Iterative-Sampling Search for Job Shop Scheduling with Setup Times

Angelo Oddi [1] and **Riccardo Rasconi** [1] and **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 the job-shop scheduling problem with sequence dependent setup times (SDST-JSSP). The algorithm relies on a core constraint-based search procedure, which generates consistent ordering of activities requiring the same resource by incrementally imposing precedence constraints on a temporal 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.

Introduction

This paper considers a variant of the job-shop scheduling problem with ready times, deadlines and *sequence dependent* setup times (SDST-JSSP), a common problem in semiconductor manufacturing (Ovacik and Uzsoy 1994; 1997). The work proposes a heuristic algorithm which relies on a *core* constraint-based search procedure, which generates a consistent ordering of activities requiring the same resource by incrementally imposing precedence constraints between pair of activities on a temporal feasible solution. Such algorithm is an extension to the case of scheduling problems with setup times of the stochastic version of the SP-PCP procedure proposed in (Oddi and Smith 1997). This constraint-based search is then embedded within a larger *iterative-sampling* search framework (Cesta, Oddi, and Smith 2002) to broaden search space coverage and promote solution optimization.

We observe within the current literature there are other examples of procedures for solving scheduling problems with setup times, which are extension of them counterpart procedures targeted on the same (or similar) scheduling problem without setup times. This is the case of the work (Brucker and Thiele 1996), which relies on the results of the papers (Brucker, Jurisch, and Sievers 1994) or the recent paper (Vela, Varela, and González 2009), which proposes a hybrid approach based on genetic algorithms and local search. The local search procedure extends to the setup times case a procedure proposed in the previous paper (Nowicki and Smutnicki 2005) for the classical job-shop scheduling problem. We can also consider the paper (Balas, Simonetti, and Vazacopoulos 2008), which extends the well-know *shifting bottleneck* procedure (Adams, Balas, and Zawack 1988) to the SDST-JSSP case. The paper (Balas, Simonetti, and Vazacopoulos 2008) is relevant for this work, because it presents the current best results for a previously studied benchmark set of SDST-JSSP problems (Ovacik and Uzsoy 1994) used for direct comparison in the experimental section of the paper. The proposed procedure is not the only one which relies on the constraint-solving paradigm, another example is described in (Focacci, Laborie, and Nuijten 2000), which proposes an more elaborate procedure based on the integration of two different models for the problem. A strength of the proposed 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.

The Scheduling Problem

The job-shop scheduling problem with sequence dependent setup times (SDST-JSSP) involves synchronizing the use of a set of resources $R = \{r_1, \ldots, r_m\}$ to perform a set of n activities $A = \{a_1, \ldots, a_n\}$ over time. The set of activities is partitioned in 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 $a_i \in J_k$, and the execution of each activity a_i is subject to the following constraints:

- *resource availability* - each 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 times of a_i.

- *separation constraints* - for each pair of successive activities a_{i_j} and $a_{i_{j+1}}$, $j = 1, \ldots, m-1$, in job J_k, there is a separation (ordering) constraints $s_{i_{j+1}} - e_{i_j} \geq 0$, $j = 1, \ldots, m-1$.
- *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 $e_i + st^r_{ij} \leq s_j$. The setup times st^r_{ij} satisfies the so-called *triangular inequality* (Brucker and Thiele 1996; Artigues and Feillet 2008), that is, for each 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.
- *job release and due dates* - every 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 must be completed. The due date is not a mandatory constraint and can be violated (see below).

Let C_k the completion time for the job J_k, the objective is to minimize the maximum lateness of the problem, that is the value $L_{max} = max_{1 \leq k \leq nj}\{C_k - d_k\}$.

As introduced above, the problem is relevant in semiconductor manufacturing and in general, as observed in the recent work (Allahverdi et al. 2008), in the last ten years there has been an increasing interest in solving scheduling problems with setup time. 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.

A CSP Representation

There are different ways to formulate this problem as a *Constraint Satisfaction Problem* (CSP) (Montanari 1974). In analogous way to (Cheng and Smith 1994; Oddi and Smith 1997), the problem is treated 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. 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 (Brucker and Thiele 1996; Balas, Simonetti, and Vazacopoulos 2008; Vela, Varela, and González 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 two dummy activities 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 precedences 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 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 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. 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 $e_j + st^r_{ji} \leq s_i$ ($a_j \preceq a_i$).

Temporal Constraints

To support the search for a consistent assignment to the set of decision variables x_{ijr}, we can define for any SDST-JSSP a directed graph $G_d(V, E)$ which is an extended version of the disjunctive graph G. The set of nodes V represents time points (i.e., the origin point and the start and end time points, s_i and e_i, of each activity a_i) and the set of edges E represents all the imposed temporal constraints (i.e., precedences, durations and setup times). For every constraint of the form $a \leq tp_j - tp_i \leq b$ 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* (Dechter, Meiri, and Pearl 1991) and its consistency can be efficiently determined via shortest path computations. 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 (i.e., no negative weight cycles). 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 (Dechter, Meiri, and Pearl 1991). Hence, the minimal allowed distance between tp_j and tp_i is $-d(tp_j, tp_i)$ and the maximal distance is $d(tp_i, tp_j)$. This information will be used in the following section.

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 (Oddi and Smith 1997) which utilizes shortest path information in $G_d(V, E)$ for guiding the search process. In similar way to the case of SP-PCP, shortest path information can be used in two ways to enhance the search process. First, it is possible to define new *dominance conditions*, which *propagate* problem constraints and identify unconditional decisions for promoting early pruning of alternatives. For any pair of activities a_i and a_j that are competing for the same resource r, four possible cases of conflict are defined:

1. $d(e_i, s_j) < st^r_{ij} \wedge d(e_j, s_i) < st^r_{ji}$
2. $d(e_i, s_j) < st^r_{ij} \wedge d(e_j, s_i) \geq st^r_{ji} \wedge -d(s_i, e_j) < st^r_{ji}$
3. $d(e_i, s_j) \geq st^r_{ij} \wedge d(e_j, s_i) < st^r_{ji} \wedge -d(s_j, e_i) < st^r_{ij}$
4. $d(e_i, s_j) \geq st^r_{ij} \wedge d(e_j, s_i) \geq st^r_{ji}$

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

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 the presence of the condition $-d(s_i, e_j) < st^r_{ji}$, which states the minimal distance between the end time e_j and the start time a_i is lesser than the 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 conflict*. 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. In this context, $flex(e_i, s_j) = d(e_i, s_j) - st^r_{ij}$ and $flex(e_j, s_i) = d(e_j, s_i) - st^r_{ji}$ provide measures of the degree of *sequencing flexibility* that remains with respect to 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 ordering decision that is closest to being forced). 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:

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

and

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

The $\sqrt{S}$ bias is introduced to hedge when the conflict with the overall $min\{flex(e_i, s_j), flex(e_j, s_i)\}$ has a very large $max\{flex(e_i, s_j), flex(e_j, s_i)\}$, 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.

The *value* ordering heuristic used within SP-PCP 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.

Figure 1 gives the basic overall PCP solution procedure, which starting from an empty solution (Step 1), where for each job J_k, it is imposed a completion time $C_k = d_k + L_{max}$. The procedure interleaves the application of dominance conditions (Steps 3 and 6) with variable and value ordering (Steps 9 and 13 respectively) and incremental updating of the solution graph $G_d(V, E)$ (Steps 7 and 14) to conduct a single pass through the search tree. The proposed PCP procedure is able to generate feasible solutions to SDST-JSSP instances, such that, when exits with success, no one of the four dominance conditions holds and the set of imposed precedence constraints induces a total ordering on each subset of activities requiring the same resource, and all the problem constraints are satisfied, in particular the setup constraints. This fact can be easily proved by contradiction. In fact, let us suppose that the PCP procedure exits

PCP(*Problem,* L_{max})
1. $S \leftarrow$ EmptySolution(*Problem*, L_{max})
2. **loop**
3. **if** UnresolvableConflict(S)
4. **then return**(nil)
5. **else**
6. **if** UniquelyResolvableConflict(S)
7. **then** PostUnconditionalConstraints(S)
8. **else begin**
9. $C \leftarrow$ChooseResolvableConflict(S)
10. **if** ($C = nil$)
11. **then return**(S)
12. **else begin**
13. $Prec \leftarrow$ ChoosePrecConstraint(S, C)
14. PostConstraint(S, $Prec$)
15. **end**
16. **end**
17. **end-loop**
18. **return**(S)

Figure 1: Basic PCP algorithm

with success and exists at least two activities a_i and a_j, requiring the same resource, which can overlap or not satisfy the setup constraints $e_i + st^r_{ij} \leq s_j$ or $e_j + st^r_{ji} \leq s_i$. In this case, at least one of the dominance conditions holds and the procedure cannot exit with success.

It is worth noting among the constraints imposed on the SDST-JSSP problem there is the so-called *triangular inequality*, an hypothesis generally assumed in the literature (Brucker and Thiele 1996; Artigues and Feillet 2008), that is, for each three activities a_i, a_j, a_k requiring the same resource, the following inequality $st^r_{ij} \leq st^r_{ik} + st^r_{kj}$ holds. Previous condition guarantees that the duration of the *direct* transition $a_i \preceq a_j$ between two generic activities a_i and a_j is the shortest one and there is not possible to find an *indirect* transition (a sequence of activities $a_i \rightsquigarrow a_k \rightsquigarrow a_j$) which has a shorter duration. This fact is relevant for the PCP procedure, in fact given the resolution procedure, for each pair of activities a_i, a_j, either the constraint $e_i + st^r_{ij} \leq s_j$ or $e_j + st^r_{ji} \leq s_i$ is still imposed or checked. Hence, in the case the triangular inequality does not hold, the procedure could overcommit the partial solution with the consequence of rejecting a valid solution (see a specific example in the experimental section).

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 (Oddi and Smith 1997)), providing both

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 subtrees 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. This 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.

Stochastic Variable and Value Ordering

Our design of stochastic versions of PCP's variable and value ordering heuristics follows from the simple intuition that makes more sense to follow a heuristic's advice when it clearly distinguishes one alternative as superior and 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 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 with respect to the set of pending resolvable conflicts and expanding the *ChooseResolvable-Conflict* step of PCP to:

1. Calculate the overall minimum value $\omega = min\{VarEval(a_i, a_j)\}$ as before
2. Determine the subset 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 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 cases where $S = \frac{min\{flex(e_i,s_j), flex(e_j,s_i)\}}{max\{flex(e_i,s_j), flex(e_j,s_i)\}}$ 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*:

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

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

Figure 2: Iterative sampling algorithm

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 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 gives the least information).

The Optimization Algorithm

Figure 2 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., (Cesta, Oddi, and Smith 2002)), we adopt a multi-pass approach; the current best value L_{max}^{best} of the feasible solution generator is repeatedly applied to solve problems with increasingly tighter constraint on the deadlines on the jobs (Steps 5-13). Analogously to the paper (Cesta, Oddi, and Smith 2002), during an initial tuning phase of the algorithm this "dynamic backward", multi-pass approach was found to outperform alternative schemes where the horizon parameter for successive calls was uniformly varied between established lower and upper bound values.

Experimental Analysis

The SDST-JSSP benchmark we have tackled in our experiments are proposed in (Ovacik and Uzsoy 1994), and are available at `http://cobweb.ecn.purdue.edu/~uzsoy/ResearchGroup`. In order to comprehensively interpret the

experimental results, it is necessary to provide a brief description of how such benchmarks have been produced.

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[1], and (2) the R value, which determines the range of I, whose bounds are defined by: $[\mu - R/2, \mu + R/2]$. All the benchmark instances used in the present work 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 modelling different due date variation levels. The particular combination of the τ and R values allows 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 and 20 jobs on 5, 10, 15 and 20 machines, yielding a total of 8 subclasses for each benchmark.

From what precedes, an important issue worth pointing out is that this benchmark do not satisfy the triangular inequality, i.e., given any three activities a_i, a_j, a_k requiring the same resource, the following inequality $st^r_{ij} \leq st^r_{ik} + st^r_{kj}$ is not guaranteed to hold (as all setup times are computed in the interval $[1, 200]$) *at random*. As a consequence, the Iterative Sampling Algorithm (see Figure 2) may be prone to disregard a number of valid solutions due to constraint overcommitment. In order to show this, let us provide the following 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 example, the triangular inequality is obviously not satisfied; in fact, our procedure will surely fail despite the solution $a_1 \preceq a_3 \preceq a_2$ exists, because the first *dominance condition* is verified, which reveals the presence of an unresolvable conflict.

However, a straightforward probabilistic computation allows to easily determine the probability to have the the triangular inequality unsatisfied; such value is as low as 4.04%, and this explains the globally good performances of the algorithm (see below). In other words, the triangular inequality assumption can still remain valid, as such inequality is in fact satisfied in 96% of the cases.

Going back to the example, even if the horizon were long enough to accommodate all the activities, there would still be cases where the triangular inequality issue will steer the constraint posting mechanism towards bad decisions: sequencing a_1 directly before a_2 (and thus allowing a setup time of 15), however remains a bad choice. For this reason, we have decided to make each new solution undergo a post-processing procedure similar to *Chaining* (Policella et al. 2007) embedded in the PCP algorithm (Figure 1, line 6), in order to eliminate all the possibly present constraint overcommitments, and thus improve the solution quality by left-shifting some of the jobs. This post-processing phase works as follows: given an input solution S a polynomial transformation method is used to remove the previous pointed overcommitments. This operation can be accomplished in two steps: (1) all the previously posted ordering constraints are removed from the input solution; (2) for each resource and for each activity a_i (according to the increasing order of the activities using the resource), the unique successor a_j is considered, and the precedence constraints $e_i + st^r_{ij} \leq s_j$ is posted. The last step is iterated until all the activities are linked by the *real* sequence dependent setup times.

The choice of these particular benchmarks for the experiments is motivated by the fact that in this preliminary phase of our investigation we are mainly interested in assessing the validity of the iterative PCP procedure in tackling problems it is not strictly designed for. Having assessed its promising overall performances, next step will be to apply modifications to the same procedure (e.g., by adjusting the dominance conditions) in order to explicitly take into account the non-verification of the triangular inequality. The main results of the conducted experiments are shown in Table 1. For every benchmark set (left column) three complete runs have been performed, with increasing CPU time limit (and a common large value for $MasRestart = 1000$); such limit is the maximum CPU time that the scheduling procedure can use to find a solution. In each complete run, we measure (1) the average percentage deviation from the results in (Balas, Simonetti, and Vazacopoulos 2008), considered as the best known results obtained from this benchmark, and (2) the number of improved instances (in square brackets). All the experiments have been conducted by selecting the following PCP parameters values: $\alpha = 0.5$ and $\beta = 0$. All runs have been performed on a Windows Xp machine with 0.9 Ghz CPU, using Allegro Common Lisp 6.0.

Table 1: Summary of the main experimental results

Set	800 (secs)	1600 (secs)	3200 (secs)
	Δ^{avg} [#impr.]	Δ^{avg} [#impr.]	Δ^{avg} [#impr.]
i305	92.7 [38]	79.0 [35]	74.8 [32]
i315	24.7 [36]	1.5 [51]	-7.0 [59]
i325	20.7 [24]	4.1 [37]	0.8 [53]
i605	24.5 [29]	15.3 [28]	11.07 [29]
i615	22.1 [33]	11.3 [37]	6.3 [38]
i625	19.1 [48]	6.1 [60]	1.2 [66]

Though this analysis is preliminary, the results are interesting: the employed scheduling procedure finds a considerable amount of improved solutions in all cases. Yet, the best performance seems to involve the benchmarks associated to higher values of the R parameter; as the table shows, the outcomes are much more convincing when R is greater

[1]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.

or equal than 1.5, i.e., when the variation level of the due dates is higher. One possible explanation for this behaviour is the following. As the value of R increases, the jobs' due dates are randomly chosen from a wider set of uniformly distributed values; this means that, among all the produced due dates, there will be a subset containing particularly demanding deadlines (i.e., the earliest deadlines). As far as the PCP scheduling procedure is conceived (see Figure 1), each solution is found by imposing the deadlines of the most "critical" jobs (i.e., the jobs characterized by the earliest deadlines)[2]. 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.

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 (Oddi and Smith 1997; Cesta, Oddi, and Smith 2002), the proposed iterative sampling algorithm uses as core procedure an extended version of the SP-PCP procedure proposed in (Oddi and Smith 1997). Key to the effectiveness of the core procedure are the new extended *dominance conditions* for pruning the search space and the new variable and value ordering heuristics. In an experimental study on a set of well-studied randomly generated benchmarks, the stochastic procedure was found to significantly improve the current best 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, merits further studies and developments. 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. For example the ones proposed in (Brucker and Thiele 1996), where current best results can be found in the recent works (Balas, Simonetti, and Vazacopoulos 2008; Vela, Varela, and González 2009; Artigues and Feillet 2008). A second step for future work will be the development of extended iterative sampling strategies. For example, a strategy which uses a core search procedure performing 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).

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

References

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

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

Artigues, C., and 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.; and Vazacopoulos, A. 2008. Job shop scheduling with setup times, deadlines and precedence constraints. *Journal of Scheduling* 11(4):253–262.

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

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

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

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

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

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

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

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

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

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

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

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