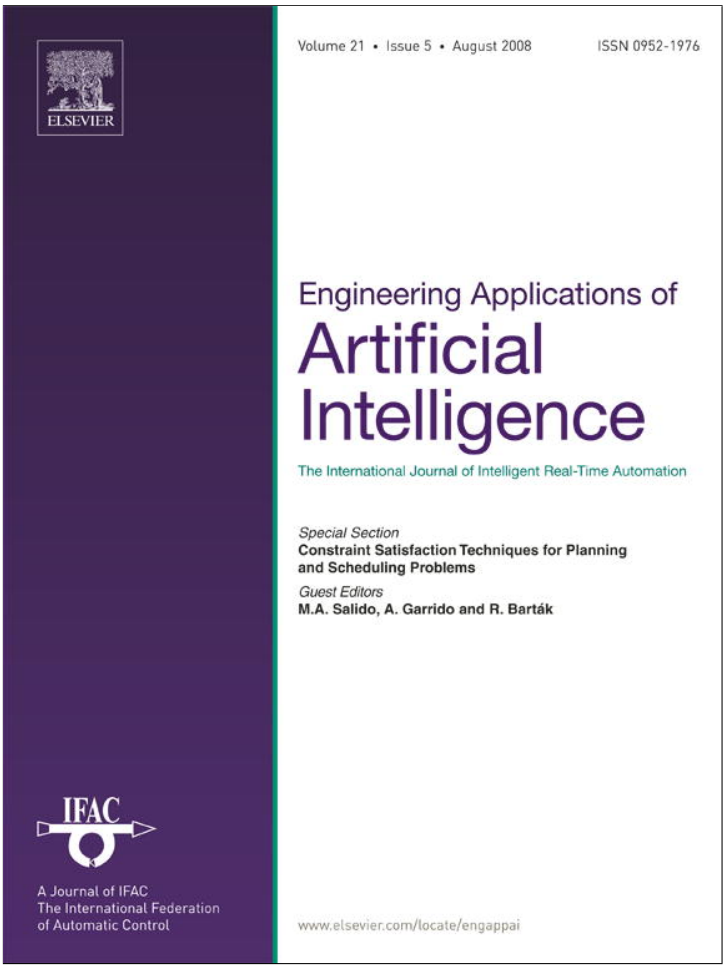

Engineering Applications of Artificial Intelligence 21 (2008) 683–690

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

Combining variants of iterative flattening search

Angelo Oddi [a,*], Amedeo Cesta [a], Nicola Policella [b], Stephen F. Smith [c]

[a] Italian National Research Council, ISTC-CNR, Via San Martino della Battaglia, 44, 00185 Rome, Italy
[b] European Space Agency, Darmstadt, Germany
[c] Carnegie Mellon University, Pittsburgh, PA, USA

ARTICLE INFO

Article history:
Received 1 February 2008
Accepted 1 March 2008
Available online 16 May 2008

Keywords:
Scheduling
Local search
Constraint satisfaction
Iterative improvement
Stochastic search

ABSTRACT

Iterative flattening search (IFS) is an iterative improvement heuristic schema for makespan minimization in scheduling problems. Given an initial solution, IFS iteratively interleaves a *relaxation-step*, which randomly retracts some search decisions, and an incremental solving step (or *flattening-step*) to recompute a new solution. The process continues until a stop condition is met and the best solution found is returned.

In recent work we have created a uniform software framework to analyze component techniques that have been proposed in IFS approaches. In this paper we combine basic components to obtain hybrid variants and perform a detailed experimental evaluation of their performance.

Specifically, we examine the utility of: (1) operating with different relaxation strategies and (2) using different searching strategies to built a new solution. We present a two-step experimental evaluation: (a) an extensive explorative evaluation with a spectrum of parameter combination; (b) a time-intensive evaluation of the best IFS combinations emerged from the previous. The experimental results shed light on weaknesses and strengths of the different variants improving the current understanding of this family of meta-heuristics.

1. Introduction

Iterative flattening search (IFS), or IFLAT, (Cesta et al., 2000) is an iterative improvement heuristic schema for solving scheduling problems with a makespan minimization objective. Given an initial solution, IFS iteratively executes a two-step cycle: first, a subset of solving decisions are randomly retracted from the current solution (called the relaxation-step); second, a set of new solving decisions are incrementally added to construct a new solution (called the flattening-step). This process continues until a specified termination condition is met and then the best solution found is returned. The basic idea resembles the Shifting Bottleneck Procedure for Job-Shop Scheduling from Adams et al. (1988).

In the original work, IFLAT was shown to produce high-quality solutions in reasonable time on a challenging set of large multi-capacitated job-shop scheduling problem MCJSSP benchmarks.

In two subsequent works, the performance of the original procedure was significantly improved through different refinements of the basic search schema. In Michel and Van Hentenryck (2004), the authors identified an anomaly in IFLAT search and proposed a simple extension: iterate the relaxation-step multiple times within each cycle. This extension substantially improved the quality of the schedules while preserving computational efficiency. The resulting algorithm found many new upper bounds and produced solutions within 1% of the best upper bounds on average. In Godard et al. (2005) additional optimal solutions and improvements on known upper bounds for MCJSSP benchmarks were obtained by substituting different mechanisms for the flattening and relaxation steps of the IFS schema.

Each of these different IFS proposals can be seen as instantiating the basic IFS schema with different relaxation and flattening strategies. However, since each proposal has been investigated using different implementation platforms, it is difficult to understand the relative effectiveness of various "component" strategies. To provide a basis for directly analyzing these design tradeoffs, we have designed and implemented a uniform framework for combining and experimentally evaluating different IFS relaxation and flattening strategies. In this paper, we present the results of an initial experimental analysis conducted with this framework. Specifically, we examine IFS to understand the relative utility of two basic sets of strategy alternatives extracted from previous IFS proposals:

- *Relaxation strategies*—We consider the performance impact of two different relaxation strategies, one strategy that is targeted to remove decisions on the solution critical path (cp) and another that considers all decisions as potential candidates for retraction.

* Corresponding author. Tel.: +39 06 44595 214; fax: +39 06 44595 243.
E-mail address: angelo.oddi@istc.cnr.it (A. Oddi).

0952-1976/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.engappai.2008.03.003

- *Flattening strategies*—We consider the performance effect of two different strategies for constructing a new solution, one strategy which operates in the space of ordering decisions and posts precedence constraints to eliminate potential resource conflicts and another which alternatively searches in the space of activity start times and computes solutions by setting activity start times.

Our experimental evaluation is carried out in two phases. First, an exploratory evaluation of the set of target IFS component configurations is performed. In this phase, different values for the tuning parameters associated with base components are considered. Then in the second step a more intensive evaluation of the best IFS configurations found in the previous phase is carried out. The results of our study provide insights into the potential of combining these IFS components in different ways, and also suggest some additional IFS configurations that warrant future investigation.

The remainder of the paper is organized as follows. We start by summarizing the MCJSSP problem that has served as a reference for IFS research, and introduce the benchmark problem sets to be used in the evaluation. Next we review the IFS schema that has evolved from previous work, beginning with some basic representational assumptions relevant to the design of IFS procedures and then summarizing the core algorithm. The central part of the paper introduces the alternative relaxation and solving (flattening) strategies to be analyzed in this paper and presents a framework for comparing them. Performance results are then given and design implications are discussed. The paper closes with a discussion of further opportunities to extend and enhance the basic IFS concept.

2. Reference problem

Research with various IFS strategies has utilized the Multi-Capacity Job-Shop Scheduling Problem, MCJSSP, as a common evaluation reference. This problem involves synchronizing the use of a set of resources $R = \{r_1, \ldots, r_m\}$ to perform a set of jobs $J = \{j_1, \ldots, j_n\}$ over time. The processing of a job j_i requires the execution of a sequence of m activities $\{a_{i_1}, \ldots, a_{i_m}\}$, 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 ($c_j \geqslant 1$). A *feasible solution* to an 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.

Experimental analysis of various MCJSSP solution procedures proposed in the literature has centered on a set of benchmark problems first introduced in Nuijten and Aarts (1996). These problems are derived from the Lawrence job-shop scheduling problem benchmarks (Lawrence, 1984) by increasing both the number of activities and the capacity of the resources. They are organized into four problem sets, distinguished as follows:

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

Set B: *LA*11–20 × 2 × 3. Five problems each of sizes 40 × 5(2), 60 × 5(3), 20 × 10(2), 30 × 10(3).

Set C: *LA*21–30 × 2 × 3. Five problems each of sizes 30 × 10(2), 45 × 10(3), 40 × 10(2), 60 × 10(3).

Set D: *LA*31–40 × 2 × 3. Five problems each of sizes 60 × 10(2), 90 × 10(3), 30 × 15(2), 45 × 15(3).

It is worth observing that these benchmark sets continue to represent a challenging benchmark for comparing algorithms. In particular, (a) they cover a wide range of problem sizes in relatively few instances; (b) they provide a direct basis for comparative evaluation; and (c) room for improvement of best known results achieved still remains. Additionally, as noted in Nuijten and Aarts (1996), one consequence of the problem generation method is that the optimal makespan for the original JSSP also provides a tight upper bound for the corresponding MCJSSP (Lawrence upper bounds). Hence, even if better solutions are known for many instances, the distance from these upper-bound solutions still provides a useful measure of solution quality. In the experimental study we present later in this paper, we will consider perfor mance of various IFS configurations relative to this benchmark problem suite.

3. Basic representations

Before presenting our general IFS schema and introducing alternative relaxation and flattening components, we first describe the basic representational assumptions that underlie our IFS framework.

3.1. Representations for constraint-based scheduling

The IFS search schema is based on the graph representation $G(A, E)$ of a scheduling problem, where A is the set of activities, plus two fictitious activities, a source a_{source} and a sink a_{sink}, and E is the set of precedence constraints defined between activities in A. A solution S is given as an extended graph G_S of G, such that an additional set of precedence constraints is added to "solve" the original problem. This means that the Set E is partitioned in two subsets, $E = E_{\text{prob}} \cup E_{\text{post}}$, 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.

In searching for a solution different search strategies may be applied. Following a *precedence constraint posting* (PCP) approach the set of E_{post} is naturally created by identifying and resolving (or flattening) contention peaks. For example in Cesta et al. (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 it is also used in Cesta et al. (2000), Michel and Van Hentenryck (2004) to deal with multi-capacity resource contention peaks in MCJSSPs.

The approach of Godard et al. (2005) uses a different strategy. Here the decision to *set a start time* (SST) of an activity is made by imposing a *rigid* temporal constraint between the a_{source} and the start-time of the interested activity. This "fixed-times" strategy for representing scheduling decisions is very common in the literature.

The IFS procedure, as originally introduced in Cesta et al. (2000), iterates two steps: (1) a *relaxation step* where a feasible schedule is relaxed into a possibly resource infeasible, but temporally feasible, schedule by removing some search decisions represented as precedence constraints between pair of activities; (2) a *flattening step* where 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, i.e., the solution representation used in a PCP approach. To apply the same philosophy with an SST we need to relax the "solution rigidity" introduced by the absolute temporal constraints that are inserted as decisions. One way of accomplishing this is to transform the SST solution into a partial order schedules (POSs) (Policella et al., 2007).

3.2. 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". Given an input solution, a polynomial transformation method, called CHAINING, has been defined that creates sets of chains of activities (Policella et al., 2007). This operation is accomplished in 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 can collectively supply 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.

One property of the POS is that it provides a flexible solution. As an example, in Godard et al. (2005) a POS is created from a fixed-time schedule to inject temporal flexibility into the solution before performing a relaxation step. 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 final chained solution is always $O(n)$. This because the chaining procedure creates POSs with only the "necessary" precedence constraints.

4. Iterative flattening search

Given these preliminaries we introduce a general IFSSEARCH-procedure in Fig. 1. Our goal is to create a uniform implementation framework for integrating component procedures defined in the various IFS procedures mentioned earlier. The idea is to decompose the effects of parts of the algorithms and hopefully understand how the effectiveness of the parts influences the effectiveness of the complete algorithm.

The algorithm in Fig. 1 alternates *relaxation* and *flattening* steps until a better solution is found or a maximal number of iterations is executed. The procedure takes two parameters as input: (1) an initial solution S; (2) a positive integer *MaxFail* which specifies the maximum number of non-makespan-improving moves that the algorithm will tolerate before terminating. 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

```
IFSSEARCH(S, MaxFail)
begin
1.  S_best ← S
2.  counter ← 0
3.  while (counter ≤ MaxFail) do
4.      RELAX(S)
5.      S ← FLATTEN(S)
6.      if Mk(S) < Mk(S_best) then
7.          S_best ← S
8.          counter ← 0
9.      else
10.         counter ← counter + 1
11. return (S_best)
end
```

Fig. 1. The IFSSEARCH general schema.

```
CPRELAX(S, p_r, MaxRlxs)
begin
1.  for 1 to MaxRlxs
2.    forall (a_i, a_j) ∈ CriticalPath(S) ∩ E_post
3.      if random(0,1) < p_r
4.        S ← S \ (a_i, a_j)
end
```

Fig. 2. Relaxation procedure based on critical-path removal.

to 0. Otherwise, if no improvement is found in *MaxFail* moves, the algorithm terminates and returns the best solution found. Before introducing the different relaxation and flattening steps in detail, it is worth saying that even if we are trying to reproduce different approaches in a uniform software framework, at the present stage we do not have a complete re-production of the algorithm in Godard et al. (2005) whose engineering aspects are well customized within the ILOG framework. For example, we do not have any of the resource propagation rules provided in that environment. Rather we have developed several components inspired by what is possible to reconstruct from Godard et al. (2005).

4.1. Relaxation procedures

In general, a relaxation procedure transforms a feasible schedule into a possibly resource infeasible, but temporally feasible, schedule by adopting different strategies for removing some search decisions. We have reproduced two of these strategies. The first, used in Cesta et al. (2000) and Michel and Van Hentenryck (2004), removes precedence constraints between pairs of activities on the critical path (cp) of the solution, hence is called *cp-based relaxation*. The second, introduced in Godard et al. (2005), starts from a POS-form solution and randomly *breaks* some chains in the POS—hence the name *chain-based relaxation*.

cp relaxation: The relaxation step in this case is based on the concept of *cp*. 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 *cp* 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 *cp*, since these constraints collectively determine the solution's current makespan. Following this observation, the relaxation step introduced in Cesta et al. (2000) is designed to retract some number of posted precedence constraints on the solution's cp. Fig. 2 shows the CPRELAX procedure. Steps 2–4 consider the set of posted precedence constraints ($pc_i \in E_{\text{post}}$), which belong to the current cp. A subset of these constraints is randomly selected on

```
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 ∈ pred(a_k) ∩ E_ch
         and (a_k, a_s), a_s ∈ succ(a_k) ∩ E_ch
4. Apply the CHAINING procedure
       to the subset of unselected activities
end
```

Fig. 3. Relaxation based on random removal from the POS chains.

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 et al. (2000) and Michel and Van Hentenryck (2004). In the former approach Steps 2–4 are performed only once (i.e., MaxRlxs = 1), whereas in Michel and Van Hentenryck (2004) these steps are iterated several times (from 2 to 6), such that a new cp 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 cp.

Chain relaxation: This relaxation requires a POS as input. A POS is an extension of the original precedence graph representing the input scheduling problem. In particular it is a graph $G_S(N, E_{\text{prob}} \cup E_{\text{ch}})$, such that the set $E = E_{\text{prob}} \cup E_{\text{ch}}$ is partitioned into a set of *chains* $\text{CH}_1, \text{CH}_2, \ldots, \text{CH}_{nc}$. 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, $\text{pred}(a_k) = \{a_p | \exists \text{CH} : (a_p, a_k) \in \text{CH}\}$ is the set of its predecessor activities and $\text{succ}(a_k) = \{a_s | \exists \text{CH} : (a_k, a_s) \in \text{CH}\}$ is the set of its successors activities. In particular, $\text{pred}(a_{\text{source}}) = \text{succ}(a_{\text{sink}}) = \emptyset$.

Given a POS S as input the chain-based relaxation procedure, Fig. 3, consists of the following steps:

- randomly select a subset of activities from the input solution S on the basis of the parameter $p_r \in (0,1)$,
- for each selected activity a_k, remove the edges (a_p, a_k), $a_p \in \text{pred}(a_k)$ and (a_k, a_s), $a_s \in \text{succ}(a_k)$ preserving the start times est_i of a_k and
- apply the *Chaining* procedure to the set of unselected activities, that is, the activities not removed by the random selection.

It is worth observing that the unselected activities still represent a feasible solution to a scheduling sub-problem, which can be transformed into a POS, in which the randomly selected activities *float* outside the solution (thus re-creating *contention peaks*).

4.2. Flattening procedures

Both relaxation schemas create an intermediate solution with contention peaks that must be *flattened* (removed from the current solution) to restore resource feasibility. We have implemented two general solution schemas for accomplishing this step, one based on the PCP strategy and the second on SST. Both solving algorithms are capable of performing a complete search through backtracking.

PCP *Search* (PCPS): The flattening step (see Fig. 4) used in Cesta et al. (2000) is inspired by prior work on the ESTA from Cesta et al. (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 et al., 1991)

```
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 ← ChoosePrecedence(S, mcs)
9.          PCPS(P, S ∪ {pc})
10.       else return(fail)
end
```

Fig. 4. The PCPS algorithm.

temporal constraint network[1] 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 then *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 (MCSs) as in Cesta et al. (2002), are then resolved by iteratively posting simple precedence constraints between pairs of competing activities. The constraint posting process of PCPS is based on the earliest start solution (ESS) consistent with currently imposed temporal constraints. The algorithm then proceeds analyzing possible resource conflicts (Steps 2–5). If this set is empty the ESS is also resource feasible and a solution is found. Otherwise if a conflict exists it can either admit a solution or not. In the first case a new precedence constraint is posted (Steps 8–9). If alternatively the conflict is unresolvable, then the process fails (Step 10). For further details on the functions *SelectConflict*(), and *ChoosePrecedence*() (a 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) which adopts the *latest finish time* (LFT) priority rule for selecting activities, and branches the search on the possible earliest start times of the current selected activity (Dorndorf et al., 2000). Note that other search procedures are possible by incorporating different priority rules; this will be stimulus for further experiments in the near future.

A recursive and non-deterministic version of the SSTS solver is shown in Fig. 5. At Step 1 the procedure *Propagate* propagates the current temporal constraints. In particular, for each activity a_i it 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 have 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 minimum latest finish time *lft* (with ties being 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.

[1] An STP (Simple Temporal Problem) is a temporal constraint network in which no disjunction of constraints is allowed between pairs of temporal variables. In our STP 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
```

Fig. 5. The SSTS algorithm.

4.3. IFS variants

The concept of iterative flattening (Cesta et al., 2000) is quite general and provides an interesting basis for designing diverse and effective local search procedures for scheduling optimization. The IFLATRELAX procedure proposed in Michel and Van Hentenryck (2004) is an example of an IFLAT extension which obtains substantial improvements over its original version. The version of iterative flattening proposed in Godard et al. (2005) produced further improvements over both previous procedures. This last procedure uses a solving strategy similar to SSTS and a chain-based relaxation schema.[2]

Our current goal is to perform a study that uniformly evaluates the strengths and the weaknesses of the individual IFS component strategies for relaxation and flattening proposed so far. According to this idea we propose the following IFS procedures:

- Two procedures based on PCPS, one that uses the precedence relaxation on the cp—identified with CPRELAX—and another one that uses the chain relaxation—identified with CHAINRELAX. 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 activities in the problem. We observe that the combination of PCPS and CPRELAX with $\alpha = 0$ reproduces the algorithm in Michel and Van Hentenryck (2004) extended with a backtracking search procedure.
- Two IFS procedures based on SSTS, one using cp-based relaxation—called SSTS + CPRELAX—and another one using chain relaxation—called SSTS + CHAINRELAX. As before, SSTS uses a parameter α to restrict 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 + COMBORELAX—which coincides with the combination of PCPS and a relaxation phase that mixes the two options: this procedure uses the *chain-based* relaxation, except when an improved solution is found within the iterative flattening loop (see Steps 3–10 in Fig. 1): and in this case the relaxation procedure is temporarily switched to the *cp* component, CPRELAX, until the makespan is improved (i.e., until the condition at Step 6 in Fig. 1 remains true).
- A new IFS procedure—called SSTS + COMBORELAX—which mirrors the previous one with respect to the relaxation strategy, but uses the SSTS procedure.

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

It is worth noting that the first four IFS strategies combine already known procedures, even if two of them (PCPS + CHAINRELAX and SSTS + CPRELAX) are relatively new algorithms. Alternatively, the last two procedures propose two new algorithms based on the following intuition. We may observe that the *cp* relaxation is more targeted to directly reduce the makespan of a solution, because it specifically relaxes its cp, which is directly correlated to the solution's makespan. However, such a procedure also seems more prone to becoming trapped in *local minima*. On the contrary, the *chain-based* relaxation removes activities without regard to whether they are on the cp, hence promoting a search with an higher degree of *diversification*. The last two IFS procedures represent two attempts to interleave intensification and diversification mechanisms within the same IFS procedure in order to improve performance. In the next section we present the results of an empirical evaluation of the procedures defined in this section.

5. Experimental analysis

Using the benchmark problem set described in Section 2, we conducted a two-step experimental evaluation of the IFS variants defined above. First, we performed an explorative evaluation using benchmark Set C only, to test the effectiveness of all variants across a range of parameter settings. This phase aims at selecting the best variants for the second phase. The second more intensive set of experiments was then performed on the entire benchmark problem suite. All algorithms were implemented in Allegro Common Lisp and were run on a Pentium 4 processor 2.6 GHz, under Linux.

Set C was chosen for testing in the initial exploratory phase because it is a quite representative subset of the MCJSSP instances. It contains instances structurally very interesting and challenging that range from 300 to 600 activities and is quite suitable for exploring interesting trends among the IFS strategies before a time consuming intensive testing.

5.1. Explorative experiments

For the exploratory stage experiments, the following general settings for the IFS strategies were assumed:

(i) The initial solution for all IFS strategies was generated by using the same flattening strategy incorporated within the IFS improving loop. For each instance, we set a large horizon constraint (in our case 5 times the makespan of the infinite capacity solution) and thus initial solution generation did not require any backtracking steps.
(ii) For both PCPS and SSTS, different amounts of backtracking were considered by setting α to the following percentage values $\alpha \in \{0, 5, 10, 15, 20, 25, 30\}$—for example, the value 10 means that the procedure executes a maximum number of backtracking steps equal to 10% of the number of activities.
(iii) The probability values p_r for cp relaxation and chain-based relaxation strategies were set to the same percentage values $p_r \in \{10, 15, 20\}$.
(iv) The parameter *MaxRlxs* was set to MaxRlxs = 6 for the cp-based relaxation.
(v) A 400 s timeout value was imposed for each problem instance.
(vi) For each strategy we set *MaxFail* = 3200 (the maximum number *non-improving* moves that the algorithm will tolerate before terminating).

Table 1
Set C—preliminary experiments

	α	PCPS			SSTS		
		$p_r = 10$	$p_r = 15$	$p_r = 20$	$p_r = 10$	$p_r = 15$	$p_r = 20$
cpRelax	0	9.4	9.2	8.7	15.4	15.4	16.4
	5	6.8	6.5	6.8	16.0	16.3	16.3
	10	7.3	6.2	6.3	16.8	16.3	15.9
	15	6.9	6.6	6.4	15.5	15.7	16.3
	20	7.3	6.0	6.4	16.1	16.2	16.9
	25	7.2	6.6	6.9	15.8	15.5	15.3
	30	6.8	6.7	6.7	16.7	15.9	15.8
chainRelax	0	5.8	6.9	8.3	15.5	16.1	15.1
	5	3.0	3.8	5.4	15.7	15.1	15.1
	10	2.8	3.9	5.4	14.8	15.8	15.0
	15	2.6	3.4	5.3	15.1	15.2	14.7
	20	2.5	3.4	5.3	14.5	14.5	15.3
	25	2.5	3.5	5.5	15.7	15.1	15.0
	30	2.6	3.9	5.2	14.8	15.2	14.4
comboRelax	0	5.7	6.8	8.3	15.4	16.3	15.2
	5	3.0	4.1	5.5	14.8	14.4	15.1
	10	3.0	4.3	5.1	15.8	15.8	15.6
	15	2.1	3.4	5.7	15.2	15.2	15.5
	20	2.7	3.6	4.9	15.1	15.1	14.9
	25	2.6	3.3	5.1	14.7	14.7	14.6
	30	2.5	3.4	5.1	15.0	15.0	15.4

In addition, in order to meet the imposed timeout, we adopted the same restarting scheme used in previous works (Cesta et al., 2000; Michel and Van Hentenryck, 2004). Specifically, in the case that 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 proposed IFS strategies with respect to the value $\Delta LWU_{\%}$, which represents the average percentage deviation from the Lawrence (1984) upper bound. In particular, given a numeric value in the table (for example the value 2.1) 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 cpRelax, chainRelax or comboRelax), representing the adopted relaxation strategy. Hence, within the identified sub-table, given the numeric value, we have the corresponding values for the parameters α and p_r (for example, performance 2.1 is obtained by using the combination PCPS + comboRelax with values $\alpha = 15$ and $p_r = 10$).

Some trends emerge from considering the results in Table 1:

(i) The results show evidence of the fact that within the same computational framework, PCPS search performs better than SSTS.
(ii) chainRelax variants perform much better than cpRelax. A possible explanation has been discussed earlier—namely that cp relaxation is more targeted to directly reduce the makespan of a solution but is more prone to becoming trapped in local minima. On the contrary, the chain-based relaxation removes activities independently from the cp, and hence its search has an inherently higher degree of *diversification* that explains the better performance observed.
(iii) comboRelax performs a bit better than chainRelax and the best performance is obtained by combining PCPS and comboRelax (value 2.1 obtained with $\alpha = 15$ and $p_r = 10$). This is a first confirmation of the intuition behind the definition of comboRelax proposed in Section 4.3. That is, the use of a cp relaxation has the function of converging quickly to a local minima when the default chain-based relaxation strategy has driven the search close to a local minima. In particular, when the objective function improves within the inner IFS loop, this is the trigger for switching to the cp-based strategy, which is more prone to descend quickly to the local minima.

On the basis of these results the combinations of the PCPS procedure with chainRelax and comboRelax were selected for the following intensive test phase. It is worth observing that the two winning strategies are both obtained as hybridization of component procedures previously proposed in the literature. This result is in line with our original goal to propose and experimentally evaluate new relaxation strategies that further boost performance of the IFS schema.

5.2. Intensive experiments

Here we analyze the promising behavior of the chain-based relaxation strategy and the mixed relaxation strategy on the full benchmark set MCJSSP described in Section 2. For this intensive experimentation we adopted the settings which gave the best results in the preliminary analysis and allocated more time to the experimentation. In particular:

(i) Parameter settings $\alpha = 0.15$ and $p_r = 10$ were chosen (and as earlier, we use the same parameter p_r for both relaxation strategies); these settings yielded the best performance in the preliminary results of Table 1.
(ii) A timeout of 3200 s was imposed for each problem instance, and for each strategy we set *MaxFail* = 3200 (the maximum number of *non-improving* moves that the algorithm will tolerate before terminating).

Table 2 shows the average values $\Delta LWU_{\%}$ for both the full set MCJSSP and the single subsets A–D. The results shown reasonably confirm the behavior found in the first phase. In fact, on average (column *All*) the hybrid schema comboRelax outperforms the chain-based relaxation strategy.

However, we can notice some differences when we consider the results for each subset (A–D) separately. In particular, chainRelax outperforms comboRelax on Set A, the subset containing the smallest problem sizes (ranging from 100 to 225 activities). comboRelax maintains a positive advantage over chainRelax on each of the other subsets, with the largest differential occurring for sets C and D (0.81 vs. 0.48 and 0.43 vs. 0.23). Notice that subsets A, B, C and D have different sizes and structures: they range in size from 100 to 900 activities and the number of activities in a single job ranges from five activities (in Set A) to 15 activities (in Set D). To explain the observed performance results, we hypothesize that for problems in Set A (and also in set B) both PCPS + chainRelax and PCPS + comboRelax enter into a *stable* phase within the 3200 s of runtime allocated, and hence they exhibit similar performance. In contrast, in the case of the more challenging and larger problems of Sets C and D, the search being conducted by both algorithms is still in a transient phase at the end of the allotted run time, and comboRelax maintains a performance advantage over chainRelax.

Table 2
$\Delta LWU_{\%}$ values on the complete benchmark

	Set A	Set B	Set C	Set D	All
PCPS + chainRelax	−0.13	−1.62	0.81	0.43	−0.12
PCPS + comboRelax	−0.11	−1.66	0.48	0.23	−0.26

With this motivation in mind, consider Figs. 6 and 7, which show a *run-time analysis* of the search progress being made with respect to $\Delta LWU_{\%}$ over time for the two proposed relaxation strategies. Fig. 6 shows this behavior across the whole problem set while Fig. 7 shows the trend within each experimental subset. We clearly see that our hypothesis is confirmed. For the Sets A and B, we observe a *transient* behavior for PCPS + CHAINRELAX and PCPS + COMBORELAX for the first 1000 s of the run, wherein the mixed relaxation strategy has a clear advantage over the pure chain-based one. After 1000 s, we observe a *stable* behavior, where the two strategies give similar performance. On the other hand, for the Sets C and D, we observe a *transient* behavior over the entire run, where PCPS + COMBORELAX maintains an advantage on PCPS + CHAINRELAX. Such advantage is quite clear for the Set C and more uncertain for the Set D (the subset with "longer" 15 activity jobs). In the latter case both relaxation strategies have similar *transient* performance, but COMBORELAX seems to have a stable improvement over CHAINRELAX at the end (0.23 vs. 0.43 in Table 2). Further experiments can further clarify the differences in this specific test set.

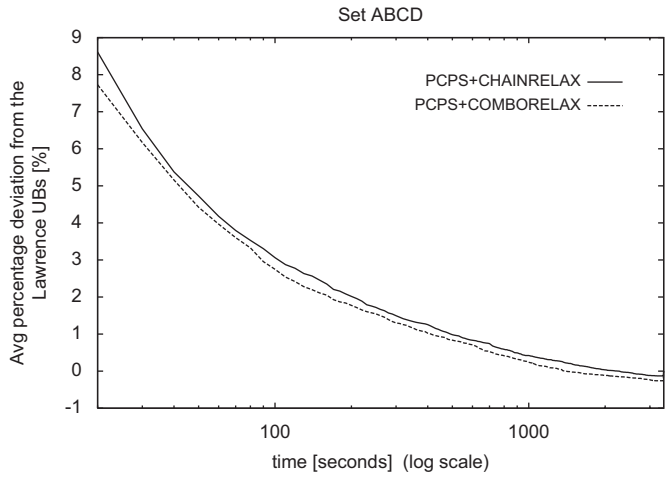

Fig. 6. Cumulative run-time performance.

5.3. Discussion

The results reported above give us some initial insights into the relative merits of different component IFS strategies that have been proposed by previous research and hence some guidelines for the design of more effective IFS procedures. Perhaps most striking is the dominance of CHAINRELAX over CPRELAX as a relaxation strategy. The broader-based search that follows from CHAINRELAX seems to consistently give a performance advantage, and the usefulness of CPRELAX appears to be only as a secondary supporting strategy. Interestingly though, the use of CPRELAX as an intensification device in conjunction with CHAINRELAX (i.e., the

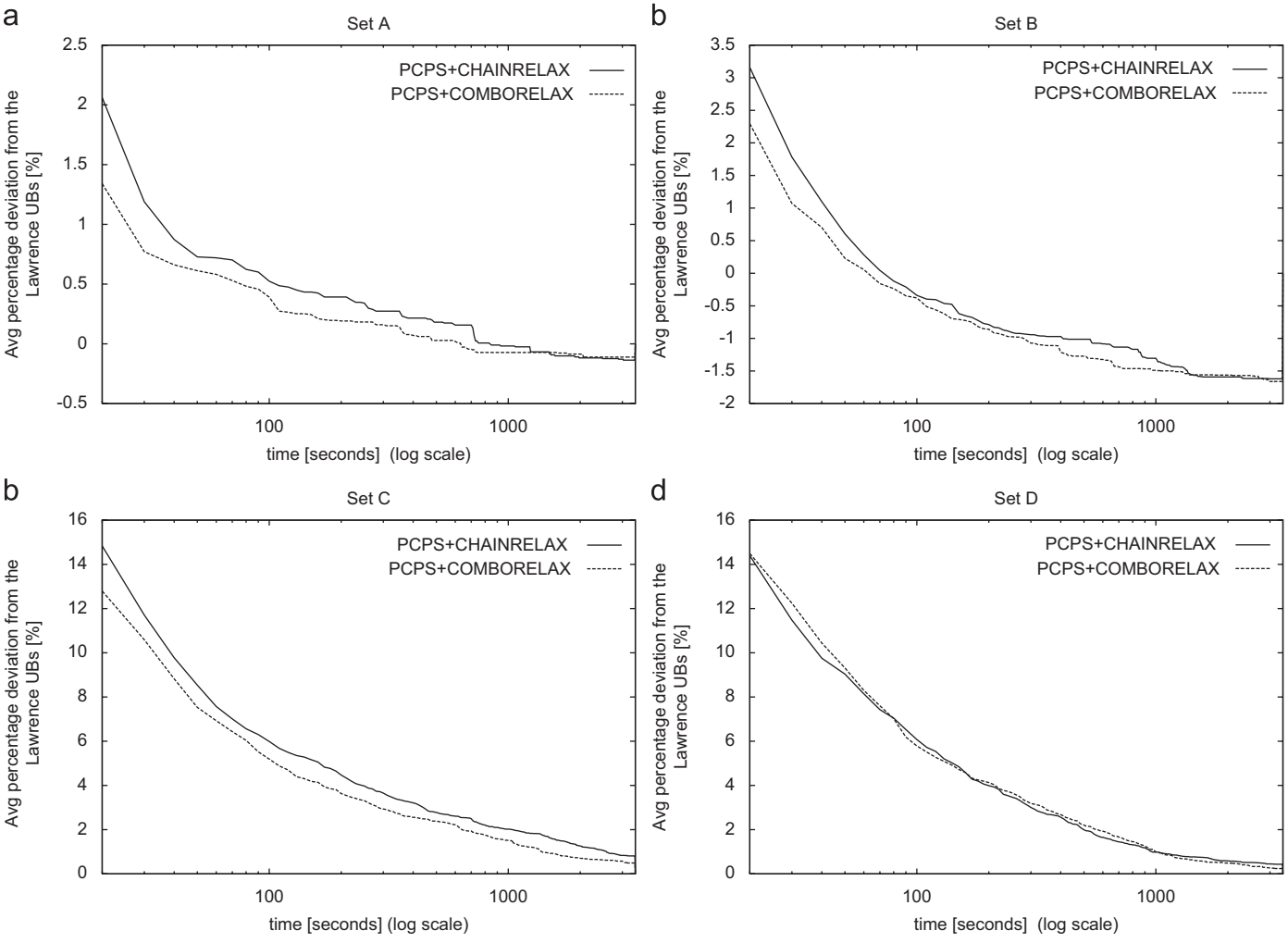

Fig. 7. Run-time performance for each testset. (a) Set A. (b) Set B. (c) Set C. (d) Set D.

COMBORELAX composite strategy), yielded the best performing IFS procedure.

With regard to solving strategy, the results suggest the superiority of PCPS over SSTS. This is somewhat surprising, since the procedure of Godard et al. (2005), which takes an SSTS search approach, remains the procedure with the best reported results on the benchmark problem suite. One possible explanation is the fact that an additional resource propagation technique (timetabling) not considered in our analysis is in fact incorporated and exploited in their implementation. In addition, the same paper adopts an incomplete backtracking scheme for SSTS using information provided by constraint propagation. This raises the question of whether similar introduction of such mechanisms into PCPS would have a similar performance boosting effect. Further experimental analysis is required to gain better understanding here.

6. Conclusions

In this paper we have explored several different instantiations of the iterative flattening search (IFS) schema within a common computational framework. Our goal has been to evaluate the relative effectiveness of different component strategies proposed in the literature for performing the relaxation and flattening steps of IFS. Specifically, we examined the performance tradeoffs associated with (i) restricting retraction of decisions during the relaxation step to those on the critical path (CPRELAX) vs. allowing decisions to be retracted in an unconstrained manner (CHAINRELAX) and (ii) conducting a flattening search by posting precedence constraints between activities that share resources (PCPS) vs. conducting a search based on setting the start times of activities (SSTS). Different IFS procedures were constructed by combining this set of strategies in different ways and evaluated on a standard reference set of Multi-Capacity Job-Shop Scheduling Problem MCJSSP benchmarks.

The experimental results clarified some weaknesses and strengths of previously proposed IFS procedures, and identified several new hybrid IFS procedures. With respect to relaxation strategy, CHAINRELAX was found to significantly outperform CPRELAX but a hybrid scheme incorporating both strategies was found to perform best overall. With regard to search strategy, PCPS was somewhat surprisingly found to perform better than SSTS once placed in a common computational framework.

In a future work, we intend to consider more sophisticated versions of the SSTS and PCPS solving strategies. In particular, we would like to investigate the effect of adding various constraint propagation algorithms to the different backtracking search schema that these strategies utilize. In another direction, we are also interested in exploring the potential benefit of extending CHAINRELAX to incorporate different chaining procedures that increase the flexibility of the partial solutions they create.

Acknowledgments

Amedeo Cesta and Angelo Oddi's work is partially supported by MIUR (Italian Ministry for Education, University and Research) under project VINCOLI E PREFERENZE (PRIN), CNR under project Integration of Planning Methodologies in a Constraint Programming Environment (RSTL funds 2007) and ESA (European Space Agency) under the APSI initiative. Nicola Policella is currently supported by a Research Fellowship of the European Space Agency, Human Spaceflight and Explorations Department. Stephen F. Smith's work is supported in part by the Department of Defense Advanced Research Projects Agency under contract #FA8750-05-C-0033 and by the CMU Robotics Institute.

References

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

Cesta, A., Oddi, A., Smith, S., 1998. Profile based algorithms to solve multiple capacitated metric scheduling problems. In: AIPS-98. Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems, pp. 214–223.

Cesta, A., Oddi, A., Smith, S.F., 2000. Iterative flattening: a scalable method for solving multi-capacity scheduling problems. In: AAAI/IAAI. 17th National Conference on Artificial Intelligence. Wiley, NY, pp. 742–747.

Cesta, A., Oddi, A., Smith, S.F., 2002. A constraint-based method for project scheduling with time windows. Journal of Heuristics 8 (1), 109–136.

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

Dorndorf, U., Pesch, E., 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., Nuitjen, W., 2005. Randomized large neighborhood search for cumulative scheduling. In: ICAPS-05. Proceedings of the 15th International Conference on Automated Planning & Scheduling, pp. 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, 1984.

Michel, L., Van Hentenryck, P., 2004. Iterative relaxations for iterative flattening in cumulative scheduling. In: ICAPS-04. Proceedings of the 14th International Conference on Automated Planning & Scheduling, pp. 200–208.

Nuijten, W., 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., Cesta, A., Oddi, A., Smith, S., 2007. From precedence constraint posting to partial order schedules. AI Communications 20 (3), 163–180.