Complete MCS-Based Search: Application to Resource Constrained Project Scheduling

Content areas: constraint programming, scheduling, search

Abstract

This paper describes a simple complete search based on the detection and resolution of *minimal critical sets* (MCS) for cumulative scheduling. The heuristics for selecting MCSs relies on an estimation of the related reduction of the search space. An extension of the search procedure using *self-adapting shaving* is proposed. The approach was implemented on top of classical constraint propagation algorithms and tested on resource constrained project scheduling problems (RCPSP). We were able to close 15% of the previously open problems of the PSPLIB [Kolisch and Sprecher, 1996] and improve more than 30% of the best known lower bounds on those heavily studied problems. Other new results on open-shop and cumulative job-shop scheduling are reported.

1 Introduction

The resource constrained project scheduling problem (RCPSP) is one of the most general scheduling problem that is extensively studied in the literature. It consists in scheduling a project, which is a set of activities linked with precedence constraints, by means of a set of limited resources while minimizing the total duration of the project. The decision variant of the RCPSP, i.e., the problem of determining whether there exists a feasible project of makespan smaller than a given deadline, is NP-hard in the strong sense. The RCPSP is a very popular and frequently studied NP-hard optimization problem and the last 20 years have witnessed a tremendous improvement of both heuristic and exact solution procedures (cf. e.g. the recent surveys given in [Demeulemeester and Herroelen, 2002; Hartmann and Kolisch, 2004]). The currently best lower bounds on the makespan for the general RCPSP are based on solving linear programs using adequate cutting planes [Brucker and Knust, 2000; Baptiste and Demassey, 2004]. State-of-the-art techniques for upper-bounds rely on meta-heuristics such as Genetic Algorithms, Ant Colony Optimization or Large Neighborhood Search. Many scheduling problems such as job shop, cumulative job shop and open-shop can be seen as special cases of RCPSP.

In this article, we present a pure constraint programming approach based on the exploration of a complete search tree to prove that the project cannot be achieved within a given deadline or to exhibit a feasible project if one exists. The search procedure is based on the detection and resolution of *minimal critical sets* (MCS) [Laborie and Ghallab, 1995] at each node of the search. MCSs are carefully chosen using a heuristics that tries to minimize the size of the search space. During the search, strong constraint propagation is enforced using classical scheduling constraint propagation techniques such as *time-tabling*, *edge-finding*, *precedence energy* and *balance* constraints [Laborie, 2003].

Next section recap the definition of the resource constrained project scheduling problem and introduces some notations. Section 3 describes our basic search procedure as well as the heuristics to select MCSs. Section 4 extends the basic search procedure to perform *self-adapting shaving*. The last part of the paper consists of experimental results on classical benchmarks (general RCPSP, open-shop and cumulative jobshop problems). For general RCPSP, we show that our approach closes 15% of previously open instances and improves more than 30% of best known lower bounds of the well known PSPLIB instances [Kolisch and Sprecher, 1996]. The same approach was used to close all the hard open-shop instances of [Guéret and Prins, 1999] in less than 5s CPU time and to improve the best known lower bounds and close several instances of cumulative jobshop.

2 Model and notations

The resource constrained project scheduling problem (RCPSP) can be formally stated as follows. A project is made of a set of activities $\mathcal{A}$ linked by precedence constraints. Precedence constraints can be represented by a directed acyclic graph $G = (\mathcal{A}, \mathcal{E})$ where each node in $\mathcal{A}$ represents an activity and each arc $(A, B) \in \mathcal{E}$ represents a precedence constraint between A and B. Let $d(A)$ denote the fixed duration of activity $A \in \mathcal{A}$ and $s(A)$ (resp. $e(A)$) denote the decision variable representing the start (resp. end) time of activity A. A set of discrete capacity resources $\mathcal{R}$ is considered, each resource $R \in \mathcal{R}$ having a maximal available capacity $Q(R)$ over the entire scheduling horizon. Each activity $A \in \mathcal{A}$ requires a non-negative quantity $q(A, R)$ of resource R. The problem is to find a feasible instantiation s of the activity start times such that precedence and resource

constraints are satisfied and the schedule makespan is minimal. More formally:

$$\begin{aligned} \text{minimize} \quad & \max_{A \in \mathcal{A}} e(A) \\ \text{subject to :} \quad & \\ \forall A \in \mathcal{A}, \quad & 0 \leq s(A) \\ & e(A) = s(A) + d(A) \\ \forall (A,B) \in \mathcal{E}, \quad & e(A) \leq s(B) \\ \forall R \in \mathcal{R}, \forall t \in \mathbb{Z}^+, \quad & \sum_{A \in \mathcal{S}(t)} q(A,R) \leq Q(R) \end{aligned}$$

where $\mathcal{S}(t)$ is the set of activities executing at time t:

$$\mathcal{S}(t) = \{A \in \mathcal{A}, s(A) \leq t < e(A)\}$$

A resource requirement of activity A on resource R is a 3-uple $u = (A, R, q)$ where $q = q(A, R) > 0$. If $u = (A, R, q)$ is a resource requirement, we will denote $A(u) = A$ the activity of u, $R(u) = R$ the required resource, $q(u) = q$ the required quantity, $s(u)$ (resp. $e(u)$) will denote the start (resp. end) time of the activity of u. We will also denote $U(R) = \{u/R(u) = R\}$ the set of resource requirements on resource R. If $\varphi \subseteq U(R)$ is a subset of resource requirements on a resource R, we will denote $q(\varphi) = \sum_{u \in \varphi} q(u)$ the global resource consumption of R by activities in φ.

3 Search

3.1 Branching scheme

Our branching scheme assumes that a temporal network representing the relations between the time-points of all activities (start and end) using the point algebra of [Vilain and Kautz, 1986] is maintained during the search. In our implementation, this is ensured by the precedence graph constraint of ILOG Scheduler [Laborie, 2003]. We denote $\{\emptyset, \prec, \preceq, =, \succ, \succeq, \neq, ?\}$ the set of qualitative relations between time points. If u and v represent two resource requirements, we will denote $u \preceq v$ if and only if $e(u) \preceq s(v)$.

The branching scheme relies on the notion of minimal critical sets (MCS) and their resolvers as introduced in [Laborie and Ghallab, 1995]. A MCS is a minimal set of resource requirements that could be executed simultaneously and would, in this case, over-consume a given resource. MCSs are a natural generalization to cumulative scheduling of the pairs of activities conflicting for the same unary resource in disjunctive scheduling.

Definition 1 (Minimal critical set) *A minimal critical set on a resource R is a subset $\phi \subseteq U(R)$, such that:*

1. $Q(R) < q(\phi)$
2. $\forall \varphi \subsetneq \phi, q(\varphi) \leq Q(R)$
3. $\bigwedge_{(u,v) \in \phi \times \phi} s(u) \prec e(v)$ *is consistent with the current temporal network*

Informally, the different ways to resolve a minimal critical set consist in posting a precedence constraint between any two of its activities.

Definition 2 (Resolvers of a minimal critical set) *If $\phi \subseteq U(R)$ is a MCS, we call resolvers of ϕ the disjunctive set of temporal constraints $Res(\phi) = \{u \preceq v\}_{(u,v) \in \phi^2, u \neq v}$.*

As described in [Laborie and Ghallab, 1995], the set of resolvers $Res(\phi)$ of a MCS ϕ can be simplified so as to remove those resolvers $\rho \in Res(\phi)$ that are so that there exists another resolver $\rho' \in Res(\phi)$ such that $\rho \Rightarrow \rho'$ given the current temporal network. Indeed, in such case, the resolver ρ is redundant. Such a simplification procedure can be achieved in $O(k^3)$ if k is the size of the MCS using the naive Algorithm 1. In what follows, we assume that the set of resolvers of a MCS has been simplified.

Algorithm 1 Resolver simplification algorithm

```
1:  procedure SIMPLIFY_RESOLVERS(φ)
2:      Res(φ) ← ∅
3:      for all u in φ do
4:          for all v in φ such that u ≠ v do
5:              keep_uv ← TRUE
6:              for all w in φ do
7:                  if s(v) ⪯ s(w) or e(w) ⪯ e(u) then
8:                      keep_uv ← FALSE
9:                      break
10:             if keep_uv then
11:                 Res(φ) ← Res(φ) ∪ (u ⪯ v)
12:     return Res(φ)
```

At each search node, our branching scheme consists in selecting a MCS ϕ and branching on its possible resolvers in the children nodes until there is no more MCS. This approach is clearly complete.

3.2 Heuristics

As all the resolvers consist of temporal constraints of the form $x \preceq y$ where x and y are two time-points (start or end of an activity), we are interested in an estimation of the size of the search space after posting such a precedence constraint. The ratio of the search space that is preserved when adding a precedence constraint is estimated using the complementary of the commitment measure introduced in [Laborie, 2003].

Let x and y be two time-points with respective lower and upper bound for time value: x_{min}, x_{max} and y_{min}, y_{max}. The size of the search space is estimated by the Cartesian product of the domain of the two variables, that is, the area of the rectangle $x_{min}, x_{max}, y_{min}, y_{max}$. The size of the search space that is preserved when adding the constraint $x \preceq y$ is the part of that rectangle above the line $x = y$ as illustrated in Fig 1. The ratio of the search space that is preserved can thus be estimated as follows. Let:

$$A = (y_{max} - y_{min} + 1)(x_{max} - x_{min} + 1)$$

$$B = (y_{max} - x_{min} + 1)^2$$

$$C_{min} = max(0, y_{min} - x_{min})^2$$

$$C_{max} = max(0, y_{max} - x_{max})^2$$

The ratio is then equal to:

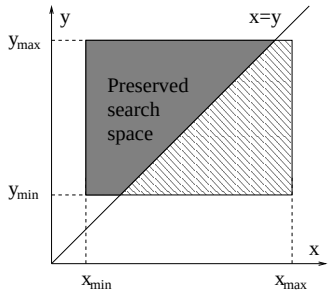


Figure 1: Preserved search space when adding $x \leq y$

$$preserved(x \preceq y) = \frac{B - C_{min} - C_{max}}{2A}$$

If Ω is the size of the search space below the current search node, the size of the search space after posting a temporal constraint $x \preceq y$ can be estimated by $\Omega.preserved(x \preceq y)$. If ϕ is the MCS that is selected to be resolved at the current search node, the size of the search space below the current node can be estimated as the sum of the sizes of the search space below each child node, that is: $\Omega.\sum_{\rho \in Res(\phi)} preserved(\rho)$. Thus, $preserved(\phi) = \sum_{\rho \in Res(\phi)} preserved(\rho)$ estimates the search space ratio that is preserved when choosing ϕ as the next MCS to solve[1].

Our heuristics simply chooses to resolve next the MCS ϕ that minimizes $preserved(\phi)$ as described in next section.

3.3 MCS selection algorithm

At each search node, the MCS selection procedure develops a tree of partial MCSs where the current MCS ϕ is extended adding one resource requirement in each child node. By definition of $preserved(\phi)$, it is clear that $\phi' \subset \phi \Rightarrow preserved(\phi') \leq preserved(\phi)$. Thus, if ϕ^* is the best MCS found so far, once a partial MCS ϕ has been reached such that $preserved(\phi^*) \leq preserved(\phi)$, the sub-tree of the MCS selection tree rooted at ϕ can be abandoned.

The algorithm for selecting and branching on a MCS is more precisely described in Algorithm 2 using the following notations: $id(u)$ is a unique index associated with resource requirement u used to break ties. $Unranked(u)$ represents all the resource requirements v that can possibly overlap u given the current temporal constraints, and that are such that $q(v) < q(u)$ or $q(v) = q(u)$ and $id(v) < id(u)$. ϕ is the (partial) MCS that is currently being extended. A (partial) MCS is represented by a list of resource requirements: $\phi = (u_1, ..., u_k)$. We denote $u_k = Last(\phi)$ and define the operator $\oplus$ as follows: $\phi \oplus u = (u_1, ..., u_k, u)$. $q = q(\phi)$ is the consumption of the current (partial) MCS; $p = preserved(\phi)$ is the preserved search space so far in the current (partial) MCS; ϕ^* is the best MCS so far and $p^* = preserved(\phi^*)$ is the preserved search space of the best MCS so far. The procedure at line 1 calls the MCS rating and selection process on each resource. At line 6, to rate and select MCSs for a given resource, the procedure first sorts the relevant sets of requirements v at lines 7 and 9 by decreasing order of $q(v)$, using $id(v)$ to break ties in order to ensure that each MCS is scanned only once, starting with the smallest MCSs, that is, the ones containing the most greedy requirements. The procedure at line 23 returns TRUE if and only if a given requirement u can possibly overlap all the requirements of a partial MCS given the current temporal network. The procedure at line 28 computes the incremental increase of preserved space due to the insertion of a new requirement u in the current MCS ϕ. The value of $preserved$ has been described in section 3.2. The main recursive function for selecting MCSs is described at line 12.

Algorithm 2 MCS selection algorithm

```
 1: procedure SELECT_MCS
 2:     p* ← +∞
 3:     for R in ℛ do
 4:         SELECT_MCS(R)
 5:     return φ*

 6: procedure SELECT_MCS(R)
 7:     Sort U(R) by decreasing q
 8:     for all u in U(R) do
 9:         Sort Unranked(u) by decreasing q
10:     for u in U(R) do
11:         RSELECT_MCS(R,(u),q(u),0)

12: procedure RSELECT_MCS(R,φ,q,p)
13:     if q > Q(R) then                          ▷ φ is a MCS
14:         if p < p* then                ▷ φ is the best MCS so far
15:             p* ← p
16:             φ* ← φ
17:     else                                  ▷ φ needs to be extended
18:         u ← Last(φ)
19:         for all v in Unranked(u) do
20:             if IS_UNRANKED(v,φ) then
21:                 dp ← DELTA_PRESERVED(v,φ)
22:                 RSELECT_MCS(R,φ⊕v,q+q(v),p+dp)

23: procedure IS_UNRANKED(u,φ)
24:     for all v in φ do
25:         if u ≼ v or v ≼ u then
26:             return FALSE
27:     return TRUE

28: procedure DELTA_PRESERVED(u,φ)
29:     dp ← 0
30:     for all v in φ do
31:         dp ← dp + preserved(u,v) + preserved(v,u)
32:     return dp
```

The best MCS ϕ^* that has been scanned by the above procedure is selected as the one to be solved at the current search node. This MCS is simplified using Algorithm 1 and the search explores all of its resolvers $\rho \in Res(\phi^*)$ in the child nodes by decreasing order of $preserved(\rho)$. This order has

[1]Note that this is of course only a rough estimate and in particular, the estimated ratio $preserved(\phi)$ can be greater than 1.

no effect when the schedule is not feasible as in this case the complete search tree needs to be explored but it helps finding a solution quicker when a solution exists.

4 Self-adapting shaving

Shaving techniques [Torres and Lopez, 2000] provide a good framework for strengthening constraint propagation and avoiding late failures to be discovered in the search tree. They are all based on the following principle: if adding a constraint C in the current node of the search leads to a failure of the propagation, then, constraint $\neg C$ can be inferred. Due to the cost of propagating a constraint C and the potential number of constraints C to try to shave on, shaving techniques are in general computationally expensive.

To improve the pruning of the search tree, we implemented the following shaving technique based on MCSs. If a MCS ϕ with resolvers $Res(\phi) = \{\rho_1, ..., \rho_k, \rho_{k+1}\}$ is such that $\forall i \in [1..k]$, adding ρ_i in the current schedule leads to a failure of the propagation, then ρ_{k+1} can be inferred. The complexity for shaving a given MCS ϕ is thus in $O(n^2P)$ where n is the size of the MCS and P is the cost of full constraint propagation at the current node. Potentially, there is of course an exponential number of MCSs to shave on at each search node and we can expect that many of those MCS do not allow inferring any precedence constraint. An idea to speed-up the shaving process is thus to only try shaving on a subset of MCSs for which the probability to infer a precedence constraint is greater than a given threshold α. Parameter α is an input of the shaving algorithm. We can roughly estimate that the probability that adding a precedence constraint $x \preceq y$ in the current schedule will lead to a failure of the propagation is proportional[2] to $1 - preserved(x \preceq y)$. If $\rho_m = argmax_{\rho \in Res(\phi)} preserved(\rho)$ is the resolver of the MCS ϕ with maximal preserved search space, we are interested in the MCSs that get a high probability that all their resolvers but ρ_m will fail, that is, if we assume all the probabilities are independent, the ones such that $\Pi_{\rho \in Res(\phi) \setminus \{\rho_m\}}(1 - preserved(\rho))$ is greater than a given threshold. For those MCSs, if the threshold is close enough to 1, we can assume that $preserved(\rho)$ is small enough so that the first order approximation $\Pi_{\rho \in Res(\phi) \setminus \{\rho_m\}}(1 - preserved(\rho)) \approx 1 - \sum_{\rho \in Res(\phi) \setminus \{\rho_m\}} preserved(\rho)$ is reasonable.

To summarize, we thus only consider for shaving those MCS scanned by the procedure described in algorithm 2 that are such that $preserved(\phi) - preserved(\rho_m) \leq \beta$, β being a threshold. The computation of this criterion only adds a negligible overhead related with the maintenance of ρ_m for each MCS in the MCS selection procedure. Due to the numerous approximations, β is not taken to be constant (theoretically equal to $1 - \alpha$). The threshold β is computed by self-adaptation in such a way that in average, among the last H shaving attempts, $\alpha.H$ lead to the inference of a new precedence. Whenever the number of successful shaving s among the last H ones deviates from $\alpha.H$, the parameter β is adapted accordingly: if $s < \alpha.H$, β is decreased by $\epsilon.\beta$ and if $s > \alpha.H$, β is increased by $\epsilon.\beta$. For all our experiments with shaving, we took $H = 20$, $\alpha = 0.75$, $\epsilon = 0.01$ and start with $\beta = 1$.

[2]Note that this estimation is exact at the extreme points when $preserved(x \preceq y) = 0$ (propagation will fail for sure) and when $preserved(x \preceq y) = 1$ (propagation cannot fail because $x \preceq y$ has already been discovered given the current domains of x and y).

5 Experimental evaluation

The approach has been implemented on top of ILOG Scheduler 6.0 using the *timetable*, *disjunctive*, *edge-finder*, *precedence energy* and *balance* constraints. All the experiments described in this section were run on a Dell Latitude D600 laptop, 1.4 GHz[3].

5.1 Results on general RCPSP

We evaluated our approach on the instances of the PSPLIB[Kolisch and Sprecher, 1996] with 60, 90 and 120 activities (resp. sets J60, J90, J120). For each instance, we solve the feasibility problem of finding a schedule with a makespan lower than T, starting with a valid lower bound for T[4] and incrementing T until the problem is shown to be feasible (in this case, T is the optimal makespan) or a until a given time limit for solving the problem with makespan T is exceeded (in this case, T is a valid lower-bound).

In a first series of experiments, we use the basic search described in section 3 without shaving with a time limit of $300s$.

The previous best lower and upper bounds we compare with are the ones reported in the PSPLIB together with the recent improvements on the J60 instances reported in [Baptiste and Demassey, 2004]. The results are summarized on Table 1 with the following columns:

- **#O** : number of instances previously open
- **#I** : number of improved lower bounds (% of #O)
- **AGR**: average gap (distance from the lower to the upper bound) reduction when a bound is improved
- **#C** : number of closed instances (% of #O)

Out of the 620 previously open instances, we are able to improve 178 lower-bounds with an average gap reduction of 10.8% and to close 85 instances.

To show the effect of self-adapting shaving, we run a version of our approach using self-adapting shaving with the same time-limit of 300s. The results are summarized on table 2. Out of the 620 previously open instances, we are able

[3]In the final version of the paper we will refer to a public web page containing the detailed results as well as all the optimal solutions found.

[4]For instance the lower-bound of the PERT of temporal constraints

Inst.	#O	#I	(%I)	AGR	#C	(%C)
J60	99	39	(39.8%)	17.0%	21	(21.2%)
J90	129	51	(39.5%)	18.1%	26	(20.2%)
J120	392	88	(22.4%)	7.6%	38	(9.7%)
ALL	620	178	(28.7%)	10.8%	85	(13.7%)

Table 1: Results on RCPSP without self-adapting shaving with a time-limit of 300s

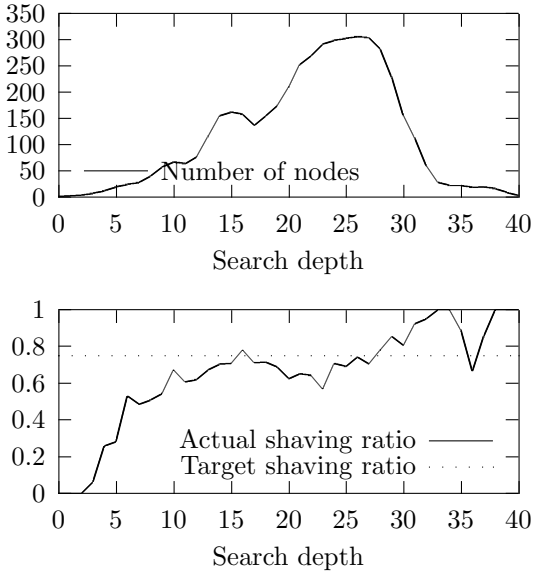
Inst.	#O	#I	(%I)	AGR	#C	(%C)
J60	99	40	(40.4%)	17.0%	22	(22.2%)
J90	129	52	(40.3%)	18.3%	26	(20.2%)
J120	392	90	(23.0%)	7.7%	38	(9.7%)
ALL	620	182	(29.4%)	10.9%	86	(13.9%)

Table 2: Results on RCPSP with a self-adapting shaving and a time-limit of 300s

Inst.	#O	#I	(%I)	AGR	#C	(%C)
J60	99	44	(44.4%)	21.1%	25	(25.3%)
J90	129	53	(41.1%)	20.4%	28	(21.7%)
J120	392	95	(24.2%)	8.3%	40	(10.2%)
ALL	620	192	(31.0%)	12.3%	93	(15.0%)

Table 3: Results on RCPSP with a self-adapting shaving and a time-limit of 1800s

to improve 182 lower-bounds with an average gap reduction of 10.9% and to close 86 instances. The main conclusion is that, within the same time-limit, the addition of self-adapting shaving slightly increases the performances. The two curves below respectively show, on a particular instance (`J60_5_2`), the number of search nodes with a given depth in the search tree and, for each node depth, the ratio between the number of selected MCSs that effectively lead to the inference of a new precedence and the total number of MCSs selected for shaving. One clearly see that in the region of the search space where most of the nodes are concentrated (between depths 10 and 35), this ratio is effectively close to the target ratio of 0.75. In this instance, 4667 nodes where explored, 4843 MCSs where selected for shaving and among them, 3291 effectively lead to the inference of a new precedence.



For a time-limit of 1800s using self-adapting shaving, the results are summarized on table 3. Out of the 620 previously open instances, we improve 192 lower-bounds (that is 31% of the previously open instances) with an average gap reduction of 12.3% and close 93 instances (that is 15% of the previously open instances).

Instance	UB	Optim.	Instance	UB	Optim.
gp06-03	1255	**1255***	gp09-02	1112	**1110***
gp06-07	1290	**1290***	gp09-03	1117	**1115***
gp06-09	1243	**1243***	gp09-06	1093	**1093***
gp06-10	1254	**1254***	gp09-07	1097	**1090***
gp07-01	1159	**1159***	gp09-08	1106	**1105***
gp07-02	1185	**1185***	gp09-09	1126	**1123***
gp07-04	1167	**1167***	gp09-10	1120	**1110***
gp07-06	1193	**1193***	gp10-01	1099	**1093***
gp07-08	1180	**1180***	gp10-02	1099	**1097***
gp07-09	1220	**1220***	gp10-03	1081	**1081***
gp08-02	1135	**1135***	gp10-04	1089	**1077***
gp08-04	1154	**1153***	gp10-05	1080	**1071***
gp08-06	1116	**1115***	gp10-06	1072	**1071***
gp08-07	1126	**1126***	gp10-07	1081	**1079***
gp08-08	1148	**1148***	gp10-08	1098	**1093***
gp08-10	1161	**1161***	gp10-09	1120	**1112***
gp09-01	1135	**1129***	gp10-10	1092	**1092***

Table 4: Results on open-shop with a self-adapting shaving and a time-limit of 5s

5.2 Results on open-shop problems

Open-shop scheduling can be seen as a special case of RCPSP where all resources have a unit capacity and additional unary resource are used to model the fact that activities of the same job do not overlap. We tested our approach, with the same settings as in previous section on the open-shop problems instances proposed in [Guéret and Prins, 1999]. Those instances are considered to be very hard instances of open-shop problems and serve as classical benchmark in open-shop scheduling (see for instance recent work in [Blum, 2005]). The benchmark consists of 80 instances ranging from $3jobs \times 3machines$ problems until $10jobs \times 10machines$ problems. Out of these 80 problems, 34 instances are still open. Using our approach, we were able to close all those instances in less than 5s CPU time. The optimal makespan for the 34 previously open instances is summarized on table 4 where the column UB corresponds to the currently best known upper-bound for which no optimality proof did exist.

We also tried our approach on the open instances of the benchmark of [Brucker *et al.*, 1997]. We closed 2 of these 6 open instances: namely `j8-per0-2` (optimal makespan: 1052) and `j8-per10-0` (optimal makespan 1017).

5.3 Results on cumulative jobshop problems

We tested our approach, with the same settings, on the cumulative job-shop problem benchmark described in [Nuijten, 1996]. These instances are derived from classical jobshop scheduling problems by multiplying the number of jobs (and thus the number of activities) and the capacity of the resources by a given factor ($\times 2$ or $\times 3$). Our results are summarized on table 5 where LB is the lower bound using the consistency checking described in [Nuijten, 1996] and $NewLB$ the new lower bound of our approach. We were able to close the ft06$\times$2 and ft06$\times$3 instances as well as to improve 12 lower bounds out of these 38 open instances.

6 Conclusions

We presented a simple complete search procedure implemented on top of classical constraint propagation algorithms

Instance	LB	New LB	Instance	LB	New LB
ft06×2	53	**55***	ft06×3	53	**55***
ft10×2	835	**837**	ft10×3	828	828
la03×2	593	593	la03×3	590	590
la04×2	572	572	la04×3	570	570
la16×2	888	**892**	la16×3	884	**887**
la17×2	754	754	la17×3	753	753
la18×2	783	**803**	la18×3	776	**783**
la19×2	731	**756**	la19×3	724	**740**
la20×2	830	**849**	la20×3	829	**842**
la21×2	1017	1017	la21×3	1010	**1012**
la22×2	913	913	la22×3	913	913
la24×2	885	885	la24×3	884	884
la25×2	907	907	la25×3	903	903
la29×2	1117	1117	la29×3	1116	1116
la36×2	1229	1229	la36×3	1227	1227
la37×2	1378	1378	la37×3	1370	1370
la38×2	1092	1092	la38×3	1087	1087
la39×2	1221	1221	la39×3	1221	1221
la40×2	1180	1180	la40×3	1176	1176

Table 5: Results on cumulative job-shop with a self-adapting shaving and a time-limit of 1800s

and applied it to resource constrained project scheduling problems. In average, this approach outperforms the best algorithms for finding lower bounds on those scheduling problems, even with a time limit of 300s per optimization step[5]. Using this approach in conjunction with a self-adapting shaving procedure, we were able to close 15% of the previously open problems of the PSPLIB and improve more than 30% of the best known lower bounds. What is even more remarkable is that this very same approach allows closing all the hard open-shop instances of [Guéret and Prins, 1999] in less than 5s CPU time although the approach was not particularly designed to tackle disjunctive scheduling and does not exploit the open-shop nature of the problems.

The understanding of why our method works so well on the instances of the PSPLIB and on many open-shop problems would deserve a deeper study. From one hand, if the problems are highly cumulative, our approach is clearly limited by the explosion of the number of MCSs to consider. From the other hand, when problems are highly disjunctive, we could expect other approaches dedicated to disjunctive scheduling to work better. A first possible explanation could be a good fit between our approach and the "disjunctivity" degree of the hard instances of the PSPLIB as suggested by some recent work [Garaix *et al.*, 2005]. A result of this study could be some kind of hybridizing of MCS-based search with techniques more adapted to highly cumulative problems, MCS-based search being restricted to the resolution of MCSs with small preserved search space (thus small MCSs) at the top of the search tree. A second direction for future work is the generalization of the notion of *self-adapting* shaving introduced in this paper to other shaving techniques in scheduling.

[5]When this time limit is exceeded at an optimization step, usually, the previous steps where fairly quick so that the overall time for computing the lower bound is close to 300s.

References

[Baptiste and Demassey, 2004] P. Baptiste and S. Demassey. "Tight LP bounds for resource constrained project scheduling". OR Spectrum (2004) 26:251-262, 2004.

[Blum, 2005] C. Blum. "Beam-ACO - hybridizing ant colony optimization with beam search: an application to open-shop scheduling". Computers & Operations Research, 32(6):1565-1591, 2005.

[Brucker *et al.*, 1997] P. Brucker, J. Hurink, B. Jurisch and B. Wöstmann. "A branch & bound algorithm for the open-shop problem". Discrete Applied Mathematics 76:43-59, 1997.

[Brucker and Knust, 2000] P. Brucker and S. Knust. "A linear programming and constraint propagation-based lower bound for the RCPSP". European Journal of Operational Research 127:355-362, 2000.

[Demeulemeester and Herroelen, 2002] E. Demeulemeester and W. Herroelen. "Project scheduling - A research handbook". Kluwer Academic Publishers, Boston, 2002.

[Garaix *et al.*, 2005] T. Garaix, C. Artigues et S. Demassey. "Bornes basées sur les ensembles interdits pour le problème d'ordonnancement de projet à moyens limités". Proc. ROADEF 2005.

[Guéret and Prins, 1999] C. Guéret and C. Prins. "A new lower bound for the open-shop problem". Annals of Operations Research, 92:165-183, 1999.

[Hartmann and Kolisch, 2004] S. Hartmann and R. Kolisch. "Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem: An Update". Technical University of Munich. 2004.

[Kolisch and Sprecher, 1996] R. Kolisch and A. Sprecher. "PSPLIB - A project scheduling problem library". European Journal of Operational Research, 96:205-216. 1996.

[Laborie and Ghallab, 1995] P. Laborie and M. Ghallab. "Planning with Sharable Resource Constraints". Proc. IJCAI-95, 1995.

[Laborie, 2003] P. Laborie. "Algorithms for propagation resource constraints in AI planning and scheduling: Existing approaches and new results". Artificial Intelligence 143 (2003) 151-188. 2003.

[Nuijten, 1994] W. Nuijten. "Time and resource constrained scheduling: A constraint satisfaction approach". PhD Thesis. Eindhoven University of Technology, 1994.

[Nuijten, 1996] W. Nuijten. "A computational study of constraint satisfaction for multiple capacitated job shop scheduling". European Journal of Operational Research. 90(2) 269-284. 1996.

[Torres and Lopez, 2000] P. Torres and P. Lopez. "Overview and possible extensions of shaving techniques for Job-Shop problems". Proc. CP-AI-OR'2000, pp.181-186.

[Vilain and Kautz, 1986] M. Vilain and H. Kautz. "Constraint propagation algorithms for temporal reasoning". Proc. Fifth National Conference on Artificial Intelligence. p377-382,1986.