Job Shop Scheduling with Setup Times: Exploring the Applicability of a Constraint-based Iterative Sampling Approach

Angelo Oddi[1], Riccardo Rasconi[1], Amedeo Cesta[1], and Stephen F. Smith[2]

[1] Institute of Cognitive Science and Technology, CNR, Rome, Italy
angelo.oddi,riccardo.rasconi,amedeo.cesta@istc.cnr.it

[2] Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA
sfs@cs.cmu.edu

Abstract

This paper presents a heuristic algorithm for solving a job-shop scheduling problem with sequence dependent setup times (SDST-JSSP). The algorithm relies on a core constraint-based search procedure, which generates a consistent ordering of activities requiring 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 conflict 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 on a set of previously studied job-shop scheduling benchmark problems with sequence dependent setup times.

1 Introduction

This paper considers a variant of the job-shop scheduling problem with ready times, deadlines and *sequence dependent* setup times (SDST-JSSP). Similar problems are common in a semiconductor manufacturing environment [18, 19], and in general, over the last ten years, there has been an increasing interest in solving scheduling problems with setup times [2, 3]. This fact stems mainly from the observation that in various real-word industry or service environments there are tremendous savings when setup times are explicitly considered in scheduling decisions.

In this paper we propose an heuristic algorithm for solving this problem. At the algorithm's core is a constraint-based search procedure, which generates a consistent ordering of activities requiring the same resource by imposing precedence constraints between pairs of such activities and incrementally extending a temporally feasible solution. The algorithm is an extension of the stochastic version of the SP-PCP procedure proposed in [17] (which does not consider sequence-dependent setups), and bases its solving capabilities on a general meta-heuristic that has proven to be quite effective in generating competitive results despite its simplicity. The resulting procedure is then embedded within a larger *iterative-sampling* search framework [9] to broaden search space coverage and promote solution optimization.

We observe that within the current literature there are other examples of procedures for solving scheduling problems with setup times, which are extensions of counterpart procedures targeted at solving the same (or similar) scheduling problem without setup times. This is the case in the work [8], which relies on the results in [6]. It is also the case in the more recent work of [21, 13], which proposes effective heuristic procedures based on genetic algorithms and local search. These local search procedures extend a procedure proposed in [16] for the classical job-shop scheduling problem to the setup times case. Such neighborhood structure relies upon a set of properties based on the concept of critical path, defined on top of a disjunctive graph representation of the problem. We can also note the work of [5], which extends the well-know *shifting bottleneck* procedure [1] to the SDST-JSSP case. Finally, we mention the work of [5, 13] which presents the reference results for a previously studied benchmark set of SDST-JSSP problems [18] that are used for direct comparison in the experimental section of this paper.

The procedure we propose is not unique in its reliance on the constraint-solving paradigm, another example is described in [12], which introduces a more elaborate procedure based on the integration of two different extensions to the classic job shop problem. A strength of our procedure is its simplicity in spite of its effectiveness in solving a set of difficult instances of SDST-JSSPs.

The paper is organized as follows. An introductory section defines the reference SDST-JSSP problem and its representation. A central section describes the core constraint-based procedure and the related iterative sampling search strategy. An experimental section describes the performance of our algorithm and the most interesting results are explained. Some conclusions and a discussion on the future work end the paper.

2 The Scheduling Problem with Setup Times

The job-shop scheduling problem with sequence dependent setup times (SDST-JSSP) involves synchronizing the use of a set of resources $R = \{r_1, \dots, r_m\}$ to perform a set of n activities $A = \{a_1, \dots, a_n\}$ over time. The set of activities is partitioned into the set of mj jobs $\mathcal{J} = \{J_1, \dots, J_{nj}\}$. The processing of a job J_k requires the execution of a strict sequence of m activities $a_i \in J_k$, and the execution of each activity a_i is subject to the following constraints: (1) ***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; (2) ***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; (3) ***sequence dependent setup times***: for each resource

r, the value st^r_{ij} represents the setup time between two generic activities a_i and a_j requiring the same resource r, such that $e_i + st^r_{ij} \leq s_j$. According to the most common approach in literature, we consider verified the so-called *triangle inequality* [8, 4], that is, for any three activities a_i, a_j, a_k requiring the same resource, the inequality $st^r_{ij} \leq st^r_{ik} + st^r_{kj}$ holds; (4) ***job release and due dates***: each Job J_k has a release date rd_k, which specifies the earliest time that the 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 have to be completed. The due date is not a mandatory constraint and can be violated (see below).

A *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. Let C_k the completion time for the job J_k, the *maximum lateness* L_{max} is the value $L_{max} = max_{1 \leq k \leq nj}\{C_k - d_k\}$. An *optimal* solution S^* is a solution S with minimum value L_{max}. The proposed problem is strongly *NP-hard*, because it is an extension of the known problem $1|r_i|L_{max}$ [7].

3 A CSP Representation

There are different ways to formulate this problem as a *Constraint Satisfaction Problem* (CSP) [15]. In analogous way to [10, 17], we treat the problem as one of establishing *precedence constraints* between pairs of activities that require the same resource, so as to eliminate all possible conflicts in resource use. This 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 [8, 5, 21, 4].

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}, representing respectively 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 origin activity a_i of the edge. 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 the current case we have to take in to account the presence of sequence dependent setup time, which must be included when an activity a_i is executed on the same resource *before* another activity a_j.

For example, as we will see in the next sections, in case the setup times verify the triangle inequality, previous decisions on the x_{ijr} can be represented as the two temporal constraints: $e_i + st^r_{ij} \leq s_j$ ($a_i \preceq a_j$) or $e_j + st^r_{ji} \leq s_i$ ($a_j \preceq a_i$).

To support the search for a consistent assignment to the set of decision variables x_{ijr}, for any SDST-JSSP we define the directed graph $G_d(V, E)$, called *distance graph*. The set of nodes V represents time points, that is, the origin point tp_0 (the reference point of the problem) together the start and end time points, s_i and e_i, of each activity a_i). The set of edges E represents all the imposed temporal constraints, that is, precedences, durations and setup times. All the constraints have the form $a \leq tp_j - tp_i \leq b$ and for each constraint specified in SDST-JSSP, 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 efficiently determined via shortest path computations on G_d (see [11] for more details on the STP). Thus, a search for a solution to SDST-JSSP *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 (or 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 (node tp_j to node tp_i), the following constraint $-d(tp_j, tp_i) \leq tp_j - tp_i \leq d(tp_i, tp_j)$ holds [11]. Hence, the *minimal* allowed distance between tp_j and tp_i is $-d(tp_j, tp_i)$ and the maximal distance is $d(tp_i, tp_j)$. In particular, any time point tp_i is associated to a given *feasibility interval* $[lb_i, ub_i]$, which determines the set of feasible values for tp_i with respect to the origin point tp_0. 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 to 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 [11]). In particular, the two set of assignment values $S_{lb} = \{-d(tp_1, tp_0), -d(tp_2, tp_0), \ldots, -d(tp_n, tp_0)\}$ and $S_{ub} = \{d(tp_0, tp_1), d(tp_0, tp_2), \ldots, d(tp_0, tp_n)\}$ to the variables tp_i represent the respectively, the so-called *earliest-time solution* and *latest-time solution* for the given STP.

4 A Precedence Constraint Posting Procedure

The proposed procedure for solving instances of SDST-JSSP is an extension of the SP-PCP scheduling procedure (Shortest Path-based Precedence Constraint Posting) proposed in [17], 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 can be used in two ways to enhance the search process. First, it is possible to define new *dominance conditions* with respect to [17] which *propagate* problem constraints and identify unconditional decisions for promoting early pruning of alternatives in presence of setup times.

The concepts of $slack(e_i, s_j)$ and $co\text{-}slack(e_i, s_j)$ (complementary slack) play

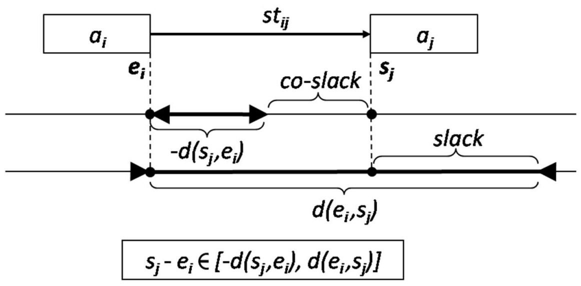

Figure 1: $slack(e_i, s_j) = d(e_i, s_j) - st^r_{ij}$ Vs. $co\text{-}slack(e_i, s_j) = -d(s_j, e_i) - st^r_{ij}$

a central role in the definition of the 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] in the graph G_d, these two concepts are defined as follows (see Figure 1):

- $slack(e_i, s_j) = d(e_i, s_j) - st^r_{ij}$ 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 1 is represented a *negative* co-slack), 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 follows:

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

Condition 1 represents an *unresolvable conflict*. There is no 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.

[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

Conditions 2, and 3, alternatively, distinguish *uniquely resolvable conflicts*. Here, there is only one feasible ordering of a_i and a_j and the decision of which constraint to post is thus unconditional. In the case of Condition 2, only $a_j \preceq a_i$ leaves $G_d(V, E)$ consistent. It is worth noting that the presence of the condition $co\text{-}slack(e_j, s_i) < 0$ means that the minimal distance between the end time e_j and the start time s_i is smaller than the minimal required setup time st^r_{ji}. Hence, we still need to impose the constraint $e_j + st^r_{ji} \leq s_i$. Condition 3 is similar and only $a_i \preceq a_j$ 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 necessary to make a *choice*.

The second way in which shortest path information is exploited is in the definition of *variable* and *value* ordering heuristics for selecting and resolving conflicts in the set characterized by Condition 4. As indicated earlier, $slack(e_i, s_j)$ and $slack(e_j, s_i)$ provide measures of the degree of *sequencing flexibility* between a_i and a_j in this context. 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[4]:

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

and

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

The *value* ordering heuristic used to resolve a selected conflict (a_i, a_j) simply chooses the precedence constraint that retains the most sequencing flexibility. Specifically, $a_i \preceq a_j$ is selected if $bd_{ij} > bd_{ji}$ and $a_j \preceq a_i$ otherwise.

4.1 The PCP Algorithm

Figure 2 gives the basic overall PCP solution procedure, which starts from an empty solution (Step 1), where the graph G_d is initialized according to Section 3 and for each job J_k an upper bound constraint $d_k + L_{max}$ is imposed on the completion time C_k. The PCP algorithm shown in Figure 2 analyses all such pairs (a_i, a_j) (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 2 interleaves the application of dominance conditions (Steps 4 and 7) with variable and value ordering (Steps 10 and

[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, $L_{max}$)
1.  $S \leftarrow$ InitSolution(Problem, $L_{max}$)
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.         $C \leftarrow$ChooseResolvableConflict($S$)
11.         if ($C = nil$)
12.           then return($S$)
13.           else begin
14.             $Prec \leftarrow$ ChoosePrecConstraint($S$, $C$)
15.             PostConstraint($S$, $Prec$)
16.           end
17.        end
18. end-loop
19. return($S$)
```

Figure 2: Basic PCP algorithm

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 2 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 assignments to the activities start times, such that all resource conflicts are resolved (i.e., all the setup times st_{ij} are satisfied), we introduce the following proposition.

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

This fact can be proved *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 $e_i + st^r_{ij} \leq s_j$ and $e_j + st^r_{ji} \leq s_i$. Since the *triangle inequality* holds for the input problem, it is guaranteed that the length of

[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 *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. Let us suppose that at least two activities a_i and a_j do not satisfy the setup constraints $e_i + st^r_{ij} \leq s_j$ (a similar proof is given for $e_j + st^r_{ji} \leq s_i$). Hence, $st^r_{ij} > s_j - e_i$ 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 $st^r_{ij} > -d(s_j, e_i)$, that is $co\text{-}slack(e_i, s_j) < 0$, which contradicts that none of the four dominance conditions is verified on S.

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 $S_{est} = \{S_i = -d(tp_i, tp_0) : i = 1 \dots n\}$.

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 sections we define: (1) a random counterpart of our conflict selection heuristic (in the style of [17]), providing both stochastic variable and value ordering heuristics, and (2) an iterative sampling search framework for optimization that embeds the stochastic procedure.

This choice is motivated by the observation that in many cases systematic backtracking search can explore large sub-trees without finding any solution. 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

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 selected conflict (a_i, a_j) is clearly distinguished. However, if other $VarEval(a_k, a_l)$ values are instead quite "close" to $VarEval(a_i, a_j)$, then the preferred choice is not clear and selection of any of these conflicts may be reasonable. We formalize this notion by defining an *acceptance band* β associated with the set of pending resolvable conflicts, and by 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 of resolvable conflicts $SC, SC = \{(a_i, a_j) : \omega \leq VarEval((a_i, a_j)) \leq \omega(1 + \beta)\}$, and (3) randomly select a conflict (a_i, a_j) in the set SC. In other words, β defines a range around the minimum heuristic evaluation within which any differences in evaluations 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 taken 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 bd, in the 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 $\approx$ equal, $pc(a_i, a_j)$ does not give clear guidance (both choices appear equally good). Accordingly, we define the following randomized version of *ChoosePrecConstraint*:

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

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 at 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 yields the least significant information).

5.2 The Optimization Algorithm

Figure 3 depicts the complete iterative sampling algorithm for generating a near-optimal solutions to SDST-JSSP instances. It is designed simply 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 L_{max}. Similar to other CSP procedures for makespan minimization (e.g., [9]), we adopt a multi-pass approach; the current best value L^{best}_{max} of the feasible solution generator is repeatedly applied to solve problems with increasingly tighter upper bound constraints ($d_k + L^{best}_{max}$) on the completion times C_k of the jobs (Steps 5-13).

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

Figure 3: Iterative sampling algorithm

6 Experimental Analysis

In this section we propose a set of empirical evaluations of the PCP algorithm. The benchmark we have tackled in our experiments are proposed in [18], and are available at http://cobweb.ecn.purdue.edu/∼uzsoy/ResearchGroup. In all benchmark instances, the setup times st^{r}_{ij} and the processing times p_i at each machine are values randomly computed in the interval $[1, 200]$. The job due dates d_i are assumed to be uniformly distributed on 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[6], 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 the benchmark instances used in the present work are calculated using τ values of 0.3 and 0.6, corresponding to loose and tight due dates respectively, and R values of 0.5, 1.5 and 2.5, respectively modelling different due date variation levels. The particular combination of the τ and R values allows us to categorize all instances in six different benchmarks, namely: $i305$, $i315$, $i325$, $i605$, $i615$, $i625$. Each benchmark contains 160 randomly generated problem instances, divided in subclasses determined by the different combinations of the number of machines and jobs involved; more precisely, all instances are synthesized by choosing 10 or 20 jobs on 5, 10, 15 or 20 machines, yielding a total of 8 subclasses for each benchmark.

From what precedes, it is clear that this benchmark does not satisfy the triangular inequality, as all setup times are computed in the interval $[1, 200]$ *at random*. As a consequence, the Iterative Sampling Algorithm (see Figure 3) may be prone to

[6]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.

disregard a number of valid solutions due to constraint overcommitment. 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 and hence, the non-verification of the triangular inequality may induce the procedure to post constraints that are stronger than they have to be during the solving process. For example, let us 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 $st_{23} = st_{32} = 3$. Let us also suppose that the available scheduling horizon is equal to 10. In this case, the triangle inequality for the setup times is clearly not satisfied; under the previous conditions our PCP procedure will surely fail (the first *dominance condition* is verified, detecting the presence of an unresolvable conflict) despite the fact that the solution $a_1 \preceq a_3 \preceq a_2$ does exist. In fact, the algorithm imposes one of the two setup constraints $st_{12} = st_{21} = 15$ and $15 > 10$, the horizon constraint[7]. Given the algorithm's reliance on the triangular inequality assumption, it is susceptible to two problems: (1) the probability of finding sub-optimal solutions increases, and (2) some existing solutions may be disregarded. However, a straightforward probabilistic computation allows easy determination of the probability to have the triangular inequality unsatisfied. This value is as low as 4.04%, which explains the globally strong performance of the algorithm in practice (see below).

The above example clearly shows the main problem caused by the non verification of the triangular inequality, and to partially compensate we opted to make each new solution undergo a post-processing procedure similar to *Chaining* [20] embedded in the PCP algorithm (Figure 2, line 6). This post-processing procedure acts to eliminate any possible constraint overcommitments that are present in the final solution, and thus can improve the solution quality by left-shifting some of the jobs. (See [20] for the details of the *Chaining* procedure.)

Table 1: Summary of the experimental results for α-randomization, the table shows the values Δ^{avg} and among square brackets the number of improved solutions.

Set	200 (secs)	400 (secs)	800 (secs)	Bests
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]

The main results of the experiments conducted are shown in Table 1 and Table 2, which respectively present the results of the random value-ordering heuris-

[7]Even if the horizon were long enough to accommodate all the activities, there can still be cases where the triangular inequality issue steers the constraint posting mechanism towards bad decisions: sequencing a_1 directly before a_2 (and thus allowing a setup time of 15) remains a bad choice.

tic (also called α-randomization) and the random variable-ordering heuristic (also called β-randomization). For every benchmark set (left column) three complete runs were performed, with increasing CPU time limit - 200, 400 and 800 seconds - this limit is the maximum CPU time that the scheduling procedure can use to find a solution. In addition, a common large value for $MaxRestart = 1000$ is imposed on all the runs. In each complete run, we measure (1) the average percentage deviation[8] from the results in [5] Δ^{avg}, considered as the best known results obtained from this benchmark, and (2) the number of improved instances (in square brackets). All runs have been performed using CMU Common Lisp version 20a on a Dell Optiplex 740, 3.5 Ghz AMD Athlon CPU, under Linux Ubuntu 8.0.

The experiments results shown in Table 1 have been conducted by selecting the following PCP parameters values: $\alpha = 0.5$ and $\beta = 0$. Though this analysis is still preliminary, the results are interesting: the employed scheduling procedure finds a considerable amount of improved solutions in all cases. As the table shows, the best performance seems to involve the benchmarks where the values of the R parameter, (i.e., the variation level of the due dates) is greater than or equal to 1.5 (as explained in the first part of this section, the corresponding benchmarks are $i315$, $i325$, $i615$ and $i625$). One possible explanation for this behavior is the following. As the value of R increases, the jobs' due dates are randomly chosen from a wider set of uniformly distributed values; as far as the PCP scheduling procedure is conceived (see Figure 2), each solution is found by imposing the deadlines of the most "critical" jobs (i.e., the jobs characterized by the earliest deadlines)[9]; in other words, our procedure *naturally* proceeds by accommodating the most critical jobs first, by imposing "hard" deadline constraints, and secondly proceeds towards the "easier" task of accommodating the remaining jobs. On the contrary, when the R values are lower, all the produced due dates tend to be critical, as all their values are comparable. This circumstance may represent an obstacle to good performance in the current version of the procedure, as it cannot always guarantee a low-lateness scheduling for all the jobs by means of imposing hard constraints. Yet, since we are running a random solving procedure, the overall results can be improved by considering the best solutions over the set of the performed runs, as shown in the column labelled *Bests* of Table 1.

Table 2 shows the results for random variable ordering (β-randomization) obtained using the following PCP parameters values: $\alpha = 0$ and $\beta = 0.2$. As in the case of α-randomization, the best performance seems to involve the benchmarks associated to higher values of the R parameter (1.5 and 2.5). However, the most evident results of Table 2 in some sense appear to be complementary with the results of Table 1. In fact, Table 2 contains the best results obtained for the problems with

[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 [5] and L^{ISP}_{max} is the best results obtained with the algorith ISP shown in Figure 2. In the particular case where $L^{B}_{max} = 0$, we consider the value $100 \times L^{ISP}_{max}$

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

Table 2: Summary of the experimental results for β-randomization the table shows the values Δ^{avg} and among square brackets the number of improved solutions.

Set	200 (secs)	400 (secs)	800 (secs)	Bests
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]

the smallest value of the R parameter ($R = 0.5$, benchmarks $i305$ and $i605$), while Table 1 contains the best results obtained with the other values of the R (which represent the best improvements over the results proposed in [5]).

In Table 3 we report the best results obtained both for the α and β randomization procedures (the two columns with names Bests(α) and Bests(β)) and also the best overall performance (the column with name BESTS). We observe a significant improvement over the two partial results, in particular we are able to improve 396 of the 960 total instances in the benchmark set.

Table 3: Summary of the main best results.

Set	Bests(α)	Bests(β)	BESTS
i305	63.3 [45]	39.6 [53]	38.5 [56]
i315	-9.0 [70]	-8.3 [60]	**-12.9 [75]**
i325	0.3 [55]	1.9 [44]	**-0.1 [61]**
i605	10.0 [40]	5.9 [47]	5.6 [52]
i615	5.4 [48]	3.4 [52]	2.8 [64]
i625	0.7 [77]	1.3 [59]	**-0.2 [88]**

As anticipated in Section 1, in order to fairly assess the efficacy of the meta-heuristic used in this work, it is also of great interest to compare such results with those obtained through recent and more specialized algorithms for the SDST-JSSP problem. To the best of our knowledge, the most recent and best results can be officially found in [13], even though they are related to the $i305$ benchmark set only. However, the authors of [13] have kindly provided us with the yet unreleased results extended to all the benchmark sets [14].According to these unreleased results, 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. Clearly, the previous results are very strong, especially on the $i305$ set, where our procedure is outperformed. In the five remaining sets however, the difference between the performances are less evident, which demonstrates the overall validity of our Precedence Constraint Posting meta-heuristic, especially when we consider the fact that we are comparing a general solving procedure with an ad-hoc algorithm, specifically devised to tackle SDST-JSSP instances.

7 Discussion, Conclusions and Future Work

In this paper we have investigated the use of iterative sampling as a means of effectively solving scheduling problems with sequence dependent setup times. Building from prior research [17, 9], the proposed iterative sampling algorithm uses an extended version of the SP-PCP procedure proposed in [17] as its core procedure.

A set of experiments have been performed on a set of well-studied randomly generated benchmarks, with the purpose of demonstrating the versatility of the procedure, which is not tailored to the job-shop problem and does not require any exploration and/or tuning of SDST-JSSP-specific parameters. We have shown that adapting the original algorithm presented in [17] to the scheduling problem version with setup times, preserving the generality of the original algorithm, was a rather straightforward process.

This procedure is to the best of our knowledge the only one that exploits a Precedence Constraint Posting approach, as opposed to the constructive search which entails the synthesis of each solution systematically from origin to horizon. Besides its versatility, key to the effectiveness of the core procedure are the newly extended *dominance conditions* for pruning the search space and the new variable and value ordering heuristics.

In the experiments conducted, the stochastic procedure was found to significantly improve the reference results in a significant set of cases. We have proposed a first interpretation of the obtained results and we think that the proposed search framework, despite its simplicity in comparison to other state-of-the-art strategies, deserves further study and development. As first steps for our future work we will explore the use of a larger set of parameters for our procedure and solve other interesting and difficult benchmarks available in the current literature. The problems proposed in [8] provide one such challenge, where the current best results can be found in the recent work of [5, 21, 4]. A second step for future work will be the development of extended iterative sampling strategies. One strategy of interest is an extension of our current core search procedure that would incorporate a limited amount of backtracking.

Acknowledgments

CNR authors are partially supported by CNR under project RSTL (funds 2007), ESA (European Space Agency) under the APSI initiative and by EU project ULISSE (Call "SPA.2007.2.1.01 Space Science". Contract FP7.218815). Stephen F. Smith is supported in part by the US Air Force Research Laboratory under contract #FA8750-10-1-0132 and Carnegie Mellon University Robotics Institute.

References

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

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

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

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

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

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

[7] P. Brucker, J. Lenstra, and A. R. Kan. Complexity of machine scheduling problems. *Ann. Discrete Math*, 1:343–362, 1977.

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

[9] A. Cesta, A. Oddi, and S. F. Smith. A constraint-based method for project scheduling with time windows. *J. Heuristics*, 8(1):109–136, 2002.

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

[11] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.

[12] F. Focacci, P. Laborie, and W. Nuijten. Solving scheduling problems with setup times and alternative resources. In *AIPS*, pages 92–111, 2000.

[13] M. A. González, C. R. Vela, and R. Varela. 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*, 2009.

[14] M. A. González, C. R. Vela, and R. Varela. Private Communication. 2009.

[15] U. Montanari. Networks of Constraints: Fundamental Properties and Applications to Picture Processing. *Information Sciences*, 7:95–132, 1974.

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

[17] A. Oddi and S. Smith. Stochastic Procedures for Generating Feasible Schedules. In *Proceedings 14th National Conference on AI (AAAI-97)*, pages 308–314, 1997.

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

[19] I. Ovacik and R. Uzsoy. *Decomposition Methods for Complex Factory Scheduling Problems*. Kluwer Academic Publishers, 1997.

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

[21] C. R. Vela, R. Varela, and M. A. González. Local search and genetic algorithm for the job shop scheduling problem with sequence dependent setup times. *Journal of Heuristics*, 2009.