Artificial Intelligence 182–183 (2012) 58–67

Contents lists available at SciVerse ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

A min-flow algorithm for Minimal Critical Set detection in Resource Constrained Project Scheduling

Michele Lombardi, Michela Milano *

DEIS Università di Bologna, Viale Risorgimento 2, 40136 Bologna, Italy

ARTICLE INFO

Article history:
Received 19 October 2010
Received in revised form 27 October 2011
Accepted 8 December 2011
Available online 9 December 2011

Keywords:
Constraint-based scheduling
Constraint programming
Precedence Constraint Posting
Minimal Critical Set
Min-flow algorithm

ABSTRACT

We propose a min-flow algorithm for detecting Minimal Critical Sets (MCS) in Resource Constrained Project Scheduling Problems (RCPSP). The MCS detection is a fundamental step in the Precedence Constraint Posting method (PCP), one of the most successful approaches for the RCPSP. The proposed approach is considerably simpler compared to existing flow based MCS detection procedures and has better scalability compared to enumeration- and envelope-based ones, while still providing good quality Critical Sets. The method is suitable for problem variants with generalized precedence relations or uncertain/variable durations.

1. Introduction

The Resource Constrained Project Scheduling Problem (RCPSP) is one of the most important problems in project management, manufacturing and resource optimization. The RCPSP has received enormous attention in Operations Research, [3] and in Artificial Intelligence [1]; the aim is to schedule at minimal duration a set of activities subject to precedence constraints (represented in a *project graph*) and to limited resource availability.

One of the most successful techniques for solving the RCPSP is Precedence Constraint Posting (PCP – see [18]) where the set of initial precedence constraints is augmented during the solution process to avoid resource over-usage, detected in the form of Minimal Critical Sets (MCS). A MCS is a minimal set of potentially overlapping activities collectively overusing a resource; due to minimality, the removal of a single activity from the set wipes out the conflict. To obtain a feasible solution, all MCSs should be removed. However, their number is exponential in the size of the project graph. Therefore, their efficient discovery is a fundamental ingredient for any RCPSP solver.

Building on our results in [16], we propose to cast the detection of a (non-minimal) Critical Set to a min-flow problem on a resource graph where activities are annotated with their resource requirements; an MCS is then extracted by removing activities in a greedy fashion. The flow optimization at the core of our method is considerably simpler than those found in the literature [12,17]. As a second contribution, we carried on extensive experimentation on MCS detection algorithms, employed within a tree-search scheme to solve the PSPlib benchmarks [13,14]; the min-flow approach exhibits better scalability compared to existing enumeration- and envelope-based MCS detection procedures [15,19], featuring a significant speed-up as the number of MCSs grows.

* Corresponding author.
E-mail address: michela.milano@unibo.it (M. Milano).

0004-3702/$ – see front matter
doi:10.1016/j.artint.2011.12.001

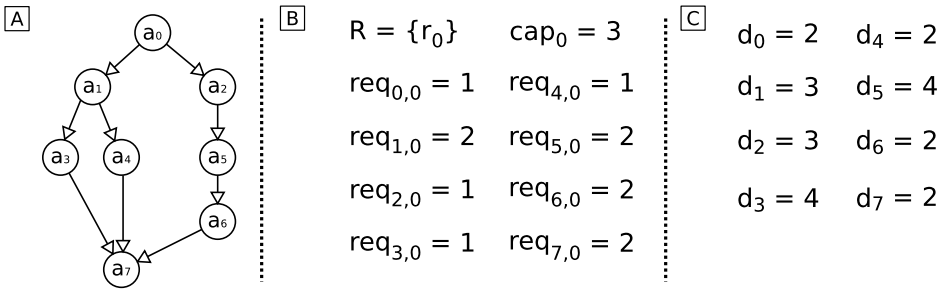

Fig. 1. A RCPSP instance: (A) precedence constraints; (B) resource requirements; (C) durations.

The PCP technique is particularly well-suited to tackle RCPSP variants with variable/uncertain durations (e.g. the Stochastic or the Multi-mode RCPSP), where fixed start times cannot be assigned, while precedence constraints can still be used to avoid resource conflicts; as a consequence, here we restrict to MCS detection algorithms suitable to deal with such a case.

2. Resource Constrained Project Scheduling

The classical Resource Constrained Project Scheduling Problem (RCPSP) is defined on a directed acyclic graph $\langle A, E\rangle$ (referred to as a project graph), where A is a set of n activities a_i having fixed duration d_i, and E is a set of directed edges (a_i, a_j), defining precedence relations over activities. Without loss of generality, we assume there is a single *source* activity (a_0) with no ingoing arcs and a single *sink* activity (a_{n-1}) with no outgoing arcs. Each activity a_i requires a certain amount $req_{i,k}$ of one or more renewable resources r_k from a set R (source and sink nodes require no resource); each resource r_k has finite capacity cap_k. The problem consists of finding a *schedule* (that is, an assignment of start times to activities), such that no resource capacity is exceeded and the overall completion time (*makespan*) is minimized. Fig. 1 shows an example of a small RCPSP instance. As a common extension, release times and deadlines can be specified on the execution of each activity.

In this work we are interested in some variants of the RCPSP [3]: in the *Multi-mode RCPSP* each activity can be executed in one of a set of possible *modes*, corresponding to a different resource requirement/duration combination; as a consequence, durations are *variable* until the mode is fixed. A second RCPSP variant considers durations that are *uncertain*, and vary at execution time according to a probability distribution; the best known example is the *Stochastic RCPSP*, where the objective is to minimize the expected makespan.

3. PCP: background and related work

The Precedence Constraint Posting approach is one of the most successful techniques for the RCPSP; in PCP resource conflicts are resolved by adding precedence constraints between the involved activities. The resulting augmented graph defines a set of possible contention-free schedules, rather than a specific schedule; this makes the approach particularly well suited for RCPSP variants where start times can be assigned only late during search (e.g. when all activity modes are decided) or only at execution time (e.g. in case of uncertain durations). A feasible schedule can be obtained from the augmented graph by assigning start times according to the precedence constraints.

A key step for any PCP approach is the detection of possible resource conflicts, usually encoded as Minimal Critical Sets [11]. A MCS is a minimal set S of activities, collectively overusing a resource r_k; the activities must potentially overlap in time. We provide the following MCS definition (from [15]):

Definition 1. A MCS for a resource $r_k \in R$ is a set of activities such that:

1. $\sum_{a_i \in MCS} req_{i,k} > cap_k$;
2. $\forall a_i \in MCS$: $\sum_{a_j \in MCS \setminus \{a_i\}} req_{j,k} \leqslant cap_k$;
3. $\forall a_i, a_j \in MCS$: a_i and a_j may overlap, given the temporal constraints,

where (1) requires the set to be a conflict, while (2) and (3) respectively are the minimality/overlapping conditions. By relaxing the minimality requirement we get a so-called *Critical Set* (CS). A MCS can be resolved by posting a precedence constraint (i.e. a *resolver*) between any pair of activities in the set.

PCP approaches proceed by the repeated identification and resolution of MCSs, until the graph becomes conflict free. The method in [15] adopts a tree search scheme where choice points correspond to MCSs and each branch represents a possible resolver; the paper introduces the so-called *preserved space* measure to rank resolvers and MCSs. Other tree search methods include [2,5], where min/max time lags are also considered and [9] for the Multi-mode RCPSP. Many PCP approaches target the Stochastic RCPSP [10,11,21]. In [18,19] the PCP technique is used to generate robust schedules; the combination of uncertain durations and time lags has been considered in [16].

A MCS can be identified via enumeration [15,21], allowing one to find the optimal set according to a ranking heuristic, but with exponential worst case complexity. Alternatively [19], one can start from a so-called resource usage envelope [17,20]: each possible over-usage instant in the envelope corresponds to a CS; then a *Minimal* CS can be extracted via enumeration (with exponential complexity) or via heuristic sampling.

In this work, we identify a MCS by sampling a *single* usage peak; this is detected from the solution of a min-flow problem by exploiting the transitivity of temporal precedence graphs [8]. Related techniques are outlined in [12,17,20], the latter being the core for the resource envelope computation in [19]. Compared to those approaches our method has similar computational complexity, but is considerably simpler.

Finally, note that in case durations are fixed one can restrict to peaks in a specific schedule, e.g. assuming all activities are started as soon as possible [19,18]: the method is fast and obtains good results, but does not work with variable or uncertain durations and is therefore not considered here.

4. Minimal Critical Set detection

We propose to detect possible conflicts by: (1) solving a min-flow problem on a specific resource r_k to identify the maximal-weight CS; (2) sampling a MCS via steepest descent. As a quality measure, we adopt the *preserved space* heuristic from [15]; our method always finds a MCS unless the graph is conflict free. In this section each step is described in detail, while Section 5 investigates the efficiency and effectiveness of the approach in guiding MCS based tree-search.

4.1. MCS detection as a maximal weight independent set problem

The input for the MCS detection process is a so-called *Resource Graph* (*RG*):

Definition 2. A Resource Graph for r_k is a directed acyclic graph $\langle A_{r_k}, E_{r_k}\rangle$; A_{r_k} is a set of *requirements* (nodes) ρ_i, each representing a time span when the resource is used; $w(\rho_i)$ is the requirement value; E_{r_k} is the set of arcs and contains the pair (ρ_i, ρ_j) if and only if requirement ρ_i is expired when ρ_j starts.

Without loss of generality, we assume the RG has a single source node with no predecessor and a single sink node with no successor; source/sink nodes are connected to all other nodes in the graph. Observe that, in the context of the RG, the term "requirement" refers to a node. Since arcs in the RG represent temporal precedence relations, the transitivity property holds[1] and the (undirected) RG is a *comparability graph* [8], i.e. an undirected graph connecting comparable pairs of elements in a partial order.

A Resource Graph can be associated to the problem from Section 2 by:

(1) building a requirement ρ_i for each activity[2] a_i, with $w(\rho_i) = req_{ik}$;
(2) building an arc (ρ_i, ρ_j) if $(a_i, a_j) \in E$.

The RG formalism is not restricted to the basic RCPSP definition: release times and deadlines can be taken into account by adding arcs (ρ_i, ρ_j) for each activity pair such that a_i is bound to end before a_j starts[3]; more complex situations can also be modeled. In general, if a Resource Graph can be built as from Definition 2, our MCS detection approach can be applied.

Fig. 2(A) shows the RG corresponding to the RCPSP instance from Fig. 1, where an additional deadline constraint has been specified on a_7 (namely $dl_7 = 13$); the deadline leads to additional arcs: in particular, by reasoning on longest paths we deduce activity a_2 must end before time 5; since a_3, a_4 cannot start earlier than 5, arcs (a_2, a_3) and (a_2, a_4) are added to the graph.

Since the activities in a CS must be possibly overlapping, we obtain that *activities in a CS always form an Independent Set* (*IS*) *in the Resource Graph*, i.e. a subset $S \subseteq A_{r_k}$ such that no pair of requirements $\rho_i, \rho_j \in S$ is connected by an arc in E_{r_k}. *An Independent Set S on the RG is a CS if and only if* $\sum_{\rho_i \in S} w(\rho_i) > cap_k$.

4.2. Maximal weight independent set as a minimum flow problem

We can therefore test the existence of a CS on a resource r_k by checking the weight of the maximal weight IS on the Resource Graph. As stated in [8], this is a polynomial complexity problem on *comparability graphs* and can be solved via flow-theory results. Since no detailed algorithm is given in the reference, in the following we provide and prove a simple solution method.

1. Problem formulation: Let Π be the set of all source-to-sink paths π_j in RG (an exponential number) and let $n(\pi_j)$ be the number of nodes in the path. Due to the transitivity, each path corresponds to a maximal size, fully connected set of nodes

[1] Transitively implied arcs are usually omitted in the figures for sake of simplicity.
[2] One may restrict to activities with $req_{ik} > 0$ for the sake of readability.
[3] Checked (e.g.) via longest path or Temporal Constraint Network reasoning [6].

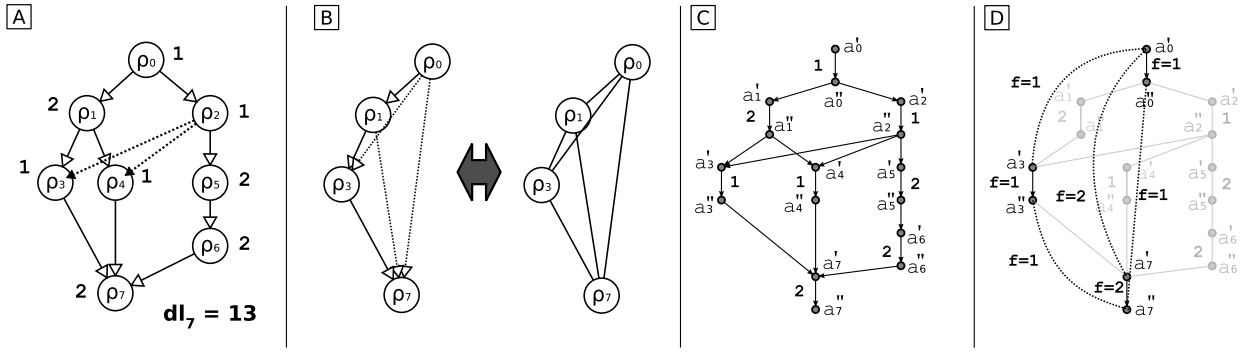

Fig. 2. (A) Resource graph, corresponding to the instance from Fig. 1; (B) Path-clique analogy; (C) Graph for min-flow computation; (D) Part of the initial flow.

in the (undirected) RG, i.e. a *maximal clique* (see Fig. 2(B), where the dotted arcs are implied by the transitivity property); this is often referred to as *Clique Path*. In any IS no two nodes can be selected from the same maximal clique, therefore the problem of finding a maximal weight IS can be formulated as the Integer Linear Program P′:

$$\mathbf{P'}: \quad \max \sum_{\rho_i \in A_{r_k}} w(\rho_i) \cdot x_i, \qquad \text{s.t.} \sum_{\rho_i \in \pi_j} x_i \leqslant 1, \quad \forall \pi_j \in \Pi, \qquad (1)$$
$$x_i \in \{0,1\},$$

$$\mathbf{P''}: \quad \min \sum_{\pi_j \in \Pi} y_j + \sum_{\rho_i \in A_{r_k}} u_i, \qquad \text{s.t.} \sum_{\rho_i \in \pi_j} y_j + u_i \geqslant w(\rho_i), \quad \forall \rho_i \in A, \qquad (2)$$
$$y_j \geqslant 0, \qquad u_i \geqslant 0,$$

where x_i are the decision variables and $x_i = 1$ if requirement ρ_i is in the selected set; Constraints (1) ensure no two requirements are chosen from the same Clique Path. Note P′ has an exponential number of constraints. By relaxing the integrality restriction, we can get a dual problem P″, with an exponential number of variables. *There exists an optimal solution for* P″ *using* y_j *variables only*.

Proof. Each u_i variable in P″ appears in exactly one Constraint (2), while each y_j variable in $n(\pi_j)$ constraints, with $n(\pi_j) \geqslant 1$; hence, for each optimal solution for P″ with a u_i variable equal to 1, it is possible to replace u_i with some y_j with no change in the objective value. □

Observe that each y_i variable represents a value assigned to path π_j in order to satisfy the covering Constraints (2); equivalently, one may think of each y_i as an amount of flow to be routed from source to sink along path π_j. Therefore, we deduce that *Program* P″ *consists of routing the minimum amount of flow from source to sink, such that all covering constraints are satisfied*; by duality, the minimum flow f^* is an *upper bound* on the maximal IS weight.

2. Solving P″: Program P″ can be solved by:

(1) splitting each node ρ_i into two sub-nodes ρ_i', ρ_i'', connected by an arc;
(2) labeling each arc (ρ_i', ρ_i'') with minimum flow requirement $w(\rho_i)$;
(3) labeling remaining arcs in the RG with zero minimum flow requirement;

Fig. 2(C) shows the modified resource graph after node splitting; as one can see, the result is very similar to a Temporal Constraint Network, where all arcs represent precedence constraints with non-negative time lag. An optimal solution can be found by (1) starting from an initial feasible flow and (2) performing iterative reductions with the *inverse* Ford–Fulkerson's method[4]; the complexity is $O(|E_{r_k}| \cdot \mathcal{F})$ (where $\mathcal{F}$ is the value of the initial flow).

3. Computing an initial flow: An initial solution can be computed by adding, for each requirement ρ_i, a flow of $w(\rho_i)$ units on the path $\rho_0' \to \rho_i' \to \rho_i'' \to \rho_{n-1}''$ (where ρ_0, ρ_{n-1} respectively denote the source and the sink requirements). Fig. 2(D) shows (part of) the initial solution for the example.

4. Extracting a solution for P′: We still have to show that an optimal (integer) solution for P′ can be extracted by the optimal solution of P″. By definition, the flow through the source/sink cut is f^*; moreover, all arcs in the cut have zero residual

[4] In detail, we use the inverted version of the classical Edmonds–Karp algorithm [7]; the pseudo-code is available at http://ai.unibo.it/node/424.

flow, i.e. the flow exactly equals the minimum flow requirement; now, let P^* be the set of ρ_i such that (ρ_i', ρ_i'') is in the source/sink cut; we have

$$\sum_{\rho_i \in P^*} w(\rho_i) = f^*.$$

Let us assume $x_i = 1$ if $\rho_i \in P^*$ ($x_i = 0$ otherwise) as a candidate P′ solution; by duality of P″, such a solution is optimal; hence we just have to show it is feasible, i.e. no two requirements ρ_i are covered by the same path π_j.

Proof. Since $f(\rho_i', \rho_j'') = \sum_{\rho_i \in \pi_j} y_j$ we get

$$f^* = \sum_{\rho_i \in P^*} w(\rho_i) = \sum_{\rho_i \in P^*} f\left(\rho_i', \rho_j''\right) = \sum_{\rho_i \in P^*} \sum_{\rho_i \in \pi_j} y_j. \tag{3}$$

Since the solution is optimal, we have $y_j = 0$ if path π_j is not covering any requirement in the source/sink cut and therefore, from the objective of P″:

$$f^* = \sum_{\pi_j \in \Pi} y_j = \sum_{\exists \rho_i \in P^*:\ \rho_i \in \pi_j} y_j. \tag{4}$$

By combining (3) and (4) we deduce that no y_j covers more than one ρ_i in the source/sink cut; hence, not two $\rho_i, \rho_j \in P^*$ belong to the same clique. □

Final remarks: In a tree search scheme where MCS are resolved by adding precedence constraints, *a starting feasible flow at each search node is provided by the minimum flow of its parent*. Since the complexity of Ford–Fulkerson's algorithm depends on the initial flow value, such an incremental approach allows a substantial performance improvement.

Thanks to the similarity between the graph used for flow minimization and a Temporal Constraint Network, in many cases it is possible to perform CS detection on the temporal model itself, with little or no graph transformation (unlike in [17,20]). Moreover, extracting the maximal weight IS from the source/sink is simpler compared to the re-routing method described in [12].

4.3. Reduction to MCS

If at the end of each minimization process the weight of the IS is higher than cap_k, then a CS has been identified. However: (1) the detected CS is not necessarily minimal (branching on a non-minimal CS can result in exploring unnecessary search paths); (2) the detected CS does not necessarily yield a good choice point when used in tree search. We propose to tackle both problems by a simple greedy minimization step.

As evaluation criterion for a given set S we adopt the so-called *preserved space* heuristic, introduced in [15]; the preserved space of a resolver (a_i, a_j) is an estimate of the amount of search space left after the addition of the precedence constraint; the preserved space of the MCS is the total preserved space of its resolvers. Once a CS is identified via the min-flow based approach, we extract a Minimal CS by iteratively removing the activity causing the largest reduction of preserved space (i.e. via steepest descent); the process stops when removing any activity causes the set weight to drop under cap_k (i.e. the CS is minimal). The approach has time complexity $O(|CS|^2)$, where $|CS|$ is the size of the original CS.

5. Experimental results

In order to assess the effectiveness of the proposed method w.r.t. approaches from the literature, we perform an extensive experimentation on RCPSP benchmarks from the PSPlib; in detail, we consider three MCS detection techniques, using the (smallest) preserved space as ranking heuristic:

(1) the one described in this work, referred to as MINFLOW;
(2) the enumerative procedure defined in [15], referred to as ENUM;
(3) the procedure in [19] (based on heuristically sampling MCS from peaks in the resource usage envelope), referred to as PEAKS.

Our ENUM implementation is analogous to the one in the original paper; as an exception, the incremental computation of the CS score is replaced by a caching scheme (each CS is evaluated only once): preliminary results showed the performance difference to be limited. In the PEAKS procedure, the usage envelope is computed with the method introduced in [17]; the incremental computation steps described in [19] and (more prominently) in [20] are not implemented, actually leaving room for further improvements. MCS are sampled from peaks as in [4]: for a peak with size n, we extract up to n MCS, with no upper limit on the set cardinality.

The MCS detection methods are employed in a Depth First Search scheme: at each node of the search tree an MCS is selected for each resource r_k; then the best set among all resources is chosen for branching. We use binary choice points: we identify the resolver (a_{i^*}, a_{j^*}) with the highest preserved space; this is *posted* on the left branch, *forbidden* on the right branch:

$$end(a_{i^*}) \leqslant start(a_{j^*}) \quad \textbf{or} \quad start(a_{j^*}) > end(a_{i^*}).$$

We perform resolver simplification as described in [15]; the technique consists of discarding (for each MCS) the resolvers (a_i, a_j) such that another resolver (a_h, a_k) exists and $(a_i, a_j) \Rightarrow (a_h, a_k)$. Timetable, disjunctive and edge-finding filtering is used for all resources. Optimization (i.e. makespan minimization) is done in the usual CP fashion (i.e. by posting constraints on the problem objective whenever an improving solution is found).

We have run the optimization process on the j30, j60, j90 and j120 benchmarks, the number in the benchmark name identifying the size of the graph; each solution attempt was capped at 600 seconds. The approach is implemented in IBM-ILOG Solver and Scheduler 6.7 and all experiments are performed on an Intel Xeon E5410, 2.33 GHz, 7 GB RAM. In the following, an excerpt of the experimentation results is reported: detailed information is available on-line[5].

Table 1 reports results for the MINFLOW approach on the j30, j60, j90 benchmarks; the j120 set is considered separately in Table 4, since it is generated with different parameters. Instances are grouped by *Resource Factor* (**RF** – the average number of required resources, averaged over the number of activities) and Resource Strength (**RS** – the ratio between a resource capacity and its average requirement); each group counts 30 graphs and row **OP/TO** shows the number of optimally solved and timed-out ones; for optimally solved instances, row **time** and **fail** are the average solution time and number of fails (backtracks), while **tmcs** is the average portion of the solution time spent in MCS detection.

The toughest challenge comes from high **RF** and low **RS** instances, usually having many, large size MCS and yielding the biggest search trees. As the number of nodes grows, detecting MCS (min-flow + steepest descent) takes an increasing portion of the total solution time; in detail, solving the flow problem accounts for up to 25% of the MCS detection, with peaks for low **RS** values; conversely, the CS greedy minimization step seems to be heavier for higher **RS** and more sensitive to the size of the graph: this explains the growing **tmcs** for increasing **RS** values on j120.

Table 2 reports results for the comparison between MINFLOW and ENUM on j30, j60 and j90 (j120 is considered in Table 4). Here, row **OP/TO** is the number of optimally-solved/timed-out instances for the ENUM approach, while **OP*****/TO*** counts the number of instances where both ENUM and MINFLOW find the optimal solution/hit the time limit. Row **time**, is the average solution time gap, defined for a single instances as

$$\textbf{time} = \begin{cases} 0 & \text{if } time_{MINFLOW} < 10 \text{ msec and } time_{ENUM} < 10 \text{ msec}, \\ \frac{time_{MINFLOW} - time_{ENUM}}{\max(time_{MINFLOW}, time_{ENUM})} & \text{otherwise} \end{cases}$$

where $time_{MINFLOW}$ and $time_{ENUM}$ are the solution time of the two approaches; analogously, the **fail** and **mk** columns are the average fail and makespan gap. Time and fail gaps are computed on instances solved to optimality by both approaches; the makespan gap is computed only when at least one method reports a time-out and is $\pm 100\%$ in case one of the two approaches has found no solution. In the table, results are in bold font whenever MINFLOW is doing better than or as well as ENUM.

As expected, MCS detection for the ENUM approach is heavier than MINFLOW, taking from 8%–85% (on j30) to basically all the solution time (for the larger graphs). This is a consequence of the exponential worst case complexity of the enumeration; the gap is more evident for large **RF** and **RS**, corresponding to the highest number of MCS: no instance for the corresponding groups in j90 is solved to optimality by ENUM, while MINFLOW reports no time-outs. In some cases on large graphs the ENUM approach was not able to find a solution at all, getting stuck when searching for the optimal MCS; this is the reason for the large negative **mk** on j60 and j90, caused by MINFLOW scoring -100% when ENUM does not find any feasible schedule. Even when ENUM is able to reach an optimal or suboptimal solution, the MINFLOW approach is usually faster and achieves comparable makespan. In general, ENUM seems to perform best for small graphs and small **RF**, **RS** values; this is why the scalability issues are less apparent on j120, since the highest **RS** values are not considered there. Quite unexpectedly, the ENUM approach does not seem to produce consistently smaller search trees when both approaches achieve optimality, raising interest in investigating the effectiveness of the preserved space as a search heuristic.

Table 3 reports results for the comparison between MINFLOW and PEAKS; results for j120 are in Table 4. The PEAKS approach can be considered a compromise between the ENUM and the MINFLOW approach: the set of all peaks in the resource usage envelope is obtained via a polynomial time procedure, then MCS are extracted from each peak via an enumerative process, stopped once a given number of Critical Sets has been considered. PEAKS performs very well on the smaller instances, considerably better than MINFLOW and ENUM in terms of number of time-outs; the method is particularly effective for low/average **RF** and low **RS** values; in this case, the quality of the returned schedules is very good, even when the optimal schedule is not reached (see the **mk** values). On instances solved to optimality by both approaches,

[5] At the address http://ai.unibo.it/node/424.

Table 1
Results for the MINFLOW approach.

	RF	0.25	0.25	0.25	0.25	0.50	0.50	0.50	0.50	0.75	0.75	0.75	0.75	1.00	1.00	1.00	1.00
	RS	0.20	0.50	0.70	1.00	0.20	0.50	0.70	1.00	0.20	0.50	0.70	1.00	0.20	0.50	0.70	1.00
J30	OP/TO	30/0	30/0	30/0	30/0	29/1	30/0	30/0	30/0	7/23	30/0	30/0	30/0	0/30	29/1	30/0	30/0
	time	0.02	0.01	0.01	0.01	25.66	3.84	0.02	0.01	229.45	6.10	1.91	0.02	–	32.55	3.63	0.03
	fail	72	22	18	13	83154	14236	36	23	492517	17167	4450	26	–	62493	7025	26
	tmcs	15.6%	11.7%	10%	3.3%	35.4%	20.9%	21.7%	20%	38.4%	39.4%	32.7%	26.7%	–	49.3%	39.5%	22%
J60	OP/TO	25/5	30/0	30/0	30/0	0/30	28/2	30/0	30/0	0/30	18/12	30/0	30/0	0/30	8/22	30/0	30/0
	time	10.54	0.05	0.04	0.03	–	15.05	0.12	0.11	–	8.38	0.28	0.22	–	15.03	0.62	0.37
	fail	27 372	98	68	46	–	21869	124	92	–	8892	197	106	–	13601	303	114
	tmcs	28.7%	35.2%	33.7%	30.2%	–	49.8%	55.4%	56.5%	–	58.2%	63.5%	66.7%	–	61.3%	66.9%	66.2%
J90	OP/TO	11/19	29/1	30/0	30/0	0/30	22/8	30/0	30/0	0/30	17/13	30/0	30/0	0/30	12/18	30/0	30/0
	time	70.90	0.22	0.16	0.13	–	6.70	0.66	0.70	–	50.15	1.69	1.62	–	6.79	2.76	2.70
	fail	84 016	244	131	107	–	7316	245	201	–	43734	343	240	–	1762	353	252
	tmcs	44.3%	55.3%	57.4%	61.8%	–	66.3%	73.5%	75.7%	–	68.9%	78.7%	79.6%	–	82.7%	80.3%	81.8%

Table 2
Results for the comparison with the ENUM approach.

	RF	0.25	0.25	0.25	0.25	0.50	0.50	0.50	0.50	0.75	0.75	0.75	0.75	1.00	1.00	1.00	1.00
	RS	0.20	0.50	0.70	1.00	0.20	0.50	0.70	1.00	0.20	0.50	0.70	1.00	0.20	0.50	0.70	1.00
J30	OP/TO	30/0	30/0	30/0	30/0	29/1	30/0	30/0	30/0	12/18	30/0	30/0	30/0	0/30	26/4	30/0	30/0
	OP*/TO*	**30**/0	**30**/0	**30**/0	**30**/0	28/**0**	**30**/0	**30**/0	**30**/0	6/**17**	**30**/0	**30**/0	**30**/0	0/30	**26**/**1**	**30**/0	**30**/0
	time	**−12.1%**	3.3%	8.3%	**−10%**	36.6%	**−18.3%**	**−28.9%**	**−34.4%**	10.1%	**−28.3%**	**−42.1%**	**−54.5%**	–	**−48.7%**	**−70.7%**	**−74.7%**
	fail	**−3.6%**	**−2.9%**	2.5%	**−6.5%**	49.9%	**−2.4%**	**−1.2%**	**−1.1%**	32.1%	**−0.4%**	0.9%	**−2.9%**	–	0.5%	**−7.9%**	**−10.2%**
	mk	–	–	–	–	**−5.7%**	–	–	–	0.2%	–	–	–	0.7%	**−1.9%**	–	–
J60	OP/TO	25/5	30/0	30/0	30/0	0/30	24/6	30/0	30/0	0/30	15/15	30/0	27/3	0/30	2/28	16/14	17/13
	OP*/TO*	23/**3**	**30**/0	**30**/0	**30**/0	0/30	23/**1**	**30**/0	**30**/0	0/30	13/**10**	**30**/0	**27**/**0**	0/30	**2**/**22**	**16**/**0**	**17**/**0**
	time	**−6.2%**	**−40%**	**−42.2%**	**−47.2%**	–	**−66.9%**	**−89.8%**	**−91.8%**	–	**−97.7%**	**−98%**	**−98.8%**	–	**−98.5%**	**−99.3%**	**−99.7%**
	fail	0.9%	2.6%	**−4%**	**−6.2%**	–	**−0.4%**	**−4.9%**	**−6.6%**	–	**−40.9%**	**−11.6%**	**−13.6%**	–	25.8%	**−19.3%**	**−23.2%**
	mk	**−0.6%**	–	–	–	**−0.8%**	**−1%**	–	–	**−0.2%**	**−1.2%**	–	**−100%**	**−1.5%**	**−30%**	**−93%**	**−100%**
J90	OP/TO	7/23	30/0	29/1	30/0	0/30	15/15	22/8	17/13	0/30	0/30	5/25	0/30	0/30	0/30	0/30	0/30
	P*/TO*	6/**18**	29/0	**29**/**0**	**30**/0	0/30	14/**7**	**22**/**0**	**17**/**0**	0/30	0/**13**	**5**/**0**	0/**0**	0/30	0/**18**	0/**0**	0/**0**
	time	9.4%	**−70.6%**	**−80.3%**	**−87.2%**	–	**−98.7%**	**−99.2%**	**−99.2%**	–	–	**−99.8%**	–	–	–	–	–
	fail	16.1%	**−3.4%**	**−11%**	**−3%**	–	**−13.1%**	**−12.7%**	**−5.3%**	–	–	**−12.4%**	–	–	–	–	–
	mk	**−2.7%**	1.1%	**−5.7%**	–	**−1.4%**	**−27.8%**	**−100%**	**−100%**	**−30.4%**	**−74.2%**	**−100%**	**−100%**	**−67.8%**	**−100%**	**−100%**	**−100%**

Table 3
Results for the comparison with the PEAKS approach.

	RF	0.25	0.25	0.25	0.25	0.50	0.50	0.50	0.50	0.75	0.75	0.75	0.75	1.00	1.00	1.00	1.00
	RS	0.20	0.50	0.70	1.00	0.20	0.50	0.70	1.00	0.20	0.50	0.70	1.00	0.20	0.50	0.70	1.00
J30	OP/TO	30/0	30/0	30/0	30/0	30/0	30/0	30/0	30/0	26/4	30/0	30/0	30/0	13/17	30/0	30/0	30/0
	P*/TO*	**30**/0	**30**/0	**30**/0	**30**/0	29/0	**30**/0	**30**/0	**30**/0	7/4	**30**/0	**30**/0	**30**/0	0/17	29/0	**30**/0	**30**/0
	time	**−28.8%**	**−31.9%**	**−25%**	**−11.7%**	51.6%	**−57.4%**	**−73.2%**	**−77.2%**	77.5%	**−37.3%**	**−78.5%**	**−86.3%**	–	**−16.1%**	**−80.7%**	**−90.3%**
	fail	8.4%	9.2%	7.4%	**−0.6%**	88.2%	17.7%	11.6%	7.2%	97.8%	36.8%	23.4%	9.3%	–	73.3%	14.8%	7.5%
	mk	–	–	–	–	5%	–	–	–	3.2%	–	–	–	3.8%	0%	–	–
J60	OP/TO	30/0	30/0	30/0	30/0	4/26	29/1	30/0	30/0	0/30	17/13	30/0	30/0	0/30	7/23	29/1	28/2
	P*/TO*	25/0	**30**/0	**30**/0	**30**/0	0/26	28/1	**30**/0	**30**/0	0/30	16/**11**	**30**/0	**30**/0	0/30	5/**20**	**29**/**0**	**28**/**0**
	time	**−28%**	**−76.6%**	**−80.2%**	**−79.4%**	–	**−56.6%**	**−95.2%**	**−95.4%**	–	**−85.4%**	**−98%**	**−98.1%**	–	**−63.8%**	**−98.6%**	**−99.1%**
	fail	13.4%	10.8%	6.5%	6%	–	23.7%	15%	10.2%	–	11.5%	12%	10.3%	–	24.1%	13%	7.8%
	mk	4.4%	–	–	–	5.5%	6.2%	–	–	4.2%	1%	–	–	2.7%	0.4%	**−100%**	**−100%**
J90	OP/TO	22/8	30/0	30/0	30/0	0/30	23/7	28/2	24/6	0/30	4/26	14/16	12/18	0/30	1/29	7/23	9/21
	P*/TO*	11/8	29/0	**30**/0	**30**/0	0/30	**20**/**5**	**28**/**0**	**24**/**0**	0/30	3/**12**	**14**/**0**	**12**/**0**	0/30	**1**/**18**	**7**/**0**	**9**/**0**
	time	**−0.2%**	**−87.5%**	**−91.2%**	**−92.1%**	–	**−89.8%**	**−98.7%**	**−98.8%**	–	**−99.5%**	**−99.3%**	**−99.4%**	–	**−99.6%**	**−99.7%**	**−99.6%**
	fail	54.1%	11.6%	6.7%	8.8%	–	25.3%	8.6%	9.9%	–	**−18.9%**	16.6%	11.4%	–	16.1%	4.6%	17.9%
	mk	2.7%	1.1%	–	–	3.8%	**−9.7%**	**−100%**	**−83.3%**	4.1%	**−44.6%**	**−100%**	**−100%**	**−36%**	**−75.9%**	**−100%**	**−100%**

Table 4
Results for the J120 benchmark.

	RF	0.25	0.25	0.25	0.25	0.25	0.50	0.50	0.50	0.50	0.50	0.75	0.75	0.75	0.75	0.75	1.00	1.00	1.00	1.00	1.00
	RS	0.10	0.20	0.30	0.40	0.50	0.10	0.20	0.30	0.40	0.50	0.10	0.20	0.30	0.40	0.50	0.10	0.20	0.30	0.40	0.50
MINFLOW	OP/TO	0/30	6/24	21/9	23/7	29/1	0/30	0/30	0/30	6/24	16/14	0/30	0/30	0/30	0/30	14/16	0/30	0/30	0/30	1/29	9/21
	time	–	98.75	25.51	2.72	0.83	–	–	–	8.83	15.37	–	–	–	–	36.69	–	–	–	18.55	16.89
	fail	–	27145	21746	1689	331	–	–	–	3717	8170	–	–	–	–	8717	–	–	–	915	734
	tmcs	–	60.3%	54%	53.5%	61%	–	–	–	60.1%	73.2%	–	–	–	–	76.5%	–	–	–	88.2%	88.3%
vs ENUM	OP/TO	0/30	2/28	15/15	19/11	27/3	0/30	0/30	0/30	0/30	2/28	0/30	0/30	0/30	0/30	0/30	0/30	0/30	0/30	0/30	0/30
	P*/TO*	0/30	2/**24**	13/**7**	17/**5**	26/**0**	0/30	0/30	0/30	0/**24**	1/**13**	0/30	0/30	0/30	0/30	0/**16**	0/30	0/30	0/30	0/**29**	**0**/**21**
	time	–	36.3%	**−78.2%**	**−81.7%**	**−89.8%**	–	–	–	–	**−99.6%**	–	–	–	–	–	–	–	–	–	–
	fail	–	64.5%	**−4.3%**	**−9.1%**	**−0.8%**	–	–	–	–	**−22.8%**	–	–	–	–	–	–	–	–	–	–
	mk	**−0.2%**	**−3%**	**−2.4%**	**−0.6%**	**−0.1%**	0.6%	**−24.1%**	**−45.4%**	**−67.6%**	**−79.7%**	**−52.7%**	**−83.5%**	**−96.7%**	**−100%**	**−100%**	**−96.5%**	**−100%**	**−100%**	**−100%**	**−100%**
vs PEAKS	OP/TO	1/29	11/19	20/10	26/4	29/1	0/30	0/30	1/29	1/29	6/24	0/30	0/30	0/30	0/30	0/30	0/30	0/30	0/30	0/30	0/30
	P*/TO*	0/29	6/19	17/**6**	22/**3**	28/**0**	0/30	0/30	0/29	1/**24**	6/**14**	0/30	0/30	0/30	0/30	0/**16**	0/30	0/30	0/30	0/**29**	0/**21**
	time	–	**−33.3%**	**−81.6%**	**−80%**	**−93.9%**	–	–	–	**−99.4%**	**−94.6%**	–	–	–	–	–	–	–	–	–	–
	fail	–	41.7%	6.4%	19.5%	0.3%	–	–	–	16.2%	19.6%	–	–	–	–	–	–	–	–	–	–
	mk	3%	2%	**−0.9%**	1.5%	2%	8.2%	4.7%	**−17.3%**	**−33.9%**	**−58.6%**	**−47%**	**−65.9%**	**−79.6%**	**−93.2%**	**−100%**	**−100%**	**−100%**	**−100%**	**−100%**	**−100%**

the MINFLOW method is usually much faster, but reports a larger number of fails; therefore, even if finding *the* optimal MCS according to the preserved space heuristic does not necessarily produce smaller search trees, the heuristic is still providing a valuable guide. This is especially appealing since the conflict detection time of PEAKS could be reduced by introducing incremental computation techniques. On large instances, for high **RF** and **RS** the PEAKS approach suffers from the same scalability issues as ENUM and in several cases it is not able to find any feasible solution, as hinted by the large negative makespan gap. Results on the j120 set follow the same general trends, stressed by the large instance size.

5.1. Other tested variants

Since only possibly overlapping activities are considered by enumerative techniques, the ENUM approach may be strongly affected by the width of the time windows. This is in part the rationale behind the search method used in [15]: an initial lower bound is set as a global deadline constraint, then the resulting scheduling problem is solved to feasibility; in case infeasibility is proven, the lower bound is improved by one unit, until an optimal solution is found. We tested this search method on the whole benchmark set: for optimally solved instances, the number of fails is reduced by 70–80% on average, the solution time improves by 25–35% for MINFLOW, 35–45% for ENUM and even more for PEAKS; obviously, no feasible solution is returned in case of time-outs. Interestingly, despite the improvements on optimally solved instances, the scalability issue of ENUM and PEAKS are neither solved nor consistently reduced. When it comes to pairwise comparisons, the general trends observed in Tables 2 and 3 for the solution time and number of fails still hold. Detailed results for this second experimentation are available on-line[5].

Finally, on the purpose to further investigate the trade-off between MCS quality and detection time, we experimented a MINFLOW variant where the greedy CS minimization step is replaced by random activity removal (computationally lighter). On j30 and j60 this leads to a considerably larger number of time-outs and more fails (10% on average); when optimality is reached, search is however faster (10% on average). On the larger benchmarks, the number of time-outs becomes comparable and the solution time gap broadens; very interestingly, this is not only a by-product of the increased MCS detection time: random minimization often produces *smaller search trees*, in particular on the j120 benchmarks (13% fewer fails). Somehow, the reliability of the preserved space heuristic seems to decrease as the graph gets bigger. Detailed results for this experimentation are available on-line[5].

6. Conclusion

We propose a min-flow algorithm for detecting MCS in Precedence Constraint Posting; the idea is simple and easier to implement than current approaches. Evidence of the method's effectiveness is provided through an extensive experimentation with the PSPlib benchmark, where the min-flow approach exhibits improved scalability compared to two state-of-art MCS detection procedures. Additionally, the results give insight into the effectiveness of the preserved space heuristic, apparently decreasing as the instance size grows. Finally, our method could be employed for envelope computation in the peak-based MCS detection, providing a (considerably simpler) alternative to the technique currently in use; this is a relevant remark, considered the results obtained by PEAKS in our experimentation.

References

[1] P. Baptiste, C. Le Pape, W. Nuijten, Constraint-Based Scheduling, Kluwer Academic Publishers, 2001.
[2] M. Bartusch, R.H. Möhring, F.J. Radermacher, Scheduling project networks with resource constraints and time windows, Ann. Oper. Res. 16 (1) (1988) 199–240.
[3] P. Brucker, A. Drexl, R.H. Möhring, K. Neumann, E. Pesch, Resource-constrained project scheduling: Notation, classification, models, and methods, European J. Oper. Res. 112 (1) (1999) 3–41.
[4] A. Cesta, A. Oddi, S.F. Smith, A constraint-based method for project scheduling with time windows, J. Heuristics 8 (1) (2002) 109–136.
[5] B. De Reyck, W. Herroelen, A branch-and-bound procedure for the resource-constrained project scheduling problem with generalized precedence relations, European J. Oper. Res. 111 (1) (1998) 152–174.
[6] R. Dechter, I. Meiri, J. Pearl, Temporal constraint networks, Artificial Intelligence 49 (1–3) (1991) 61–95.
[7] J. Edmonds, R.M. Karp, Theoretical improvements in algorithmic efficiency for network flow problems, J. ACM 19 (2) (1972) 248–264.
[8] M. Golumbic, Algorithmic Graph Theory and Perfect Graphs, second ed., Elsevier, 2004.
[9] R. Heilmann, A branch-and-bound procedure for the multi-mode resource-constrained project scheduling problem with minimum and maximum time lags, European J. Oper. Res. 144 (2) (January 2003) 348–365.
[10] G. Igelmund, F.J. Radermacher, Algorithmic approaches to preselective strategies for stochastic scheduling problems, Networks 13 (1) (January 1983) 29–48.
[11] G. Igelmund, F.J. Radermacher, Preselective strategies for the optimization of stochastic project networks under resource constraints, Networks 13 (1) (January 1983) 1–28.
[12] D. Kagaris, S. Tragoudas, Maximum independent sets on transitive graphs and their applications in testing and CAD, in: Proc. of ICCAD, IEEE Computer Society, 1997, pp. 736–740.
[13] R. Kolisch, PSPLIB – A project scheduling problem library, European J. Oper. Res. 96 (1) (January 1997) 205–216.
[14] R. Kolisch, C. Schwindt, A. Sprecher, Benchmark instances for project scheduling problems, in: Handbook on Recent Advances in Project Scheduling, Kluwer, 1999, pp. 197–212 (Chapter 9).
[15] P. Laborie, Complete MCS-based search: Application to resource constrained project scheduling, in: Proc. of IJCAI, Professional Book Center, 2005, pp. 181–186.
[16] M. Lombardi, M. Milano, A precedence constraint posting approach for the RCPSP with time lags and variable durations, in: Proc. of CP, 2009, pp. 569–583.

[17] N. Muscettola, Computing the envelope for stepwise-constant resource allocations, in: Proc. of CP, 2002, pp. 139–154.
[18] N. Policella, A. Cesta, A. Oddi, S.F. Smith, From precedence constraint posting to partial order schedules: A CSP approach to Robust Scheduling, AI Commun. 20 (3) (2007) 163–180.
[19] N. Policella, S. F. Smith, A. Cesta, A. Oddi, Generating robust schedules through temporal flexibility, in: Proc. of ICAPS, 2004, pp 209–218.
[20] T.K. Satish Kumar, Incremental computation of resource-envelopes in producer-consumer models, in: Proc. of CP, 2003, pp. 664–678.
[21] F. Stork, Stochastic resource-constrained project scheduling, PhD thesis, Technische Universitat Berlin, 2001.