \frac{slack(e_i, s_j)}{\sqrt{S}}, \quad bd_{ji} = \frac{slack(e_j, s_i)}{\sqrt{S}}$$

and

$$S = \frac{min\{slack(e_i, s_j), slack(e_j, s_i)\}}{max\{slack(e_i, s_j), slack(e_j, s_i)\}}$$

As opposed to variable ordering, the *value* ordering heuristic attempts to resolve the selected conflict (a_i, a_j) simply by choosing the precedence constraint that retains the highest amount of sequencing flexibility. Specifically, $a_i \preceq a_j$ is selected if $bd_{ij} > bd_{ji}$ and $a_j \preceq a_i$ is selected otherwise.

4.1 The PCP Algorithm

Figure 3 gives the basic overall PCP solution procedure, which starts from an empty solution (Step 1) where the graphs G_d is initialized according to Section 3. Also, the

[4] The $\sqrt{S}$ bias is introduced to take into account cases where a first conflict with the overall $min\{slack(e_i, s_j), slack(e_j, s_i)\}$ has a very large $max\{slack(e_i, s_j), slack(e_j, s_i)\}$, and a second conflict has two shortest path values just slightly larger than this overall minimum. In such situations, it is not clear which conflict has the least sequencing flexibility.

```
PCP(Problem, V_max)
1.  S ← InitSolution(Problem, V_max)
2.  loop
3.   Propagate(S)
4.   if UnresolvableConflict(S)
5.    then return(nil)
6.    else
7.     if UniquelyResolvableConflict(S)
8.       then PostUnconditionalConstraints(S)
9.       else begin
10.        C ←ChooseResolvableConflict(S)
11.        if (C = nil)
12.         then return(S)
13.         else begin
14.          Prec ← ChoosePrecConstraint(S, C)
15.          PostConstraint(S, Prec)
16.         end
17.      end
18. end-loop
19. return(S)
```

Fig. 3 Basic PCP algorithm

procedure accepts a *never-exceed* value (V_{max}) of the objective function of interest. If $V_{max} = L_{max}$ (maximum lateness), this value is used to initially impose a maximum completion time $C_k = d_k + L_{max}$ to each job J_k; in case $V_{max} = C_{max}$ (makespan), the parameter is used to impose an initial *global* makespan to all the jobs.

The PCP algorithm shown in Figure 3 analyses all pairs (a_i, a_j) of activities that require the same resource (i.e., the *decision variables* of the corresponding CSP problem), and decides their *values* in terms of precedence ordering (i.e., $a_i \preceq a_j$ or $a_j \preceq a_i$, see Section 3), on the basis of the response provided by the *dominance conditions*.

In broad terms, the procedure in Figure 3 interleaves the application of dominance conditions (Steps 4 and 7) with variable and value ordering (Steps 10 and 14 respectively) and updating of the solution graph G_d (Steps 8 and 15) to conduct a single pass through the search tree. At each cycle, a propagation step is performed (Step 3) by the function `Propagate`(S), which propagates the effects of posting a new solving decision (i.e., a constraint) in the graph G_d. In particular, `Propagate`(S) updates the shortest path distances on the graph G_d. We observe that within the main loop of the procedure PCP shown in Figure 3 new constraints are added incrementally (one-by-one) to G_d, hence the complexity of this step [5] is in the worst case $O(n^2)$.

A solution is found when the PCP algorithm finds a feasible assignment to the activity start times such that all resource conflicts are resolved (i.e., all the setup times st_{ij} are satisfied), according to the following proposition:

Proposition 1 *A solution S is found when none of the four dominance conditions is verified on S.*

[5] Let us suppose we have a consistent G_d, in the case we add a new edge (tp_x, tp_y) with weight w_{xy}, if $w_{xy} + d(tp_y, tp_x) \geq 0$ (G_d remains consistent, because no negative cycles are added, see Section 3), then the generic shortest path distance can be updated as $d(tp_i, tp_j) = min\{d(tp_i, tp_j), d(tp_i, tp_x) + w_{xy} + d(tp_y, tp_j).\}$

The previous assertion can be demonstrated *by contradiction*. Let us suppose that the PCP procedure exits with success (none of the four dominance conditions is verified on S) and that at least two sequential activities a_i and a_j, requiring the same resource r do not satisfy the setup constraints $e_i + st^r_{ij} \leq s_j$ or $e_j + st^r_{ji} \leq s_i$. Since the *triangle inequality* holds for the input problem, it is guaranteed that the length of the *direct* setup transition $a_i \preceq a_j$ between two generic activities a_i and a_j is the shortest possible (i.e., no *indirect* transition $a_i \rightsquigarrow a_k \rightsquigarrow a_j$ having a shorter overall length can exist). This fact is relevant for the PCP approach, because the solving algorithm proceeds by checking/imposing either the constraint $e_i + st^r_{ij} \leq s_j$ or the constraint $e_j + st^r_{ji} \leq s_i$ for each pair of activities. Hence, when none of the four dominance conditions is verified, each subset of activities A^r requiring the same resource r is totally ordered over time. Clearly, for each pair (a_i, a_j), such that $a_i, a_j \in A^r$, either *co-slack*$(e_i, s_j) \geq 0$ or *co-slack*$(e_j, s_i) \geq 0$; hence, all pairs of activities (a_i, a_j) requiring the same resource r *satisfy* the setup constraints $e_i + st^r_{ij} \leq s_j$ or $e_j + st^r_{ji} \leq s_i$. In fact, by definition *co-slack*$(e_i, s_j) \geq 0$ implies $-d(s_j, e_i) \geq st^r_{ij}$ and together the condition $s_j - e_i \geq -d(s_j, e_i)$ (which holds because G_d is consistent, see Section 3), we have $e_i + st^r_{ij} \leq s_j$ (a similar proof is given for *co–slack*$(e_j, s_i) \geq 0$).

To wrap up, when none of the four dominance conditions is verified, and the PCP procedure exits with success, the G_d graph represents a consistent Simple Temporal Problem and, as described in Section 3, one possible solution of the problem is the so-called *earliest-time solution*, such that $S_{est} = \{S_i = -d(tp_i, tp_0) : i = 1 \ldots n\}$.

5 An Iterative Sampling Procedure

The PCP resolution procedure, as defined above, is a deterministic (partial) solution procedure with no recourse in the event that an unresolved conflict is encountered. To provide a capability for expanding the search in such cases without incurring the combinatorial overhead of a conventional backtracking search, in the following two subsections we define:

1. a random counterpart of our conflict selection heuristic (in the style of Oddi and Smith (1997)), providing stochastic variable and value ordering heuristics;
2. an iterative sampling search framework for optimization embedding the stochastic procedure.

This choice is motivated by the observation that in many cases systematic backtracking search can explore large sub-trees without finding a solution (this behavior is also known as *thrashing* in the Constraint Proramming area, see Cambazard et al (2008)). On the other hand, if we compare the whole search tree created by a systematic search algorithm with the non systematic tree explored by repeatedly restarting a randomized search algorithm, we see that the randomized procedure is able to reach "different and distant" leaves in the search tree. The latter property could be an advantage when problem solutions are uniformly distributed within the set of search tree leaves interleaved with large sub-trees which do not contain any problem solution.

5.1 Stochastic Variable and Value Ordering

The stochastic versions of PCP's variable and value ordering heuristics follows from the simple intuition that it generally makes more sense to follow a heuristic's advice when

the heuristic clearly distinguishes one alternative as superior, while it makes less sense to follow its advice when several choices are judged to be equally good.

Let us consider first the case of *variable ordering*. As previously discussed, PCP's variable ordering heuristic selects the conflict (a_i, a_j) with the overall minimum value of $VarEval(a_i, a_j) = min\{bd_{ij}, bd_{ji}\}$. If $VarEval(a_i, a_j)$ is $<<$ than $VarEval(a_k, a_l)$ for all other pending conflicts (a_k, a_l), then the conflict (a_i, a_j) is clearly recognized as the one to be selected. However, if more $VarEval(a_k, a_l)$ values are instead quite "close" to $VarEval(a_i, a_j)$, then the choice to be preferred is not so clear and the selection of any of these conflicts may be reasonable. We formalize this notion by defining an *acceptance band* β with respect to the set of pending resolvable conflicts and expanding the PCP *ChooseResolvable-Conflict* routine in the following three steps:

1. Calculate the overall minimum value
$$\omega = min\{VarEval(a_i, a_j)\}$$
2. Determine the subset SC of resolvable conflicts
$$SC = \{(a_i, a_j) : \omega \leq VarEval((a_i, a_j)) \leq \omega(1 + \beta)\}$$
3. Randomly select a conflict (a_i, a_j) in the set SC.

Thus, β defines a range around the minimum (i.e., best) heuristic evaluation within which all differences in value are assumed to be insignificant and non-informative. The smaller the value of β, the higher the assumed discriminatory power of the heuristic.

A similar approach can be chosen for *value ordering* decisions. Let $pc(a_i, a_j)$ be the deterministic value ordering heuristic used by PCP. As previously noted, $pc(a_i, a_j) = a_i \preceq a_j$ when $bd_{ij} > bd_{ji}$, and $a_j \preceq a_i$ otherwise. Recalling the definition of the biased distance bd, in cases where $S = \frac{min\{slack(e_i,s_j),slack(e_j,s_i)\}}{max\{slack(e_i,s_j),slack(e_j,s_i)\}}$ is ≈ 1, and hence bd_{ij} and bd_{ji} are almost equal, $pc(a_i, a_j)$ does not yield a clear guidance (both choices appear equally good). Therefore, we define the following randomized version of *ChoosePrec-Constraint*:

$$rpc(a_i, a_j) = \begin{cases} \overline{pc(a_i, a_j)} : U[0, 1] + \alpha < S \\ pc(a_i, a_j) : otherwise \end{cases}$$

where α represents a threshold parameter, $U[0, 1]$ represents a random value in the interval $[0, 1]$ with uniform distribution function, and $\overline{pc(a_i, a_j)}$ is the complement of the choice advocated by $pc(a_i, a_j)$. Under this random selection method, it is simple to demonstrate that the probability of deviating from the choice of PCP's original value ordering heuristic pc is $(S - \alpha)$ when $S \geq \alpha$ and 0 otherwise. If α is set to 0.5, then each ordering choice can be seen to be equally likely in the case where $S = 1$ (i.e., the case where the heuristic gives the least information).

5.2 The Optimization Algorithm

Figure 4 depicts the complete iterative sampling algorithm for generating near-optimal solutions to SDST-JSSP/MAX instances. It is designed to invoke the random version of the PCP resolution procedure a fixed number ($MaxRestart$) of times, such that each restart provides a new opportunity to produce a different feasible solution with lower V_{max}. In this optimization version we adopt a multi-pass approach; the current best value of the solution quality metric of interest V_{max}^{best} is retained and re-applied as a parameter of the PCP procedure to solve problems with increasingly tighter constraints on the jobs (Steps 5-13).

ISP($Problem$, $V_{max}^{(0)}$, $MaxRestart$)
1. $S \leftarrow$ EmptySolution($Problem$, $V_{max}^{(0)}$)
2. $S_{best} \leftarrow S$
3. $V_{max}^{best} \leftarrow V_{max}^{(0)}$
4. $count \leftarrow 0$
5. **while** ($count \leq MaxRestart$) **do begin**
6. $S \leftarrow$ PCP($Problem$, V_{max}^{best})
7. **if** ($V_{max}(S) < V_{max}^{best}$)
8. **then begin**
9. $S_{best} \leftarrow S$
10. $V_{max}^{best} \leftarrow V_{max}(S)$
11. **end**
12. $count \leftarrow count + 1$
13. **end-while**
14. **return**(S_{best})

Fig. 4 Iterative sampling algorithm

6 Experimental Analysis

In this section we propose a set of empirical evaluations of the ISP algorithm shown in Figure 4. We remark that in the tested version of the ISP algorithm we use a uniform distribution probability to implement both random variable and value ordering heuristics [6]. The ISP algorithm has been implemented in CMU Common Lisp Ver. 20a and run on a AMD Phenom II X4 Quad 3.5 Ghz under Linux Ubuntu 10.04.1.

We observe that, to the best of our knowledge, there are no standard benchmarks available for the full version of the SDST-JSSP/MAX problems (i.e., SDST-JSSP problems that additionally include *min/max* separation constraints). For this reason, we first consider some classical benchmarks of SDST-JSSP described in literature and available on the Internet. Such benchmarks represent a relaxed version of the SDST-JSSP/MAX problem introduced in the first part of the paper. Later, we will define a new benchmark set for the full SDST-JSSP/MAX, and for this new benchmark a first set of experimental results will be provided.

Since we are proposing a heuristic algorithm that incorporates two basic parameters - α and β, we face the problem of *tuning* our heuristic approach. We use the racing procedure *F-Race* proposed in Birattari et al (2002) to find a configuration of the algorithm that performs as well as possible on a given instance class of a combinatorial optimization problem. This method takes inspiration from the Machine Learning (ML) literature for model selection through cross-validation. The *F-Race* procedure empirically evaluates a set of candidate configurations by discarding bad ones as soon as *statistically sufficient evidence* has been gathered against them. The advantage in using such approach for tuning parameters is twofold. First, *F-Race* is able to quickly reduce the number of candidates, and narrow focus to the most promising ones. Second, this methodology allows comparison of results on different instances independently of the

[6] In particular, for variable ordering we select at random one element in the set $SC = \{(a_i, a_j) : \omega \leq VarEval((a_i, a_j)) \leq \omega(1+\beta)\}$ with uniform distribution probability (see Section 5.1). About random value ordering, we exchange the precedence constraint determined via the *deterministic* value ordering heuristic on the basis of the test $U[0,1] + \alpha < S$, where $U[0,1]$ represents a random value in the interval $[0,1]$ with uniform distribution (see Section 5.1).

inherent difference in the cost functions. More specifically in our analysis we perform a *F-Race* selection based on Friedman analysis of variance by ranks as implemented in the *Race package* (Birattari (2010)) available for the *R statistical software* - R Development Core Team (2010).

6.1 Empirical Analysis on Existing Benchmarks

As indicated above, we start in this section by analyzing the performance of the ISP algorithm on selected SDST-JSSP benchmark sets that can be found in literature. This allows us to test our algorithm against the best results currently available, in order to get an idea of its effectiveness when compared with different existing solving approaches. In this way, we intend to provide an added value to the subsequent analysis of the algorithm's performance on problem instances of the full SDST-JSSP/MAX (see Section 6.3), for which, we have no competitors as yet.

Our analysis on SDST-JSSP benchmarks proceeds as follows. In Section 6.1.1 the ISP algorithm is employed to solve the SDST-JSSP benchmark proposed in Ovacik and Uzsoy (1994); Demirkol et al (1998). In this case, the objective is minimization of *maximum lateness* (L_{max}). We perform a series of both explorative and selective experimental runs that vary the algorithm's randomization parameters α and β. Section 6.1.2 then is dedicated to analyzing the performance of our algorithm on a second SDST-JSSP benchmark originally proposed in Brucker and Thiele (1996). In this case, the objective is to minimize the *makespan* (C_{max}). We again perform a series of both explorative and selective experimental runs using different α and β randomization parameters.

6.1.1 L_{max} minimization

The first benchmarks we consider are proposed in Demirkol et al (1998); Ovacik and Uzsoy (1994), and are available at `http://pst.istc.cnr.it/~angelo/OUdata/`. In order to comprehensively interpret the experimental results, it is necessary to provide a brief description of how these benchmarks were originally produced. In all benchmark instances, the setup times st_{ij}^r and the processing times p_i of every activity a_i are values randomly computed in the interval $[1, 200]$. The job due dates d_i are assumed to be uniformly distributed over an interval I characterized by the following two parameters: (1) the mean value $\mu = (1 - \tau)E[C_{max}]$, where τ denotes the percentage of jobs that are expected to be tardy and $E[C_{max}]$ is the expected makespan[7], and (2) the R value, which determines the range of I, whose bounds are defined by: $[\mu(1-R/2), \mu(1+R/2)]$. All benchmark instances are calculated using the τ values of 0.3 and 0.6, corresponding to loose and tight due dates respectively, and the R values of 0.5, 1.5 and 2.5, respectively modeling different due date variation levels. The particular combination of the τ and R values allows us to categorize all instances into six different benchmarks, namely: $i305$, $i315$, $i325$, $i605$, $i615$, $i625$. Each benchmark contains 160 randomly generated problem instances, divided in subclasses based on the numbers of machines and jobs involved; more precisely, all instances are synthesized by choosing 10 and 20 jobs on 5, 10, 15 and 20 machines, yielding a total of 8 subclasses for each benchmark.

[7] Calculated by estimating the total setup and processing time required by all jobs at all machines and dividing the result by the number of available machines.

One implication of this procedure for generating problem instances is that this benchmark does not satisfy the *triangle inequality*, as all setup times and activity durations are computed in the interval $[1, 200]$ *at random*. This fact has important consequences on our Iterative Sampling Algorithm (see Figure 4), as it may cause the algorithm to disregard a number of valid solutions due to constraint overcommitment. As indicated earlier, the triangle inequality condition guarantees that the length of the *direct* setup transition $a_i \preceq a_j$ between two generic activities a_i and a_j is the shortest possible and it is not possible to find an *indirect* transition (a sequence of activities $a_i \rightsquigarrow a_k \rightsquigarrow a_j$) which has a shorter overall length. The PCP approach relies on this property, as the solving algorithm proceeds by checking/imposing either the constraint $e_i + st^r_{ij} \leq s_j$ or the constraint $e_j + st^r_{ji} \leq s_i$ for any given pair of activities competing for the same resource. Hence, if the triangular inequality does not hold, the procedure might post over-committing constraints during the solving process, which (a) increases the probability of finding sub-optimal solutions and (b) introduces the possibility that some solutions may be disregarded.

To illustrate the potential problem due to over-commitment, consider three activities a_1, a_2 and a_3 requiring the same resource, with processing times $p_1 = p_2 = p_3 = 1$ and setup times $st_{12} = st_{21} = 15$, $st_{13} = st_{31} = 3$ and $st_{23} = st_{32} = 3$. Suppose also that the overall scheduling horizon is limited to 10. Under these conditions, the triangle inequality for the setup times is clearly not satisfied, and our PCP procedure is prone to failure. Specifically, the first *dominance condition* is verified for the activity pair $\langle a_1, a_2 \rangle$ (detecting an unresolvable conflict), despite the fact that the solution $a_1 \preceq a_3 \preceq a_2$ does exist. In fact, the algorithm will try to impose one of the two setup constraints $st_{12} = st_{21} = 15$, but both are in conflict with the horizon constraint $(15 > 10)$.

This characteristic of the benchmark definitely puts the ISP algorithm at a disadvantage. However, a straightforward probabilistic computation allows us to easily determine the probability that the triangular inequality will not be satisfied by a given triple of activities; and this value is as low as 4.04%. This suggests that use of the triangular inequality assumption can still be considered as viable, as it is in fact satisfied in the vast majority of the cases. Indeed, this explains the globally strong performance of the algorithm on these instances (see below).

Despite the low probability of the triangular inequality being violated, we opted to apply a post-processing step similar to the *Chaining* of Policella et al (2007) to the solution S generated by the PCP algorithm in Figure 3, to eliminate any constraint overcommitments that may have been introduced, and thus improve the solution quality by left-shifting some of the jobs. This post-processing phase is accomplished in two steps. First, all previously posted ordering constraints are removed from the solution S. Second, for each resource and for each activity a_i (in increasing order of the determined sequence in which activities use the resource), the unique successor a_j is considered, and the precedence constraints $e_i + st^r_{ij} \leq s_j$ is posted. This last step is iterated until all activities are linked by the correct sequence dependent setup times provided in the problem definition. Then, a new earliest start time solution is recomputed on the basis of the newly imposed constraints via shortest path computation on the graph G_d representation. We observe that the set of imposed constraints surely represents a solution of the problem because this new set of constraints is less restrictive than the previous one.

In what follows we perform a two-step analysis. An initial, broad analysis of the independent effects of random value ordering and random variable ordering configu-

rations is first carried out on all six benchmark sets ($i305$, $i315$, $i325$, $i605$, $i615$, and $i625$) with different computational time limits. Next, a second, more fine-grained analysis of the effects of the α (random value ordering) and β (random variable ordering) parameters is performed using the above cited *F-Race* technique on the two benchmark sets for which the ISP algorithm achieved its worst and best performance respectively in the initial analysis step.

Table 1 Summary of the experimental results for the α and β randomization for different CPU time limits. In each column, the table shows the values Δ^{avg} and among square brackets, the number of improved solutions. The *Bests* column refers to the best solutions found over the three runs.

α, β	**Set**	200 (secs)	400 (secs)	800 (secs)	Bests
$\alpha = 0.5$ $\beta = 0$	i305	92.7 [38]	79.0 [35]	74.8 [32]	63.3 [45]
	i315	24.7 [36]	1.5 [51]	-7.0 [59]	-9.0 [70]
	i325	20.7 [24]	4.1 [37]	0.8 [53]	0.3 [55]
	i605	24.5 [29]	15.3 [28]	11.07 [29]	10.0 [40]
	i615	22.1 [33]	11.3 [37]	6.3 [38]	5.4 [48]
	i625	19.1 [48]	6.1 [60]	1.2 [66]	0.7 [77]
$\alpha = 0$ $\beta = 0.2$	i305	53.3 [34]	52.7 [45]	48.3 [46]	39.6 [53]
	i315	26.2 [28]	10.7 [38]	-2.8 [51]	-8.3 [60]
	i325	12.8 [21]	6.7 [25]	4.0 [39]	1.9 [44]
	i605	8.4 [37]	8.3 [37]	7.0 [35]	5.9 [47]
	i615	8.0 [36]	6.0 [42]	4.8 [43]	3.4 [52]
	i625	8.6 [31]	5.8 [38]	2.6 [47]	1.3 [59]

A Preliminary and Broader Analysis. The results of the first step of the analysis are shown in Table 1, where the first column indicates the two sets of α (random value ordering) and β (random variable ordering) parameter values evaluated. For every benchmark set (second column) three complete runs have been performed with increasing CPU time limit, 200, 400 and 800 seconds, where such limit is the maximum CPU time that the scheduling procedure can use to find a solution. In addition, a common value for $MaxRestart = 1000$ is imposed on all the runs. In each complete run, we measure (1) the Δ^{avg} average percentage deviation[8] from the results in Balas et al (2008), and (2) the number of improved solutions with respect to Balas et al (2008) (in square brackets). The *Bests* column refers to the best solutions found over the three runs. It is worth noting that the results in Balas et al (2008) are not the current best known, yet we use these as reference because they represent the only complete set of published results on all 960 instances contained in the sets $i305$, $i315$, $i325$, $i605$, $i615$, and $i625$. The full set of results is available at `http://www.andrew.cmu.edu/user/neils/tsp/t2ps/`, there is a table for each benchmark set ($i305$, $i315$, $i325$, $i605$, $i615$, and $i625$) showing both the L_{max} values (for two different settings of the algorithm) and the correlated CPU times. Such values range from few seconds, for the instances with the smallest

[8] The value Δ^{avg} is the average over the set of values $100 \times \frac{L^{ISP}_{max} - L^{B}_{max}}{|L^{B}_{max}|}$, where L^{B}_{max} is the best results from Balas et al (2008) and L^{ISP}_{max} is the best results obtained with the algorithm ISP shown in Figure 3. In the particular case where $L^{B}_{max} = 0$, we consider the value $100 \times L^{ISP}_{max}$

size, to about one hundred seconds for the largest ones. More recent results representing the current best are published in González et al (2009a) for the $i305$ set only. The authors of González et al (2009a) have also kindly provided us with (as yet unreleased) results extended to all the benchmark sets [González et al (2009b)]. These results are summarized in the $GVV09$ column of Table 2.

Returning to Table 1, the top half indicates the results obtained when randomization is restricted to value ordering decisions and variable ordering decisions are made deterministically (i.e., $\alpha = 0.5$ and $\beta = 0$). Even though this analysis represents an initial baseline configuration, the results are nonetheless interesting as this configuration of the algorithm finds a considerable number of improved solutions in all cases. The best performance seems to involve the benchmarks associated with higher values of the R parameter; as the table shows, the outcomes are much more convincing when R is greater or equal than 1.5, i.e., when the level of the due date variation is higher. One possible explanation for this behavior is the following. As the value of R increases, the due dates of jobs are randomly chosen from a wider set of uniformly distributed values; this implies that, among all generated due dates, there will be a subset that are particularly tight (i.e., the earliest deadlines). Given the design of the PCP scheduling procedure (see Figure 3), solutions are found by imposing the deadlines of the most "critical" jobs (i.e., the jobs characterized by the earliest deadlines)[9]. In other words, the procedure *naturally* proceeds by first accommodating the most critical jobs, by imposing "hard" deadline constraints, and then secondly addressing the "easier" task of accommodating the remaining jobs. On the contrary, when the R values are lower, all generated due dates tend to be critical, as all their values are comparable and hence indistinguishable. This circumstance may represent an obstacle to good performance in the current version of the procedure, in that low-lateness scheduling for all jobs by means of imposing hard constraints cannot be always guaranteed. As we are running a random solving procedure, we finally observe that the overall results can be improved by considering the best solutions over the set of the performed runs. In Table 1, the column labelled *Bests* shows a significant performance improvement in comparison to the three individual runs.

The bottom half of Table 1 shows the results for random variable ordering (i.e., $\alpha = 0$ and $\beta = 0.2$). As in the previous case, the best performance seems to involve the benchmarks associated with higher values of R (1.5 and 2.5). However, there are also some differences. The most evident is that the results obtained with random variable ordering are in some sense complementary with the results for random value ordering. In fact, Table 1 shows that the best results for the $i305$ and $i605$ benchmarks ($R = 0.5$) are obtained in the random *variable* ordering case (lower half of the table), while the best results for the $i315$, $i325$, $i615$ and $i625$ benchmarks ($R = 1.5$ and $R = 2.5$), are obtained in the random *value* ordering case (upper half of the table).

In Table 2 we summarize the best results obtained using both the random value randomization and the random variable randomization procedures (the two columns labeled Bests(α) and Bests(β)), as well as the best overall performance (the column labeled BESTS). Observing the best overall performance, we see that our algorithm yields significant improvement in terms of number of improved solutions; in fact, we are able to find better solutions for 396 of the 960 total instances in the overall benchmark. Moreover, we also observe very good performance in average percentage deviation

[9] Due to the "most constrained first" approach used in the PCP procedure on conflict selection, line 9.

(Δ^{avg}), globally improving this measure for 3 of the 6 benchmark sets (see the figures in bold).

Table 2 Summary of the main best results obtained from the α and β randomization.

Set	Bests(α)	Bests(β)	BESTS	GVV09
i305	63.3 [45]	39.6 [53]	38.5 [56]	−56.2[155]
i315	-9.0 [70]	-8.3 [60]	**-12.9 [75]**	−25.9[134]
i325	0.3 [55]	1.9 [44]	**-0.1 [61]**	−1.7[98]
i605	10.0 [40]	5.9 [47]	5.6 [52]	−7.8[154]
i615	5.4 [48]	3.4 [52]	2.8 [64]	−7.5[152]
i625	0.7 [77]	1.3 [59]	**-0.2 [88]**	−4.1[144]

As indicated above, the algorithm proposed in González et al (2009a) provides better results for the $i305$ benchmark set, and the authors have provided us with unofficial performance results for the remaining benchmark sets González et al (2009b). These results are reported in Table 2 in the column named GVV09 according to the best L_{max} values obtained imposing the same CPU limits reported at `http://www.andrew.cmu.edu/user/neils/tsp/t2ps/` for each single problem instance. These results surpass the improvements achieved by our ISP algorithm to the results reported in Balas et al (2008). The $i305$ set is solved down to $-56.2[155]$, while the $i315$ set is solved with a score of $-25.9[134]$. As for the remaining sets $i325$, $i605$, $i615$ and $i625$, the final scores are close to $-1.7[98]$, $-7.8[154]$, $-7.5[152]$ and $-4.1[144]$, respectively. In the case of the $i305$ set, our procedure is significantly outperformed. In the five remaining sets however, the difference in performance is less evident, which demonstrates the overall validity of our Precedence Constraint Posting meta-heuristic approach. As we will analyze in more detail later (see Section 6.2), the procedure developed in González et al (2009a) is ad-hoc and specifically devised to tackle SDST-JSSP instances, whereas our procedure is a much more general solving procedure.

Analysis for the Parameters α and β. Given the above results, a second set of experiments were performed to conduct a more thorough analysis of the effects of the α (random value ordering) and β (random variable ordering) parameters on algorithm performance. In this experimental phase, attention was restricted to the $i305$ and $i315$ sets where, in the first phase, the ISP algorithm obtains its worst and the best performance, respectively. A CPU time limit of 800 seconds was imposed throughout along with a common value for $MaxRestart = 10000$ for all the runs. We performed a *F-Race* selection (see Birattari et al (2002)) based on Friedman analysis of variance by ranks as implemented in the Race package Birattari (2010) available for the *R statistical software* - R Development Core Team (2010). At each step of the selection procedure each solver configuration was tested on one instance (among the available ones) and assigned a rank according to the value of the cost function. Exploiting the fact that this methodology allows comparison of results on different instances independently of the inherent difference in the cost functions, a given solver configuration can be discarded as soon as statistical evidence is found that it is inferior to the others, and the comparison then proceeds with the set of solver configurations that remain. We set the confidence level employed in the *F-Race* procedure at 95%, the first test was set to be performed after 10 steps (instances).

This phase of the analysis was carried out in two steps. In the first step, for each of the two sets $i305$ and $i315$ we selected 20 instances (in particular the instances 21-40) and performed a *F-Race* selection of the effects on performance of various α and β parameter configurations. In the second step, we ran a selective analysis for the values α and β that yielded the best results in the first step, and compared the results obtained to the best known results available in the literature.

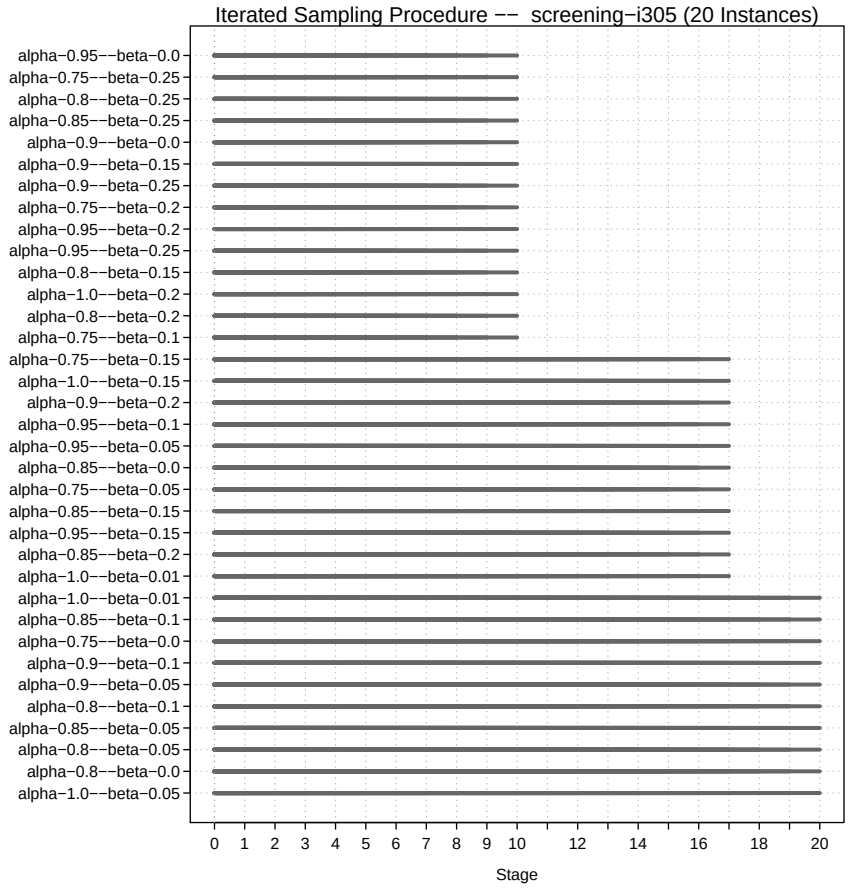

Fig. 5 Benchmark $i305$: *F-Race* analysis for the instances 21-40, the graphic shows the *surviving* pairs of α and β values as the number of tested instances increases.

Figure 5 shows the F-Race analysis for the instances 21-40, the graphic shows the *surviving* pairs of α and β values as the number of tested instances (the Stage value) increases. Figure 5 shows that the best performance are obtained for small values of β and values of α close to 1.0. Hence a lower degree of randomization gives the best performance. We observe that the previous analysis does not generate a single best; rather, a subset of α and β values survived the race (see Figure 5). In addition, if we consider the graphic of Figure 6, showing the boxplots of the solutions generated during the race depicted in Figure 5, it shows a large variance in L_{max} values. This

behavior complicates the ability to make meaningful comparisons of the performance of different solvers.

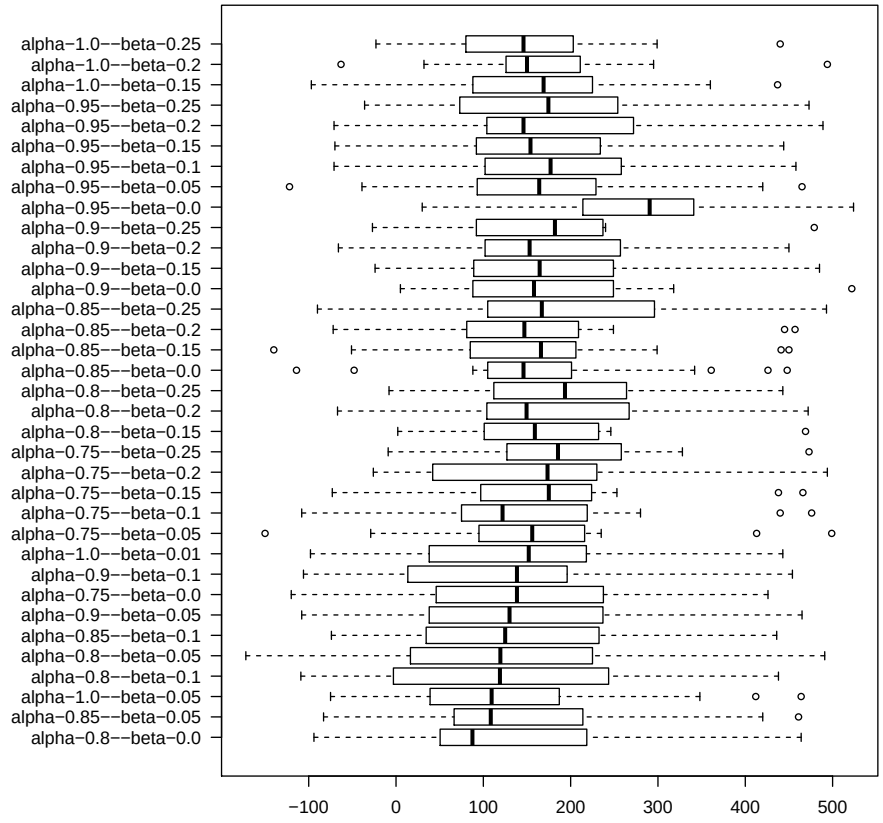

Fig. 6 Benchmark $i305$: boxplots corresponding to the *F-Race* analysis of Figure 5, L_{max} in shown on the x-axis

In accordance with the two-step analysis mentioned above, we present the results for the full benchmark set $i305$ (160 instances) in Table 3 . In this table, the N_i column represents the number of improved results with respect to Balas et al (2008), while the AL_{max} column contains the average lateness over the 160 instances. As before, we compare our results with both the results in Balas et al (2008), and those provided in González et al (2009a) (included in the last row labeled GVV09). We observe that for the $i305$ benchmark set, a more accurate selection of the parameters α and β improves over the results of the ISP algorithm shown in Table 1 and Table 2. In particular, the number of improved solutions is seen to increase from $N_i = 56$ to $N_i = 79$, and the average deviation from best is decreased from $\Delta^{avg} = 38.5$ to $\Delta^{avg} = 7.9$. We observe however, that these results still remain significantly below the best known results from

Table 3 Benchmark $i305$: results for the full benchmark set (160 instances); the table shows the number of improved solutions N_i as well as the Δ^{avg} and AL_{max} values for different α and β values.

α	β	N_i	Δ^{avg}	AL_{max}
0.8	0.02	53	20.2	1303.7
	0.05	62	27.0	1296.4
	0.1	59	30.1	1294.3
0.85	0.02	45	21.6	1296.9
	0.03	55	18.3	1293.2
	0.05	56	22.9	1291.9
0.9	0.02	40	22.7	1309.2
	0.03	46	20.8	1299.2
0.95	0.02	27	27.3	1321.4
	0.03	38	25.1	1310.1
1.0	0.02	43	24.6	1309.9
	0.05	45	27.7	1298.1
BESTS	-	79	7.9	1244.3
GVV09	-	155	-56.2	1216.1

González et al (2009a). As we will see shortly, things improve for the $i315$ benchmark set.

Figure 7 shows a *F-Race* selection of the effects on performance of the α and β parameters on the set $i315$ for the instances 21-40. In contrast to benchmark $i305$, we observe that the best performance for the set $i315$ is obtained for larger values of β and smaller values of α. Hence the best performance on this benchmark set is obtained with a higher degree of randomization. Like in the previous case, Table 4 indicates the results for the the full benchmark set $i315$ (160 instances). Once more, we compare our results with both the results in Balas et al (2008), and those provided in González et al (2009a) (included in the last row labeled GVV09). We observe that relative to the $i315$ benchmark set, a more accurate selection of α and β significantly improves on the results of the ISP algorithm shown in Table 1 and Table 2. In particular, we increase the number of improved solutions from $N_i = 75$ to $N_i = 99$, the average deviation is decreased from $\Delta^{avg} = -12.9$ to $\Delta^{avg} = -17.28$. Moreover, these new results are quite close to the best known results from González et al (2009a); in fact, the comparative numbers of improved solutions are $N_i = 99$ versus $N_i = 134$, and the comparative average deviation values are $\Delta^{avg} = -17.28$ versus $\Delta^{avg} = -25.9$.

Table 4 Benchmark $i315$: results for the full benchmark set (160 instances); the table shows the number of improved solutions (N_i), as well as the Δ^{avg} and AL_{max} values for different α and β.

α	β	N_i	Δ^{avg}	AL_{max}
0.5	0.2	87	-13.61	1185.26
0.5	0.4	83	-14.20	1199.96
0.6	0.3	88	-14.50	1182.34
1.0	0.5	70	-11.34	1204.51
BESTS	-	99	-17.28	1169.86
GVV09	-	134	-25.9	-

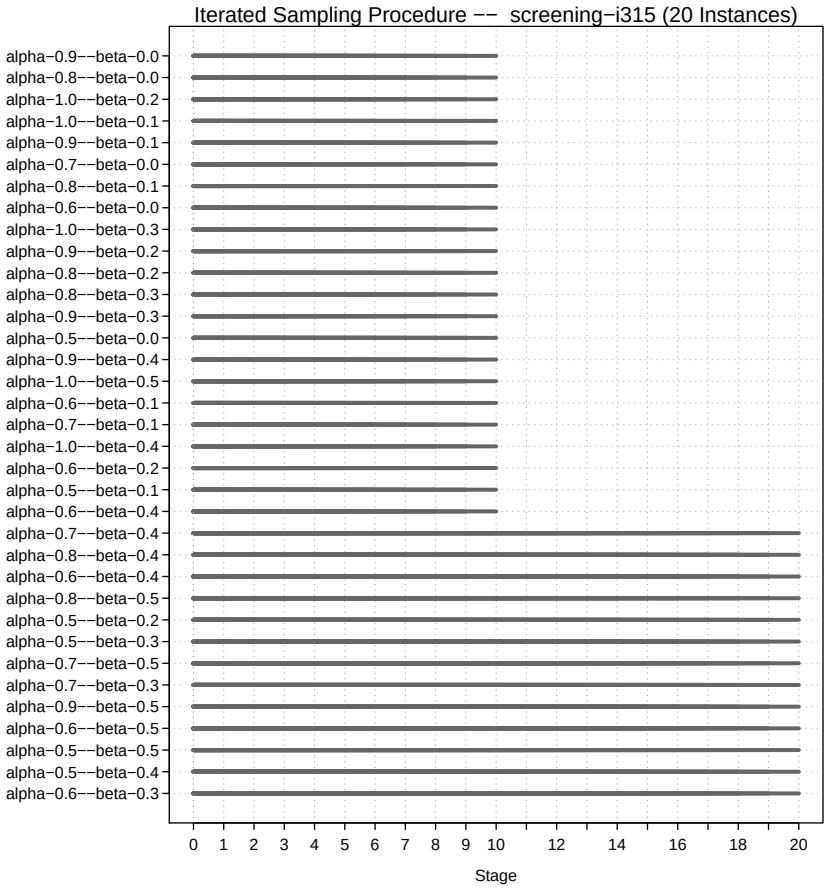

Fig. 7 Benchmark *i*315: *F-Race* analysis for the instances 21-40, the graphic shows the surviving α and β values as the number of tested instances increases.

6.1.2 C_{max} minimization

The second testset we considerd in our experiments with existing benchmarks is the set proposed in Brucker and Thiele (1996). This set is composed of 15 instances initially provided by Brucker and Thiele (1996) and later integrated with other 10 instances; they are available at `http://www.andrew.cmu.edu/user/neils/tsp/t2ps/`. Each instance is characterized by the configuration ($nJ \times nA$) where for every instance, nJ is the number of jobs present and nA is the number of activities per job. The original benchmark of 15 problems is divided into sets of 5 instances each, composed as follows: the first set contains 10×5 problems, the second set contains 15×5 problems, and the third set contains 20×5 problems. The 10 problems successively added are divided in two sets of 5 instances each: the first set contains 15×5 problems, while the second set contains 20×5 problems. Hence, our benchmark is therefore composed of 25 instances that range from 50 to 100 activities. In the following, this benchmark will be referred to as BTS25.

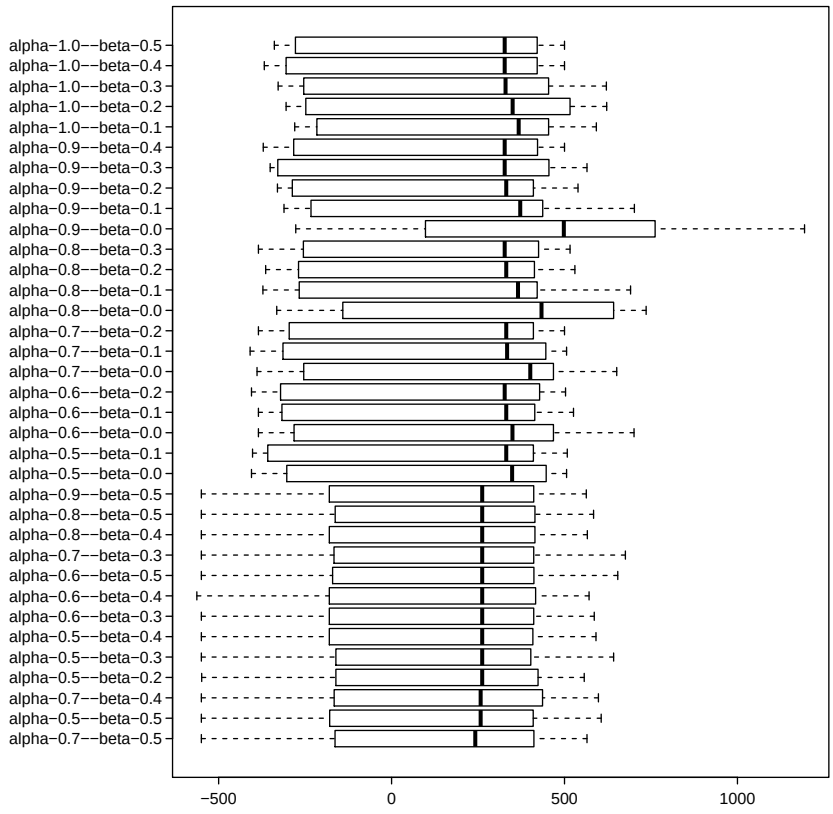

Fig. 8 Benchmark *i*315: boxplots corresponding to the *F-Race* analysis of Figure 7.

Conducting the same two-step analysis pattern, Figure 9 shows the results obtained from a wide *F-Race* based exploration of α and β on this benchmark set. As before, we set 95% as the confidence level employed in the *F-Race* procedure and the first test was performed after 10 steps (instances). With regard to the the explorative results shown in Figure 9, we note that the best performances is obtained for small values of β together with α values close to 1.0. In other words, the best performance is obtained in a neighborhood close to the point $(\alpha = 1, \beta = 0)$, which corresponds to the deterministic version (no randomization) of the PCP procedure. As for the previous cases, the *F-Race* analysis of Figure 5, does not generate a single best configuration, but rather a subset of ISP configurations. As was also the case earlier, if we consider the companion graphic of Figure 8 which shows the boxplots of the data of previous Figure 7, the large variance in C_{max} values (shown on the x-axis) partially explain the difficulty of making a comparison among the different solvers.

Table 5 shows the results of the selective run for the benchmark BTS25 based on the *F-Race* analysis of Figure 9. In this case, the parameter α spans over the values $0.7, 0.8, 0.9, 1.0$, while the parameter β spans over the values $0.0, 0.03, 0.05, 0.1$. The

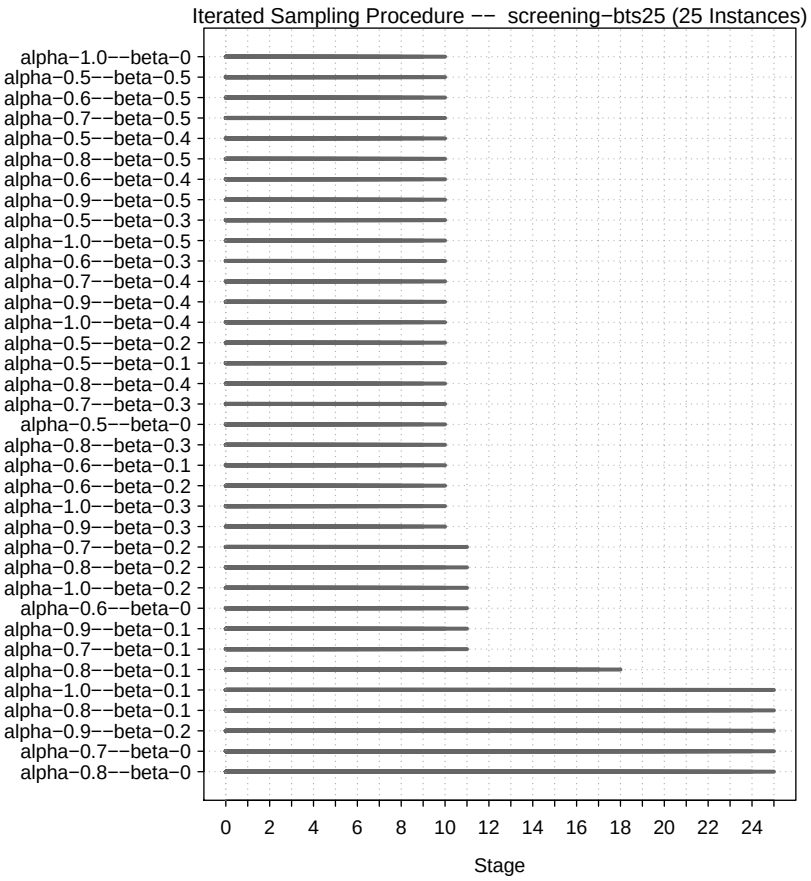

Fig. 9 *F-Race* analysis for the BTS25 instances: the graphic shows the remaining α and β values as the number of tested instances increases.

column labeled Δ^0 shows the average percentage variation from the infinite capacity makespan, while N_i represents the number of improved solutions with respect to the best known solutions for BTS25. For the first 15 BTS25 instances, the best known solutions are selected as the union of the best results proposed in the papers of Balas et al (2008), Vela et al (2009) and Artigues and Feillet (2008). For the last 10 instances, the best results are the ones proposed in Balas et al (2008). The column labeled Δ^{bests} contains the average percentage variation from the best solutions, while the last two columns N_i^{bsv} and Δ^{avg} respectively represent the number of improved solutions and the average percentage deviations from the best solutions proposed in Balas et al (2008). As shown, our best results are close to the best known results (see row BESTS); in particular, the average percentage deviation from the best known results is 2.11% and the average percentage deviation from the best results proposed in Balas et al (2008) is 1.54%. Finally, we note that the best known result for 1 problem (i.e., the 15×5 problem named `pss09`, 19th instance in the BTS25 benchmark) has been improved from an original best makespan value of $C_{max} = 1181$ to the new value of $C_{max} = 1170$.

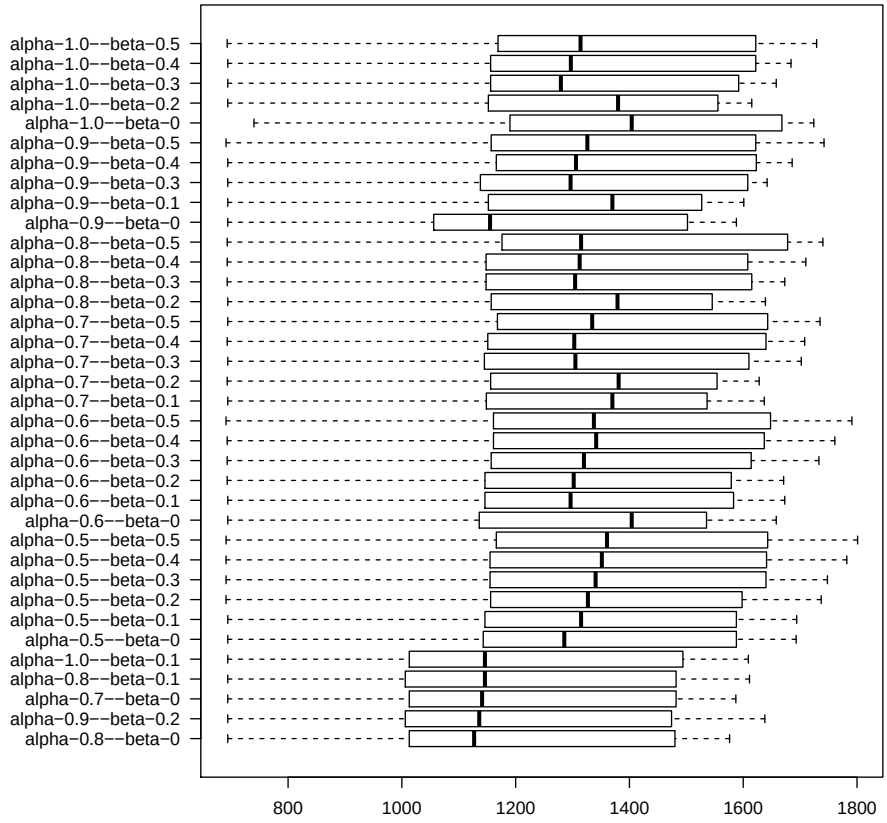

Fig. 10 Benchmark BTS25: boxplots corresponding to the F-Race analysis of Figure 9.

We conclude here the experimental analysis of existing SDST-JSSP benchmark sets. From this exploration we can draw the conclusion that on average our algorithm's performance is in line with that of the known best algorithms, and in some cases it is able to improve the best known results. In Section 6.3 we will introduce a new benchmark set on which we will perform a set of empirical evaluations for further comparisons, showing how our algorithm can be directly applied to solve an extended version of the problems just analyzed (i.e., the SDST-JSSP/MAX with min/max constraints). However, before proceeding to the next empirical evaluation, we first show in the next section that the the best performing algorithms for SDST-JSSP (i.e., the algorithm proposed in González et al (2009a) for L_{max} and the algortihm proposed in Vela et al (2009) for C_{max}), cannot be directly applied on SDST-JSSP/MAX instances.

Table 5 Benchmark BTS25: selective results for all the 25 instances. The table shows the average percentage variation from the infinite capacity makespan (Δ^0), the number of improved solutions (N_i), and the average percentage deviations from the best-known solutions for the BTS25 (Δ^{bests}), the number of improved solutions (N_i^{bsv}) and the average percentage deviations from the best solutions (Δ^{avg}) proposed in (Balas et al 2008), for different values of α and β.

α	β	Δ^0	N_i	Δ^{bests}	N_i^{bsv}	Δ^{avg}
0.7	0.03	174.14	1	3.65	2	3.07
	0.05	172.82	1	3.30	2	2.73
	0.1	173.91	1	3.64	2	3.06
0.8	0.03	172.74	1	3.21	2	2.64
	0.05	172.30	1	3.14	2	2.57
	0.1	171.89	1	2.98	2	2.41
0.9	0.03	173.02	1	3.36	2	2.79
	0.05	171.87	1	3.09	2	2.52
	0.1	171.62	1	2.92	2	2.35
1.0	0.03	172.40	1	3.13	2	2.56
	0.05	172.37	1	3.37	2	2.80
	0.1	172.44	1	3.38	2	2.81
BESTS	-	169.43	1	2.11	2	1.54

6.2 A Note on Algorithm Generality When Adding Time Windows

It is well known that scheduling problems with time windows constraints are not trivial problems. Generally, moving from a given base scheduling problem to a counterpart problem with generalized temporal constraints entails a significant increase in complexity. As we have seen in the previous section, the solution presented in González et al (2009a) performs very well against the previous SDST-JSSP benchmarks for L_{max}, basically implementing a hybrid procedure that integrates a genetic algorithm with tabu search techniques. In Vela et al (2009) a similar procedure is proposed for C_{max}. Key to the effectiveness of both procedures is an innovative tabu search neighborhood structure. This neighborhood structure is created by swapping pairs of activities (v, w) that belong to a *critical block*, defined by the authors as the maximal subsequence of at least two activities in the critical path that require the same resource. The feasibility of performing a given swap, as well as the efficiency with which this can be decided, is based on relevant properties that are demonstrated by means of two theorems introduced in González et al (2009a); these theorems are shown to be applicable on job shop scheduling problems without time windows. The question we answer in this section is whether the same theorems can be applied to the case of generalized temporal constraints. In particular, the first theorem (Theorem 1 in González et al (2009a)) provides a sufficient condition for allowing swap operations between v and w by demonstrating the non-existence of an alternative path between the same activities (of course, the existence of a path $v \rightsquigarrow w$ would imply that it is not possible to swap v and w). In order to evaluate this sufficiency condition, the theorem reasons on early start time solutions, and hence guarantees that the condition can always be evaluated in constant time. This last circumstance represents a great advantage for the overall algorithm, as it enables extremely fast computations of the neighborhood structures. On the other hand, all algorithms based on the distance graph (G_d) data structure (see

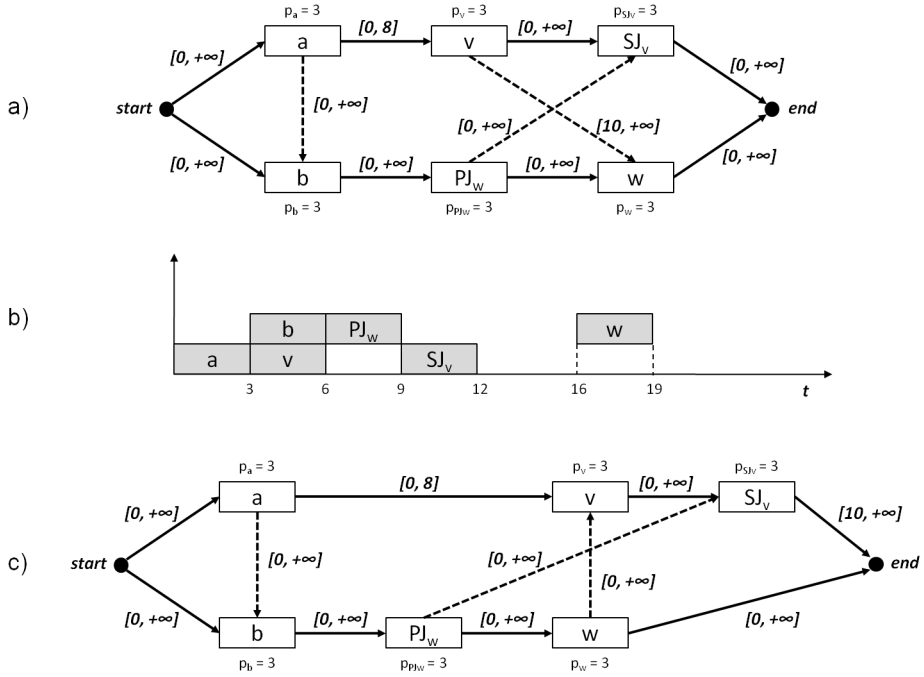

Fig. 11 a) a SDST-JSSP/MAX instance with generalized time constraints (note the time window $[0, 8]$ between a and v); b) a possible solution for the instance in a); c) the graph representation of a different (temporally infeasible) solution for the instance in a).

Section 3) are notoriously subject to heavy propagations to obtain causal and temporal information.

However, the example in Figure 11 shows that this efficiency comes at a price, *i.e., the inability to tackle problem instances characterized by maximum temporal constraints*. Part a) of Figure 11 represents a SDST-JSSP/MAX instance where minimum and maximum time lags $[min, max]$ can exist between activities of the same job (see the labels on the continuous arrows), and where the dashed arrows represent the setup constraints currently imposed on the solution[10]. Part b) of the same figure shows a possible solution that satisfies all the problem constraints, as can be easily verified by visual inspection. It is straightforward to prove that in this solution, the condition establishing the non-existence of an alternative path between the activities v and w as provided by the theorem is verified (it is enough to observe that the start time of w's job predecessor PJ_w is smaller than the end time of v's job successor SJ_v). As a consequence, according to the theorem it is possible to swap v with w (i.e., building an alternative solution where $w \preceq v$). But in fact, this is impossible in the current situation; As shown in part c) of Figure 11, sequencing v after w leads to an inconsistency caused by the presence of a maximum temporal constraint between activities v and a (i.e., the path $a \rightsquigarrow b \rightsquigarrow PJ_w \rightsquigarrow w \rightsquigarrow v$ has length $= 9$, exceeding the maximum distance of 8 that is allowed between a's end time and v's start time). The existence of an

[10] As required by the SDST-JSSP definition provided in Section 2, all pairs of activities that are connected by setup constraints share the same resource.

alternative path between v and w escaped the theorem's analysis because such paths are only visible if the G_d data structure is employed. Employing this data structure would however increase the computational burden of the procedure, and impact its efficiency. However, without an empirical study on the adapted procedure employing the G_d data structure, we cannot draw any reliable conclusions about the quality of the solutions.

6.3 An Extended Benchmark and its Analysis

In order to test our procedure against SDST-JSSP/MAX instances, we have chosen to create our own benchmark starting from the BTS25 benchmark described in the previous sections. For each of the 25 original SDST-JSSP benchmark instances, a new SDST-JSSP/MAX instance is constructed as follows:

1. For each original SDST-JSSP benchmark instance, a solution $S = \{est(a_i) : i = 1 \dots n\}$ is computed ($est(a_i)$ represents the earliest start time of the activity a_i);
2. For each pair of successive activities a_{ik} and $a_{(i+1)k}$ in each job J_k, a *min/max* separation constraint $l_{ik}^{min} \leq s_{(i+1)k} - e_{ik} \leq l_{ik}^{max}$ is imposed, where:

$$l_{ik}^{min} = \frac{1}{s}\delta_{ik}$$
$$l_{ik}^{max} = s\delta_{ik}$$

where $\delta_{ik} = est(a_{(i+1)k}) - est(a_{ik}) - p_{ik}$ represents the temporal distance[11], computed from the solution, between the contiguous activities a_{ik} and $a_{(i+1)k}$ of job J_k, and s is a positive real number.

To proceed with our analysis of the ISP procedure, we have generated two different SDST-JSSP/MAX benchmarks, using two values for the s parameter, i.e., $s = 1.5$ and $s = 2$. We respectively call these benchmarks BTS25-TW15 and BTS25-TW20 (downloadable from `http://pst.istc.cnr.it/~angelo/bts25tw/`). From the way the SDST-JSSP/MAX benchmarks are synthesized, it is clear that the smaller the value of s, the smaller the amount slack that is allowed between any two contiguous activities of the same job, and intuitively the resulting benchmark should be more difficult to solve.

Figures 12 and 14 - and the companion boxplot graphics of Figures 13 and 15 - respectively show the results of a F-Race analysis on BTS25-TW15s and BTS25-TW20s for a wide range of α and β values. Even in the absence of competitors for the SDST-JSSP/MAX benchmarks, a number of interesting observations can be drawn from the previous figures. For example, in comparing both Figures 13 and 15 with Figure 10 we observe that the C_{max} values in the former are greater (i.e., entailing a worse result) than the C_{max} values in the latter. In fact, the SDST-JSSP/MAX benchmarks are characterized by a greater infinite capacity makespan C_{max}^0 (on average, twice as long as their SDST-JSSP counterparts), due to the presence of minimum time lags l_{ik}^{min} in the job separation constraints. These increased C_{max}^0 values obviously push the C_{max} values higher.

Let us now compare the results obtained in Figure 13 with those in Figure 15. Immediately observable is the fact that the C_{max} values of Figure 13 (shown on the

[11] In the case δ_{ik}, then we impose the separation constraint $[0, 0]$.

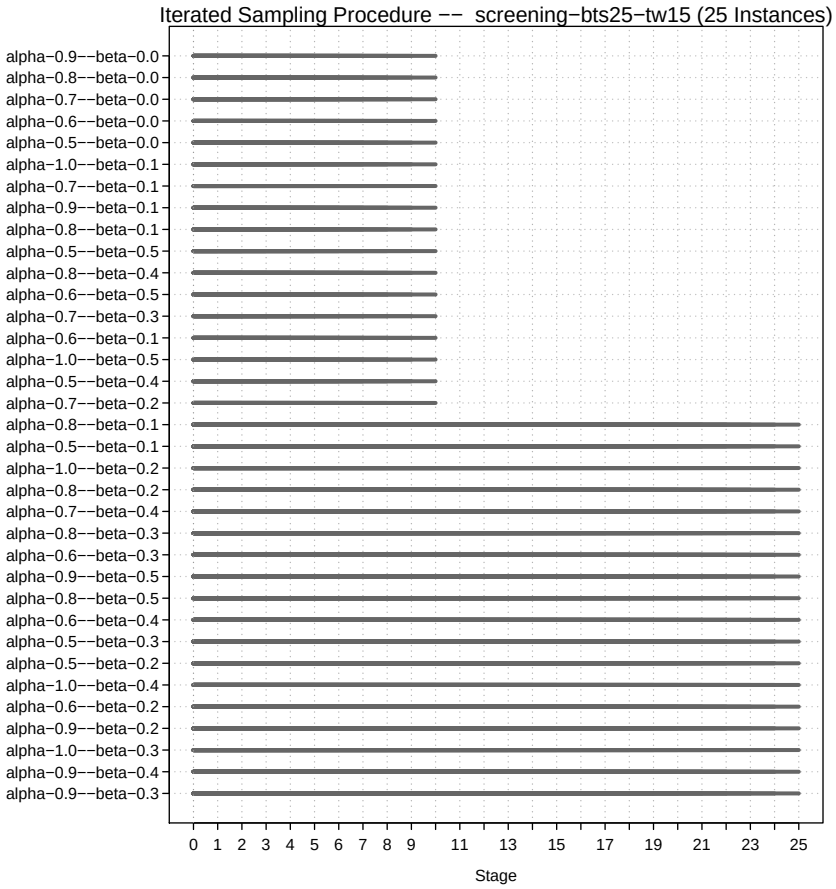

Fig. 12 *F-Race* analysis for the BTS25-TW15 instances: the graphic shows the surviving α and β values as the number of tested instances increases.

x-axis) are generally greater than the C_{max} in Figure 15. This is clearly due to the fact that the BTS25-TW15 benchmark has larger l_{ik}^{min} minimum separation constraints than does BTS25-TW20, with a consequent increase of the C_{max} values in the former case.

We also note that the best global performance is obtained with α and β values close to $\alpha = 0.6$ and $\beta = 0.2$ for both the BTS25-TW15s, and the BTS25-TW20s. This represents a difference with respect to the BTS25 case (see Section 6.1.2). When generalized temporal constraints are present it is necessary to inject a strong randomization bias in the PCP procedure to obtain the best results.

Finally, even if no meaningful comparison can be performed as yet due to the absence of other techniques capable of solving SDST-JSSP/MAX problems, some of the best results obtained on the BTS25-TW15 and BTS25-TW20 sets are available at `http://pst.istc.cnr.it/~angelo/bts25tw/`, with the aim of fostering research and competition on a class of scheduling problems that we believe are extremely interesting from the practical point of view. In fact, there are many real-world problems that are

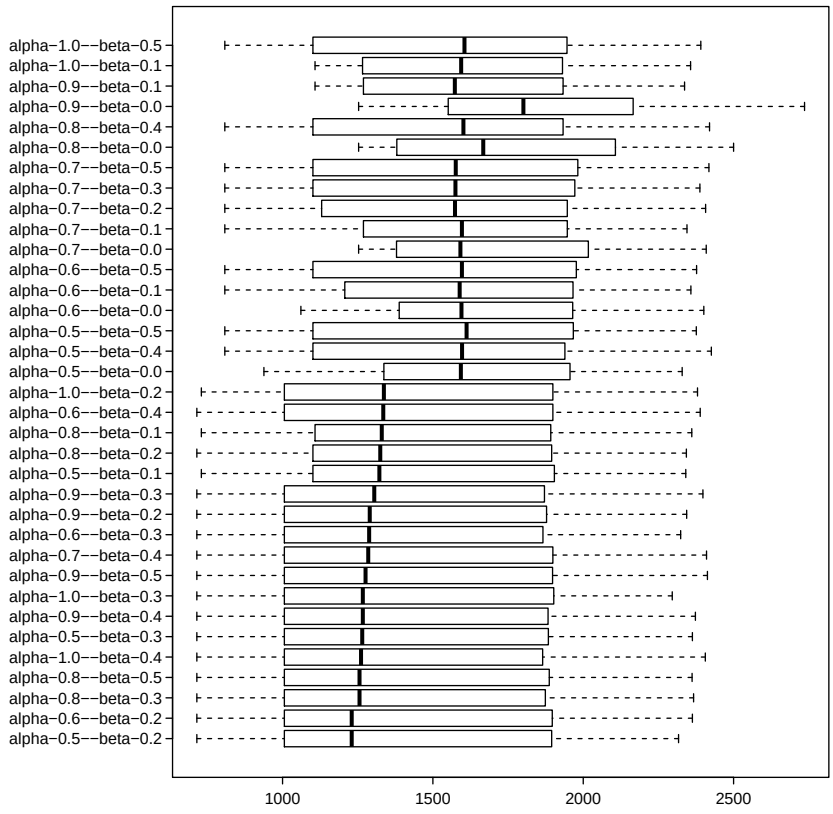

Fig. 13 Benchmark BTS25-TW15: boxplots corresponding to the *F-Race* analysis of Figure 12.

accurately modeled with the SDST-JSSP/MAX pattern. We therefore encourage the scheduling research community to accept the challenge, and expect that new techniques will be available soon for fair comparison.

7 Conclusions

Building from prior research (Oddi and Smith 1997; Cesta et al 2002), this paper has investigated the use of iterative sampling and precedence constraint posting as a means of effectively solving scheduling problems with sequence dependent setup times. The proposed iterative sampling algorithm uses at its core an extended version of the SP-PCP procedure proposed by Oddi and Smith (1997). Keys to the effectiveness of the inner procedure are a newly extended set of *dominance conditions* for pruning the search space, and correspondingly generalized variable and value ordering heuristics for directing the search. We have considered the solution of two classes of problems involving scheduling with setup times, distinguished by the presence or absence of

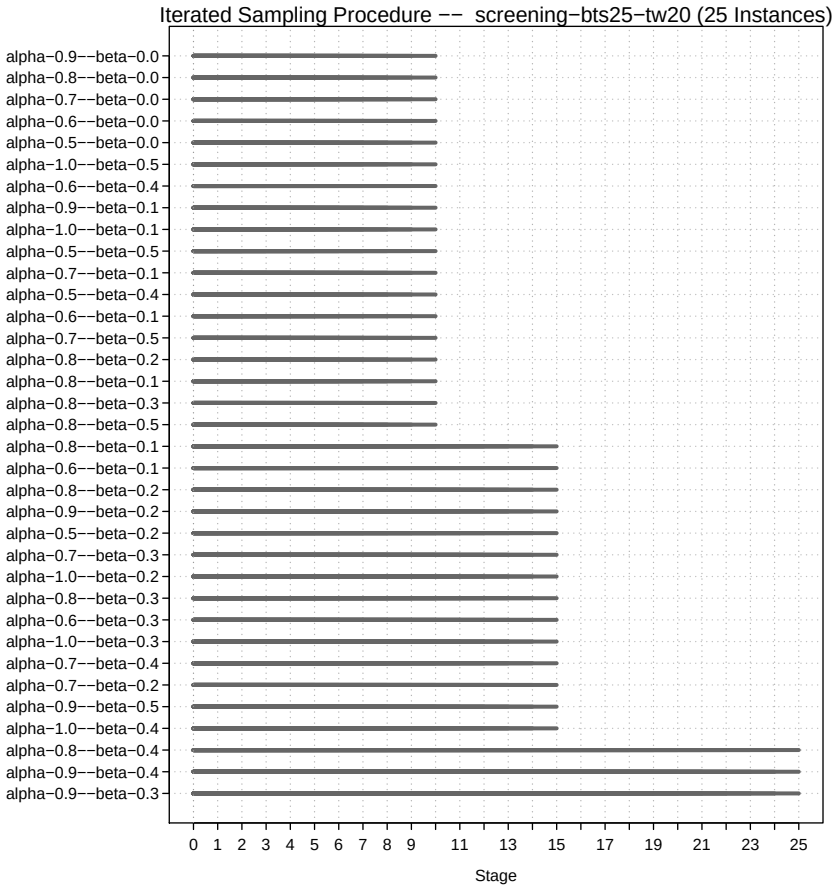

Fig. 14 *F-Race* analysis for the BTS25-TW20 instances: the graphic shows the remaining solvers (each one with different α and β values) as the number of tested instances increases.

min/max separation constraints among the activities (designated SDST-JSSP/MAX and SDST-JSSP, respectively). Overall, we have shown that adaptation of the original procedure in (Oddi and Smith 1997) to solve SDST-JSSP and SDST-JSSP/MAX instances entailed a rather limited effort, and that the generality of the original algorithm was fully preserved. To demonstrate the effectiveness of the procedure, a set of experiments were performed on two previously studied sets of SDST-JSSP benchmark problems and a new benchmark set for the problem of scheduling with both setups and min/max separation constraints (SDST-JSSP/MAX) was proposed and analyzed.

In contrast to the techniques previously applied to the setup problems, our iterative sampling procedure is not tailored to the job-shop problem in any way. For example, since some of the randomly generated benchmark problems that were analyzed do not guarantee satisfaction of the triangular inequality in all cases, the iterative sampling procedure is actually put at a disadvantage in solving these problems. However, despite this fact and the fact that the algorithm does not exploit specialized problem structure,

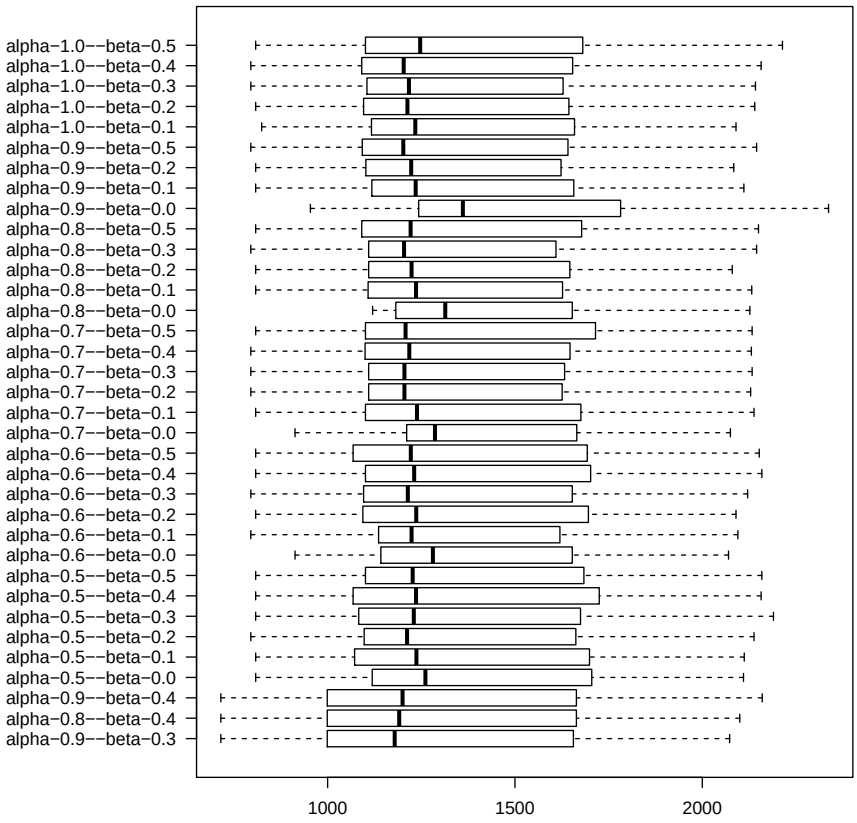

Fig. 15 Benchmark BTS25-TW20: boxplots corresponding to the *F-Race* analysis of Figure 14.

results indicate its ability to perform competitively with more problem-specific, state-of-the-art procedures.

The algorithm was also shown to be directly applicable to the full SDST-JSSP/MAX where minimum and maximum separation constraints are introduced. Reference results obtained by solving the new set of SDST-JSSP/MAX benchmark problems were presented to demonstrate this generality of our procedure, while the assumptions of the dominant job-shop specific solution proposed in González et al (2009a) were shown not to extend to the full SDST-JSSP/MAX.

As for future work, one direction of interest is towards the investigation of extended iterative sampling procedures. An example would be a strategy that enhances the search through the use of a core constraint posting procedure that performs some limited amount of backtracking.

Acknowledgements CNR authors are partially supported by CNR internal funds under project RSTL (funds 2007), EU under the ULISSE project (Contract FP7.218815), and MIUR under the PRIN project 20089M932N (funds 2008). Stephen F. Smith's work is funded in

part by the US Department of Defense Advanced Research Projects Agency (DARPA) under contract W911NF1120009 and the CMU Robotics Institute.

References

Adams J, Balas E, Zawack D (1988) The shifting bottleneck procedure for job shop scheduling. Management Science 34(3):391–401

Allahverdi A, Soroush H (2008) The significance of reducing setup times/setup costs. European Journal of Operational Research 187(3):978–984

Allahverdi A, Ng C, Cheng T, Kovalyov M (2008) A survey of scheduling problems with setup times or costs. European Journal of Operational Research 187(3):985–1032

Artigues C, Feillet D (2008) A branch and bound method for the job-shop problem with sequence-dependent setup times. Annals OR 159(1):135–159

Balas E, Simonetti N, Vazacopoulos A (2008) Job shop scheduling with setup times, deadlines and precedence constraints. Journal of Scheduling 11(4):253–262

Birattari M (2010) race: Racing methods for the selection of the best. URL `http://CRAN.R-project.org/package=race`, r package version 0.1.58

Birattari M, Stützle T, Paquete L, Varrentrapp K (2002) A racing algorithm for configuring metaheuristics. In: GECCO, pp 11–18

Brucker P, Thiele O (1996) A branch & bound method for the general-shop problem with sequence dependent setup-times. OR Spectrum 18(3):145–161

Brucker P, Lenstra J, Kan AR (1977) Complexity of machine scheduling problems. Ann Discrete Math 1:343–362

Brucker P, Jurisch B, Sievers B (1994) A branch and bound algorithm for the job-shop scheduling problem. Discrete Applied Mathematics 49(1-3):107–127

Cambazard H, Hebrard E, Barry O, Papadopoulos A (2008) Local search and constraint programming for the post-enrolment-based course timetabling problem. In: PATAT '08 – Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling

Cesta A, Oddi A, Smith SF (2002) A constraint-based method for project scheduling with time windows. J Heuristics 8(1):109–136

Cheng C, Smith S (1994) Generating Feasible Schedules under Complex Metric Constraints. In: Proceedings 12th National Conference on AI (AAAI-94)

Dechter R, Meiri I, Pearl J (1991) Temporal constraint networks. Artificial Intelligence 49:61–95

Demirkol E, Mehta S, Uzsoy R (1998) Benchmarks for shop scheduling problems. European Journal of Operational Research 109(1):137–141

Focacci F, Laborie P, Nuijten W (2000) Solving scheduling problems with setup times and alternative resources. In: AIPS, pp 92–111

González MA, Vela CR, Varela R (2009a) A Tabu Search Algorithm to Minimize Lateness in Scheduling Problems with Setup Times. In: Proceedings of the CAEPIA-TTIA 2009 13th Conference of the Spanish Association on Artificial Intelligence

González MA, Vela CR, Varela R (2009b) Private Communication

Montanari U (1974) Networks of Constraints: Fundamental Properties and Applications to Picture Processing. Information Sciences 7:95–132

Nowicki E, Smutnicki C (2005) An advanced tabu search algorithm for the job shop problem. Journal of Scheduling 8(2):145–159

Oddi A, Smith S (1997) Stochastic Procedures for Generating Feasible Schedules. In: Proceedings 14th National Conference on AI (AAAI-97), pp 308–314

Ovacik I, Uzsoy R (1994) Exploiting shop floor status information to schedule complex job shops. Journal of Manufacturing Systems 13(2):73–84

Ovacik I, Uzsoy R (1997) Decomposition Methods for Complex Factory Scheduling Problems. Kluwer Academic Publishers

Policella N, Cesta A, Oddi A, Smith S (2007) From Precedence Constraint Posting to Partial Order Schedules. AI Communications 20(3):163–180

R Development Core Team (2010) R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria, URL `http://www.R-project.org/`, ISBN 3-900051-07-0

Sotskov YN, Shakhlevich NV (1995) Np-hardness of shop-scheduling problems with three jobs. Discrete Appl Math 59(3):237–266, DOI http://dx.doi.org/10.1016/0166-218X(93)E0169-Y

Vela CR, Varela R, González MA (2009) Local search and genetic algorithm for the job shop scheduling problem with sequence dependent setup times. Journal of Heuristics