Iterative Improvement Strategies for Multi-Capacity Scheduling Problems

Angelo Oddi and **Amedeo Cesta**
ISTC-CNR
Rome, Italy
{*name.surname*}*@istc.cnr.it*

Nicola Policella
European Space Agency
Darmstadt, Germany
nicola.policella@esa.int

Stephen F. Smith
The Robotics Institute
Carnegie Mellon University,USA
sfs@cs.cmu.edu

Abstract

Iterative Flattening Search (IFS) is an iterative improvement heuristic schema for solving scheduling problems with a makespan minimization objective. 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 incrementally recomputed (flattening-step). Since its introduction, several variations of IFS have been proposed over the original strategy. Such variations involve both different strategies for the relaxation step and for the incremental solving procedure. This paper investigates a missing point in the related literature and present initial results of a uniform study to evaluate the effectiveness of the "single component" strategies among those proposed till now. This paper introduces a framework to combine and experimentally evaluate different IFS strategies. Specifically, we examine the utility of: (i) operating with different relaxation strategies; (ii) using different strategies to built a new solution. We evaluate these extensions on benchmark instances of the Multi-Capacity Job-Shop Scheduling Problem (MCJSSP) also used in previous IFS studies. The experimental results shed light on the weaknesses and the strengths of the ideas proposed over the past years and suggest potentials for more effective IFS procedures.

Introduction

Iterative Flattening Search (IFS or IFLAT) (Cesta, Oddi, & Smith 2000) is an iterative improvement heuristic schema for solving scheduling problems with a makespan minimization objective. 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 incrementally recomputed (flattening-step). IFLAT performance was measured on a set of challenging Multi-Capacity Job-Shop Scheduling Problem (MCJSSP) instances. Since its original introduction, several variations have been proposed. Two works in particular (Michel & Van Hentenryck 2004; Godard, Laborie, & Nuitjen 2005) have extended the performance of the original paper through refinement of the basic IFLAT search schema. (Michel & Van Hentenryck 2004) identified an anomaly in IFLAT search and proposed a simple extension, which dramatically improved the quality of its schedules while preserving its computational efficiency. The key idea was to iterate the relaxation step multiple times, the resulting algorithm found many new upper bounds and produced solutions within 1% of the best upper bounds on average. Additional improvements were obtained in (Godard, Laborie, & Nuitjen 2005) with an approach which follows the same schema of IFLAT but uses different engines for the flattening and relaxation steps. Such a procedure was able to find additional optimal solutions and furtherly improved known upper-bounds for MCJSSP benchmarks.

The different IFS proposals involve both different strategies for the relaxation step and for the incremental solving procedure. However, till now, an uniform study to evaluate the effectiveness of the "single component" strategies were an open issue. To this purpose this paper proposes an uniform framework to combine and experimentally evaluate different IFS strategies. Specifically, we examine the utility of:

i) operating with different relaxation strategies, one targeted on removing decisions on the solution critical path and another one considering the whole solution;

ii) using different strategies to built a new solution, one posting precedence constraints among the activities and another one based on setting the start time of the activities.

We present here some initial experimental results that shed light on the relative weaknesses and strengths of the previously proposed strategies and start suggesting more effective and efficient IFS procedures.

The paper is organized as follows. We first review the iterative flattening search schema as evolved from previous work. The central part of the paper introduces the available relaxation and solving strategies and present a framework for comparing all of them. Next we recall the MCJSSP problem domain and benchmark problem sets used in our evaluation. Performance results are then given that demonstrate the leverage provided by the extended search procedures. We conclude by briefly discussing further opportunities to extend and enhance the basic iterative flattening search concept.

Problem Representation

Before describing the iterative flattening approach we introduce the modeling perspective on which this schema is based. According to this a scheduling problem is represented as a directed graph $G(A, E)$. A is the set of activities

specified in MCJSSP, plus a dummy activity a_{source} temporally constrained to occur before all others and a dummy activity a_{sink} temporally constrained to occur after all others. E is the set of precedence constraints defined between activities in A.

In the IFS search schema the reference representation of a solution S is given as an extended graph G_S of G, such that an additional set of precedence constraints is added to the original problem representation. This means that the set E is partitioned in two subsets, $E = E_{prob} \cup E_{post}$, where E_{prob} is the set of precedence constraints originating from the problem definition, and E_{post} is the set of precedence constraints posted to resolve resource conflicts. In general the directed graph $G_S(A, E)$ represents a set of temporal solutions. The set E_{post} is added in order to guarantee that at least one of those temporal solutions is also resource feasible.

In searching for a solution different search strategies apply. Following a *Precedence Constraint Posting* (PCP) approach the set of E_{post} is naturally created reasoning on contention peaks. For example in (Cesta, Oddi, & Smith 1998; 2000; 2002) the precedences are selected by a basic Earliest Start Time Algorithm (ESTA). ESTA is designed to address more general, multi-capacity scheduling problems with generalized precedence relations between activities (i.e., corresponding to metric separation constraints with minimum and maximum time lags) but is used also in (Cesta, Oddi, & Smith 2000; Michel & Van Hentenryck 2004) to deal with multi-capacity resource contention peaks in MCJSSPs. Others use a different strategy, the decision to Set a Start Time (SST) of an activity, by imposing a *rigid* temporal constraint between the a_{source} and the start-time of the interested activity. This strategy is very common in the scheduling literature, for example it is used in (Godard, Laborie, & Nuitjen 2005) and many others.

The original Iterative Flattening Search procedure (Cesta, Oddi, & Smith 2000) iterates two steps:

Relaxation step: a feasible schedule is relaxed into a possibly resource infeasible, but precedence feasible, schedule by removing some search decisions represented as precedence constraints between pair of activities;

Flattening step: a sufficient set of new precedence constraints is posted to re-establish a feasible schedule.

This schema integrates naturally with the graph representation G_S for a solution, that is the solution representation used in a PCP approach.[1] To apply the same philosophy with an SST we need to relax the "solution rigidity" introduced by the absolute temporal constraints inserted as decisions. One way of doing it consists in transforming the SST solution in a Partial Order Schedules (POS) (Cesta, Oddi, & Smith 1998; Policella *et al.* 2004).

In general a POS can be also obtained from a PCP solution produced by ESTA. Both G_S and POS are graph representations. The difference is that while ESTA solutions guarantee that at least one of the temporal solutions they represent is also resource feasible, a POS guarantees that *all* delineated temporal solutions are also resource feasible. The use of a POS in general increases the possibilities for rearranging relaxed activities. This is why we dedicate a short paragraph to POS basics.

[1] This representation is sometimes called *Flexible Schedule*.

Partial Order Schedules

The common thread underlying a POS is the characteristic that activities which require the same resource units are linked via precedence constraints into precedence *chains*. Given this structure, each constraint becomes more than just a simple precedence. It also represents a *producer-consumer* relation, allowing each activity to know the precise set of predecessors that will supply the units of resource it requires for execution. In this way, the resulting network of chains can be interpreted as a flow of resource units through the schedule; each time an activity terminates its execution, it passes its resource unit(s) on to its successors. It is clear that this representation is flexible if and only if there is temporal slack that allows chained activities to move "back and forth". Polynomial methods for producing a POS from an input solution represented as a precedence graph (or equivalently as a set of start times) have been introduced in (Cesta, Oddi, & Smith 1998; Policella *et al.* 2004). Given an input solution, a transformation method, named *Chaining*, is defined that proceeds to create sets of chains of activities. This operation can be accomplished over three steps: (1) all the previously posted leveling constraints are removed from the input partial order; (2) the activities are sorted by increasing activity earliest start times; (3) for each resource and for each activity a_i (according to the increasing order of start times), one or more predecessors a_j are chosen, which supplies the units of resource required by a_i – a precedence constraint (a_i, a_j) is posted for each predecessor a_j. The last step is iterated until all the activities are linked by precedence chains.

Having a flexible solution is not the only benefit in considering the use of partial order schedules. A second property that appears to be relevant is the reduction in the number of additional precedence constraints that must be posted to obtain a solution: given a problem with n activities to be scheduled, the number of constraints appearing in the solution is always $O(n)$. This because the chaining procedure creates POSs with only the "necessary" precedence constraints, and eliminates all "redundant" constraints. For example applying POS to a G_S result on a PCP version of IFLAT removes redundant precedence constraints and tends to intensify the effect of the IFLAT relaxation step in the procedure. More solutions are accessible at each flattening cycle,because the removal of redundant constraints increases possibilities for rearranging relaxed activities. In (Godard, Laborie, & Nuitjen 2005) a POS is created from an SST solution to insert temporal flexibility in the solution before performing a relaxation step.

Iterative Flattening Search

Given these preliminaries we introduce a general IFSSEARCH procedure in Figure 1. The algorithm basically alternates Relaxation and Flattening steps until a better so-

IFSSEARCH(S,$MaxFail$)
begin
1. $S_{best} \leftarrow S$
2. $counter \leftarrow 0$
3. **while** ($counter \leq MaxFail$) **do**
4. RELAX(S)
5. $Sol \leftarrow$FLATTEN(S)
6. **if** Mk(Sol) < Mk(S_{best}) **then**
7. $S_{best} \leftarrow S$
8. counter $\leftarrow$ 0
9. **else**
10. counter $\leftarrow$ counter + 1
11. **return** (S_{best})

end

Figure 1: The IFSSEARCH general schema

lution is found or a maximal number of iterations is executed. The procedure takes two parameters as input: (1) a starting solution S; (2) a positive integer $MaxFail$ which specifies the maximum number of non-makespan-improving moves that the algorithm tolerates before termination. After initialization (Steps 1-2), a solution is repeatedly modified within the while loop (Steps 3-10) by the application of the RELAX and FLATTEN procedures. In the case that a better makespan solution is found (Step 6), the new solution is stored in S_{best} and the *counter* is reset to 0. Otherwise, if no improvement is found in $MaxFail$ moves, the algorithm terminates and returns the best solution found. Our goal is to create a uniform implementation framework in which integrate procedures from the various IFS papers mentioned till now. The idea is to decompose effects of parts of the algorithms and hopefully understand how effectiveness of parts influences effectiveness of complete algorithm.[2] In the rest of this section different relaxations and the flattening steps are introduced in more detail.

Relaxation Procedures

In general, a relaxation procedure transforms a feasible schedule into a possibly resource infeasible, but temporal feasible, schedule by adopting different strategies for removing some search decisions. We have reproduced two of these strategies. The first, introduced in the paper (Cesta, Oddi, & Smith 2000; Michel & Van Hentenryck 2004), which removes precedence constraints between pair of activities on the critical path of the solution, hence we call it *pc-based relaxation*; the second, introduced in the work (Godard, Laborie, & Nuitjen 2005), which starting from a POS-form solution, basically randomly *breaks* some chains in the input POS schedule, hence the name *chain-based relaxation*.

[2]Even if we are trying to reproduce different approaches in a uniform software framework, it is worth underscoring that, in the present stage we do not have a complete re-production of the algorithm (Godard, Laborie, & Nuitjen 2005) whose engineering aspects are well customized within the ILOG framework. For example, we do not have any of the resource propagation rules of that environment. We rather have several components inspired by what is possible to reconstruct from the (Godard, Laborie, & Nuitjen 2005) paper. In our analysis the common implementation base is useful to explore approaches and contributions to problem solving by different components.

PCRELAX($S, p_r, MaxRlxs$)
begin
1. **for** 1 **to** $MaxRlxs$
2. **forall** $(a_i, a_j) \in$ CriticalPath(S) $\cap E_{post}$
3. **if** random(0,1) < p_r
4. $S \leftarrow S \setminus (a_i, a_j)$

end

Figure 2: pc-based relaxation procedure

CHAINRELAX(S, p_r)
begin
1. **for** k = 1 **to** n
2. **if** random(0,1) < p_r **then**
3. Remove the edges (a_p, a_k), $a_p \in pred(a_k) \cap E_{ch}$ and (a_k, a_s), $a_s \in succ(a_k) \cap E_{ch}$
4. Apply the CHAINING procedure to the subset of unselected activities

end

Figure 3: Chain-based relaxation procedure

Precedence relaxation. The relaxation step is based on the concept of *critical path*. A *path* in $G_S(A, E)$ is a sequence of activities $a_1, a_2, \ldots, a_k$, such that, $(a_i, a_{i+1}) \in E$ with $i = 1, 2, \ldots, (k-1)$. The length of a path is the sum of the activities processing times and a *critical path* is a path from a_{source} to a_{sink} which determines the solution's makespan. Any improvement in makespan will necessarily require change to some subset of precedence constraints situated on the *critical path*, since these constraints collectively determine the solution's current makespan. Following this observation, the relaxation step introduced in (Cesta, Oddi, & Smith 2000) is designed to retract some number of posted precedence constraints on the solution's critical path. Figure 2 shows the PCRELAX procedure. Steps 2-4 consider the set of posted precedence constraints ($pc_i \in E_{post}$), which belong to the current critical path. A subset of these constraints is randomly selected on the basis of the parameter $p_r \in (0, 1)$ and then removed from the current solution. Step 1 represents the crucial difference between the approaches of (Cesta, Oddi, & Smith 2000) and (Michel & Van Hentenryck 2004). In the former approach Steps 2-4 are performed only once (i.e., $MaxRlxs = 1$), whereas in (Michel & Van Hentenryck 2004) these steps are iterated several times (from 2 to 6), such that, a new critical path of S is computed at each iteration. Notice that this path can be completely different from the previous one. This allows the relaxation step to also take into account those paths that have a length very close to the one of the critical path.

Chain Relaxation. This second relaxation requires an input solution in POS-form. A solution in POS form is an extension of the original precedence graph representing the

```
PCPS(P, S)
begin
1.  Propagate(S)
2.  if IsSolution(S)
3.      then return(S)
4.      else
5.          mcs ← SelectConflict(P,S)
6.          if Solvable(mcs, S)
7.              then
8.                  pc ← Choose Precedence(S, mcs)
9.                  MCSS(P, S ∪ {pc})
10.             else return(fail)
end
```

Figure 4: The PCPS algorithm

input scheduling problem. As previously introduced, a POS form solution is a graph $G_S(N, E_{prob} \cup E_{ch})$, such that the set $E = E_{prob} \cup E_{ch}$ is partitioned into a set of *chains* $CH_1, CH_2, \ldots, CH_{nc}$. Each chain CH_i imposes a total order on a subset of problem activities requiring the same resource. Hence, given a generic activity a_k, $pred(a_k) = \{a_p | \exists\, CH : (a_p, a_k) \in CH\}$ is the set of its predecessor activities and $succ(a_k) = \{a_s | \exists\, CH : (a_k, a_s) \in CH\}$ is the set of its successors activities. In particular, $pred(a_{source}) = succ(a_{sink}) = \emptyset$ Figure 3 shows the chain-based relaxation procedure. The procedure (a) randomly selects a subset of activities from the input solution S on the basis of the parameter $p_r \in (0, 1)$, (b) removes the edges (a_p, a_k), $a_p \in pred(a_k)$ and (a_k, a_s), $a_s \in succ(a_k)$ without updating the start times est_i of the activities; (c) the *Chaining* procedure (previously described) is applied on the set of unselected activities, that is, the activities not removed by the random selection. It is worth observing that such activities still represents a feasible solution to a scheduling sub-problem, which can be transformed in POS-form, in which the randomly selected activities *float* outside the solution thus re-creating *contention peaks*.

Flattening Procedures

Both relaxation schema create a solution with contention peaks that should be *flattened*. We have implemented two general solution schema, one based on the PCP idea, the second on the SST strategy. Both solving algorithms are able to perform a complete search through backtracking.

PCP Search (PCPS). The flattening step (see Figure 4) used in (Cesta, Oddi, & Smith 2000) is inspired by prior work on the Earliest Start Time Algorithm (ESTA) from (Cesta, Oddi, & Smith 1998). The algorithm is a variant of a class of PCP scheduling procedures characterized by a two-phase solution generation process. The first step *constructs an infinite capacity solution*. The current problem is formulated as an STP (Dechter, Meiri, & Pearl 1991) temporal constraint network[3] where temporal constraints are modeled and satisfied (via constraint propagation) but resource constraints are ignored, yielding a time feasible solution that assumes infinite resource capacity. The second step *levels resource demand by posting precedence constraints*. Resource constraints are super-imposed by projecting "resource demand profiles" over time. Detected resource conflicts, which are Minimal Conflict Sets (MCS) as in (Cesta, Oddi, & Smith 2002), are then resolved by iteratively posting simple precedence constraints between pairs of competing activities. The constraint posting process of ESTA is based on the Earliest Start Solution (ESS) consistent with currently imposed temporal constraints. It then proceeds to compute a resource conflict (Step 2-5). If this set is empty the ESS is also resource feasible and a solution is found; otherwise if a conflict exists that can be solved, a new precedence constraint is posted to do so (Steps 8-9); otherwise the process fails (Step 10). For further details on the functions *SelectConflict()*, and *ChoosePrecedence()* (non deterministic version of the precedence selection operator) the reader should refer to the original references.

SST Search (SSTS). The second solving procedure is based on the idea of searching the set of possible assignments to the activity start-times. In particular, our implementation of SSTS can be seen as a *serial scheduling schema* (Kolisch 1996) adopting the *latest finish time* (LFT) priority rule, which branches the search on the possible earliest start times (Dorndorf, Pesch, & Phan Huy 2000). However, other search schemas are possible, with different priority rules, this will be motivation for further experiments in the near future. A recursive and non deterministic version of the solver is shown in Figure 5. At Step 1 the procedure *Propagate* propagates the current temporal constraints. In particular, for each activity a_i updates its earliest stat-time est_i and latest finish time lft_i of the activities. When the output solution S is a complete and resource feasible solution (all the activities has a start-time assigned), the procedure returns it (Steps 2-3). Otherwise an activity is selected on the basis of a *priority rule*. Currently, we select the activity with the minimal latest finish time lft (ties are broken by the est values). Given a selected activity a_i, the search *branches* (Step 8) on the possible resource feasible assignments of the earliest start-time est_i.

Iterative Flattening Variants

The concept of Iterative Flattening introduced in (Cesta, Oddi, & Smith 2000) is quite general and provided an interesting new basis for designing more sophisticated and effective local search procedures for scheduling optimization. The IFLATRELAX procedure proposed in (Michel & Van Hentenryck 2004) is a nice example of an IFLAT extension which obtains substantial improvements over its original version. In addition, the version of Iterative Flattening proposed in (Godard, Laborie, & Nuitjen 2005) produced further improvements on both the previous procedures. This

[3] In a STP (Simple Temporal Problem) network we make the following representational assumptions: temporal variables (or time-points) represent the start and end of each activity, and the beginning and end of the overall temporal horizon; distance constraints represent the duration of each activity and separation constraints between activities including simple precedences.

```
SSTS(P, S)
begin
1.  Propagate(S)
2.  if IsSolution(S)
3.      then return(S)
4.      else
5.          a_i ← SelectActivity(P,S)
6.          if ExistFeasibleEST(a_i, S)
7.              then
8.                  est_i ← ChooseEST(S, a_i)
9.                  SSTS(P, S ∪ est_i)
10.             else return(fail)
end
```

Figure 5: The SSTS algorithm

procedure uses a solving strategy similar to SSTS and a chain-based relaxation schema.[4]

However, till now, an uniform study to evaluate the effectiveness of the *single component strategies* proposed in the literature were an open issue. In this spirit, the paper proposes an uniform framework to combine and experimentally evaluate different IFS strategies. Specifically, we examine the utility of: (i) operating with different relaxation strategies; (ii) using different strategies to built a new solution. Hence, our idea is to shed light on the weaknesses and the strengths of the ideas proposed over the past years and suggest more effective and efficient IFS procedures. According to this idea we propose the following IFS procedures:

- Two procedures based on PCPS search, one uses the precedence relaxation – identified with PCs – and another one the chain relaxation - identified with ACTs. PCPS is implemented as a depth-first backtracking procedure using an input parameter α, which is used to limit the number of backtracking steps. In particular, the PCPS procedure returns the solution found with minimal makespan, within αn steps, where n is the number of problem's activities. We observe that the combination of PCPS and *PC-based* relaxation with $\alpha = 0$ reproduces the algorithm in (Michel & Van Hentenryck 2004) furtherly extended with a backtracking search procedure.
- Two IFS procedures based on SSTS search, one with precedence relaxation – called SSTS-PCs – and another one with chain relaxation – called SSTS-ACTs. Also in this case, SSTS search uses the same parameter α to bound the number of backtracking steps to the value αn and returns the best solution found with regard to the makespan.
- A new IFS procedure – called PCPS-ACTs-iPCs – which coincides with the combination of PCPS and *Chain-based* relation, except when an improved solution is found within the Iterative Flattening loop (see Steps 3-10 in Figure 1). In this case the relaxation procedure is temporary switched to the *precedence based* one.
- A new IFS procedure –called SSTS-ACTs-iPCs – which mirrors the previous one on the relaxation strategy, but uses the SSTS search procedure.

As introduced above, the main goal of this paper is perform a first uniform study for evaluating the strengths and the weaknesses of the single IFS component strategies. In particular, the first four IFS strategies, basically combines already know procedures, even if two of them (PCPS-ACTs and SSTS-PCs) are relatively new algorithms. Whereas, the last two procedures, proposes two new algorithm based on the following intuition. We observe, the *PC-based* relaxation is more targeted on directly reducing the makespan of a solution, because specifically relaxes its critical path, which is directly correlated to the solution's makespan. However, such procedure seems also more incline to be trapped in a *local minima*. On the contrary, the *Chain-based* relaxation removes activities independently from the critical path, hence it promotes a search with an higher degree of *diversification*. The last two IFS procedures are two attempts to interleave intensification and diversification mechanisms within the same IFS procedure in order to improve performance. In the next section, after a short summary on the used benchmarks, we propose a first empirical evaluation of the procedures defined in this section.

The MCJSSP Scheduling Problem

We consider the Multi-Capacity Job-Shop Scheduling Problem, MCJSSP, as a basis for evaluating the performance of our search procedures. This problem involves synchronizing the use of a set of resources $R = \{r_1 \dots r_m\}$ to perform a set of jobs $J = \{j_1 \dots j_n\}$ over time. The processing of a job j_i requires the execution of a sequence of m activities $\{a_{i_1} \dots a_{i_m}\}$, each a_{ij} has a constant processing time p_{ij} and requires the use of a single unit of resource $r_{a_{ij}}$ for its entire duration. Each resource r_j is required only once in a job and can process at most c_j activities at the same time ($c_j \geq 1$). A *feasible solution* to a MCJSSP is any temporally consistent assignment to the activities' start times which does not violate resource capacity constraints. An *optimal solution* is a feasible solution with minimal overall duration or makespan. Generally speaking, MCJSSP has the same structure as JSSP but involves multi-capacitated resources instead of unit-capacity resources.

Benchmark Sets

For our analysis, we refer to the benchmarks introduced in (Nuijten & Aarts 1996). They consist of four sets of problems which are derived from the Lawrence job-shop scheduling problems (Lawrence 1984) by increasing the number of activities and the capacity of the resources.

Set A: *LA1-10x2x3* (Lawrence's problems numbered 1 to 10, with resource capacity duplicated and triplicated). Using the notation #jobs × #resources (resource capacity), this set consists of 5 problems of sizes 20x5(2), 30x5(3), 30x5(2), 45x5(3).

[4] It is worth noting that at present neither PCPS nor SSTS include any resource propagation algorithm (e.g., timetabling, edge finding, etc.).

Set B: *LA11-20x2x3*. 5 problems each of sizes 40x5(2), 60x5(3), 20x10(2), 30x10(3).

Set C: *LA21-30x2x3*. 5 problems each of sizes 30x10(2), 45x10(3), 40x10(2), 60x10(3).

Set D: *LA31-40x2x3*. 5 problems each of sizes 60x10(2), 90x10(3), 30x15(2), 45x15(3).

We observe that the proposed benchmark set still represents a challenging benchmark for comparing algorithms. In fact, (a) in relatively few instances they cover a wide range of problem sizes; (b) they also provide a direct basis for comparative evaluation. In fact, as noted in (Nuijten & Aarts 1996), one consequence of the problem generation method is that the optimal makespan for the original JSSP is also a tight upper bound for the corresponding MCJSSP (Lawrence upper bounds). Hence, even if for many instances there are known better solutions, distance from these upper-bound solutions can provide a useful measure of solution quality.

Current Experimental Results

This section proposes a first explorative evaluations of the IFS procedures introduced in the previous sections. In this phase of our work, we are using the Set C benchmark, which is a quite representative sub-set of the proposed full benchmark of MCJSSP instances. It contains very interesting instances ranging from 300 to 600 activities and is really suitable for exploring interesting trends before a time consuming intensive testing. All algorithms were implemented in Allegro Common Lisp and were run on a Pentium 3 processor 800 MHz, under Windows XP.

The general settings for the tested IFS strategies were the following:

1. we have limited the amount of backtracking for the procedures PCPS and SSTS by setting $\alpha = 2$;
2. the parameters for the precedence-based relaxation were $p_r = 0.2$ and $MaxRlxs = 6$;
3. the parameter p_r of the chain-based relaxation was set to 0.1 and 0.2;
4. we imposed a timeout of 3200 seconds for each problem instance and for each strategy we set $MaxFail = 1600$ (the maximum number of *non improving* moves that the algorithm tolerates before termination).

In addition, in order to met the imposed timeout, we adopt the same restarting schema used in previous works (Cesta, Oddi, & Smith 2000; Michel & Van Hentenryck 2004). In the case a first run finishes before the imposed time limit, the random procedure restarts from the initial solution until the time bound is reached. At the end, the best solution found is returned.

Table 1 compares the performance of the IFS strategies with respect to the value $\Delta LWU_{\%}$, which represents the average percentage deviation from the Lawrence upper bound (Lawrence 1984). In particular, given a numeric value in the table, (for example 9.84) the corresponding IFS strategy is given by reading the column's label (PCPS or SSTS), representing the solving strategy, and the row's label (one among PCs, ACTs or ACTs-iPCs) representing the adopted relaxation strategy. In particular:

IFS	PCPS	SSTS
PCs	6.97	12.92
ACTs ($p_r = 10$)	1.97	9.95
ACTs ($p_r = 20$)	4.03	8.77
ACTs-iPCs ($p_r = 10$)	1.48	9.84
ACTs-iPCs ($p_r = 20$)	4.10	8.95

Table 1: Comparative performance ($\mathbf{\Delta}LWU_{\%}$) on Set C

- PCs row represents the precedence-based relaxation on the solution critical path,
- ACTs represents the chain-based relaxation,
- ACTs-iPCs represents the chain-based relaxation with the switching to the PCs relaxation when the makespan improves within the iterative flattening loop.

Some of the relaxation strategies are differentiated with respect to the value of the parameter p_r (the probability to randomly remove an activity in a POS-form solution). Hence, the value 9.84 in Table 1 refers to an IFS algorithm using SSTS search and the relaxation strategy ACTs-iPCs with $p_r = 10$. The remaining IFS procedures can be easily deduced in analogous way.

First of all, the results shown in Table 1 gives a first empirical evidence of the fact that within the same computational framework, PCPS search performs better than SSTS. We remember our implementation of SSTS can be seen as a *serial scheduling schema* adopting the *latest finish time* (LFT) priority rule, which branches the search on the possible earliest start times. Other search schemas are possible, with different priority rules, this will be investigated stimulus in the near future.

When we consider the first three rows of Table 1, we clearly see that ACTs always outperforms PCs. In particular, the best performance is obtained by the combination of PCPS and ACTs. A possible explanation of this fact is that *precedence-based relaxation* is more targeted on directly reducing the makespan of a solution, because specifically relaxes its critical path (which is directly correlated to the solution's makespan). Hence, such procedure seems also more incline to be trapped in a local minima. On the contrary, the *chain-based* relaxation removes activities independently from the critical path, hence it might promotes a search with a higher degree of *diversification* thus explaining the better performance observed.

Things get even more interesting when we read also the last two rows of Table 1, where we see that the IFS procedures using PCPS and the relaxation strategy ACTs-iPCs improves over the other PCPS-based procedures. Notice that the last two IFS procedures are a first attempt to interleave intensification and diversification within the same IFS procedure. In particular, the idea is that when an improvement of the makespan is detected within the IFS loop, the relaxation strategy is temporary switched to the PCs one, which should be the more suited procedure to explore the detected

local minima. When no more improvement is found, the relaxation strategy is restored back to the ACSs one, which promotes a search with an higher degree of diversification.

A last comment concerns the parameter p_r, representing the probability of removing at random an activity in a POS-form solution. Here we consider two different values (0.1 and 0.2) just to test the sensibility of the performance measure with respect to p_r. Again, we remark the need for a more in depth experimentation. Nevertheless, we observe the opposite effect with respect to the IFS flattening procedures. In fact, the best performance for the PCPS-based procedures is obtained with $p_r = 0.1$, whereas the best performance for the SSTS-based ones are obtained with $p_r = 0.2$.

Conclusions

In this paper we have discussed a set of extensions to the Iterative Flattening Search procedure. IFLAT is a local search procedure for solving large-scale scheduling problems with a makespan minimization objective criterion. The presented extensions were motivated to perform an uniform study to evaluate the effectiveness of the "single component" IFS strategies proposed in the literature. In this spirit, we propose an uniform framework to combine and experimentally evaluate different IFS strategies. Specifically, we examine the utility of:

i) operating with different relaxation strategies, one targeted on removing decisions on the solution critical path and another one considering the whole solution;

ii) using different strategies to built a new solution, one posting precedence constraints among the activities and another one based on setting the start time of the activities.

We proposed a first experimental evaluation on benchmark instances of the Multi-Capacity Job-Shop Scheduling Problem, which have been used in previous studies of IFS procedures. The present experimental results start to clarify some weaknesses and strengths of the ideas proposed over the past years and suggest more effective and efficient IFS procedures. Some of the proposed extensions were found to improve the performance of the reference strategies. We are planning now to start an intensive experimentation on the complete set of MCJSSP benchmarks.

Acknowledgments

Amedeo Cesta, and Angelo Oddi 's work is partially supported by MIUR (Italian Ministry for Education, University and Research) under the project VINCOLI E PREFERENZE (PRIN). Nicola Policella is currently supported by a Research Fellowship of the European Space Agency, Directorate of Operations and Infrastructure. Stephen F. Smith's work is supported in part by the National Science Foundation under contract #9900298, by the Department of Defense Advanced Research Projects Agency under contract # FA8750-05-C-0033 and by the CMU Robotics Institute.

References

Cesta, A.; Oddi, A.; and Smith, S. 1998. Profile Based Algorithms to Solve Multiple Capacitated Metric Scheduling Problems. In *Proceedings of the* 4^{th} *International Conference on Artificial Intelligence Planning Systems, AIPS-98*, 214–223.

Cesta, A.; Oddi, A.; and Smith, S. F. 2000. Iterative Flattening: A Scalable Method for Solving Multi-Capacity Scheduling Problems. In *AAAI/IAAI,* 17^{th} *National Conference on Artificial Intelligence*, 742–747.

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.

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

Dorndorf, U.; Pesch, E.; and Phan Huy, T. 2000. A Branch-and-Bound Algorithm for the Resource-Constrained Project Scheduling Problem. *Mathematical Methods of Operations Research* 52:413–439.

Godard, D.; Laborie, P.; and Nuitjen, W. 2005. Randomized Large Neighborhood Search for Cumulative Scheduling. In *Proceedings of the* 15^{th} *International Conference on Automated Planning & Scheduling, ICAPS'05*, 81–89.

Kolisch, R. 1996. Serial and parallel resource-constrained project scheduling methods revised: Theory and computation. *European Journal of Operational Research* 90:320–333.

Lawrence, S. 1984. Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques (Supplement). Technical report, Graduate School of Industrial Administration, Carnegie Mellon University.

Michel, L., and Van Hentenryck, P. 2004. Iterative Relaxations for Iterative Flattening in Cumulative Scheduling. In *Proceedings of the* 14^{th} *International Conference on Automated Planning & Scheduling, ICAPS'04*, 200–208.

Nuijten, W., and Aarts, E. 1996. A Computational Study of Constraint Satisfaction for Multiple Capacitated Job Shop Scheduling. *European Journal of Operational Research* 90(2):269–284.

Policella, N.; Smith, S. F.; Cesta, A.; and Oddi, A. 2004. Generating Robust Schedules through Temporal Flexibility. In *Proceedings of the* 14^{th} *International Conference on Automated Planning & Scheduling, ICAPS'04*, 209–218.