Iterative Relaxations for Iterative Flattening in Cumulative Scheduling

Laurent Michel and Pascal Van Hentenryck

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-03-21
October 2003

Iterative Relaxations for Iterative Flattening in Cumulative Scheduling

Laurent Michel and **Pascal Van Hentenryck**
University of Connecticut, Storrs, CT 06269-3155
Brown University, Box 1910, Providence RI 02912

Abstract

Cumulative scheduling is a generalization of jobshop scheduling, where machines have discrete capacities and activities may require several capacity units. This paper considers iterative flattening, a heuristic procedure aiming at producing high-quality solutions to large cumulative problems in reasonable time. On standard benchmarks (with as much as 900 activities), iterative flattening quickly delivers solutions that are within 10% of the best upper bounds in average. This paper analyzes iterative flattening and identifies a pathological behaviour which explains why it may result in solutions where resources are under-utilized in early parts of the schedules. It proposes a simple extension of the algorithm which has dramatic effect on the quality of the solutions, while preserving its computational efficiency. The new algorithm found 21 new best upper bounds to the same benchmarks and quickly delivers solutions that are within 1% of the best available upper bounds in the average.

Introduction

Cumulative scheduling is a generalization of jobshop scheduling where each activity requires some capacity of the discrete resource on which it must execute. The goal is to minimize the makespan while satisfying the precedence constraints and making sure that, at each time step, the capacity of each machine is not exceeded by the activities executing on the machine.

In recent years, increased attention has been devoted to local search and heuristic procedures for cumulative scheduling. These include the iterative flattening algorithm (IFLAT) (Cesta, Oddi, & Smith 2000) which emerged from a series of papers, e.g., (Oddi & Smith 1997; Cesta, Oddi, & Smith 1999a; 1999b). Iterative flattening was motivated by the desire to obtain high-quality solutions to large cumulative scheduling problems (e.g., involving as much as 900 activities) in reasonable time. Starting from an infeasible schedule violating the resource constraints, IFLAT iterates two steps: a flattening step which adds precedence constraints to remove infeasibilities and a relaxation step which removes some of the added precedence constraints to provide a new starting point for flattening. Algorithm IFLAT was shown to produce high-quality solutions (usually within 10% of the best known upper bounds) in reasonable time on a variety of traditional benchmarks. As a consequence, iterative flattening provides an interesting and orthogonal alternative to constraint programming approaches (e.g., (Nuijten 1994; Nuijten & Aarts 1996)) which typically combine systematic search with domain reductions based on edge-finding algorithms.

This paper reconsiders iterative flattening and makes two main contributions. First, it provides a careful analysis of the behavior of algorithm IFLAT. The study reveals a pathological behavior which explains why iterative flattening sometimes produces solutions where resources are significantly under-utilized in early parts of the schedule. It also shows that this anomaly can be explained by IFLAT's failure to identify and alleviate some of the bottlenecks in the schedule. Second, the paper proposes a simple extension to algorithm IFLAT which dramatically improves the quality of its schedules, while preserving its computational efficiency. They key idea is to iterate the relaxation step multiple times in order to isolate the schedule bottlenecks in a sequential fashion. The resulting algorithm IFlatIRelax found many (21) new upper bounds to cumulative scheduling benchmarks and typically produces solutions that are on average within 1% from the best upper bounds. As a consequence, IFlatIRelax broadens the scope of the original algorithm, while retaining its benefits, and makes it a very appealling approach for many large-scale problems.

The rest of this paper is organized as follows. It first specifies the problem, describes the high-level modeling used by the algorithms, and introduces precedence graphs and various related concepts. It then describes the iterative flattening algorithm precisely and studies its limitation. Iterative relaxation and the new algorithm are presented next and the paper explains why they address the limitations of IFLAT. Experimental results on the new algorithm are then presented and contrasted with the original procedures. The last section contains the conclusion of this research.

Problem Specification

A cumulative scheduling problem can be specified as follows. We are given a set of jobs J, each of which consisting of a sequence of activities $\langle a_1, \ldots, a_n \rangle$. Each activity a_i has a duration $d(a_i)$, a machine $m(a_i)$ on which it executes, and a demand $r(a_i)$ for the capacity of this machine. There is

Copyright © 2003

a precedence constraint (a_{i-1}, a_i) for $i \geq 2$ in the job sequence $\langle a_1, \ldots, a_n \rangle$. Each machine $c \in M$ has an available capacity $cap(c)$. The problem is to minimize the earliest completion time of the project (i.e., the makespan), while satisfying all the precedence and resource constraints.

More formally, let $\mathcal{A}$ be the set of activities, $\mathcal{P}$ the set of precedence constraints (a_i, a_j) induced by the job sequences, $\mathcal{A}(m)$ the activities requiring machine m, i.e.,

$$\mathcal{A}(m) = \{a \in \mathcal{A} \mid m(a) = m\},$$

and $H = [0, \sum_{a \in \mathcal{A}} d(a)]$ the schedule horizon. A *schedule* is an assignment $\sigma : \mathcal{A} \rightarrow H$ which assigns a starting date $\sigma(a)$ with each activity a. A schedule is *feasible* if it satisfies the precedence and cumulative constraints, i.e.,

$$\forall (a_i, a_j) \in \mathcal{P} : \sigma(a_j) \geq \sigma(a_i) + d(a_i); \quad (1)$$

$$\forall t \in H, \forall m \in M : \sum_{a \in \mathcal{A}(\sigma, m, t)} r(a) \leq cap(c) \quad (2)$$

where $\mathcal{A}(\sigma, m, t)$ is the set of activities requiring machine m at time t, i.e.,

$$\mathcal{A}(\sigma, m, t) = \{a \in \mathcal{A}(m) \mid \sigma(a) \leq t \leq \sigma(a) + d(a)\}.$$

An *optimal* schedule is a feasible schedule σ which minimizes the makespan, i.e.,

$$\max_{a \in \mathcal{A}} \sigma(a) + d(a).$$

Problem Modeling

At a very high level, the algorithms described in this paper model the cumulative scheduling problem by dividing the constraints into two sets: the *hard* constraints which are the precedence constraints (1) and the *soft* constraints which are the cumulative constraints (2). All schedules considered in this paper satisfy the precedence constraints induced by the job sequences, as well as the additional precedence constraints introduced during search. These schedules are called *precedence feasible*. The cumulative constraints may be violated by the schedule and the main goal of the algorithms is to reduce the number of violations until the schedule becomes feasible. In the following, we often abuse notations and use machine and cumulative constraint interchangeably.

Given a schedule σ, a *violation* for a cumulative constraint $c \in M$ is a time t such that

$$\sum_{a \in \mathcal{A}(\sigma, c, t)} r(a) > cap(c)$$

and we use $V(\sigma, c)$ to denote the violations of c in σ and $nbv(\sigma, c)$ to denote $|V(\sigma, c)|$. The violation degree $vd(\sigma, c, t)$ of violation t for the cumulative constraint c in schedule σ is the demand exceeding the capacity, i.e.,

$$\max(0, cap(c) - \sum_{a \in \mathcal{A}(\sigma, c, t)} r(a)).$$

Given a violation t for constraint c and schedule σ, a critical set of activities for (σ, c, t) is a set of activities $C \in \mathcal{A}(\sigma, c, t)$ such that

$$\sum_{a \in C} r(a) > cap(c).$$

A critical set is *minimal* if all its proper subsets satisfy the constraint, i.e.,

$$\forall C' \subset C : \sum_{a \in C'} r(a) \leq cap(c).$$

Minimal critical sets were introduced in (Laborie & Ghallab 1995) but there may be exponentially many of them for a violation t. The algorithms discussed in this paper uses a heuristic to identify a quadratic number of "interesting" critical sets. The details are discussed in (Cesta, Oddi, & Smith 2000) and we denote these subset by $quad(\sigma, c, t)$ in the following.

Precedence Graphs

As mentioned, the algorithms in this paper are always precedence feasible. As a consequence, it is useful to associate a precedence graph with a set of precedence constraints, since it helps defining a variety of concepts and implicitly specifies a schedule.

A *precedence graph* for $\mathcal{A}$ and $\mathcal{P}$ is a graph $G = (\mathcal{A} \cup \{s, t\}, E)$, where the nodes are the activities, a source s, and a sink t, and where E contains three sets of arcs

1. $s \rightarrow a \quad (a \in \mathcal{A})$;
2. $a \rightarrow t \quad (a \in \mathcal{A})$;
3. $a \rightarrow b \quad ((a, b) \in \mathcal{P})$;

An arc $a \rightarrow b$ has weight $d(a)$ with the convention that $d(s) = 0$. The precedence graph is instrumental in computing the earliest starting dates and latest finishing date of every activity. The earliest starting date of an activity a, denoted by $esd(a)$, is the longest path from s to a in the precedence graph. The makespan ms is the earliest starting date of the sink $esd(t)$. The latest starting date of an activity a, denoted by $lsd(a)$, is the makespan ms minus the longest path from a to t. An activity is *critical* if $esd(a) = lsd(s)$. An arc $a \rightarrow b$ is *critical* if a and b are critical and if $esd(a) + d(a) = esd(b)$.

The precedence graph for $\mathcal{A}$ and $\mathcal{P}$ implicitly defines a precedence feasible schedule σ defined as

$$\forall a \in \mathcal{A} : \sigma(a) = esd(a).$$

As a consequence, given a set S of precedence constraints, we use $precedenceShedule(S)$ to denote the corresponding precedence feasible schedule σ, $esd(S, a)$, $lsd(S, a)$, $ms(S)$, and $Critical(S)$ to denote the earliest and latest starting dates of a for S, the makespan for S, and the set of critical arcs in the precedence graph associated with S.

The main operations of the algorithms are the addition and removal of precedence constraints or, equivalently, of arcs in the precedence graph. There exist good incremental algorithms to implement these operations (Michel & Van Hentenryck 2003). In particular, the addition and removal of an arc respectively takes time $O(\Delta \log \Delta)$ and $O(\Delta)$, where Δ is the sets of vertices whose earliest and latest starting dates are updated.

```
function IFLAT()
begin
1.    S := P;
2.    σ := precedenceSchedule(S);
3.    bestMs := ∞;
4.    nbStableIt := 0;
5.    forall (i ∈ 1..maxIterations)
6.      (S, σ) := FLATTEN(S, σ);
7.      if (ms(S) < bestMs) {
8.         σ* := σ;
9.         bestMs := ms(S);
10.        nbStableIt := 0;
11.     } else
12.        nbStableIt++;
13.     if (nbStableIt < maxStableIt)
14.        break;
15.     (S, σ) := RELAX(S, σ);
16.   return σ*;
end
```

Figure 1: The Iterative Flattening Algorithm IFLAT

Iterative Flattening

Iterative flattening (Cesta, Oddi, & Smith 2000) is a heuristic procedure that iterates two main steps:

1. a *flattening* step which transforms a precedence feasible schedule into a feasible schedule by introducing precedence constraints;
2. a *relaxation* step which relaxes a feasible schedule into a possibly infeasible, but precedence feasible, schedule by removing some precedence constraints introduced in the flattening step.

These two steps are executed for a number of iterations (i.e., *maxIterations*) or until no improved feasible schedule has been found for a number of iterations (i.e., *maxStableIt*). The algorithm returns the best feasible schedule found after the flattening step (step 1).

Figure 1 depicts the core of the iterative flattening algorithm. Observe line 6 which applies procedure FLATTEN to generate a feasible schedule and a new set of precedence constraints, as well as line 15 which applies the relaxation algorithm RELAX to remove some of the precedence constraints introduced by FLATTEN. These two main steps are iterated for *maxIterations* (line 5) or until no improved schedule has been found for *maxStableIt* iterations (lines 13 and 14). Lines 8-10 updates the best schedule and makespan whenever the algorithm finds a new feasible schedule which improves the makespan. The rest of this section describes the FLATTEN and RELAX procedures in more detail.

Flattening

The flattening algorithm aims at removing all violations of the cumulative constraints by additing precedence constraints. It selects a cumulative constraint c with the most violations, i.e., maximizing $nbv(\sigma, c)$. Once the constraint c is selected, it chooses the violation t with maximal violation degree $vd(\sigma, c, t)$. To try removing this violation, the flattening algorithm selects the two activities a_i and a_j in the minimal critical sets and a precedence constraint $a_i \rightarrow a_j$.

```
function FLATTEN(S, σ)
begin
1.    while (∃c ∈ M : nbv(σ, c) > 0)
2.      c* = argmax(c ∈ C) nbv(σ, c);
3.      t* = argmax(t ∈ V(σ, c)) vd(σ, c*, t);
4.      (S*, a_i*, a_j*) =
            argmax(C ∈ quad(σ, c*, t*), (a_i, a_j) ∈ S)
                lsd(S, a_j) − esd(S, a_i);
5.      S := S ∪ {a_i* → a_j*};
6.      σ := precedenceSchedule(S);
7.    return (S, σ);
end
```

Figure 2: The FLATTEN Implementation

```
function RELAX(S, σ)
begin
1.    S* := S;
2.    forall ((a_i, a_j) ∈ Critical(S) \ P)
3.      if (RANDOM(0, 1) ≤ relaxProbability)
4.         S* := S* \ {(a_i, a_j)};
5.    return (S, precedenceSchedule(S));
end
```

Figure 3: The RELAX Implementation

The two activities are chosen carefully in order to minimize the impact on the makespan. More precisely, the flattening algorithm selects the two activities a_i and a_j maximizing

$$lsd(S, a_j) - esd(S, a_i)$$

since this should keep as much flexibility as possible in the schedule. The pseudo-code is depicted in Figure 2. As long as there are violations (line 1), the implementation selects the cumulative constraints c^* with the most violations (line 2), the violation t^* with the largest violation degree (line 3), and the two activities (a_i^*, a_j^*) such that the precedence $a_i \rightarrow a_j$ "maximizes" flexibility (line 4). Lines 5 and 6 update S and σ and line 7 returns the feasible schedule σ and its associated precedene constraints S. Note that ties are broken randomly in each of these selections.

Relaxation

After the flattening, the algorithm has a feasible schedule σ and its set S of precedence constraints. Instead of restarting the flattening step from scratch (as a GRASP procedure would do), the key idea behind the RELAX procedure is to remove only some of the precedence constraints introduced by procedure FLATTEN. More precisely, the relaxation step considers the precedence constraints introduced by FLATTEN and removes them with some probability. Only critical precedence constraints (i.e., constraints which correspond to critical arcs) are considered in (Cesta, Oddi, & Smith 2000). This is natural, since only these constraints may decrease the makespan. The pseudo-code is depicted in Figure 3. Line 2 considers all critical precedence constraints which are not induced by the job sequences and lines 3 and 4 remove such a constraint with probability *relaxProbability*. Line 5 returns a new schedule which typically is infeasible, although it satisfies all the original precedence constraints.

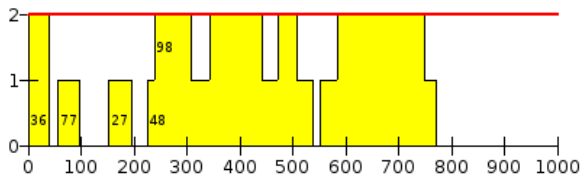

Figure 4: The Profile of Machine 3 after the First Call to FLATTEN.

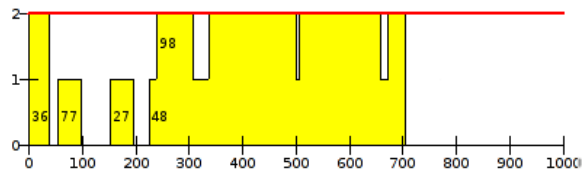

Figure 5: The Profile of Machine 3 after the First Call to RELAX.

Limitations of Iterative Flattening

Iterative flattening is an effective procedure to approach large cumulative scheduling that are out of scope for traditional systematic constraint-based scheduling systems. However, there are significant gap between the quality of the solutions returned by function IFLAT and the best known upper bounds on many benchmarks. Such a gap may be around 10% for many of these problems. This paper originated from an attempt to understand whether this was an inherent limitation to the approach or whether the algorithm could be extended to improve the quality of its solutions while retaining its efficiency.

This section tries to illustrate our findings intuitively and visually on a particular benchmark called LA01D. This benchmark has 100 activities and 5 machines. Each activity only requires 1 unit of capacity from its resource which all have capacity 2.

Consider the schedule after the first application of FLATTEN. The feasible schedule has a makespan of 770 and Figure 4 illustrates the resource profile on machine number 3. Obviously the cumulative constraint is satisfied but it exhibits a significant under-utilization before activity 48 and especially between times 100 and 200 where the resource is never used at full capacity. Machine 3 is thus severely under-utilized early in this schedule. Ideally, high quality schedules yield dense profiles where the machine is almost always operating at full capacity throughout the schedule.[1] As a consequence, we expected that the relaxation step would remove these gaps and under-utilizations, albeit at the cost of violating the cumulative constraint.

Figure 5 depicts the profile of machine 3 after the relaxation step (after execution of line 16 in Figure 1). *Observe that the early gaps/under-utilizations have not disappeared*. The profile is much more compact at the end of the schedule

[1] Of course, there are benchmarks where optimal solutions also exhibit gaps and under-utilizations in the profiles.

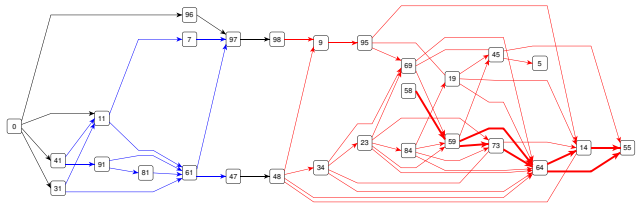

Figure 6: Projection of Precedence Graph wrt Machine 3

(it is not used after 707), but the early part of the schedule seems not affected by the relaxation. In addition, the cumulative constraint is not even violated. One may think that this is a side-effect of the randomization in choosing which precedences to remove, but other runs also exhibit the same pathological behaviour. In fact, The removal of all critical arcs does not affect the early parts of the schedule.

To understand the cause of this anomaly, it is useful to look at (parts of) the precedence graph. Figure 6 depicts a part of the precedence graph which focuses on machine 3 primarily. In the picture, black arcs are job precedences, blue arcs indicate machine precedences on machine 4 (i.e., precedence constraints introduced during FLATTEN to remove violations in machine 4), while red arcs correspond to machine precedences on machine 3 itself. Thick lines indicate arcs that belong to the critical path. Note also that the critical arcs on all machines are given by

$$\begin{aligned} C_1 &= \{\} \\ C_2 &= \{\langle 51, 66\rangle, \langle 66, 26\rangle\} \\ C_3 &= \{\langle 14, 55\rangle, \langle 59, 73\rangle, \langle 64, 14\rangle, \langle 73, 64\rangle\} \\ C_4 &= \{\} \\ C_5 &= \{\langle 3, 68\rangle, \langle 28, 93\rangle, \langle 53, 58\rangle, \langle 68, 53\rangle, \langle 93, 3\rangle\} \end{aligned}$$

A key aspect of the schedule depicted in Figure 6 is the central role of activities 48 and 98. These activities are scheduled on machine 3 just after the second large early gap. Observe that the only arcs entering activities 48 and 98 are job arcs, both of which originate from machine 4 (i.e., activities 47 and 97 respectively). *More importantly perhaps, there are no critical arcs (thick lines) from the source to these two activities*. As a consequence, removing critical arcs will not affect the release dates of the two activities. Finally, and this is also extremely pertinent to understanding the anomaly, observe that removing all critical arcs on machine 3 will not help in trying to schedule any other activities using machine 3 earlier. Indeed, as can be seen in Figure 6, all these tasks are reachable from activities 48 and 98 from multiple paths using arcs from machine 3. *In other words, activities* 48 *and* 98 *are really the bottlenecks which prevent other activities using machine 3 to be scheduled earlier and to utilize the machine more effectively in the early parts of the schedule.*

One may wonder why there is such a gap if there are no critical arcs before activities 48 and 98. The reason are the arcs from machine 4

$$\{\langle 31, 11\rangle, \langle 11, 61\rangle, \langle 61, 47\rangle\}$$

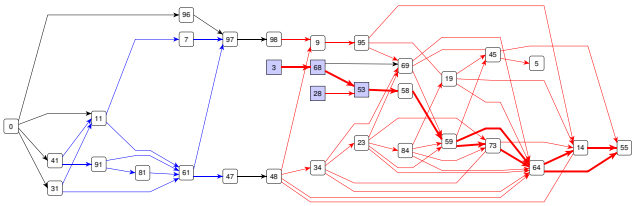

Figure 7: The Precedence Graph Projected wrt Machine 3 with some Critical Arcs from Machine 5.

but none of these arcs are critical of course. In fact, there are no critical arcs on machine 4 at all, as shown previously. The above arcs are critical if activities 48 and 98 are considered as sinks.

In conclusion, although it seems perfectly reasonable to focus on relaxing critical arcs (since it is only by removing some of them that one may hope to improve the makespan), the resulting relaxation fails to remove bottlenecks which are responsible for under-utilizations of machines even in the early parts of the schedule. *It seems thus important to relax non-critical arcs. How to detect which ones to consider is the topic of the next section and the basis of the new algorithm.*

Iterative Relaxation

We now describe how to remedy this limitation of iterative flattening. *The key observation is to recognize that the relaxation procedure has two effects. On the one hand, it packs the schedule and thus the profiles of each machine by introducing violations of the cumulative constraints. On the other hand, it changes the critical paths, exposing bottlenecks that were previously hidden.* These new bottlenecks are often responsible for under-utilizations of some of the machines in earlier parts of the schedule.

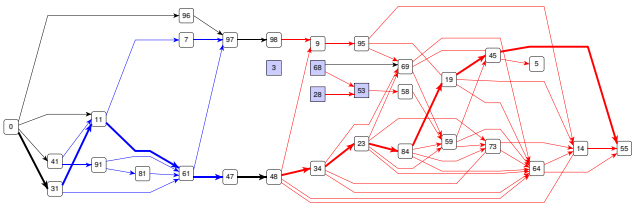

Figure 8: The Precedence Graph Projected wrt Machine 3 after the Relaxation of $3 \rightarrow 68$.

Consider Figure 7 which shows the same parts of the precedence graph with some additional critical activities and arcs from machine 5. Assume that the relaxation procedure removes arc $3 \rightarrow 68$ from the precedence graph. The effect of this removal on the precedence graph is shown in Figure 8. Observe that the arcs from machine 4

$$\{\langle 31, 11\rangle, \langle 11, 61\rangle, \langle 61, 47\rangle\}$$

are now critical. The job precedence $47 \rightarrow 48$ is also critical. Finally, there are many activities on machine 3 that are

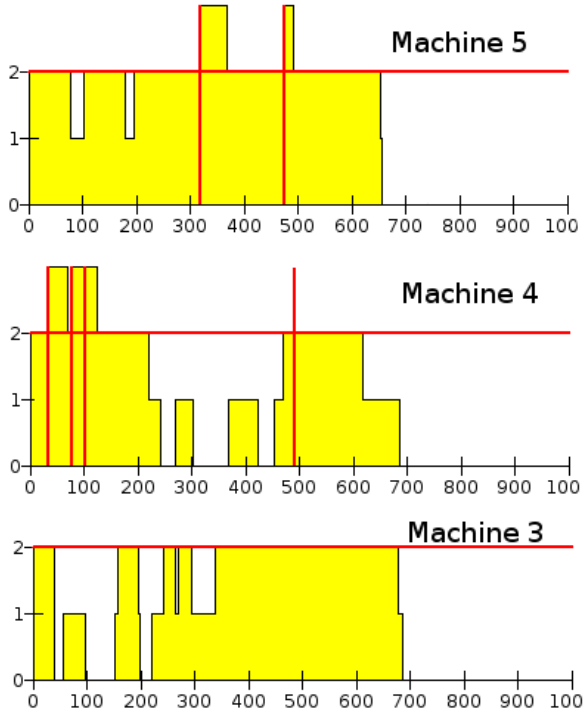

Figure 9: The Packed Profiles after a Second Relaxation

now on the critical paths and reachable by only one path. As a consequence, relaxing these critical arcs makes it possible to schedule some of these activities on machine 3 earlier, giving a better utilization of the resource. *As a consequence, it seems particularly beneficial to iterate the relaxation step, since it nicely identifies the various bottlenecks of the schedule in a sequential, step by step, fashion.*

Figure 9 illustrates the impact of a second relaxation which removes arcs the three critical arcs automatically detected on machine 4. Machine 3 is now used at full capacity in parts of the time interval $[100, 200]$, although there is still gaps. Note also that machine 4 is now exhibiting a significant under-utilization, which can be eliminated by further rounds of relaxation.

The key idea behind the new algorithm is thus surprisingly simple: it consists in iterating the relaxation step a number of times before returning to the flattening step. The pseudo-code of the new algorithm IFlatIRelax is given in Figure 10. The only change is in lines 15 and 16, which iterates the relaxation. Note that the experimental results indicate that a few iterations of RELAX may have a significant impact on the quality of the schedule.

Experimental Results

We now describe the experimental results on algorithm IFlatIRelax.

The Benchmarks The experimental results use the standard benchmarks from (Cesta, Oddi, & Smith 2000; Nui-

```
function IFLAT()
begin
1.   S := P;
2.   σ := precedenceSchedule(S);
3.   bestMs := ∞;
4.   nbStableIt := 0;
5.   forall(i ∈ 1..maxIterations)
6.     (S,σ) := FLATTEN(S,σ);
7.     if (ms(S) < bestMs) {
8.       σ* := σ;
9.       bestMs := ms(S);
10.      nbStableIt := 0;
11.    } else
12.      nbStableIt++;
13.    if (nbStableIt < maxStableIt)
14.      break;
15.    forall(j ∈ 1..maxRelaxations)
16.      (S,σ) := RELAX(S,σ);
17.  return σ*;
end
```

Figure 10: Algorithm IFlatIRelax: Iterative Flattening with Iterative Relaxation

jten & Aarts 1996). They consist of five classes of problems which are derived from jobshop scheduling problems by increasing the number of activities and the capacity of the resources:

set A Lawrence LA1-LA10 duplicated and triplicated.

set B Lawrence LA11-LA20 duplicated and triplicated.

set C Lawrence LA21-LA30 duplicated and triplicated.

set D Lawrence LA31-LA40 duplicated and triplicated.

set MT MT6-10-20 duplicated and triplicated.

Because they generalize jobshop scheduling problems, good upper bounds can be derived for these problems. Note however that this does not mean that algorithms tailored to cumulative scheduling can find these upper bounds easily, since they are not aware of this underlying structure. To the contrary, the results in (Cesta, Oddi, & Smith 2000; Nuijten & Aarts 1996) show that reaching these upper bounds is often challenging.

Sets C and D include large-scale benchmarks. In set C, the benchmarks vary in size from $(30, 10)$ to $(60, 10)$. In set D, they vary in size from $(60, 10)$ to $(90, 10)$ (first half `la31-la35`) and $(30, 15)$ to $(45, 15)$ (second half, `la36-la40`). No results on these sets are reported for the systematic, constraint-based, approach in (Nuijten & Aarts 1996).

Prior Results Table 1 summarizes prior results from (Cesta, Oddi, & Smith 2000) on iterative flattening. Each group of three rows reports results on a given algorithm. FLATTEN denotes the simple flattening procedure (without relaxation). IFLAT$_1$ and IFLAT$_5$ correspond to the iterative flattening procedure with one and five random restarts respectively. Row (Qual.) of each block gives the deviation in percentage from the best known upper bound. Row (Time$_o$) gives the running time on a sparc station (266Mhz). Row (Time) gives the scaled running time on Pentium 4

Algorithm	Result	A	B	C	D	MT
FLATTEN	Qual.	17.91	16.04	25.57	24.83	-
	Time$_o$	45	125	190	758	-
	Time	5.0	13.9	21.1	84.0	-
IFLAT$_1$	Qual.	8.99	8.29	14.61	12.22	-
	Time$_o$	60	161	286	1199	-
	Time	6.7	17.8	31.7	132.9	-
IFLAT$_5$	Qual.	7.76	7.10	13.03	11.92	-
	Time$_o$	124	329	657	1875	-
	Time	13.7	36.5	72.8	207.8	-

Table 1: The Quality and Performance of IFLAT

machines at 2.4Ghz. The scaling uses the clock frequency which favors slower machines, since machine speed usually scale sub-linearly with the clock speed. Observe that IFLAT$_5$ produces solutions which are from 7% to 13% off the best known upper bound on this set of benchmarks.

Experimental Setting for IFlatIRelax In the following, we report results for algorithm IFlatIRelax. In these results, the probability *relaxProbability* of relaxing a precedence constraint is set to 20%. Average results are always computed over 100 runs (unless specified otherwise). Other parameters will vary to demonstrate their impact on the results. Algorithm IFlatIRelax was implemented in COMET, which is constraint-based language for local search (Michel & Van Hentenryck 2002; Van Hentenryck & Michel 2003).

New Upper Bounds Table 2 shows the lower and upper bound reported in (Nuijten & Aarts 1996) and used in the evaluation in (Cesta, Oddi, & Smith 2000). It also reports the new best upper bounds found by IFlatIRelax during the course of this research. As shown later, IFlatIRelax found these these new upper bounds systematically. The table shows that IFlatIRelax improves 21 upper bounds, 19 of which (in bold face) being better than the upper bounds that can be derived from job-shop scheduling. Some of the new upper bounds are significant improvements over existing results, as indicated by the last benchmarks on Set B. Note also that IFlatIRelax finds the best known upper bounds on the other instances of set B as well. On the Fisher and Thompson instances, the results show that IFlatIRelax also produces significant improvements on `MT10D` and `MT10T`. This is interesting as the `RCS` routine of (Nuijten & Aarts 1996) produced a best upper bound at 963 and an average upper bound (over 5 runs) at 979.6. IFlatIRelax also improves the best upper bounds on the MT-20 benchmarks, since the best upper bound found by `RCS` was 1319 (average upper bound over 5 runs was 1339.4). However, IFlatIRelax does not reach the upper bounds that can be derived from jobshop scheduling on this problem. *Overall, these results show that IFlatIRelax is a very high-quality heuristic which produces significant improvements over existing results, including on large-scale benchmarks.*

set A				set B			
P	lb	ub	ub_n	P.	lb	ub	ub_n
la4d	572	590	**577**	a1d	888	935	**929**
la4t	570	590	**584**	a1t	717	935	**927**
set MT				a2d	750	765	**756**
P	lb	ub	ub_n	a2t	646	765	**761**
mt10d	835	930	**913**	a3d	783	844	**818**
mt10t	655	930	**912**	a3t	663	844	**813**
mt20d	1165	1165	*1186*	a4d	730	840	**803**
mt20t	387	1165	*1205*	a4t	617	840	**801**
				a5d	829	902	**864**
				a5t	756	902	**863**
set C				set D			
P	lb	ub	ub_n	P.	lb	ub	ub_n
a24d	704	935	**932**	a38d	943	1196	**1185**
a24t	704	935	**929**	a38t	943	1196	**1195**
a25t	723	977	**965**				

Table 2: Improved Upper Bounds

Stable	set A		set B		set MT	
1000	0.17	1.63	-0.92	1.04	0.94	4.76
3000	0	1.17	-1.09	0.59	0.37	3.65
5000	-0.01	1.07	-1.17	0.47	0.37	3.41
10000	-0.05	0.90	-1.61	0.24	0.91	3.12

Table 3: Impact of the Number of Iterations on IFlatIRelax

Impact of the Number of Stable Iterations Table 3 reports the best and average deviation (in percentage) from UB for $nbRelaxations = 4$ and *maxStableIt* varying from 1000 to 10000. The results demontrate the value of multiple relaxations. Even with 1000 stable iterations, IFlatIRelax is now within 1.63 and 1.04% of the best upper bounds in average (recall that IFLAT_5 is between 7.76 and 7.10) and sometimes improves the upper bounds. Additional iterations improve the quality steadily at the cost of increased CPU time. *These results show that IFlatIRelax is an extremely competitive algorithm on these benchmarks as it produces very high-quality solutions in very reasonable time.*

Impact of the Number of Relaxations Table 4 reports the best and average deviations in percentage from UB for 5000 stable iterations and *nbRelaxations* varying between 1 and 6. It is very interesting to observe the significant impact of moving from 1 to 2 relaxations, since the average deviations drop from above 7% to below 2% for sets A and B and from 13% to 6% on set MT. Additional relaxations further improve the quality of the solutions but the improvements tend to decrease. *Once again, these results demonstrate the sustantial impact of multiple relaxations.*

Relax.	set A		set B		set MT	
1	2.20	7.86	2.70	7.47	7.15	13.03
2	0.21	2.00	-0.33	1.86	2.01	6.07
4	-0.01	1.07	-1.17	0.47	0.37	3.41
6	-0.13	0.78	-1.23	-0.04	0.84	2.88

Table 4: Impact of the Number of Relaxations on IFlatIRelax

Parms.	set A	set B	set MT
$4/20/1000$	15.02	36.28	46.06
$4/20/3000$	31.79	79.36	79.58
$4/20/5000$	48.55	103.96	127.84
$4/20/10000$	92.53	137.56	176.64
$1/20/5000$	7.46	24.94	40.7
$2/20/5000$	20.65	44.32	80.44
$4/20/5000$	48.55	103.96	127.84
$6/20/5000$	91.04	148.98	207.41

Table 5: Computation Results of IFlatIRelax

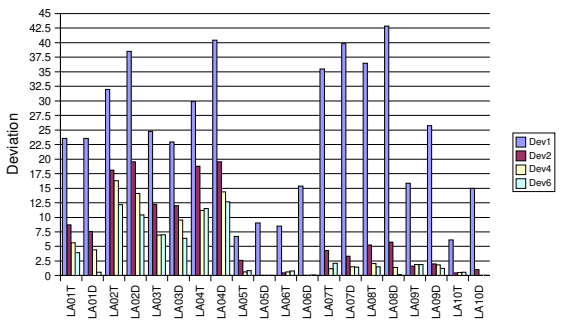

Figure 11: Visualizing the Effect of Multiple Relaxations (set A)

Performance of IFlatIRelax Table 5 shows the average running time when varying the number of stable iterations and relaxations. The average runtime always increase linearly with the number of relaxations and with the number of stable iterations. The average running time for the baseline $4/20/1000$ is comparable to IFLAT_5, while delivering a substantial improvement in quality. *These results indicate that IFlatIRelax improves the quality of iterative flattening substantially without affecting its performance.* Of course, further improvements in quality can be obtained if additional time is available.

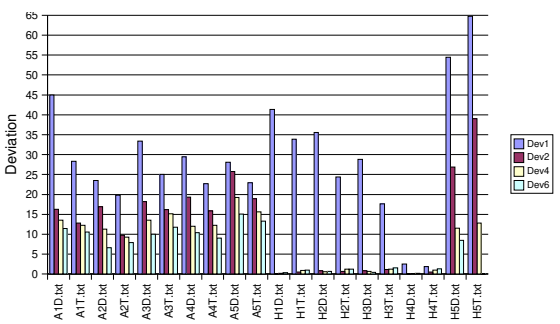

Figure 12: Visualizing the Effect of Multiple Relaxations (set B)

set C				set D			
Relax.	min	avg	T	Rnds.	min	avg	$time$
1(20)	7.7	12.8	93.9	1(20)	5	9.3	519
2(20)	3.2	6	164.9	2(10)	1.9	3.3	626
4(20)	1.7	4.2	221.3	4(10)	1.4	2.4	645
6(10)	1.8	3.5	301.2	6(20)	0.8	2.0	665

Table 6: Quality & Performance of IFlatIRelax on sets C and D

Robustness Figures 11 and 12 report some experimental results on the robustness of the algorithm. They describe the standard deviation of the solutions produced by IFlatIRelax for sets A and B. The results show that the standard deviation significantly decreases from 1 to 2 relaxations. Further relaxations keep decreasing the deviations, but at a slower pace. The histogram for set B is particularly interesting. It shows that IFlatIRelax has a dramatic effect on the robustness of H instances, essentially matching the best upper bounds systematically.

Results on Large-Scale Benchmarks Table 6 reports quality and performance results on the larger benchmarks from sets C and D. For time reasons, not all benchmarks could be executed 100 times, so the table includes the actual number of runs performed. The first and fifth column give the number of relaxations for $relaxProbability = 20\%$ and $maxStable = 5000$. The value in parenthesis gives the number of runs for each benchmark in the class. The min and avg columns report the deviation in percentage from the best known upper bound. The T column gives the running time in seconds.

Once again, the improvement in quality for the best (resp. average) solution going from 7.7% (resp. 12.8%) to 1.7% (resp. 4.2) for set C and from 5% (resp. 9.3%) to 1.4% (resp. 2.4%) when the number of relaxations goes from 1 to 4. Algorithm IFlatIRelax found 6 new upper bounds on these instances, i.e., it improved the upper bound that are derived from the jobshop scheduling algorithms. Note that these solutions have not been found by systematic algorithms for the cumulative problem, since no results being reported on these.

Conclusion

This paper studied the use of iterative flattening for cumulative scheduling and showed that iterative flattening may fail to identify bottlenecks which explain under-utilizations of the resources in early parts of the schedules. It proposed a simple extension of iterative flattening that iterates the relaxation steps to identify the bottlenecks in a sequential fashion. The resulting algorithm IFlatIRelax significantly improves the quality of the original algorithm, while retaining its computational efficiency. In particular, IFlatIRelax was instrumental in finding numerous new best upper bounds and it quickly produces solutions that are within 1% in the average of the best available upper bounds for benchmarks involving up to 900 activities. As a consequence, algorithm IFlatIRelax seems to be a very appealling approach for large-scale problems and problems where high-quality solutions must be found in very reasonable time.

There are many directions for future work. First, it would be interesting to design algorithms and visual tools to identify and explain bottlenecks in intuitive terms. In general, it is very hard to understand why resources are under-utilized in parts of the schedule and which decisions (if any) contributed to these inefficiencies. A major part of this research was devoted to understanding and analyzing the strenghts and weakenesses of iterative flattening. Automating such analyses may help in improving the scheduling algorithms further. Second, it would be interesting to generalize and apply the algorithm to more complex cumulative scheduling problems involving arbitrary distance constraints. Finally, it would be valuable to investigate the combination of IFlatIRelax with decomposition techniques on very large-scale problems.

References

Cesta, A.; Oddi, A.; and Smith, S. F. 1999a. Greedy algorithms for the multi-capacitated metric scheduling problem. In *ECP*, 213–225.

Cesta, A.; Oddi, A.; and Smith, S. F. 1999b. An iterative sampling procedure for resource constrained project scheduling with time windows. In *IJCAI*, 1022–1033.

Cesta, A.; Oddi, A.; and Smith, S. F. 2000. Iterative flattening: A scalable method for solving multi-capacity scheduling problems. In *AAAI/IAAI*, 742–747.

Laborie, P., and Ghallab, M. 1995. Planning with sharable resource constraints. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, 1643–1649.

Michel, L., and Van Hentenryck, P. 2002. A constraint-based architecture for local search. In *Conference on Object-Oriented Programming Systems, Languages, and Applications.*, 101–110. Seattle, WA, USA: ACM.

Michel, L., and Van Hentenryck, P. 2003. Maintaining longest path incrementally. In *Ninth International Conference on Principles and Practice of Constraint Programming*.

Nuijten, W. P. M., and 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.

Nuijten, W. 1994. *Time and Resource Constrained Scheduling: A Constraint Satisfaction Approach*. Ph.D. Dissertation, Eindhoven University of Technology.

Oddi, A., and Smith, S. F. 1997. Stochastic procedures for generating feasible schedules. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97)*, 308–314. Providence, Rhode Island: AAAI Press / MIT Press.

Van Hentenryck, P., and Michel, L. 2003. Control abstractions for local search. In *Ninth International Conference on Principles and Practice of Constraint Programming*.