Job Shop Scheduling with Routing Flexibility and Sequence Dependent Setup-Times

Angelo Oddi¹, Riccardo Rasconi¹, Amedeo Cesta¹, and Stephen F. Smith²

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 meta-heuristic algorithm for solving a job shop scheduling problem involving both sequence dependent setup-times and the possibility of selecting alternative routes among the available machines. The proposed strategy is a variant of the Iterative Flattening Search (IFS) schema. This work provides three separate results: (1) a constraint-based solving procedure that extends an existing approach for classical Job Shop Scheduling; (2) a new variable and value ordering heuristic based on temporal flexibility that take into account both sequence dependent setup-times and flexibility in machine selection; (3) an original relaxation strategy based on the idea of randomly breaking the execution orders of the activities on the machines with a activity selection criteria based on their proximity to the solution’s critical path. The efficacy of the overall heuristic optimization algorithm is demonstrated on a new benchmark set which is an extension of a well-known and difficult benchmark for the Flexible Job Shop Scheduling Problem.

1 Introduction

This paper describes an iterative improvement approach to solve job-shop scheduling problems involving both sequence dependent setup-times and the possibility of selecting alternative routes among the available machines. Over the last years there has been an increasing interest in solving scheduling problems involving both setup-times and flexible shop environments [3, 2]. This fact stems mainly from the observation that in various real-world industry or service environments there are tremendous savings when setup times are explicitly considered in scheduling decisions. In addition, the possibility of selecting alternative routes among the available machines is motivated by interest in developing Flexible Manufacturing Systems (FMS) [25] able to use multiple machines to perform the same operation on a job’s part, as well as to absorb large-scale changes, in volume, capacity, or capability.

The proposed problem, called in the rest of the paper Flexible Job Shop Scheduling Problem with Sequence Dependent Setup Times (SDST-FJSSP) is a generalization of the classical Job Shop Scheduling Problem (JSSP) where a given activity may be
processed on any one of a designated set of available machines and there are no setup-
times. This problem is more difficult than the classical JSSP (which is itself NP-hard),
since it is not just a sequencing problem; in addition to deciding how to sequence ac-
tivities that require the same machine (involving sequence-dependent setup-times), it
is also necessary to choose a routing policy, i.e., deciding which machine will process
each activity. The objective remains that of minimizing makespan.

Despite this problem is often met in real manufacturing systems, not many papers
take into account sequence dependent setup-times in flexible job-shop environments.
On the other hand, a richer literature is available when setup-times and flexible job-
shop environments are considered separately. In particular, on the side of setup-times
a first reference work is [7], which relies on an earlier proposal presented in [6]. More
recent works are [28] and [13], which propose effective heuristic procedures based on
genetic algorithms and local search. In these works, the introduced local search pro-
ducts extend an approach originally proposed by [19] for the classical job-shop sched-
uling problem to the setup times case. A last noteworthy work is [5], which extends the
have produced reference results on a previously studied benchmark set of JSSP with
sequence dependent setup-times problems initially proposed by [7]. About the Flexible
Job Shop Scheduling FJSSP an effective synthesis of the existing solving approaches is
proposed in [14]. The core set of procedures which generate the best results include the
genetic algorithm (GA) proposed in [10], the tabu search (TS) approach of [16] and the
discrepancy-based method, called climbing depth-bound discrepancy search (CDDS),
defined in [14]. Among the papers dealing with both sequence dependent setup times
and flexible shop environments there is the work [23], which considers a shop type
composed of pools of identical machines as well as two types of setup times: one mod-
eling the transportation times between different machines (sequence dependent) and the
other one modeling the required reconfiguration times (not sequence dependent) on the
machines. The other work that deals with sequence dependent setup times and routing
flexibility is [24], which considers a flow-shop environment with multi-purpose ma-
chines such that each stage of a job can be processed by a set of unrelated machines
(the processing times of the jobs depend on the machine they are assigned to). [26] con-
siders a problem similar to the previous one, where the jobs are composed by a single
step, but setup-times are both sequence and machine dependent. Finally, [27] considers
a job-shop problem with parallel identical machines, release times and due dates but
sequence independent setup-times.

This paper focuses on a family of solving techniques referred to as Iterative Flattening
Search (IFS). IFS was first introduced in [8] as a scalable procedure for solving
multi-capacity scheduling problems. IFS is an iterative improvement heuristic designed
to minimize schedule makespan. Given an initial solution, IFS iteratively applies two-
steps: (1) a subset of solving decisions are randomly retracted from a current solution
(relaxation-step); (2) a new solution is then incrementally recomputed (flattening-step).
Extensions to the original IFS procedure were made in two subsequent works [17, 12]
and more recently [20] have performed a systematic study aimed at evaluating the effec-
tiveness of single component strategies within the same uniform software framework.
The IFS variant that we propose relies at its core on a constraint-based solver. This
procedure is an extension of the SP-PCP procedure proposed in [21]. SP-PCP generates consistent orderings of activities requiring the same resource by imposing precedence constraints on a temporally feasible solution, using variable and value ordering heuristics that discriminate on the basis of temporal flexibility to guide the search. We extend both the procedure and these heuristics to take into account both sequence dependent setup-times and flexibility in machine selection. To provide a basis for embedding this core solver within an IFS optimization framework, we also specify an original relaxation strategy based on the idea of randomly breaking the execution orders of the activities on the machines with a activity selection criteria based on their proximity to the solution’s critical path.

The paper is organized as follows. Section 2 defines the SDST-FJSSP problem and Section 3 introduces a CSP representation. Section 4 describes the core constraint-based search procedure while Section 5 introduces details of the IFS meta-heuristics. An experimental section (Section 6) describes the performance of our algorithm on a set of benchmark problems, and explains the most interesting results. Some conclusions end the paper.

2 The Scheduling Problem

The SDST-FJSSP entails synchronizing the use of a set of machines (or 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 into a set of \( nj \) jobs \( J = \{J_1, \ldots, J_{nj}\} \). The processing of a job \( J_k \) requires the execution of a strict sequence of \( nk \) activities \( a_i \in J_k \) and cannot be modified. All jobs are released at time 0. Each activity \( a_i \) requires the exclusive use of a single resource \( r_i \) for its entire duration chosen among a set of available resources \( R_i \subseteq R \). No preemption is allowed. Each machine is available at time 0 and can process more than one operation of a given job \( J_k \) (recirculation is allowed). The processing time \( p_{ir} \) of each activity \( a_i \) depends on the selected machine \( r \in R_i \), such that \( e_i - s_i = p_{ir} \), where the variables \( s_i \) and \( e_i \) represent the start and end time of \( a_i \). Moreover, 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 [7, 4]). 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. A solution \( S = \{(\pi_1, \tau_1), (\pi_2, \tau_2), \ldots, (\pi_n, \tau_n)\} \) is a set of pairs \( (\pi_i, \tau_i) \), where \( \pi_i \) is the assigned start time of \( a_i \), \( \tau_i \) is the selected resource for \( a_i \) and all the above constraints are satisfied. Let \( C_k \) be the completion time for the job \( J_k \), the makespan is the value \( C_{max} = \max_{1 \leq k \leq nj} \{C_k\} \). An optimal solution \( S^* \) is a solution \( S \) with the minimum value of \( C_{max} \). The SDST-FJSSP is NP-hard since it is an extension of the JSSP problem [11].

3 A CSP Representation

There are different ways to model the problem as a Constraint Satisfaction Problem (CSP) [18]; here we use an approach similar to [21]. In particular, we focus on assigning
resources to activities, a distinguishing aspect of SDST-FJSSP and on *establis-

hing sequence dependent setup time constraints* between pairs of activities that require the

same resource, so as to eliminate all possible conflicts in the resource usage.

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, rep-
resenting the beginning (reference) and the end (horizon) of the schedule. Each activity
$a_i$ is labelled with the set of available resource choices $R_i$. $J$ is a set of directed edges
$(a_i, a_j)$ representing the precedence constraints among the activities (job precedences
constraints) and are labelled with the set of processing times $p_i r \ (r \in R_i)$ 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 $r$ ($R_i = R_j = \{r\}$) and the related label
represents the set of possible ordering between $a_i$ and $a_j$: $a_i \leq a_j$ or $a_j \leq a_i$. Hence,
in CSP terms, there are *two sets of decision variables*: (1) a variable $x_i$ is defined for
each activity $a_i$ to select one resource for its execution, the domain of $x_i$ is the set of
available resource $R_i$; (2) A variable $o_{ijr}$ is defined for each pair of activities $a_i$ and $a_j$,
requiring the same resource $r$ ($x_i = x_j = r$), which can take one of two values $a_i \leq a_j$
or $a_j \leq a_i$. It is worth noting that in considering either ordering we have to take into ac-
count 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 $o_{ijr}$ can be represented as the following two temporal constraints: $e_i + st_{ij}^x \leq s_j$ (i.e. $a_i \leq a_j$) or $e_j + st_{ij}^x \leq s_i$ (i.e. $a_j \leq a_i$).

To support the search for a consistent assignment to the set of decision variables
$x_i$ and $o_{ijr}$, for any SDST-FJSSP we define the directed graph $G_d(V, E)$, called *dist-
ance graph*, which is an extended version of the 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 $e_i$ have their usual meaning. The set
of edges $E$ represents all the imposed temporal constraints, i.e., precedences and du-

rations. In particular, for each activity $a_i$ we impose the interval duration constraint
$e_i - s_i \in [p_i^{min}, p_i^{max}]$, such that $p_i^{min}$ ($p_i^{max}$) is the minimum (maximum) process-
ing time according to the set of available resources $R_i$. 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 speci-
fied in the SDST-FJSSP 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 $a$ 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 (STP) and its consistency can be efficiently determined via shortest path com-
putations; the problem is consistent if and only if no closed paths with negative length
(or negative cycles) are contained in the graph $G_d$ [9]. Thus, a search for a solution to a
SDST-FJSSP 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.

A solution $S$ is given as a affine graph $G_S(A_G, J, X_S)$, such that each undirected
edge $(a_i, a_j)$ in $X$ is replaced with a directed edge representing one of the possible
orderings between $a_i$ and $a_j$: $a_i \leq a_j$ or $a_j \leq a_i$. In general the directed graph $G_S$ rep-
respects a set of temporal solutions \((S_1, S_2, \ldots, S_n)\) that is, a set of assignments to the activities’ start times which are consistent with the set of imposed constraints \(X\). 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 [9]. Hence, the interval \([lb_i, ub_i]\) of time values associated with a given time variable \(tp_i\) respect to the reference point \(tp_0\) is computed on the graph \(G_d\) as the interval \([-d(tp_i, tp_0), d(tp_0, tp_i)]\). 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 \(tp_i\) represent the so-called earliest-time solution and latest-time solution, respectively.

4 Basic Constraint-based Search

The proposed procedure for solving instances of SDST-FJSSP integrates a Precedence Constraint Posting (PCP) one-shot search for generating sample solutions and an Iterative Flattening meta-heuristic that pursues optimization. The one-shot step, similarly to the SP-PCP scheduling procedure (Shortest Path-based Precedence Constraint Posting) proposed in [21], utilizes shortest path information in \(G_d(V, E)\) to guide the search process. Shortest path information is used in a twofold fashion to enhance the search process: to propagate problem constraints and to define variable and value ordering heuristics.

4.1 Propagation Rules

The first way to exploit shortest path information is by introducing conditions to remove infeasible values from the domains of the decision variables \(x_i\), representing the assignment of resources to activities. Namely, for each activity \(a_i\) we relax the disjunctive duration constraint into the interval constraint \(e_i - s_i \in [p_i^{\min}, p_i^{\max}]\), such that \(p_i^{\min}\) \((p_i^{\max})\) is the minimum (maximum) processing time according to the set of available resources \(R_i\) (i.e., the domain of the decision variable \(x_i\)). As the search proceeds, as soon as the interval of distance between the start time and the end time of \(a_i\) \([-d(s_i, e_i), d(e_i, s_i)]\) is updated, the duration \(p_ir \not\in [-d(s_i, e_i), d(e_i, s_i)]\) is removed from the domain of \(x_i\) and a new interval \([p_i^{\min}, p_i^{\max}]\) is recomputed accordingly. In case the domain of the decision variable \(x_i\) becomes empty, the search reaches a failure state.

The second way to exploit shortest path is by introducing new Dominance Conditions (which adapt those presented in [21] to the setup times case), through which problem constraints are propagated and mandatory decisions for promoting early pruning of alternatives are identified. The following concepts of \(\text{slack}(e_i, s_j)\) and \(\text{co-slack}(e_i, s_j)\) (complementary slack) play a central role in the definition of such new dominance conditions.

Given two activities \(a_i, a_j\) both assigned to resource \(r\), and the related interval of distances \([-d(s_j, e_i), d(e_i, s_j)]\) on the graph \(G_d\), they are defined as follows:

- \(\text{slack}^r(e_i, s_j) = d(e_i, s_j) - st^r_{ij}\) 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 $a_j$ taking into account the setup time constraint $e_i + st_{ij}^r \leq s_j$. If $slack^r(e_i, s_j) < 0$, then the ordering $a_i \leq a_j$ is not feasible.

- $co-slack^r(e_i, s_j) = -d(s_j, e_i) - st_{ij}^r$ is the difference between the minimum possible distance between $a_i$ and $a_j$, $-d(s_i, e_j)$, and the setup time $st_{ij}^r$; if $co-slack^r(e_i, s_j) \geq 0$, then there is no need to separate $a_i$ and $a_j$, as the setup time constraint $e_i + st_{ij}^r \leq s_j$ is already satisfied.

In order not to overload the notation, in the rest of the paper the slack and co-slack elements will be presented without the resource $r$ superscript.

For any pair of activities $a_i$ and $a_j$ that can compete for the same resource $r$ ($R_i \cap R_j \neq 0$), given the corresponding durations $p_{ir}$ and $p_{jr}$, the Dominance Conditions, describing the four main possible cases of conflict, are defined as follows:

1. $slack(e_i, s_j) < 0 \land slack(e_j, s_i) < 0$
2. $slack(e_i, s_j) < 0 \land slack(e_j, s_i) \geq 0 \land co-slack(e_j, s_i) < 0$
3. $slack(e_i, s_j) \geq 0 \land slack(e_j, s_i) < 0 \land co-slack(e_i, s_j) < 0$
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 way to order $a_i$ and $a_j$ taking into account the setup times $st_{ij}^r$ and $st_{ji}^r$, without inducing a negative cycle in the graph $G_d(V, E)$. When Condition 1 is verified there are four different interesting sub-cases generated on the basis of the cardinality of the domain sets $R_i$ and $R_j$.

- a. $|R_i| = |R_j| = 1$: the search has reached a failure state;
- b. $|R_i| = 1 \land |R_j| > 1$: the resource requirement $r$ can be removed from $R_j$;
- c. $|R_i| > 1 \land |R_j| = 1$: the resource requirement $r$ can be removed from $R_i$;
- d. $|R_i| > 1 \land |R_j| > 1$: the activities $a_i$ and $a_j$ cannot use the same resource $r$.

Conditions 2, and 3, alternatively, distinguish uniquely resolvable conflicts, i.e., there is only one feasible ordering of $a_i$ and $a_j$ when both the activities require $r$, and the decision of which constraint to post is thus unconditional. In the particular case where $|R_i| = |R_j| = 1$ the decision $a_j \leq a_i$ is mandatory; if Condition 2 is verified, only $a_j \leq a_i$ leaves $G_d(V, E)$ consistent. It is worth noting that the presence of the condition $co-slack(e_j, s_i) < 0$ 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 $st_{ij}^r$; hence, we still need to impose the constraint $e_j + st_{ij}^r \leq s_i$. 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 $a_i \leq a_j$ ordering is feasible. In case there is at least one activity with more than one resource option ($|R_i| > 1 \lor |R_j| > 1$), it is still possible to choose different resource assignments for $a_i$ and $a_j$, and avoid posting a precedence constraint. Condition 3 works similarly, and entails that only the $a_i \leq a_j$ ordering is feasible when $|R_i| = |R_j| = 1$.

Condition 4 designates a class of resolvable conflicts with more search options. In this case, when $|R_i| = |R_j| = 1$ both orderings between $a_i$ and $a_j$ remain feasible, and

---

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 $a_j$. 
it is therefore necessary to perform a search decision. When there is at least one activity \( a_i \) or \( a_j \) with more than one resource option (\(|R_i| > 1 \lor |R_j| > 1\)), then there is also the possibility of choosing different resource assignment to \( a_i \) and \( a_j \), and avoid to post a precedence constraint.

4.2 Heuristic Analysis

Shortest path information in \( G_d \) can also be exploited to define variable and value ordering heuristics for the decision variables \( x_i \) and \( o_{ijr} \) in all cases where no mandatory decisions are deduced from the propagation phase. The idea is to evaluate both types of decision variables \( (x_i \) and \( o_{ijr}) \) and select the one (independently of type) with the minimum heuristic evaluation. The selection of the variables is based on the most constrained first (MCF) principle and the selection of values follows the least constraining value (LCV) heuristic.

Ordering decision variables. We start to analyze the case of selecting an ordering decision variables \( o_{ijr} \), under the hypothesis that both the activity \( a_i \) and \( a_j \) use the same resource \( r \in R_i \cap R_j \). As stated above, in this context \( \text{slack}(e_i, s_j) \) and \( \text{slack}(e_j, s_i) \) provide measures of the degree of sequencing flexibility between \( a_i \) and \( a_j \). More precisely, given a variable \( o_{ijr} \), related to the pair \((a_i, a_j)\), its heuristic evaluation is \( \text{VarEval}(a_i, a_j) = \)

\[
\begin{cases}
\min\{\frac{\text{slack}(e_i, s_j)}{\sqrt{S}}, \frac{\text{slack}(e_j, s_i)}{\sqrt{S}}\} & \text{if } \text{slack}(e_i, s_j) \geq 0 \land \text{slack}(e_j, s_i) \geq 0 \\
\text{slack}(e_j, s_i) & \text{if } \text{slack}(e_i, s_j) < 0 \land \text{slack}(e_j, s_i) \geq 0 \\
\text{slack}(e_i, s_j) & \text{if } \text{slack}(e_i, s_j) \geq 0 \land \text{slack}(e_j, s_i) < 0.
\end{cases}
\]

where \( S = \min\{\text{slack}(e_i, s_j), \text{slack}(e_j, s_i)\} \). The variable ordering heuristic attempts to focus first on the most constrained conflict \((a_i, a_j)\), that is, on the conflict with the least amount of temporal flexibility (i.e., the conflict that is closest to previous Condition 1.a).

As opposed to variable ordering, the value ordering heuristic attempts to resolve the selected conflict \((a_i, a_j)\) by simply choosing the activity pair sequencing that retains the highest amount of temporal flexibility (least constrained value). Specifically, \( a_i \sim a_j \) is selected if \( \text{slack}(e_i, s_j) > \text{slack}(e_j, s_i) \) and \( a_j \sim a_i \) is selected otherwise.

Resource decision variables. Decision variables \( x_i \) are also selected according to the MCF principle. Initially, all pairs of activities \((a_i, a_j)\), such that \((|R_i| > 1 \lor |R_j| > 1 \text{ and } R_i \cap R_j \neq \emptyset)\) undergo a double-key sorting, where the primary key is a heuristic evaluation based on resource flexibility and computed as \( F_{ij} = 2(|R_i| + |R_j|) - |R_i \cap R_j| \), while the secondary key is the known \( \text{VarEval}(a_i, a_j) \) heuristic, based on temporal flexibility. Then, we select the pair \((a_i^*, a_j^*)\) with the lowest value of the

\[\]
pair $\langle F_{ij}, \text{VarEval}(a_i, a_j) \rangle$, where $\text{VarEval}(a_i, a_j)$ is computed for each possible resource $r \in R_i \cap R_j$. Finally, between $x_i^*$ and $x_j^*$ we select the variable whose domain of values has the lowest cardinality.

Value ordering on the decision variables $x_i$ is also accomplished by using temporal flexibility measures. If $R_i$ is the domain of the selected decision variable $x_i$, then for each resource $r \in R_i$, we consider the set of activities $A_r$ already assigned to resource $r$ and calculate the value $F_{min}(r) = \min_{a_k \in A_r} \{ \text{VarEval}(a_i, a_k) \}$. Then, for each resource $r$ we evaluate the flexibility associated with the most critical pair $(a_i, a_k)$, under the hypothesis that the resource $r$ is assigned to $a_i$. The resource $r^* \in R_i$ which maximizes the value $F_{min}(r)$, and therefore allows $a_i$ to retain maximal flexibility, is selected.

PCP($Problem, C_{max}$)
1. $S \leftarrow \text{InitSolution}(Problem, C_{max})$
2. loop
3. Propagate($S$)
4. if UnresolvableConflict($S$)
5. then return(nil)
6. else
7. if UniquelyResolvableDecisions($S$)
8. then PostUnconditionalConstraints($S$)
9. else begin
10. $C \leftarrow \text{ChooseDecisionVariable}(S)$
11. if ($C = \text{nil}$)
12. then return($S$)
13. else begin
14. $vc \leftarrow \text{ChooseValueConstraint}(S, C)$
15. PostConstraint($S, vc$)
16. end
17. end
18. end-loop
19. return $S$

Fig. 1. The PCP one-shot algorithm

4.3 The PCP Algorithm

Figure 1 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 procedure accepts a never-exceed value ($C_{max}$) of the objective function of interest, used to impose an initial global makespan to all the jobs. The PCP algorithm shown in Figure 1 analyses the decision variables $x_i$ and $o_{ijr}$ and, respectively, decides their values in terms of imposing a duration constraint on a selected activity or a setup time constraint (i.e., $a_i \preceq a_j$ or $a_j \preceq a_i$, see Section 3). In broad terms, the procedure in Figure 1 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
IFS(S, MaxFail, γ)

begin
1. S_{best} ← S
2. counter ← 0
3. while (counter ≤ MaxFail) do
4. RELAX(S, γ)
5. S ← PCP(S, C_{max}(S_{best}))
6. if C_{max}(S) < C_{max}(S_{best}) then
7. S_{best} ← S
8. counter ← 0
9. else
10. counter ← counter + 1
11. return (S_{best})
end

Fig. 2. The IFS schema

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 setup time constraint) in the graph $G_d$. In particular, Propagate(S) updates the shortest path distances on the graph $G_d$. A solution $S$ is found when the PCP algorithm finds a feasible assignment of resources $\tau_i \in R_i$ to activities $a_i (i = 1 \ldots n)$ and when none of the four dominance conditions is verified on $S$. In fact, when none of the four Dominance Conditions is verified (and the PCP procedure exits with success), for each resource $r$, the set of activities $A_r$ assigned to $r$ represents a total execution order. In addition, as the graph $G_d$ represents a consistent Simple Temporal Problem (see Section 3), one possible solution to the problem is the earliest-time solution, such that $S = \{(-d(s_1, t_{p0}), \tau_1), (-d(s_2, t_{p0}), \tau_2), \ldots, (-d(s_n, t_{p0}), \tau_n)\}$.

5 The Optimization Metaheuristic

Figure 2 introduces the generic IFS procedure. The algorithm basically alternates relaxation and flattening steps until a better solution is found or a maximal number of iterations is executed. The procedure takes three parameters as input: (1) an initial solution $S$; (2) a positive integer MaxFail, which specifies the maximum number of consecutive non makespan-improving moves that the algorithm will tolerate before terminating; (3) a parameter $\gamma$, representing the selection probability of an activity for removal (relaxing factor), as explained in 5.1. After the initialization (Steps 1-2), a solution is repeatedly modified within the while loop (Steps 3-10) by applying the RELAX procedure (as explained in the following section), and the PCP procedure shown in Figure 1 used as flattening step. At each iteration, the RELAX step reintroduces the possibility of resource contention, and the PCP step is called again to restore resource feasibility. In the case a better makespan solution is found (Step 6), the new solution is saved in $S_{best}$ and the counter is reset to 0. If no improvement is found within MaxFail moves, the algorithm terminates and returns the best solution found.
5.1 Relaxation Procedure

The first part of the IFS cycle is the relaxation step, wherein a feasible schedule is relaxed into a possibly resource infeasible, but precedence feasible, schedule by retracting some scheduling decisions. Here we use a strategy similar to the one in [12] and called \textit{chain-based relaxation}. Given the graph representation described above, each scheduling decision is either a setup time constraint between a pair of activities that are competing for the same resource capacity and/or a resource assignment to one activity. The strategy starts from a solution \( S \) and randomly \textit{breaks} some total orders (or \textit{chains}) imposed on the subset of activities requiring the same resource \( r \). The relaxation strategy requires an input solution as a graph \( G_S(A, J, X_S) \) which (Section 3) is a modification of the original precedence graph \( G \) that represents the input scheduling problem. \( G_S \) contains a set of additional general precedence constraints \( X_S \) which can be seen as a set of \textit{chains}. Each chain imposes a total order on a subset of problem activities requiring the same resource.

The \textit{chain-based relaxation} proceeds in two steps. Firstly, a subset of activities \( a_i \) is randomly selected from the input solution \( S \), according to some criteria that will be explained below. The selection process is generally driven by a parameter \( \gamma \in (0, 1) \) that indicates the probability that each activity has to be selected (\( \gamma \) is called the \textit{relaxing factor}). For each selected activity, the resource assignment is removed and the original set of available options \( R_i \) is re-established. Secondly, a procedure similar to \textsc{Chain-ing} – used in [22] – is applied to the set of unselected activities. This operation is in its turn accomplished in three steps: (1) all previously posted setup time constraints \( X_S \) are removed from the solution \( S \); (2) the unselected activities are sorted by increasing earliest start times of the input solution \( S \); (3) for each resource \( r \) and for each unselected activity \( a_i \) assigned to \( r \) (according to the increasing order of start times), \( a_i \)'s predecessor \( p = \text{pred}(a_i, r) \) is considered and the setup time constraint related to the sequence \( p \preceq a_i \) is posted (the dummy activity \( a_0 \) is the first activity of all the chains). This last step is iterated until all the activities are linked by the correct setup time constraints. Note that this set of unselected activities still represents a feasible solution to a scheduling sub-problem, which is represented as a graph \( G_S \) in which the randomly selected activities float outside the solution and thus re-create conflict in resource usage.

As anticipated above, we implemented two different mechanisms to perform the random activity selection process, respectively called \textit{Random} and a \textit{Slack-based}.

\emph{Random selection} According to the random selection approach, at each solving cycle of the IFS algorithm in Figure 2, a subset of activities \( a_i \) is randomly selected from the input solution \( S \), with each activity having an uniformly distributed selection probability equal to \( \gamma \). It is of great importance to underscore that according to this approach, the activities to be relaxed are randomly picked up from the solution \( S \) \textit{with the same probability} which, as we will see shortly, entails a relaxation characterized by a greater disruption on \( S \), compared to the following selection approach.

\emph{Critical Path-biased selection} As opposed to the random selection, at each iteration the critical path-biased selection approach restricts the pool of the relaxable activities to the subset containing those activities that are closer to the \textit{critical path condition} (critical
As known, an activity \( a_i \) belongs to the critical path (i.e., meets the critical path condition) when, given \( a_i \)'s end time \( e_i \) and its feasibility interval \([lb_i, ub_i]\), the condition \( lb_i = ub_i \) holds. For each activity \( a_i \), the smaller the difference \( ub_i - lb_i \) computed on \( e_i \), the closer is \( a_i \) to the critical path condition. At each IFS iteration, the critical path set is built so as to contain any activity \( a_i \) with a probability directly proportional to the \( \gamma \) parameter and inversely proportional to the \( ub_i - lb_i \) value. For obvious reasons, the critical path-biased relaxation entails a smaller disruption on the solution \( S \), as it operates on a smaller set of activities; the activities that are farther from the critical path condition will have a minimum probability to be selected. As explained in the following section, this difference has important consequences on the experimental behavior.

### 6 Experimental Analysis

The empirical evaluation has been carried out on a SDST-FJSSP benchmark set synthesized on purpose out of the first 20 instances of the edata subset of the FJSSP HUdata testbed from [15], and will therefore be referred to as SDST-HUdata. Each one of the SDST-HUdata instances has been created by adding to the original HUdata instance one Setup-Time matrix \( st^r(nJ \times nJ) \) for each present machine \( r \), where \( nJ \) is the number of present jobs. Without loss of generality, the same randomly generated Setup-Time matrix was added for each machine of all the benchmark instances. Each value \( st^r_{ij} \) in the Setup-Time matrix models the setup time necessary to reconfigure the \( r \)-th machine to switch from job \( i \) to job \( j \). Note that machine reconfiguration times are sequence dependent: setting up a machine to process a product of type \( j \) after processing a product of type \( i \) can generally take a different amount of time than setting up the same machine for the opposite transition. The elements \( st^r_{ij} \) of the Setup-Time matrix satisfy the triangle inequality [7, 4], that is, for each three activities \( a_i, a_j, a_k \) requiring the same machine, the inequality \( st^r_{ij} \leq st^r_{ik} + st^r_{kj} \) holds. The 20 instances taken from HUdata (namely, the instances la01-la20) are divided in four groups of five \((nJ \times nA)\) instances each, where \( nJ \) is the number of jobs and \( nA \) is the number of activities per job for each instance. More precisely, group la01-la05 is \((10 \times 5)\), group la06-la10 is \((15 \times 5)\), group la11-la15 is \((20 \times 5)\), and group la16-la20 is \((10 \times 10)\). In all instances, the processing times on machines assignable to the same activity are identical, as in the original HUdata set. The algorithm used for these experiments has been implemented in Java and run on a AMD Phenom II X4 Quad 3.5 Ghz under Linux Ubuntu 10.4.1.

**Results.** Table 1 and table 2 show the obtained results running our algorithm on the SDST-HUdata set using the Random or Slack-based procedure in the IFS relaxation step, respectively. Both tables are composed of 10 columns and 23 rows (one row per problem instance plus three data wrap-up rows). The best column lists the shortest makespans obtained in the experiments for each instance; underlined values represent the best values obtained from both tables (global bests). The columns labeled \( \gamma = 0.2 \) to \( \gamma = 0.9 \) (see Section 4) contain the results obtained running the IFS procedure with a different value for the relaxing factor \( \gamma \). For each problem instance (i.e., for each row) the values in bold indicate the best makespan found among all the tested \( \gamma \) values (\( \gamma \) runs).
Table 1. Results with random selection procedure

<table>
<thead>
<tr>
<th>inst.</th>
<th>best</th>
<th>γ</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0.2</td>
</tr>
<tr>
<td>la01</td>
<td>726</td>
<td>726</td>
</tr>
<tr>
<td>la02</td>
<td>749</td>
<td>749</td>
</tr>
<tr>
<td>la03</td>
<td>689</td>
<td>689</td>
</tr>
<tr>
<td>la04</td>
<td>673</td>
<td>673</td>
</tr>
<tr>
<td>la05</td>
<td>603</td>
<td>603</td>
</tr>
<tr>
<td>la06</td>
<td>950</td>
<td>950</td>
</tr>
<tr>
<td>la07</td>
<td>916</td>
<td>916</td>
</tr>
<tr>
<td>la08</td>
<td>954</td>
<td>954</td>
</tr>
<tr>
<td>la09</td>
<td>1002</td>
<td>1002</td>
</tr>
<tr>
<td>la10</td>
<td>977</td>
<td>977</td>
</tr>
<tr>
<td>la11</td>
<td>1365</td>
<td>1365</td>
</tr>
<tr>
<td>la12</td>
<td>1088</td>
<td>1088</td>
</tr>
<tr>
<td>la13</td>
<td>1255</td>
<td>1255</td>
</tr>
<tr>
<td>la14</td>
<td>1292</td>
<td>1292</td>
</tr>
<tr>
<td>la15</td>
<td>1298</td>
<td>1298</td>
</tr>
<tr>
<td>la16</td>
<td>1012</td>
<td>1012</td>
</tr>
<tr>
<td>la17</td>
<td>864</td>
<td>864</td>
</tr>
<tr>
<td>la18</td>
<td>985</td>
<td>985</td>
</tr>
<tr>
<td>la19</td>
<td>956</td>
<td>956</td>
</tr>
<tr>
<td>la20</td>
<td>997</td>
<td>997</td>
</tr>
<tr>
<td>(2(N))</td>
<td>12</td>
<td>6.3</td>
</tr>
<tr>
<td>Av.C.</td>
<td>20.419</td>
<td>17.591</td>
</tr>
<tr>
<td>Av.MRE</td>
<td>19.34</td>
<td>18.29</td>
</tr>
</tbody>
</table>

For each γ run, the last three rows of both tables show respectively (up-bottom): (1) the number \(B\) of best solutions found locally (i.e., within the current table) and, underlined within round brackets, the number \(N\) of best solutions found globally (i.e., between both tables); (2) the average number of utilized solving cycles (Av.C.), and (3) the average mean relative error (Av.MRE)\(^6\) with respect to the lower bounds of the original \(HU\) data set (i.e., without setup times), reported in [16]. For all runs, a maximum CPU time limit was set to 800 seconds.

One significant result that the tables show is the difference in the average of utilized solving cycles (Av.C. row) between the random and the slack-based relaxation procedure. In fact, it can be observed that on average the slack-based approach uses more solving cycles in the same allotted time than its random counterpart (i.e., the slack-based relaxation heuristic is faster in the solving process). This is explained by observing that the slack-based relaxation heuristic entails a less severe disruption of the current solution at each solving cycle compared to the random heuristic, as the former generally relaxes a lower number of activities (given the same γ value). The lower the disruption level of the current solution in the relaxation step, the easier it is to re-gain solution feasibility in the flattening step. In addition of this efficiency issue, the slack-based relaxation approach also provides the extra effectiveness deriving from operating in the vicinity of the critical path of the solution, as demonstrated in [8].

The good performance exhibited by the slack-based heuristic can be also observed by inspecting the \(B(N)\) rows in both tables. Clearly, the slack-based approach finds a

\(^6\) The individual MRE of each solution is computed as follows: \(MRE = 100 \times \frac{(C_{\text{max}} - LB)}{LB}\), where \(C_{\text{max}}\) is the solution makespan and \(LB\) is the instance’s lower bound.
higher number of best solutions (17 against 12), which is confirmed by comparing the number of locally found bests ($B$) with the global ones ($N$), for each $\gamma$ value, and for both heuristics.

Another interesting aspect can be found analyzing the $\gamma$ values range where the best performances are obtained ($Av.MRE$ row). Inspecting the $Av.MRE$ values, the following can in fact be stated: (1) the slack-based heuristic finds solutions of higher quality w.r.t. the random heuristic over the complete $\gamma$ variability range; (2) in the random case, the best results are obtained in the $[0.3, 0.5]$ $\gamma$ range, while in the slack-based case the best $\gamma$ range is wider ($[0.3, 0.6]$).

### 7 Conclusions

In this paper we have proposed the use of Iterative Flattening Search (IFS) as a means of effectively solving the SDST-FJSSP. The proposed algorithm uses as its core solving procedure an extended version of the SP-PCP procedure proposed by [21] and a new relaxation strategy targeted to the case of SDST-FJSSP. The effectiveness of the procedure was demonstrated on 20 modified instances of the $\text{edata}$ subset of the FJSSP $\text{HUdata}$ testbed from [15], a well known and difficult Flexible Job Shop Scheduling benchmark set. In particular, we show as the new slack-based relaxation strategy exhibits better performance than the random selection one. Further improvement of the current algorithm may be possible by incorporating additional heuristic information and search mechanisms. One of the next steps will be the collection of the benchmarks proposed in the cited works [23, 24, 26, 27], although no one of the problems proposed
in these papers coincides with the SDST-FJSSP, basically they can be seen as slight variations of this problem, hence the proposed IFS procedure can be adapted to solve an interesting and large class of flexible manufacturing scheduling problems. This will be the focus of our future work together the realization of a web repository to collect all the interesting benchmark sets.

Acknowledgments

CNR authors are partially supported by EU under the ULISSE project (Contract FP7.218815), and MIUR under the PRIN project 20089M932N (funds 2008).

References