J Intell Manuf (2010) 21:17–30
DOI 10.1007/s10845-008-0163-8

Iterative flattening search for resource constrained scheduling

Angelo Oddi · Amedeo Cesta · Nicola Policella · Stephen F. Smith

Received: 5 August 2008 / Accepted: 14 August 2008 / Published online: 13 November 2008

Abstract Iterative flattening search (IFS) is a meta-heuristic strategy for solving multi-capacity scheduling problems. Given an initial solution, IFS iteratively applies: (1) a relaxation-step, in which a subset of scheduling decisions are randomly retracted from the current solution; and (2) a flattening-step, in which a new solution is incrementally recomputed from this partial schedule. Whenever a better solution is found, it is retained, and, upon termination, the best solution found during the search is returned. Prior research has shown IFS to be an effective and scalable heuristic procedure for minimizing schedule makespan in multi-capacity resource settings. In this paper we experimentally investigate the impact on IFS performance of algorithmic variants of the flattening step. The variants considered are distinguished by different computational requirements and correspondingly vary in the type and depth of search performed. The analysis is centered around the idea that given a time bound to the overall optimization procedure, the IFS optimization process is driven by two different and contrasting mechanisms: the random sampling performed by iteratively applying the "relaxation/flattening" cycle and the search conducted within the constituent flattening procedure. On one hand, one might expect that efficiency of the flattening process is key: the faster the flattening procedure, the greater the number of iterations (and number of sampled solutions) for a given time bound; and hence the greater the probability of finding better quality solutions. On the other hand, the use of more accurate (and more costly) flattening procedures can increase the probability of obtaining better quality solutions even if their greater computational cost reduces the number of IFS iterations. Comparative results on well-studied benchmark problems are presented that clarify this tradeoff with respect to previously proposed flattening strategies, identify qualitative guidelines for the design of effective IFS procedures, and suggest some new directions for future work in this area.

Keywords Scheduling · Meta-heuristics · Iterative sampling

A. Oddi (✉) · A. Cesta
ISTC-CNR, National Research Council of Italy, Via San Martino della Battaglia 44, 00185 Rome, Italy
e-mail: angelo.oddi@istc.cnr.it

A. Cesta
e-mail: amedeo.cesta@istc.cnr.it

N. Policella
European Space Agency, Robert-Bosch-Strasse 5, 64293 Darmstadt, Germany
e-mail: nicola.policella@esa.int

S. F. Smith
The Robotics Institute, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, USA
e-mail: sfs@cs.cmu.edu

Introduction

Given the complexity of the scheduling problem, research has investigated several non-systematic solution approaches to various classes of problems. Stochastic local search (Hoos and Stützle 2005), for example, is a broad category of such approaches that includes many specific techniques. In this paper, we focus on one family of techniques referred to as iterative flattening search (IFS). IFS was first introduced in Cesta et al. (2000) 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 (referred to as the relaxation-step); (2) a new solution is then incrementally recomputed (referred to as the flattening-step).

The original IFS procedure was extended in two subsequent works, each of which proposed refinements to the basic search schema. In Michel and Van Hentenryck (2004) an anomaly in the original algorithm was identified, and a simple extension (iterate the relaxation step multiple times) was introduced which substantially improved the quality of the schedules generated while preserving computational efficiency. In Godard et al. (2005) additional optimal solutions and improvements to known upper-bounds for the reference benchmark problems were obtained, following from the substitution of different algorithmic components for the component flattening and relaxation steps.

More recently Oddi et al. (2008) initiated a systematic study aimed at evaluating the effectiveness of single *component* strategies within the same uniform software framework. In this work a generic IFS procedure was defined and component strategies taken from previous works were implemented and evaluated in this uniform framework. In particular, the utility of operating with different relaxation strategies was analyzed. Experimental results shed light on relative performance of the previously proposed relaxation procedures, and led to the definition of a more effective, composite relaxation strategy.

Figure 1 graphically illustrates the IFS solving process. A target search space is shown where the darker areas represent regions of consistent solutions. The solving process starts from an initial, inconsistent situation (left-hand side of the picture) and then computes an initial solution (thick arrow) by applying a FLATTEN operator. Once a solution is obtained, a perturbation step is taken (dashed arrow) by applying a RELAX operator to retract one or more decisions in the current solution. This step moves the solving process back to an inconsistent situation and implicitly takes a diversification decision. Then, from this new inconsistent state, a new solution is computed by again applying the FLATTEN operator. Of course, it is possible to perform more than one relaxation step in a given iteration (as done for example by Michel and Van Hentenryck (2004)). Broadly speaking, multiple relaxation steps amplify the diversification aspect and enable exploration of a different set of solutions. In the figure, the process evolves through the production of two new solutions by applying the RELAX operator multiple times (2 and 3 respectively). Despite its simplicity, the IFS approach has several desirable characteristics. One important aspect is its scalability: empirical results have shown that the approach can efficiently and effectively solve problems of significant size. Another valuable property is its ability to provide solutions at "any time". The solving process can be terminated after any FLATTEN step, with IFS simply returning the current best solution found.

In this paper, we continue the study of component strategies initiated in Oddi et al. (2008). We take as our starting point the best relaxation strategy found in this work, and

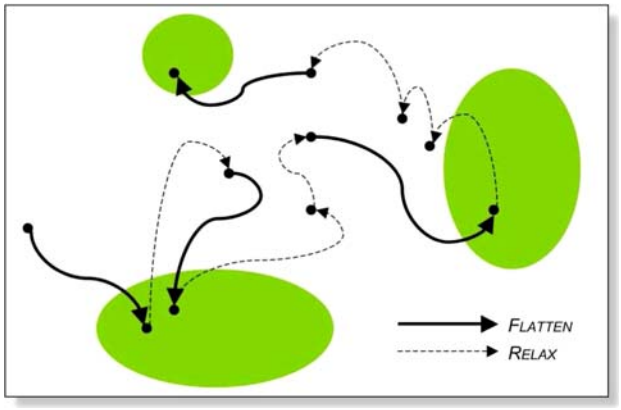


Fig. 1 A high-level description of the IFLAT solving process

focus on the complementary issue of understanding the performance characteristics and relative advantages of different flattening procedures. Our analysis follows the idea that once a time bound for the overall optimization procedure has been fixed, the IFS optimization process is driven by two different and contrasting mechanisms: the random sampling (Gomes 2003; Langley 1992) performed by application of the "relaxation/flattening" steps and the search provided by the constituent flattening procedure. In fact, we can observe that on any iteration of the IFS loop, a new solution is randomly sampled together with a set of *nearby* solutions (generated by the flattening procedure). On one hand, one might expect that efficiency of the flattening process is key: the faster the flattening procedure, the greater the number of iterations (and number of sampled solutions) for a given time bound; and hence the greater the probability of finding better quality solutions. On the other hand, the use of more accurate (and more costly) flattening procedures can increase the probability of obtaining better quality solutions even if their greater computational cost reduces the number of IFS iterations. This work attempts to evaluate this tradeoff and answer the question of whether it is better (for a given time limit) to maximize the number of IFS cycles or to rely on more effective (but slower) flattening procedures.

Plan of the paper

The remainder of the paper is organized as follows. First, the reference scheduling problem is quickly introduced. Next the IFS algorithmic template, the relaxation strategy to be assumed and the set of flattening (solving) strategies to be analyzed are described. Performance results are then given which indicate the pros and cons of using flattening procedures of different complexity and effectiveness. This is followed by a broader discussion of similar algorithmic ideas that have appeared in the literature. Finally we discuss further

opportunities for extending and enhancing the iterative flattening search concept.

The resource constrained scheduling problem

In this paper we apply IFS to instances of the Resource Constrained Scheduling Problem. This is an NP-hard optimization problem (Blazewicz et al. 1983) and is formulated as a set of n activities a_i $(i = 1 \ldots n)$ and r (renewable) resources $k = 1 \ldots r$. A constant amount of c_k units of resource k is available at any time. Activity a_i must be processed for p_i time units. During this time period a constant amount of r_{ik} units of resource k is occupied. Furthermore, precedence constraints are imposed between activities to describe a project precedence graph. These are given by relations $a_i \prec a_j$, where $a_i \prec a_j$ means that activity a_j cannot start before activity a_i is completed. The objective is to determine starting times S_i for the activities a_i $(i = 1 \ldots n)$ in such a way that: (1) for each time t the total resource demand is less than or equal to the resource availability c_k for each resource k; (2) all given precedence constraints are satisfied; and (3) the makespan $C_{max} = max_{i=1\ldots n}\{C_i\}$ is minimized, where $C_i = S_i + p_i$ is assumed to be the completion time of activity a_i.

For the experimental analysis considered later in this paper, benchmarks problem sets are drawn from two distinct resource-constrained scheduling problems: the Multi-Capacity Job Shop Scheduling Problem (MCJSSP) and the Resource Constrained Project Scheduling Problem (RCPSP). Indeed, these two problem classes provide benchmark problems which differ quite significantly both in size and in problem structure. It is worth mentioning that while MCJSSP has been a standard reference benchmark for comparing the IFS, the analysis with the RCPSP instancies represents a unique contribution of this work.

Iterative flattening search

As in Oddi et al. (2008) our goal is to use a uniform implementation framework to study component procedures that have been proposed in previous IFS models. The main idea is to isolate the effects of various component parts of IFS algorithms and understand how the effectiveness of parts influences the effectiveness of the complete algorithm. We focus in particular on examining the utility of operating with different flattening strategies, and assume as the reference relaxation strategy the best strategy found in Oddi et al. (2008).

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 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. On each iteration, the RELAX step reintroduces the possibility of resource contention, and the FLATTEN step then restores resource feasibility by leveling (or flattening) detected contention peaks(s). 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 in $MaxFail$ moves, the algorithm terminates and returns the best solution found.

```
IFS(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. 2 The IFS general schema

Preliminaries

The algorithms we are describing here are all based on a representation of the basic scheduling problem as a precedence graph $G(A, E)$, where A is the set of problem activities, plus the two fictitious activities a_{source} and a_{sink}, and E is the set of directed edges (a_i, a_j) representing the set of precedence constraints $a_i \prec a_j$ defined among the nodes 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. Hence, 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 $(S_1, S_2, \ldots, S_n)$, that is a set of assignments to the activities' start-times which are consistent with the set of constraints E and the set of imposed resource constraints.

In the following we consider flattening (solving) procedures which output as a solution either (1) a partial order G_S or (2) a set of assignments to the start-times of activities $(S_1, S_2, \ldots, S_n)$. In the latter case, it easy to eliminate the "solution rigidity" introduced by the absolute temporal

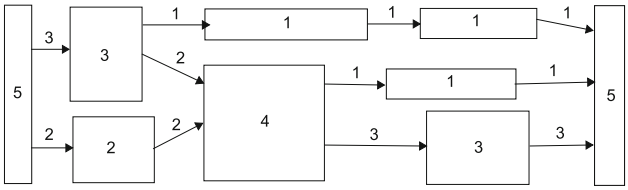


Fig. 3 An example of partial order schedule (POS)

constraints inserted as decisions and convert it into a graph G_S. One way of doing this is to transform the solution $(S_1, S_2, \ldots, S_n)$ into a partial order schedule (POS) (Policella et al. 2007).

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 constraint, but represents a *producer-consumer* relation, allowing each activity to be connected with the set of predecessors which 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 completes its execution, it passes its resource unit(s) on to its successors. Figure 3 shows an example of POS for a single resource with capacity $c = 5$. In particular, activities are represented as rectangles and edges represent the precedence constraints. The numbers inside the rectangles represent the resource requirements and the labeling numbers on the directed edges represents the flow of resource units supplied to a generic activity a_i from its predecessors in order to satisfy the imposed resource constraint. For example, the activity which requires four units of resource receives two units of resource from each of its two predecessors and supplies one and three units of resource respectively to its two successors.

In general, a solution $S = G_S(A, E_{POS})$ is in *POS-form* if for each resource k there exists a labelling function $f_k : E_{POS} \longrightarrow [0..c_k]$ representing the flow of resource units among the activities such that for each activity a_i the following constraint holds:

$$\sum_{p \in prec(a_i)} f_k(p, a_i) = \sum_{p \in succ(a_i)} f_k(a_i, s) = r_{ik} \qquad (1)$$

where $prec(a_i) = \{p | \exists (p, a_i) \in E_{POS}\}$ and $succ(a_i) = \{s | \exists (a_i, s) \in E_{POS}\}$ exist.[1] Given an input solution S (represented either as a graph or as a set of start-time values) a polynomial transformation method, named CHAINING, can be defined that creates sets of activity *chains* Policella et al. 2007. This operation can be 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 p are chosen, which supplies the units of resource required by a_i—a precedence constraint (p, a_i) is posted for each predecessor p. The last step is iterated until all the activities are linked by precedence chains and the constraints in (1) are satisfied.

[1] $prec(a_{source}) = succ(a_{sink}) = \emptyset$

Relaxation procedures

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 number of scheduling decisions. Given the above described graph representation, each such decision is a *precedence constraint* between a pair of activities that are competing for the same resource capacity. In Oddi et al. (2008), variants of three relaxation strategies were considered. The first strategy, used in Cesta et al. (2000); Michel and Van Hentenryck (2004), removes precedence constraints between pair of activities on the *critical path* of a solution; and hence is called *cp-based relaxation*. The second strategy, from Godard et al. (2005), starts from a POS-form solution and randomly *breaks* some chains in the input POS schedule, and hence is given the name *chain-based relaxation*. The third strategy, defined for the first time in Oddi et al. (2008), interleaves the first two strategies in a way that promotes a better balance between *diversification* and *intensification* (Blum and Roli 2003). This third strategy, referred to as *Combo Relaxation*, was found to yield the best performance and accordingly we adopt this as our baseline relaxation strategy in this paper. To define it more precisely we first briefly describe the two base strategies that it integrates.

Precedence relaxation

The *cp-based relaxation* strategy is centered on the solution's *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. From this observation, the relaxation step is designed to retract some number of posted precedence constraints on the solution's critical path. Figure 4 shows the CPRELAX procedure. Steps 2–4 consider the set of posted precedence constraints 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. These steps are iterated

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

Fig. 4 Relaxation procedure based on removal from critical path

```
CHAINRELAX(S, p_r)
begin
1. foreach activity a_i
2.    if random(0,1) < p_r then
3.       foreach a_p ∈ prec(a_i) ∩ E_ch remove (a_p, a_i)
4.       foreach a_s ∈ succ(a_i) ∩ E_ch remove (a_i, a_s)
5. Apply the CHAINING procedure
         to the subset of unselected activities
end
```

Fig. 5 Relaxation based on removal from POS chains

$MaxRlxs$ times (effective values range 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 whose length is very close to the critical one.

Chain relaxation

The *chain-based relaxation* strategy requires an input solution in POS-form. As explained above, a solution in POS-form is an extension of the original precedence graph that represents the input scheduling problem. A POS is a graph $G_S(N, E_{POS})$, such that the set $E_{POS} = E_{prob} \cup E_{ch}$ is composed by a set of *chains*. Each chain imposes a total order on a subset of problem activities requiring the same resource. Given a generic activity a_i, $prec(a_i)$ is the set of its predecessor activities and $succ(a_i)$ is the set of its successors activities.

The CHAINRELAX procedure, shown in Fig. 5, proceeds in three steps. First, a subset of activities from the input solution S are randomly selected on the basis of the parameter $p_r \in (0, 1)$. Second, the edges (a_p, a_i), $a_p \in prec(a_i)$ and (a_i, a_s), $a_s \in succ(a_i)$ are removed without updating the start times est_i of the activities. Finally, the CHAINING procedure is applied to the set of activities that have not been removed by the random selection. It is worth observing that this set of activities still represents a feasible solution to a scheduling sub-problem, which can be transformed into POS-form, in which the randomly selected activities *float* outside the solution thus re-creating *contention peaks*.

Combo relaxation

The *Combo relaxation* procedure interleaves the previous two strategies. It uses CHAINRELAX as its default, except when an improved solution is found within the IFS loop (i.e., when the boolean condition at Step 6 in the algorithm of Fig. 2 is found to be true). In this case the relaxation procedure is temporarily switched to CPRELAX, until the makespan can no longer be improved, (i.e., as long as the boolean condition at Step 6 $Mk(S) < Mk(S_{best})$ remains true). Hence, the procedure considers as input parameters both the parameters for the *cp-based relaxation* component (p_r and $MaxRlxs$) and for the *chain-based relaxation* component (p_r).

It is worth observing that the *critical path* relaxation is more targeted to directly reduce the makespan of a solution, because it specifically relaxes its critical path, 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 critical path, hence promoting a search with an higher degree of *diversification*. As observed in Oddi et al. (2008) the combination of the two strategies provides a better balance between diversification and intensification. In fact, as soon as CPRELAX is not able to improve the makespan, the relaxation strategy is reverted to CHAINRELAX, which increases the probability of escaping the local minima. On the contrary, when the makespan is improved, the relaxation strategy in switched to CPRELAX, which should increase the convergence rate to a local minima.

As mentioned earlier, we choose this last relaxation strategy as the reference for the experimental analysis of this paper based on the performance study of Oddi et al. (2008).

Flattening procedures

Both base relaxation procedures, CPRELAX and CHAINRELAX, create a intermediate solution with contention *peaks* (i.e., intervals of contention in resource usage). The role of the flattening step is to level (or *flatten*) these peaks, through the introduction of additional activity ordering or start time assignment decisions. We have implemented two general search templates for flattening peaks and producing a resource feasible solution, the first based on precedence constraint posting (PCP) as in Cesta et al. (2000) and the second based on set start time (SST) strategies as in Godard et al. (2005). In principle, both solving algorithms can be configured to perform a complete search through depth-first backtracking. However in our implementations, we limit the number of backtracking steps for purposes of scalability to a value dependent on the size of the input problem. Both algorithms can also be augmented to incorporate resource *constraint propagation*, for checking consistency and for

```
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. 6 The PCPS algorithm

```
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. 7 The SSTS algorithm

tightening the domains of activity start and end time variables as scheduling decisions are made. For both general flattening procedures, we define a variant of the basic template that uses a *Time Tabling* algorithm (Le Pape 1996) to this end, and a variant that does not exploit resource propagation. This gives us a variety of procedures for evaluation with different degrees of complexity and effectiveness. In the subsections below, we first describe each of the above mentioned algorithmic building blocks.

*PCP search (*PCPS*)*

The flattening step (see Fig. 6) used in Cesta et al. (2000) is inspired by prior work on the earliest start time algorithm (ESTA) from Cesta et al. (1998). The algorithm is a variant of a class of PCP scheduling procedures that utilize a two-phase solution generation process.

The first step *constructs an infinite capacity solution*. The problem is formulated as an STP (Dechter et al. 1991) temporal constraint network[2] where temporal constraints are modeled and satisfied (via constraint propagation), 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 et al. (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. At Step 1 of Fig. 6 the procedure *Propagate* deduces effects of both current resource and temporal constraints. It then proceeds to compute a resource conflict (Steps 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()* (a non deterministic version of the precedence selection operator) the reader should refer to the original implementation in Cesta et al. (2002).

*SST search (*SSTS*)*

The second solving procedure is based on the idea of searching the set of possible assignments to the activity start-times. Our implementation of SSTS is inspired by one described in Godard et al. (2005). It can be seen as a *serial scheduling schema* (Kolisch 1996), which branches the search on the possible earliest start times Dorndorf et al. 2000. A recursive and non deterministic version of the solver is shown in Fig. 7. At Step 1 the procedure *Propagate* propagates resource and current temporal constraints. In particular, for each activity a_i updates its earliest start-time est_i and latest finish time lft_i. When the output solution S is a complete and resource feasible solution (each activity has a feasible 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 earliest resource feasible start time* (and ties are broken by the minimal latest finish time values). Given a selected activity a_i, the search *branches* (Step 8) on all possible resource feasible assignments of the earliest start-time est_i.

Time-Tabling propagation

As stated above, it is possible to augment the flattening procedures to make use of Time Tabling propagation. Time-Tabling relies on the computation for each resource k and every instant t of the *aggregated necessary demand* induced by the current set of activities a_i in the schedule (Le Pape 1996). This aggregated profile is updated during the search and allows the domains of activity start and end times to be

[2] 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.

restricted by removing each instant t that would necessarily lead to an over-consumption of resource.

Let us suppose that an activity a_i requires r_{ik} units of a given discrete resource k and the following condition holds on the latest start-time (lst_i) and the earliest finish-time (eft_i) of a_i: $lst_i < eft_i$. In this case, a_i will always execute over the time interval $[lst_i, eft_i)$ and will surely require r_{ik} units of resource k over this time interval.

For each resource k, a necessary demand profile $ndp_k(t)$ is maintained that *aggregates* all these single activity demands. When there exists an instant t such that $npd_k(t)$ is strictly greater than c_k—the capacity of the resource k—the current schedule is *inconsistent*. In addition, suppose there exists an activity a_i requiring r_{ik} units of resource k and an instant t_{ub} such that: $eft_i \leq t_{ub} < lft_i$ and $\forall t \in [t_{ub}, lft_i)$ the condition $ndp_k(t) + r_{ik} > c_k$ holds. Then, activity a_i cannot end after t_{ub}, otherwise it would over-consume the resource. Similar reasoning can be applied for finding new lower bounds t_{lb} for the start-times of activities.

The main advantage of this technique is its relative simplicity and its low algorithmic complexity, which is $O(n \cdot s_{max})$, where n is the number of activities and s_{max} is the maximum slack value $lft_i - est_i - p_i$. In practice, Time-Tabling is one of the principal techniques used for scheduling discrete resources. However, it is worth noting that the above complexity reflects the time required to check the bounds of all problem activities only once. In the case that additional constraints are discovered and added to the current solution, we have to either incur the complexity of arriving to the *fix-point* or limit the number of iterations to a constant known value.

Iterative flattening search variants

Following the approach taken in Oddi et al. (2008), we aim at studying the effects of single "component" IFS strategies. As previously mentioned, we consider as given the use of the Combo Relaxation strategy previously found to perform best and concentrate on exploring the effects of the flattening strategies of increasing complexity and effectiveness. In particular we consider the following IFS procedures:

- Two IFS procedures based on PCP search named PCPS and PCPS- PROP, where PCPS- PROP uses the *Time-Tabling* propagation and PCPS does not. Both PCPS and PCPS- PROP are implemented as a depth-first backtracking procedures using an input parameter α, which is used to limit the number of backtracking steps. In particular, both procedures return the solution found with minimal make-span within $\alpha \cdot n$ backtracking steps, where n is the number of activities in the problem. The complexity of the basic solving procedure (without backtracking) is $O(d \cdot (n^2 + n \cdot s_{max}))$ in the case in which we use Time-Tabling propagation, and $O(d \cdot n^2)$ in the case in which we do not, where d is the number of precedence constraints posted, and and $n \cdot s_{max}$ is the complexity of Time Tabling propagation. In the worst case $d = O(n^2)$, but in many cases we can assume $d = O(n)$.
- Two IFS procedures based on SST search named SSTS and SSTS- PROP, where SSTS- PROP similarly uses Time-Tabling propagation and SSTS does not. As before, SSTS-PROP (or SSTS) uses a parameter α to bound the number of backtracking steps to the value $\alpha \cdot n$ and returns the best makespan solution found within $\alpha \cdot n$ steps. The complexity of this basic solving procedure (without considering backtracking) is $O(n(n^2 + n \cdot s_{max}))$ in the case that Time-Tabling propagation is used and $O(n^3)$ in the case that it is not.

The above four solving procedures vary in complexity and search structure, and the use of each as the flattening step within IFS offers an alternative tradeoff in emphasis with respect to the computational effort put into generating one solution sample and the number of solutions sampled during the search. In the experimental section that follows we test the effects of each of these procedures on two different benchmarks of Resource Constrained Scheduling containing instances of different sizes and structures.

Experimental analysis

This section presents the results of our investigation of the possibilities offered by different flattening algorithms within the IFS metaheuristic strategy. We subdivide the experimental analysis into two phases. First, we perform an *explorative* evaluation on subsets of the benchmarks described in this section, to test the effectiveness of all variants across a range of parameter settings. This phase is aimed at selecting the best variants for the second intensive phase. In this second phase, we evaluate the selected variants on the entire benchmark problem suites.

Benchmark problems

As stated earlier, we consider two types of resource-constrained scheduling problems: the multi-capacity job shop scheduling problem (MCJSSP) and the resource constrained project scheduling problem (RCPSP). These problems and benchmark problem sets are briefly summarized in the paragraphs below.

The *multi-capacity job-shop scheduling problem* involves synchronizing the use of a set of m renewable resources to perform a set of *jobs* $J = \{j_1 \ldots j_{n_j}\}$ 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 for its entire duration. Each resource k is required only once in a job and can process at most c_k activities at the same time ($c_k \geq 1$). Hence, broadly speaking, MCJSSP has the same causal structure of the Job-Shop Scheduling Problem (JSSP) but involves multi-capacitated resources instead of unit-capacity ones. For our analysis, we refer to the benchmarks introduced by Nuijten and 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. In particular we distinguish:

- Set A: $LA1\text{-}10\times2\times3$ (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 $20\times5(2)$, $30\times5(3)$, $30\times5(2)$, $45\times5(3)$.
- Set B: $LA11\text{-}20\times2\times3$. 5 problems each of sizes $40\times5(2)$, $60\times5(3)$, $20\times10(2)$, $30\times10(3)$.
- Set C: $LA21\text{-}30\times2\times3$. 5 problems each of sizes $30\times10(2)$, $45\times10(3)$, $40\times10(2)$, $60\times10(3)$.
- Set D: $LA31\text{-}40\times2\times3$. 5 problems each of sizes $60\times10(2)$, $90\times10(3)$, $30\times15(2)$, $45\times15(3)$.

These benchmark problems represent an interesting and challenging benchmark for comparing algorithms. In relatively few instances they cover a wide range of problem sizes. At the same time, the difficulty of various instances are to some extent understood. As noted in Nuijten and 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. Hence, even if better solutions are known for several instances, the distance from these upper-bound solutions still provides a useful measure of solution quality.

The *resource constrained project scheduling problem* (RCPSP) is a more general scheduling problem. According to the definition given in "The resource constrained scheduling problem", it consists of a set of n activities and a set of r renewable resources. Each resource k is available in a given constant amount. Each activity a_i has a duration p_i and requires a constant amount of resource r_{ik} to be processed. Preemption is not allowed. Activities are related by two sets of constraints: temporal constraints modelled through precedence constraints, and resource constraints that state that for each time instant and for each resource, the total demand cannot exceed the resource capacity c_k. The objective considered here is also makespan minimization, in this case of the project. We consider instances of RCPSPs available at http://129.187.106.231/psplib. These instances are generated by *ProGen*, a well-known algorithm for the generation of a general class of precedence and resource-constrained scheduling problems. In particular, benchmark instances are available with sizes of 30, 60, 90 and 120 activities, called respectively $J30$, $J60$, $J90$ and $J120$. The random generation algorithm ProGen accepts as input several parameters, which characterize problem hardness. Indeed, within the J120 benchmark, there are still several instances which are not solved to optimality in spite of a large number of powerful algorithms that have been proposed in the literature over the past twenty years. In the following we consider a subset of this benchmark for our analysis of IFS flattening strategies.

For the *explorative phase* we consider Set C of the MCJSSP benchmarks[3] (20 instances) and a random selection of 20 *hard* instances of J120 RCPSP—named J120-20—for which optimal solutions are still not known. For the *intensive phase* we consider the full benchmark suite of 80 MCJSSP instances and a benchmark set of RCPSP instances—named J120-80—obtained by the union of the previously selected 20 hard instances and a further random selection of 60 instances within the set of J120 problems proposed in the literature.

Explorative experiments

In the following we evaluate a variant of the IFS schema that incorporates each of the 4 flattening procedures introduced in "Iterative flattening search variants": PCPS and SSTS, and their respective counterparts with Time-Tabling resource constraint propagation, PCPS- PROP and SSTS- PROP. Each variant incorporates the COMBORELAX strategy as the relaxation step. All algorithms were implemented in Allegro Common Lisp and were run on a Pentium 4 processor 2.6 GHz, under Linux. For the exploratory stage, the following general settings for the IFS strategies were assumed:

1. The initial solution for each IFS variant was generated by using the same flattening strategy that is 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 step.
2. 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

[3] 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 that are very interesting structurally and challenging in size that ranges from 300 to 600 activities. Hence, it is quite suitable for exploring interesting trends among the IFS strategies before a time consuming intensive testing.

number of backtracking steps equal to 10 % of the number of activities.
3. For the COMBORELAX strategy the probability values p_r for both the critical-path relaxation and chain-based relaxation strategies were set to the same percentage values $p_r \in \{10, 15, 20\}$.
4. The parameter $MaxRlxs$ was set to $MaxRlxs = 6$ for the critical-path based relaxation.
5. A 400 s timeout value was imposed on the Set C problem instances and 200 s for the J120 problem instances.
6. For each strategy we set $MaxFail = 3200$ (the maximum number of *non-improving* moves that the algorithm will tolerate before terminating).

In addition, in order to fully exploit the allotted time, 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 same initial solution until the time bound is reached. At the end, the best solution found is returned.

Table 1 compares the performance of each IFS strategy on the MCJSSP's Set C with respect to the value $\Delta LWU_{\%}$, which represents the average percentage deviation from the Lawrence upper bound (Lawrence 1984). Given a numeric value in the table, (for example 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 (either *Propagation* or *No Propagation*), indicating the inclusion of Time-Tabling propagation in the corresponding flattening procedures. Within this identified sub-table, for a given numeric value, we have the corresponding values for the parameters α and p_r (for example, the performance value 2.1 is obtained by using the combination PCPS and *No Propagation*, with parameter values $\alpha = 15$ and $p_r = 10$).

Table 2 compares the performance of each IFS strategy across the selected 20 instances of RCPSP using the same notation as in Table 1. Some trends are evident from the results in Tables 1 and 2:

1. PCPS performs better than SSTS.
2. Resource constraint propagation has an opposite effect in solving the two benchmark sets. In fact, in the case of MCJSSP Set C (where problems range in size between 300 and 600 activities) better performance is obtained without the use of Time-Tabling propagation. Alternatively, in the case of J120 RCPSP problems, resource propagation improves performance.

Intensive experiments

In this section we analyze the behavior over time of the best performing IFS strategies from the exploratory phase on the *full* 80 instances of the MCJSSP benchmark problem suite and on the extended 80 instance subset of the J120 RCPSP benchmark suite. Recall that the aim of this study is to experimentally evaluate the tradeoff associated with using flattening procedures of varying computational complexity and effectiveness. Our analysis is centered around the idea that given a time bound to the optimization process, there is a IFS optimization process tradeoff to be made between maximizing the number of iterative sampling cycles performed and maximizing the effectiveness of the inner flattening procedure. The more complex (and effective) the flattening procedure becomes, the smaller the number of iterative sampling cycles that can be performed within the time limit becomes. Our study evaluates four different flattening procedures of increasing complexity, to analyze this algorithm design tradeoff with respect to input problem size and imposed CPU bounds. For this intensive experimentation we adopted the settings for the parameters α and p_r which gave the best results in the preliminary phase and allocated more time to the experimentation. More specifically:

1. In the case of MCJSSP instances, we adopted the parameter settings $\alpha = 15$ and $p_r = 10$ for both the PCPS and PCPS- PROP procedures, as these settings yielded the best performance in the preliminary results of Table 1. Alternatively, for the SSTS and SSTS- PROP procedures, we selected the values $\alpha = 5$ and $p_r = 20$. Precedence and chain relaxation strategies use the same parameter p_r.
2. In the case of RCPSP instances, we adopted the parameter settings $\alpha = 25$ and $p_r = 10$ for both the procedure PCPS and PCPS- PROP; these settings yielded the best performance in the preliminary results of Table 2. Whereas for the procedures SSTS and SSTS- PROP we selected the values $\alpha = 15$ and $p_r = 15$. As above, precedence and chain relaxation strategies use the same parameter p_r.
3. A timeout of 3200 seconds 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 3 shows the average values $\Delta LWU_{\%}$ for both the full set MCJSSP and the single subsets A, B, C and D. The results reasonably confirm the behavior found in the first preliminary phase: the PCP-based procedures outperform the SST procedures; and, on average (column *All*), the IFS procedures not using propagation (PCPS and SSTS) outperform their propagation counterparts (PCPS- PROP and SSTS- PROP). However, when we analyze performance for the single subsets (Set A, B, C and D), we observe that on Set A (the smallest problems, which range in size between 100 and 225 activities) the best procedures are those that incorporate Time-Tabling propagation: (PCPS- PROP and SSTS- PROP).

Table 1 Set C—preliminary experiments

	α	PCPS			SSTS		
		$p_r = 10$	$p_r = 15$	$p_r = 20$	$p_r = 10$	$p_r = 15$	$p_r = 20$
Propagation	0	5.0	6.5	7.8	12.1	11.3	9.3
	5	5.5	6.4	8.1	11.8	10.8	9.4
	10	5.6	7.5	8.6	11.9	11.0	9.7
	15	6.8	7.8	8.9	11.9	11.2	9.5
	20	7.0	8.7	10.2	12.6	11.3	10.1
	25	8.1	9.2	10.9	12.4	12.3	10.2
	30	7.9	8.7	10.1	12.4	11.3	10.8
No propagation	0	5.7	6.8	8.3	7.8	6.6	5.0
	5	3.0	4.1	5.5	6.2	5.3	**4.3**
	10	3.0	4.3	5.1	6.9	6.2	5.4
	15	**2.1**	3.4	5.7	8.2	6.0	5.6
	20	2.7	3.6	4.9	8.3	7.0	5.6
	25	2.6	3.3	5.1	8.6	7.8	6.4
	30	2.5	3.4	5.1	9.6	7.8	6.5

Table 2 J120-20—preliminary experiments

	α	PCPS			SSTS		
		$p_r = 10$	$p_r = 15$	$p_r = 20$	$p_r = 10$	$p_r = 15$	$p_r = 20$
Propagation	0	7.6	9.4	11.2	10.6	10.1	9.7
	5	7.4	8.2	10.4	9.5	9.7	10.0
	10	7.1	8.6	10.3	10.5	9.7	9.8
	15	6.9	8.4	10.3	9.5	9.7	9.6
	20	7.0	8.9	10.5	10.2	9.7	9.6
	25	**6.8**	8.2	10.5	10.1	9.5	10.0
	30	7.8	8.5	10.8	9.7	9.6	9.7
No propagation	0	9.4	11.0	11.5	10.8	10.0	9.9
	5	9.1	9.4	11.1	9.3	8.8	8.2
	10	7.8	9.8	11.1	9.0	8.8	9.1
	15	8.3	9.0	11.0	9.0	**8.2**	9.0
	20	8.1	8.5	11.3	8.9	9.0	9.0
	25	7.6	8.7	10.7	9.3	8.7	8.6
	30	7.8	9.1	10.3	9.5	8.4	9.1

An expanded view of the same data is offered by the graphics shown in Figures 8 and 10, where we represent the value $\Delta LWU_{\%}$ over computation time. This view of dynamic solving performance adds further information with respect to the content of Table 3. First of all, we see that the difference in performance between the different variants of a given solving procedure (i.e., PCPS- PROP and PCPS or SSTS- PROP and SSTS) generally tends to shrink as time progresses. In the extreme case of Set A, there is a time instant at which both solving procedures with Time-Tabling propagation begin to outperform their counterparts without propagation, and the best overall performing procedure is PCPS- PROP.

The same trend of Fig. 10 (a) for Set A is shown in Fig. 9 for the RCPSP benchmark J120-80, where we report the performance of our algorithms as the average percentage deviation from the best known makespan solutions. We observe that even in this case we have problem instances with relatively *small sizes* (120 activities).

Again, we see the PCP-based procedures outperform the SST procedures, and after a given time instant the solving

Table 3 $\Delta LWU_{\%}$ values on the complete benchmark

	Set A	Set B	Set C	Set D	All
PCPS- PROP	**−0.16**	−1.29	1.53	1.64	0.43
PCPS	−0.11	**−1.66**	**0.48**	**0.23**	**−0.26**
SSTS- PROP	0.20	−0.39	4.07	3.59	3.58
SSTS	0.41	−0.74	2.81	2.44	1.23

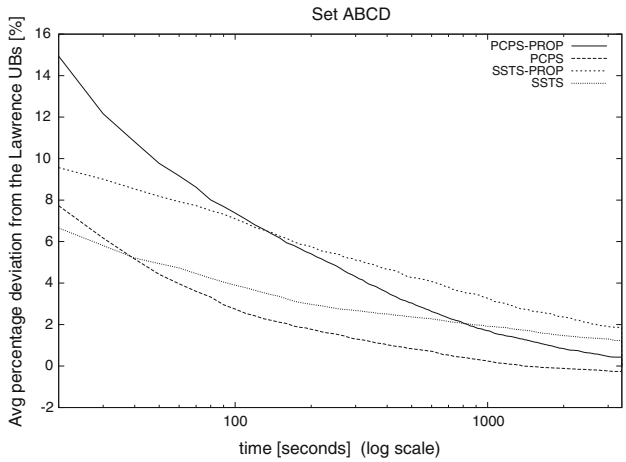


Fig. 8 Cumulative run-time performance for MCJSSP

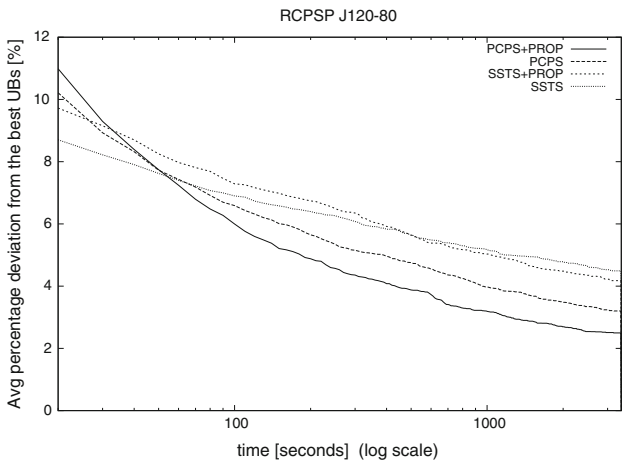


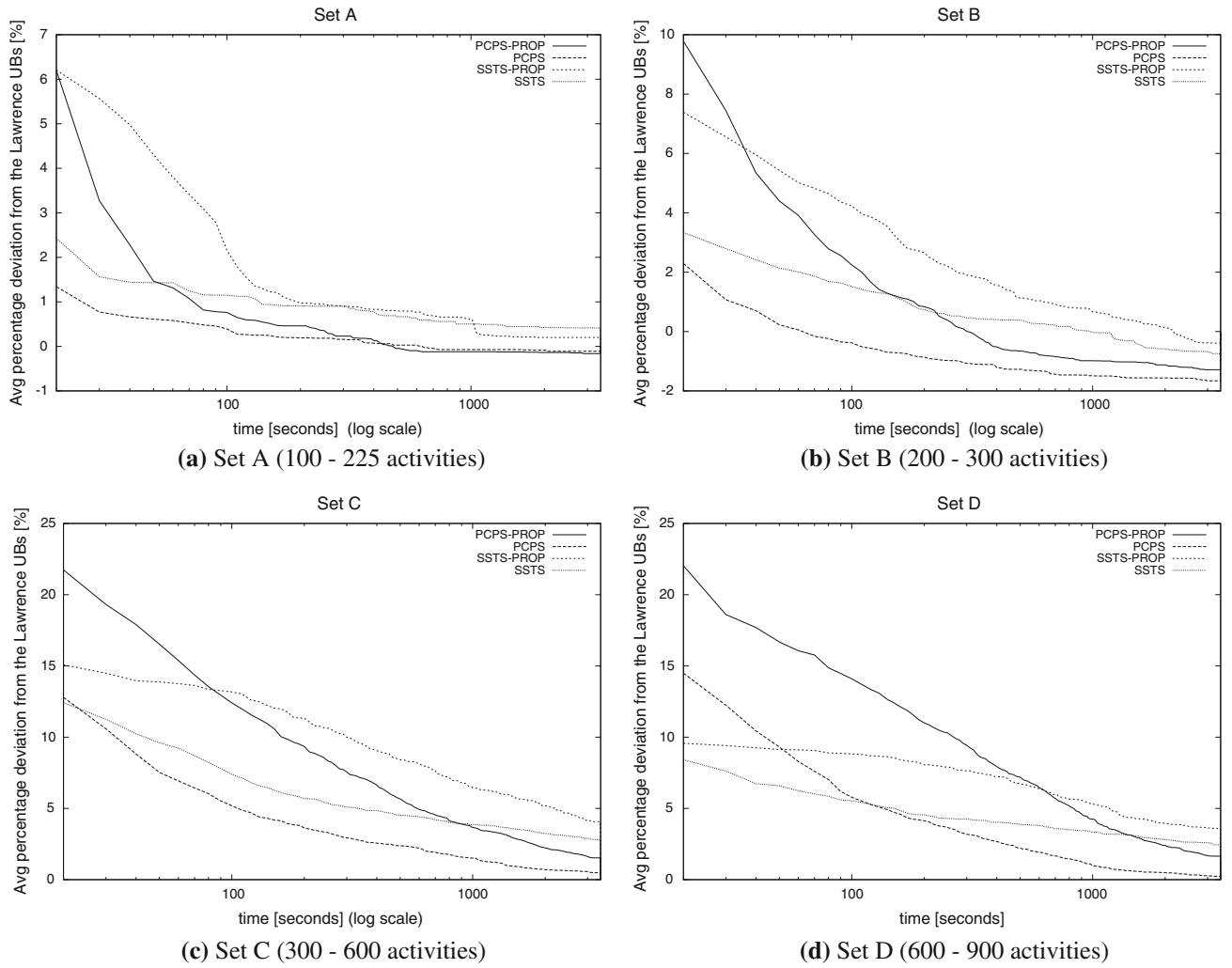
Fig. 9 Cumulative run-time performance for RCPSP (J120-80)

procedures including resource propagation (PCPS- PROP and SSTS- PROP) outperform their counterparts without propagation and the best performing procedure is PCPS- PROP.

Broadly speaking the main IFS steps can be compared directly to the concepts of *diversification* and *intensification* in a generic stochastic optimization procedure, where the diversification component is provided by the relax-flattening loop and the inner flattening search provides the intensification component. Interpreting our experimental results from this perspective, some qualitative guidelines for designing effective IFS procedures can be identified. In particular, we observe that the dominant optimization mechanism is represented by the iterative randomized sampling mechanism provided by the IFS loop; whereas, the constituent flattening procedure provides a *refinement* step to further improve the quality of the current solution. Hence, the two optimization mechanisms are generally not equivalent, and according to the size of the problem and the available computation time, it is important to maintain an *adequate* level of diversification (which implies greater emphasis on maximizing the number of iterations). Of course, when there are no tight cpu-time limitations, best performance is obtained by using the most complex (and effective) flattening procedure. But when solving larger problems under tighter time limits, it is possible to improve performance by decreasing the complexity of the inner flattening procedure. This can be accomplished by decreasing the number of backtracking steps (see Tables 1 and 2), by removing additional constraint propagation functionalities (this is the case of Set B, C and D in Fig. 10), or by choosing a computationally lighter flattening strategy. This last possibility is clearly visible for the Set D results shown in Fig. 10(d) (where problem sizes range from 600 to 900 activities). Noting that the solving procedures introduced in this paper can be ordered by increasing complexity as (SSTS, SSTS- PROP, PCPS and PCPS- PROP), we can see in Fig. 10(d) that initially SSTS (the lightest procedure) has the best performance, while the other strategies are bogged down by the larger size of the problem instances included in the Set D. However, as time passes, both the PCPS and PCPS- PROP procedures improve over SSTS.

Putting IFS in a broader perspective

Before ending the paper we would like to give a more abstract view of IFS and use this as a basis to stress connections and differences with other works. As already discussed the iterative flattening method for scheduling optimization alternates two phases: one in which the solution is partially *destroyed* and another in which the solution is *rebuilt*. In particular, the first step of IFS perturbs the current solution by selectively removing some number of ordering decisions among competing activities, while the second step applies a con-



(a) Set A (100 - 225 activities)

(b) Set B (200 - 300 activities)

(c) Set C (300 - 600 activities)

(d) Set D (600 - 900 activities)

Fig. 10 Run-time performance for each MCJSSP testset

straint-based algorithm to search for a new solution. Despite its simplicity, the approach offers the advantage of being able to provide a solution "any time", and has been shown to effectively scale to problems of significant size.

A similar two-step optimization schema has been proposed very recently for solving different types of scheduling problem than those considered here. In Ruiz and Stützle (2007) the authors propose an *Iterated Greedy* algorithm for solving instances of the well-known permutation flow-shop problem. This work was followed by Ruiz and Stützle (2008), where the same authors propose a different Iterated Greedy procedure for solving instances of flow-shop scheduling problems with sequence dependent setup times and both makespan and weighted tardiness minimization objectives. In both cases the experimental results obtained rank the proposed algorithms among the best state-of-the-art methods for solving the corresponding flow-shop problems. Some interesting similarities can be also noted between IFS and the *Shifting Bottleneck Procedure* for Job Shop Scheduling (Adams et al. 1988), which can also be case in the same general schema of *Destroy&Rebuild*. In particular, this procedure iteratively identifies the so-called *bottleneck machine* and sequences it to optimality by solving the corresponding one-machine problem. These decisions made at one resource perturb the schedules of those resources that stand upstream or downstream of scheduled activities, leading to successive re-optimization of the schedules for these machines.

The idea of iterative Destroy&Rebuild has been also used for solving other combinatorial optimization problems. Jacobs and Brusco (1995) and Marchiori and Steenbeek (1998) propose two different procedures for solving the Set Covering Problem, and in both cases very competitive experimental results were obtained. Prestwich (2000) proposes a constraint local search (CSL) hybrid algorithm for solving large instances of 3-SAT problems by randomizing its backtracking component to occur at arbitrarily-chosen variables. Finally, Shaw (1998) proposes a local search method, termed large neighbourhood search (LNS), for solving instances of

the vehicle routing problem (VRP) based upon a process of continual relaxation and re-optimization.

In general, it is worth noting that the core idea behind IFS (or Destroy&Rebuild, Iterated Greedy, etc.) represents by itself a new meta-heuristic proposal with similarities and differences with respect to other meta-heuristic scheme for solving general combinatorial optimization problems. In fact, within the current literature, there are at least two meta-heuristic methods which can be correlated to IFS. The first is greedy randomized adaptive search (GRASP) (Resende and Ribeiro 2002). This is a meta-heuristic which combines random constructive greedy heuristics and local search, such that the procedure iteratively composes two phases: construction and improvement. The best solution found is returned upon termination. The second similar meta-heuristic procedure is iterated local search (ILS) (Lourenco et al. 2002). ILS is a general meta-heuristic schema which first applies a local search to an initial solution until finds a local optimum. Then it perturbs the solution and restarts a local search procedure, and as before the best solution found is returned upon termination.

Even if IFS is correlated to GRASP and ILS, in both cases there are differences. With respect to GRASP, we note that it restarts from scratch at each cycle, whereas IFS only partially destroys the current solution. Hence, IFS provides a more general relaxation mechanism than GRASP, allowing incorporation of more flexible search procedures where it is possible to *tune* the size of the search space generated by relaxation. With regard to ILS, we can say that IFS can be seen as a ILS procedure where the perturbation mechanism is the most frequent operation, whereas in ILS the dominant operation is local search. So, IFS iteratively tries to discover a *block* of wrong decisions, remove it, and substitute a new, *better* block of decisions. Hence, a decisive step in the definition of an effective IFS procedure, is the explicit definition of a criterion for discovering wrong decisions.

Conclusions

In this paper, we have investigated the performance effects of various component strategies proposed by previous research for the flattening step of the iterative flattening search (IFS) model. The work was motivated by the observation that if a time bound is given to the overall IFS optimization process, the final solution produced will be the result of the interaction of two optimization mechanisms: (1) iterative sampling—embodied in the overall relax-flattening loop—which performs a global and broader sampling of the search space, and (2) local optimization—embodied by the inner flattening procedure—which within each iteration explores a set of nearby solutions. From a broader local search perspective, the former mechanism can be seen as promoting diversification while the latter mechanism emphasizes intensification. The important question concerns how these two mechanisms should be balanced to maximize performance, and to what extent various component mechanisms proposed by previous research strike the right balance in different time constrained circumstances.

Starting from the results of a previous study that evaluated a range of relaxation strategies, we conducted a comparative experimental analysis of four distinct flattening procedures, differentiated by the structure of the search that was performed and the extent to which resource constraint propagation was employed to promote early pruning. Each of these procedure was tested on two reference sets of benchmark problems from the resource-constrained scheduling literature. The results of this experimental analysis suggest basic guidelines for designing effective IFS procedures. In particular, we observed that the dominant optimization mechanism is the iterative randomized sampling provided by the IFS loop, whereas the inner flattening procedure was seen to provide a *refinement* step to further improve the quality of the current solution. This being the case, it becomes important to maintain an *adequate* level of diversification in the search as the size of the problems addressed is increased and/or the available computation time is decreased. For example, in solving larger problems within a given time bound, our results indicated that it is possible to improve performance by decreasing the number of backtracking steps allowed in the flattening search, by removing additional constraint propagation functionalities, or by choosing a lighter flattening strategy.

The experimental results presented in this paper have also further clarified some weaknesses and strengths of the ideas proposed over the past years about the use of different IFS component strategies for solving schedule makespan optimization problems. Most surprising was the superiority of PCPS over SSTS when both flattening procedures are placed in a uniform implementation framework. We have also analyzed the current literature and made correlations with similar approaches for solving not only scheduling problems other than Resource Constrained Scheduling, but also other hard optimization problems, like Set Covering or Vehicle Routing problems. An interesting direction for future research is the exploration of the simple and effective *Destroy&Rebuild* principle that underlies our IFS algorithm as a basis for solving broader classes of optimization problems. At present we are working to a version of the IFS for solving RCPSP/max—resource constrained scheduling problems with generalized precedence relations—a problem where the minimal and maximal time legs among activities make the problem of finding even a feasible solution *NP-hard*. However, our longer term goal is to generalize the results obtained with IFS in solving different scheduling problems toward the definition of an effective *Destroy&Rebuild* meta-heuristic strategy.

Acknowledgements 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 RSTL (Funds 2007) and ESA (European Space Agency) under the APSI project. 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 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

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

Blazewicz, J., Lenstra, J. K., & Rinnoy Kan, A. H. G.(1983). Scheduling projects subject to resource constraints: Classification and complexity. *Discrete Applied Mathematics, 5*(1), 11–24.

Blum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys, 35*(3), 268–308.

Cesta, A., Oddi, A., & Smith, S. F. (1998). Profile based algorithms to solve multiple capacitated metric scheduling problems. In *AIPS-98. Proceedings of the 4th 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 *AAAI00. Proceedings of the 17th National Conference on Artificial Intelligence* (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).

Gomes, C. P. (2003). Complete randomized backtrack search. In M. Milano (Ed.), *Constraint and integer programming: Toward a unified methodology* (pp. 233–283). Kluwer.

Hoos, H. H., & Stützle, T. (2005). *Stochastic local search. Foundations and applications*. San Francisco: Morgan Kaufmann.

Jacobs, L. W., & Brusco, M. J. (1995). A local search heuristic for large set-covering problems. *Naval Research Logistic Quarterly, 42* (7), 1129–1140.

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

Langley, P. (1992). Systematic and nonsystematic search strategies. In *AIPS92. Proceedings of the First International Conference on Artificial Intelligence Planning Systems* (pp. 145–152). San Francisco: Morgan Kaufmann Publishers Inc.

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.

Le Pape, C. (1994). Implementation of resource constraints in ILOG schedule: A library for the development of constraint-based scheduling systems. *Intelligent Systems Engineering, 3*(2), 55–66.

Lourenco, H. R., Martin, O., & Stutzle, T. (2002). Iterated local search. In F. Glover & G. Kochenberger (Eds.), *Handbook of metaheuristics, international series in operations research & management science* (Vol. 57, pp. 321–353). Norwell, MA: Academic Publishers.

Marchiori, E., & Steenbeek, A. (1998). An iterated heuristic algorithm for the set covering problem. In *Proceedings WAE'98, Saarbrücken, Germany, August 20–22* (pp. 155–166).

Michel, L., & Van Hentenryck, P. (2004). Iterative relaxations for iterative flattening in cumulative scheduling. In *ICAPS04. Proceedings of the 14th International Conference on Automated Planning & Scheduling* (pp. 200–208).

Nuijten, W. P. M., & Aarts, E. H. L. (1996). A computational study of constraint satisfaction for multiple capacitated job shop scheduling. *European Journal of Operational Research, 90*(2), 269–284.

Oddi, A., Cesta, A., Policella, N., & Smith, S. F. (2008). Combining variants of iterative flattening search. *Journal of Engineering Applications of Artificial Intelligence, 21*(5), 683–690.

Policella, N., Cesta, A., Oddi, A., & Smith, S. F. (2007). From precedence constraint posting to partial order schedules. *AI Communications, 20*(3), 163–180.

Prestwich, S. (2000). A hybrid search architecture applied to hard random 3-SAT and low-autocorrelation binary sequences. In *2000CP00. The 6th International Conference on Principles and Practice of Constraint Programming, LNCS* (Vol. 1894, pp. 337–352). Springer-Verlag.

Resende, M. G. C., & Ribeiro, C. C. (2002). Greedy randomized adaptive search procedures. In F. Glover & G. Kochenberger (Eds.), *State of the art handbook in metaheuristics*. Kluwer.

Ruiz, R., & Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research, 177*(3), 2033–2049.

Ruiz, R., & Stützle, T. (2008). An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objective. *European Journal of Operational Research, 187* (3), 1143–1159.

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In *CP98. The 4th International Conference on Principles and Practice of Constraint Programming, LNCS* (Vol. 1520, pp. 417–431). Springer-Verlag.