Precedence Constraint Posting for the RCPSP with Uncertain, Bounded Durations

Michele Lombardi and **Michela Milano**
{michele.lombardi2,michela.milano}@unibo.it
DEIS, University of Bologna
Viale del Risorgimento 2, 40136 Bologna, IT

Abstract

The Resource Constrained Project Scheduling Problem is an important problem in project management, manufacturing and resource optimization. We focus on a variant of RCPSP with time lags and uncertain activity durations. We adopt a Precedence Constraint Posting approach and add precedence constraints to the original project graph so that all resource conflicts are solved and a consistent assignment of start times can be computed for whatever combination of activity durations. We propose a novel method for computing resource conflicts based on the minimum flow on the resource graph and we use it in an efficient complete search strategy. We test the approach on instances coming from the scheduling of parallel applications on multi processor systems on chip.

1 Introduction

In this work we tackle the Resource Constrained Project Scheduling Problem (RCPSP) with minimum and maximum time lags and uncertain, bounded durations. RCPSP aims to schedule a set of activities subject to precedence constraints and the limited availability of resources.

The classical RCPSP formulation is based on a Directed Acyclic Graph (*Project Graph*) where nodes are activities and arcs are precedence relations. Activities use a certain amount of finite capacity resources; the problem consists in finding an assignment of start times, such that no resource capacity is exceed and the *makespan* is minimized. Here we take into account two variants to the classical RCPSP formulation: time lags (i.e., minimum and maximum time interval) between activities) and uncertain durations. In particular, we assume activity durations are bounded by a worst and best case value; we do not require a probability distribution to be specified. We want to schedule all activities such that, whatever their duration is, all temporal and resource constraints are satisfied.

This RCPSP variant finds several industrial and design applications; as a case study, we consider predictable scheduling of parallel computer programs on multiprocessor systems, subject to hard real time constraints.

In this case activities are threads or processes, to be scheduler over limited hardware resources (processors, memories, communication channels...). In many practical scenarios durations are data dependent and not known a-priori (despite lower and upper bounds can be identified by off-line analysis). Uncertain durations is especially troublesome in the case of hard real time applications, where providing guarantees for the worst case application behavior is more important than achieving a good average performance. Time lags can be used in this scenario to model (e.g.) inter task communication latencies due to DMA transfers.

We adopt a Precedence Constraint Posting approach (see [Policella et al., 2007]): the solution we provide is an augmented project graph; this is the original project graph plus a *fixed* set of new precedence constraints, such that all possible resource conflicts are cleared and a consistent assignment of activity start times can be computed for whatever combination of durations at run time. In practice, once a solution (i.e. an augmented graph) is given, a feasible assignment of start times can be computed in polynomial time.

The main contributions of the paper are: 1) a constraint based temporal model to allow consistency check and propagation with uncertain, bounded durations and 2) a conflict detection procedure based on the solution of a minimum flow problem (integrated in an efficient complete search strategy). We perform computational experiments on instances representing a system design problem. We compare out approach with a generalization of the method by [Laborie, 2005], modified to take into account uncertain durations.

2 Problem definition

The classical RCPSP is defined on a directed acyclic graph $\langle A, E\rangle$ (referred to as *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 (t_i, t_j), defining precedence relations. 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 requires a certain amount $req(a_i, r_k)$ of one or more renewable resources r_k within a set R; all resources have finite capacity $cap(r_k)$. The problem consists in finding a *schedule* (that is, an assignment of start times to activities), such that no resource capacity is exceed and the overall completion time (*makespan*) is minimized.

Unlike in the classical RCPSP, we assume every activity has to start after a specified *release time* (rs_i) and to end

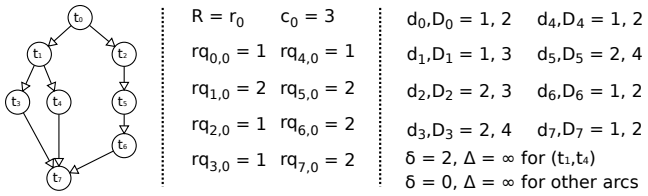

Figure 1: An instance of RCPSP with time lags and uncertain, bounded durations

before a specified *deadline* (dl_i); each arc (a_i, a_j) is labeled with a minimum and maximum value (δ_{ij}, Δ_{ij}), such that the time distance between a_i and a_j cannot be lower than δ_{ij} nor higher than Δ_{ij}. Finally, we assume activity durations are not known a priori, but range at execution time between a lower and an upper bound d_i and D_i.

We require the time window constraints to be met at run time for every possible scenario, that is for every possible combination of duration values. Therefore, all possible resource conflicts should be cleared so that a consistent assignment of start times can be computed for whatever combination of activity durations. The problem objective we take into account is to identify the best possible global deadline which can be met by a given project.

3 PCP: background and related work

We adopt a Precedence Constraint Posting approach (PCP, see [Policella et al., 2007]); in PCP possible resource conflicts are resolved off-line by adding precedence constraints between the involved activities. The resulting augmented graph defines a set of possible schedules, rather than a schedule in particular. More precisely, the graph can be provided as input to an on-line dispatcher; this will start each activity a_i: (1) within the time window (2) after the end time of all the predecessors. We assume *any* start time satisfying condition (1) and (2) can be assigned: choosing the minimal one is usually reasonable, but we allow the dispatcher to delay an activity (e.g. to react to some unexpected event).

The provided solution graph must be such that if these rules are followed, a feasible assignment of start times is guaranteed to be found for every possible combination of task durations. This amounts to enforcing dynamic controllability; this requires all deadline constraints are met when activity starting times are decided knowing only already executed activities (see [Vidal and Fargier, 1999]).

Many PCP approaches proceed by iteratively resolving Minimal Critical Sets (MCS - introduced by [G. Igelmund and F. J. Radermacher, 1983a]); an MCS is a set of activities collectively overusing one of the resources and such that the removal of a single activity from the set wipes out the conflict; additionally, the activities must have the possibility to overlap in time. Following [Laborie, 2005], we define a MCS for a resource r_k as a set of activities such that:

1. $\sum_{a_i \in MCS} req(a_i, r_k) > cap(r_k)$
2. $\forall a_i \in MCS \; : \sum_{a_j \in MCS \setminus \{a_i\}} req(a_j, r_k) \leq cap(r_k)$
3. $\forall a_i, a_j \in MCS, i < j \; : \; a_i \prec a_j$ and $a_j \prec a_i$ are both consistent with current state of the model.

Where (1) requires the set to be a conflict, (2) is the minimality condition, $a_i \prec a_j$ means a_i comes before a_j and (3) requires activities to be possibly overlapping. A MCS is *resolved* by posting a precedence constraint (i.e. a *resolver*) between any pair of activities in the set; complete search can thus be performed by using MCS as choice points and opening a branch for each possible resolver. This is the case of many PCP based works: for example [Laborie, 2005] makes use of complete search to detect MCS and proposes a heuristic to rank possible resolvers. Other approaches based on posting precedence constraints are described in [Reyck and Herroelen, 1998] (branch and bound) and [Policella et al., 2004b; 2004a]. None of those works takes into account duration uncertainty.

Detecting Critical Sets is a key issue in PCP methods; since the number of MCS is in general exponential in the size of the graph, complete enumeration can be time consuming. A way to overcome the issue is to assume a specific "execution policy": for example in [Reyck and Herroelen, 1998] activities are assumed to start as soon as their predecessors are over.

Uncertain activity durations are directly tackled by the literature on the Stochastic RCPSP; here, those are modeled as stochastic variables with known distribution; the objective function is to minimize the expected value of the makespan. Most works in this area focus on computing so-called *policies*, such that their execution by an on-line schedule avoids resource overusage and minimizes the expected value of makespan. For example, [G. Igelmund and F. J. Radermacher, 1983a; 1983b] define so-called *pre-selective policies*, which specify for each possible conflict an activity to be delayed.

The exponential number or MCS makes the straightforward use of pre-selective policies impractical; therefore restricted subclasses of pre-selective policies are introduced, for example: *earliest start policies* [R. H. Möhring, F. J. Radermacher, and G. Weiss, 1984; 1985] define as a *fixed* set of precedence constraints (similarly to PCP); in [F. Stork, 2001] AND-OR constraints are introduced to generalize earliest start policies; *linear pre-selective policies* (see [Rolf H. Möhring and Frederik Stork, 2000]) specify the activity to be delayed based on a linear size vector of priorities. All those approaches rely on complete MCS enumeration to identify the conflicts to be resolved [F. Stork, 2001; 2000]. To the best of the authors knowledge, no Stochastic RCPSP approach has considered time lags and time windows so far.

[Policella et al., 2007; 2004b] perform MCS detection in polynomial time by either computing a resource free schedule and observing the resulting usage peaks, or by analysis of the resource envelopes. In both cases activity durations are considered fixed. The mentioned works also incorporate time reasoning via simple temporal networks.

Our answer to the conflict detection issue is to cast the detection of a conflict to a (polynomial complexity) minimum flow problem by exploiting the transitivity of the solution

graph. This is a first major contribution of this work. A related technique is outlined in [Muscettola, 2002], where a maximum flow algorithm is used to extract usage envelopes at specific time instants, from a particular activity-resource graph. Finally, for excellent overviews about methods for the RCPSP problem and dealing with uncertainty in scheduling see [Brucker et al., 1999; J.C. Beck and A.J. Davenport, 2002; Herroelen and Leus, 2005].

4 Description of the approach

We propose a PCP based approach for the RCPSP with minimum and maximum time lags and uncertain, bounded activity durations. The method performs complete search branching on MCS. The first major contribution w.r.t. similar approaches like [Laborie, 2005] is the use of an efficient, polynomial time, MCS detection procedure based on the solution of a minimum flow problem.

Our second main is an expressive and efficient time model, consisting of a generalization of the temporal network adopted by [Policella et al., 2007]; in particular, the approach was modified to provide efficient time reasoning with uncertainty and enable constant time consistency/overlapping check.

4.1 The time model

The adopted temporal model consists of a constraint based formulation of the STNU formalism (with minor restrictions); we rely on constraint propagation, rather then on specialized algorithms, to enforce consistency. The model provides the following building blocks:

- *Event variables* ($\mathtt{T_i}$), associated to events τ_i; the domain of an event variable is the time span where τ_i can occur;
- *Free constraints* ($\mathtt{T_i} \xrightarrow{[a,b]} \mathtt{T_j}$), meaning that $\mathtt{T_i}$ and $\mathtt{T_j}$ must have enough flexibility to to allow τ_j to occur d' time units after τ_j, for *at least* a value d' in the interval $[a, b]$;
- *Contingent constraints* ($\mathtt{T_i} \xrightarrow{[a:b]} \mathtt{T_j}$), meaning that $\mathtt{T_i}$ and $\mathtt{T_j}$ must have enough flexibility to allow τ_j to occur d' time units after τ_j, for *every* value $d' \in [a, b]$;

Unlike interval variables (see [Laborie and Rogerie, 2008]), event variables represent *instantaneous* events and necessarily take place at run time (there is no "execution variable"). For all constraints $0 \leq a \leq b$ must hold; event variables connected by binary constraints form a directed graph. A temporal model of the problem at hand can be built by:

1. introducing two event variables $\mathtt{S_i}, \mathtt{E_i}$ for the start and the end of each activity a_i, respectively with time windows $[rs_i, \infty]$ and $[0, dl_i]$;
2. adding a contingent constraint $\mathtt{S_i} \xrightarrow{[d_i:D_i]} \mathtt{E_i}$ for each task;
3. adding a free constraint $\mathtt{E_i} \xrightarrow{[\delta_{ij},\Delta_{ij}]} \mathtt{S_j}$ for each arc (a_i, a_j) in the Task Graph.

Where we recall d_i/D_i is the minimum/maximum duration for task t_i and δ_{ij}/Δ_{ij} is the minimum/maximum time lag for arc (t_i, t_j). Figure 2B shows the temporal model for the project graph in Figure 1 (reported again to ease the reader). For each activity t_i in the graph, start/end event variables (resp. $\mathtt{S_i}, \mathtt{E_i}$) are linked by contingent constraints (solid arcs) with $a = d_i$ and $b = D_i$. Free constraints are used to represent inter task precedence relations (dotted arcs); in the example we have $a = 2, b = \infty$ and $a = 0$ and $b = \infty$ for all other arcs. Due to duration uncertainty, start and end event variables ($\mathtt{S_i}, \mathtt{E_i}$) never become bound at search time.

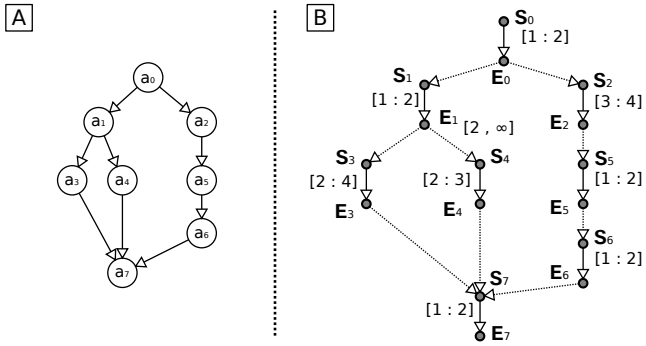

Figure 2: Temporal model for the Problem of Figure 1; time lags are $[0, \infty]$ when not specified.

We are interested in maintaining dynamic controllability, which ensures all deadlines and time lags can be met at run time. More precisely, a network is dynamically controllable if and only if we can take successive decisions such that, at any point of time, the partial sequence executed so far is ensured to extend to a complete solution, whatever durations remain to be observed. A formal definition is provided by [Vidal and Fargier, 1999].

Time Windows and Constraint Propagation The time window of each event variable $\mathtt{T_i}$ is specified by means of four values, namely $s_p(\mathtt{T_i})$, $s_o(\mathtt{T_i})$, $e_o(\mathtt{T_i})$, $e_p(\mathtt{T_i})$. Values $s_p(\mathtt{T_i})$ and $e_p(\mathtt{T_i})$ delimit the so-called *possible span* and specify the time interval where the event τ_i has some chance to take place at run time. Conversely, values $s_o(\mathtt{T_i})$ and $e_o(\mathtt{T_i})$ bound the so-called *obligatory span*; if an event is forced to occur out of its *obligatory span*, dynamic controllability is compromised. In general dynamic controllability holds if and only if $s_p \leq s_o \leq e_o \leq e_p$.

For a free time point variable (not involved in any precedence constraints), we have $s_p = s_o$ and $e_o = e_p$; user defined release times and deadlines directly constrain s_p and e_p values, while s_o, e_o are only indirectly affected (i.e. to maintain $[s_o, e_o] \subseteq [s_p, e_p]$).

Precedence constraints generally modify time window values, as depicted in Figure 3. Values s_p and e_p delimit the region where each τ_i *may* occur at run-time: for example τ_1 (corresponding to variable $\mathtt{T_1}$) can first occur at time 10, if τ_0 occurs at 0 and the constraint has duration 10; similarly τ_2 can first occur at 20 as at least 10 time units must pass between τ_1 and τ_2 due to the precedence constraint. As for the upper bounds, note that τ_2 cannot occur after time

60, or there would be a value $\theta \in [10, 20]$ with no support in the time window of τ_3; conversely, τ_1 can occur as late as time 50, since there is *at least* a value $\theta \in [10, 20]$ with a support in the time window of τ_2. Consider now bounds on the obligatory region: note that if (for instance) τ_1 is *forced* to occur before time 20 the network is no longer dynamic controllable, as in that case the time span between τ_0 and τ_1 would not be sufficient. Similarly, τ_2 cannot be forced to occur later than time 60 or there would be a value $\theta \in [10, 20]$ such that the precedence constraint between τ_2 and τ_3 cannot be satisfied.

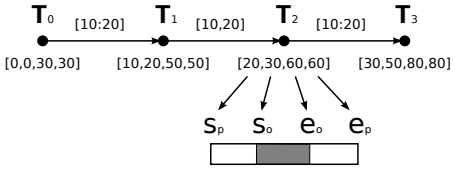

Figure 3: The 4 value time window of time event variables

Dynamic Controllability can be enforced by the iterative application of proper filtering rules for all free and contingent constraints, until a fix-point is reached. Alternatively, a propagation algorithm such as AC-3 can be used to speed up the process. The filtering rules for a free constraint $\mathtt{T_i} \xrightarrow{[a,b]} \mathtt{T_j}$ are:

$$s_p(\mathtt{T_j}) = \max\{s_p(\mathtt{T_j}), s_p(\mathtt{T_i}) + a\} \quad (1)$$
$$s_p(\mathtt{T_i}) = \max\{s_p(\mathtt{T_i}), s_p(\mathtt{T_j}) - b\} \quad (2)$$
$$s_o(\mathtt{T_j}) = \max\{s_o(\mathtt{T_j}), s_o(\mathtt{T_i}) + a\} \quad (3)$$
$$s_o(\mathtt{T_i}) = \max\{s_o(\mathtt{T_i}), s_o(\mathtt{T_j}) - b\} \quad (4)$$
$$e_o(\mathtt{T_j}) = \min\{e_o(\mathtt{T_j}), e_o(\mathtt{T_i}) + a\} \quad (5)$$
$$e_o(\mathtt{T_i}) = \min\{e_o(\mathtt{T_i}), e_o(\mathtt{T_j}) - b\} \quad (6)$$
$$e_p(\mathtt{T_j}) = \min\{e_p(\mathtt{T_j}), e_p(\mathtt{T_i}) + a\} \quad (7)$$
$$e_p(\mathtt{T_i}) = \min\{e_p(\mathtt{T_i}), e_p(\mathtt{T_j}) - b\} \quad (8)$$

In practice, $s_p(\mathtt{T_i}) + a$ is a the lower bound for $s_p(\mathtt{T_j})$, $s_p(\mathtt{T_j}) - b$ is the lower bound for $s_p(\mathtt{T_i})$ and so on. Note that free constraints apply the same kind of filtering both to the possible and the obligatory span of $\mathtt{T_i}$ and $\mathtt{T_j}$; the distinction between the two spans becomes relevant only when contingent constraints are taken into account.

Note that pruning a value from a time window may be a consequence of the value not being possible (e.g. rule (1)) or not being allowed (e.g. rule (2)). Figure 4A/B gives a pictorial intuition of the rules. Dynamic Controllability on a contingent constraint $\mathtt{T_i} \xrightarrow{[a:b]} \mathtt{T_j}$ is enforced by application of a second set of rules:

$$s_p(\mathtt{T_j}) = \max\{s_p(\mathtt{T_j}), s_p(\mathtt{T_i}) + a\} \quad (9)$$
$$s_p(\mathtt{T_i}) = \max\{s_p(\mathtt{T_i}), s_p(\mathtt{T_j}) - a\} \quad (10)$$
$$s_o(\mathtt{T_j}) = \max\{s_o(\mathtt{T_j}), s_o(\mathtt{T_i}) + b\} \quad (11)$$
$$s_o(\mathtt{T_i}) = \max\{s_o(\mathtt{T_i}), s_o(\mathtt{T_j}) - b\} \quad (12)$$
$$e_o(\mathtt{T_j}) = \min\{e_o(\mathtt{T_j}), e_o(\mathtt{T_i}) + b\} \quad (13)$$
$$e_o(\mathtt{T_i}) = \min\{e_o(\mathtt{T_i}), e_o(\mathtt{T_j}) - b\} \quad (14)$$
$$e_p(\mathtt{T_j}) = \min\{e_p(\mathtt{T_j}), e_p(\mathtt{T_i}) + b\} \quad (15)$$
$$e_p(\mathtt{T_i}) = \min\{e_p(\mathtt{T_i}), e_p(\mathtt{T_j}) - b\} \quad (16)$$

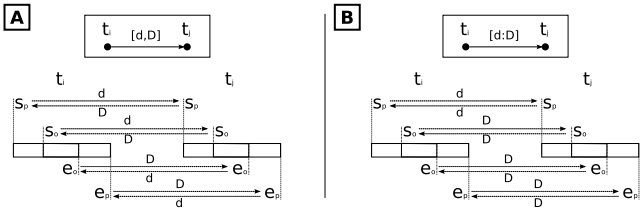

Figure 4: A) Dynamic Controllability filtering for a free constraints; B) Dynamic Controllability filtering for a contingent constraints;

Rule (9) is analogous to rule (1) for free constraints, but the duration is not under user control in this case.

Filtering and propagation of free and contingent constraints has the same (asymptotic) complexity of traditional precedence constraints (with fixed durations); moreover, the temporal model allows testing *in constant time* whether a contingent or free constraint can be consistently added to the current network; this is done by checking if the application of the corresponding filtering rules would violate the condition $s_p \leq s_o \leq e_o \leq e_p$.

4.2 Search strategy

One of the key issue with complete search based on MCS branching is how to detect and choose the Critical Set to branch on; in fact, the number of MCS is in exponential in the size of the graph and complete enumeration incurs the risk of combinatorial explosion. We propose to detect possible Critical Sets by solving a minimum flow problem on a specific resource r_k, as described by [Martin Golumbic, 2004]. As a major advantage, the method has polynomial complexity; note however the identified conflict set is not guaranteed to be minimal, nor to be well suited to open a choice point; hence we add a simple conflict minimization step. An overview of the adopted search strategy is shown in Algorithm 1. In the next section each of the steps will be described in deeper detail; the adopted criterion to evaluate the quality of a Critical Set will be given as well.

Critical Set Detection In first place, observe that, if the problem contains a *minimal* Critical Set, it contains a non (necessarily) minimal Critical Set as well. Therefore we can check the existence of an MCS on a given resource r_k

Algorithm 1 Overview of the search strategy

```
1: set best MCS so far (let this be "best") to ∅
2: for r_k ∈ R do
3:     find a conflict set S by solving a minimum flow problem
4:     if weight of S is higher than cap(r_k) then
5:         refine S to a Minimal Critical Set S'
6:         if S' is better than the best MCS so far then
7:             best = S'
8: if best = ∅ then
9:     the problem is solved
10: else
11:     open a choice point branching on possible resolvers of best
```

by checking the existence of any CS. Moreover, as the activities in a CS must have the possibility to overlap, they always form a stable set (or independent set) on the augmented project graph, further annotated with all precedence constraint which can be detected by time reasoning; if we assign to each activity the requirement $req(t_i, r_k)$ as a weight, a stable set S is a CS iff $\sum_{a_i \in S} req(a_i, r_k) > cap(r_k)$. We refer to such weighted graph as *resource graph* $\langle A, E_R \rangle$, where $(a_i, a_j) \in E_R$ iff $a_i \preceq a_j$ or $e_p(\mathtt{E_i}) \leq s_p(\mathtt{S_j})$. We can therefore check the existence of a MCS on a resource r_k by finding the maximum weight independent set on the resource graph and checking its total weight; this amounts to solve the following ILP model P':

$$\begin{aligned} \mathbf{P'}: \max & \sum_{a_i \in A} req(a_i, r_t) x_i \\ \text{s.t.} & \sum_{a_i \in \pi_j} x_i \leq 1 \quad \forall \pi_j \in \Pi \\ & x_i \in \{0, 1\} \end{aligned} \tag{17}$$

$$\begin{aligned} \mathbf{P''}: \min & \sum_{\pi_j \in \Pi} y_j \\ \text{s.t.} & \sum_{a_i \in \pi_j} y_j \geq req(a_i, r_t) \quad \forall a_i \in A \\ & y_j \in \{0, 1\} \end{aligned} \tag{18}$$

where x_i are the decision variables and $x_i = 1$ iff activity a_i is in the selected set; Π is the set of all paths in the graph (in exponential number) and π_j is a path in Π. As for the constraints (17) consider that, due to the transitivity of temporal relations, a clique on the resource graph is always a path from source to sink. In any independent set no two nodes can be selected from the same clique, therefore, no more than one activity can be selected from each path π_j in the set Π of all graph paths.

The corresponding dual problem is P'', where each variable y_j has value 1 if path π_j is selected; that is, finding the maximum weight stable set on a transitive graph amounts to find the minimum set of source-to-sink paths such that all nodes are covered by a number of paths at least equal to their requirement (constraints (18)). Note that, while the primal problem features an exponential number of constraints, its dual has an exponential number of variables. One can however see that the described dual is equivalent to route the least possible amount of flow from source to sink, such that a number of minimum flow constraints are satisfied; therefore, by introducing a real variable f_{ij} for each edge in E_R, we get:

$$\begin{aligned} \min & \sum_{a_j \in E^+(a_0)} f_{0j} \\ \text{s.t.} & \sum_{a_j \in E^-(a_i)} f_{ji} \geq req(a_i, r_t) \quad \forall a_i \in A \end{aligned} \tag{19}$$

$$\sum_{a_j \in E^-(a_i)} f_{ji} = \sum_{a_j \in E^+(a_i)} f_{ij} \quad \forall a_i \in A \setminus \{a_0, a_{n-1}\} \tag{20}$$

$$f_{ij} \geq 0$$

where $E^+(a_i)$ denotes the set of direct successors of a_i and $E^-(a_i)$ denotes the set of direct predecessors. One can note this is a flow minimization problem. Constraints (19) are the same as constraints (18), while the flow balance constraints (20) for all intermediate activities are implicit in the previous model. The problem can be solved starting for an initial feasible solution by iteratively reducing the flow with the any embodiment of the inverse Ford-Fulkerson's method, with complexity $O(|E_R| \cdot \mathcal{F})$ (where $\mathcal{F}$ is the value of the initial flow). Once the final flow is known, activities in the source-sink cut form the maximum weight independent set.

In our approach we solve the minimum flow problem by means of the Edmond-Karp's algorithm. On this purpose each activity a_i has to be split into two subnodes a'_i, a''_i; the connecting arc (a'_i, a''_i) is then given minimum flow requirement $req(a_i, r_k)$; every arc $(a_i, a_j) \in E_R$ is converted into an arc (a''_i, a'_j) and assigned minimum flow requirement 0. If the maximum independent set weight exceeds $cap(r_k)$, then a CS has been identified.

An initial solution is computed *at the root node* by a very simple algorithm; during search, the minimum flow at a search node is used to prime the method in all the children. This usually provides a very good starting flow for the Edmond-Karp's algorithm and improves performance (since the complexity depends on the initial flow value $\mathcal{F}$).

Reduction to MCS Once a critical set has been identified, a number of issues still have to be coped with; namely (1) the detected CS is not necessarily minimal and (2) the detected CS does not necessarily yield a good choice point. Branching on non-minimal CS can result in exploring unnecessary search paths.

We tackle both issues by applying a simple greedy minimization procedure; namely, the non-minimal set (let this be S) coming from the solution of the minimum flow problem is reduced by iteratively removing the activity yielding the best CS (according to a user specified heuristics). The procedure runs in time $O(|S|^2)$.

At each step of the iterative minimization process, we choose the CS with the lowest overall *preserved space*; this is an estimate of the amount of search space remaining after the addition of a new precedence relation (or *resolver*); the measure was introduced by [Laborie, 2005]. Here we distinguish between preserved *possible* space and preserved *obligatory* space; in particular, we choose the CS with the lowest sum of preserved obligatory space for its resolvers; the preserved possible space is used to break ties. Resolvers which cannot be posted do no contribute to the overall preserved space; those can be easily determined thanks to the temporal model, which allows to check in constant time whether a new constraint can be added.

Opening a choice point Once a MCS is selected, the next step is to open a choice point. Let $RS = (a_{i_0}, a_{j_0}), \ldots (a_{i_{m-1}}, a_{j_{m-1}})$ be the list of pairs of nodes in the set such that a precedence constraints can be posted.

		FLW solver						ENM solver						TO better	
	nodes	time	#MCS	tmcs	time/mcs	TO	BC/WC	time	#MCS	tmcs	time/mcs	TO	BC/WC	FLW	ENM
Platform A	41-49	0.08(0.26)	68	0.02	0.0002	0	0.50	0.06(0.48)	36	0.06	0.0014	0	0.50	0(0%)	0(0%)
	56-66	0.17(1.79)	207	0.11	0.0004	0	0.46	0.15(34.69)	268	3.31	0.0023	0	0.45	0(0%)	0(0%)
	75-82	0.26(3.81)	290	0.16	0.0004	0	0.48	0.26(1.42)	120	0.25	0.0017	0	0.48	0(0%)	0(0%)
	93-103	1.21(8.37)	460	0.61	0.0007	0	0.52	1.06(300.00)	2940	31.61	0.0035	3(2)	0.39	1(6%)	0(0%)
	110-119	3.08(28.18)	1004	2.30	0.0020	0	0.48	118.31(300.00)	6911	106.12	0.0878	3(1)	0.43	0(0%)	0(0%)
	118-128	1.60(300.00)	10677	10.60	0.0006	2	0.47	2.81(300.00)	1572	27.65	0.0043	1	0.47	0(0%)	1(0%)
Platform B	29-36	0.03(0.09)	19	0.00	0.0001	0	0.48	0.04(0.20)	17	0.03	0.0014	0	0.49	0(0%)	0(0%)
	41-52	0.05(0.10)	19	0.01	0.0004	0	0.46	0.06(5.27)	21	0.53	0.0139	0	0.46	0(0%)	0(0%)
	54-60	0.07(0.08)	23	0.01	0.0002	0	0.48	0.08(0.27)	24	0.04	0.0015	0	0.49	0(0%)	0(0%)
	65-78	0.11(0.30)	40	0.03	0.0006	0	0.43	0.30(140.31)	56	14.80	0.1720	0	0.44	0(0%)	0(0%)
	78-86	0.23(1.46)	154	0.10	0.0009	0	0.46	3.00(36.24)	306	5.69	0.0293	0	0.46	0(0%)	0(0%)
	86-96	0.23(1.09)	89	0.04	0.0005	0	0.42	0.83(3.61)	93	1.22	0.0125	0	0.42	0(0%)	0(0%)

Table 1: Results on the first group of instances (growing number of nodes) for Platform A and B

Then the choice point can be recursively expressed as:

$$CP(RS) = \begin{cases} \texttt{post}(a_{i_0}, a_{j_0}) \text{ if } |RS = 1| \\ \texttt{post}(a_{i_0}, a_{j_0}) \\ \quad \vee [\texttt{forbid}(a_{i_0}, a_{j_0}) \wedge CP(RS \setminus (a_{i_0}, a_{j_0}))] \end{cases}$$

where (a_{i_0}, a_{j_0}) always denotes the first pair in the sequence being processed. The operation $\texttt{post}(a_{i_0}, a_{j_0})$ amounts to add the constraint $e_i \xrightarrow{[0,\infty]} s_j$ in the time model, and $\texttt{forbid}(a_{i_0}, a_{j_0})$ consists in adding $s_j \xrightarrow{[1,\infty]} e_i$ (strict precedence relation). Prior to actually building the choice point, all precedence constraints are sorted by *increasing* preserved space (namely, preserved obligatory space, while the preserved possible space is used to break ties).

5 Experimental results

The described approach was implemented on top of ILOG Solver 6.7; we compare two "flavors" of the proposed method, sharing the same kind of temporal reasoning and propagation, but adopting different search strategies.

The first approach (referred to as FLW) performs Critical Set detection via the minimum flow method described in the paper; in the second approach (referred to as ENM) Minimal Critical Sets are detected with the enumeration based procedure provided by [Laborie, 2005]. Unlike FLW, the ENM method always identifies the set with the lowest preserved space (see Section 4.2); as a drawback, the enumeration can be time-expensive. The resolver simplification procedure described by [Laborie, 2005] is not applied here, as it was found to produce non-consistently better results. In both cases timetable, precedence graph and balance filtering were applied for all resources.

We performed tests on RCPSP instances derived from a system design problem. Given a computer application (described as a graph) and a target platform, the problem consists in finding a schedule guaranteed to meet a global deadline constraint; this is a very relevant issue in the design of real time systems.

Nodes in the application graph denote tasks/processes or data communication activities; each node has a priori unknown duration, bounded by a worst case and a best case execution time. We considered two target platforms; the first one (Platform A) features 16 processors (resources with capacity 1) and 32 communication devices (resources with capacity 10); the second platform has four 4-threaded processors (resources with capacity 4) and 8 communication devices. Both systems are representative of quite advanced hardware. Each task requires a processors and each data communication activity requires up to 9 units of two communication devices; both the processor and the communication channel to be used must be specified by the user prior to the scheduling process.

The testbench consists of synthetically generated instances; the random generator was devised to mimic the structure of real world applicationsThe generator, the instances and some solution files are available at http://www.lia.deis.unibo.it/Staff/MicheleLombardi/: i.e. nested parallel blocks (parallel subgrahs, with a single source and a single sink node) and some arcs between the nested structures. In particular, we generated two groups of instances by scaling the number of nodes (Group 1) in the graph and the branching factor (Group 2), i.e. the number of children when a new parallel block is initiated (branching factor).

Each graph in the testbench is fist *mapped* to the target platform via a heuristic procedure (see [Luca Benini, Michele Lombardi, and Michela Milano, 2009]); basically, this assigns processors and communication channels to activities. The result is a pure RCPSP instance (with uncertain durations) and is the input for the scheduling approaches. Note the mapping process introduces several additional nodes to allow correct modeling of inter-task communications; this explains the different number of nodes reported on the two platforms. Mapped instances are then solved to optimality, by assuming as objective function the best achievable deadline; formally, the objective function is $\max_{a_i \in A} e_p(\mathtt{E_i})$ and corresponds to the completion time (makespan) in the worst case scenario. A time limit of 300 seconds was set on the whole test process; all experiments were run on an Intel Core2 Duo with 2GB of RAM.

Table 1 summarizes results on Group 1 on Platform A and Platform B; here the branching factor ranges from 3 to 5; each row refers to a set of ten instances, ordered by

			FLW solver						ENM solver						TO better	
	BF	nodes	time	#MCS	tmcs	time/mcs	TO	BC/WC	time	#MCS	tmcs	time/mcs	TO	BC/WC	FLW	ENM
Platform A	2-4	74-84	0.21(0.82)	169	0.09	0.0005	0	0.47	0.41(19.47)	168	2.20	0.0067	0	0.47	0(0%)	0(0%)
	3-5	78-84	0.42(1.13)	223	0.23	0.0008	0	0.50	1.69(300.00)	185	17.51	0.0628	1(1)	0.42	0(0%)	0(0%)
	4-6	78-88	1.23(3.20)	389	0.72	0.0018	0	0.48	125.75(300.00)	337	124.33	0.5958	3(1)	0.43	2(13%)	0(0%)
	5-7	79-90	1.54(10.24)	796	1.59	0.0008	0	0.46	10.41(300.00)	254	21.11	0.0600	2(2)	0.33	0(0%)	0(0%)
	6-8	83-95	4.81(300.00)	10799	4.81	0.0013	2	0.52	70.77(300.00)	12248	46.97	0.0415	4(3)	0.35	1(1%)	1(1%)
Platform B	2-4	55-62	0.08(0.10)	22	0.01	0.0004	0	0.44	0.10(1.09)	22	0.18	0.0063	0	0.44	0(0%)	0(0%)
	3-5	55-66	0.11(0.25)	48	0.03	0.0007	0	0.47	0.43(8.71)	42	1.89	0.0302	0	0.46	0(0%)	0(0%)
	4-6	57-69	0.14(0.42)	55	0.07	0.0010	0	0.46	2.51(74.46)	85	20.91	0.2097	0	0.47	0(0%)	0(0%)
	5-7	52-72	0.21(1.16)	114	0.18	0.0007	0	0.42	0.97(300.00)	59	20.97	0.2827	1(1)	0.36	0(0%)	0(0%)
	6-8	58-67	0.27(0.65)	124	0.13	0.0013	0	0.48	7.50(136.34)	222	38.25	0.3078	0	0.48	0(0%)	0(0%)

Table 2: Results on the second group of instances (growing branching factor) for Platform A and B

increasing graph size. The table reports the minimum and maximum number of nodes ("nodes") and the performance of each of the approaches. In detail, the "time" column shows the median solution time for each group (the maximum solution time for the group is reported in round brackets); "#MCS" is the mean number of choice points (Minimal Conflict Sets); "tmcs" and "time/mcs" respectively are the mean time (overall) spent for MCS identification and the mean time to identify a single conflict. Column "TO" is the number of reported timeouts; finally, "BC/WC" is the ratio between the application makespan in the best and in the worst case. Timed out instances are included in the statistics, with an exception; namely, in a few cases the enumerative procedure was not able to reach any feasible solution: those instances are discarded in the statistical figures and their number is reported between round brackets in the *TO* column.

As one can see, on the problem at hand the two approaches have comparable performance in terms of solution time; the FLW method is however much more stable, reporting fewer time-outs. As expected, the FLW solver is faster in MCS identification (around one order of magnitude improvement in the *time/mcs* column). Less obviously, the approach yields a comparable number of choice points (column *#MCS*), pointing out that the MCS identified by the min-flow based procedure are not so bad when used for branching. This was kind of unexpected, since the complete enumeration finds consistently better MCS according to the specified heuristics; one shall conclude that there is a little point in branching over the "best" MCS, as long as there is not a clear understanding of what makes a scheduling heuristics actually good. In order to validate this conjecture, a deeper investigation on different problem classes is planned as future research. The *BC/WC* provides an estimate of the flexibility of the provided solutions: the lower the best case/worst case ration, the higher the retained flexibility; however, the indicator tends to be less reliable in case of time outs, since a low ratio value can result as a consequence of a very large worst case makespan.

It is interesting to observe that, in most cases, MCS detection is *not* the main bottleneck for the ENM solver (propagation time plays a key role). The tricky point is that the enumerative conflict detection process exhibits poor *stability*, taking a very large amount of time in a few cases. This is the reason (e.g.) for all the timed out instances where the ENM solver was not able to find any feasible solution. Those cases are not taken into account in the computation of the performance figures, so that the mean conflict detection time (*tmcs*) and the mean time per MCS (*time/mcs*) fail to give some grasp on the stability issue; the number of timed out instances (and in particular the number of cases where no feasible solution was found) provides in this sense a better indicator.

Observe that instances on Platform B are way easier than those on Platform A; this is mainly a consequence of the effectiveness of the heuristic mapping process, which greatly reduces the number of conflicts to be resolved at schedule time on the 4-thread processor platform. Finally, the "TO better" columns report the number of cases where either of the two approaches provides a better result than the competitor ("better" means here that a tighter deadline can be met). The relative difference is reported between round brackets and the instances where ENM provided no solution are excluded.

Table 2 shows the same results for Group 2; here the original graphs always contain 40 nodes, plus the additional activities introduced by the mapping process; the branching factor spans the interval reported for each row in the column "BF". The amount of application parallelism has a strong impact on the performance; most notably, the higher the branching factor, the larger the number of possible resource, penalizing the ENM solver. As for instance Group 1, MCS detection time is a dominant factor whenever ENM reports a time-out or poor performance (as hinted by the 4-6 row on Platform A). In the cases when conflict identification is not a bottleneck, the ENM solver performs fewer backtracks (see the #MCS value), proving the preserved space heuristic is more effective in this case.

Finally, we performed some experiments on the j30 and j60 single-mode benchmarks from the PSPLIB (from [Kolisch, 1997]), modified to introduce uncertain durations; in particular, the worst case value for each activity a_i is the duration D_i from the original graph, while the best case value is $\max(1, 0.5 \cdot D_i)$. Detailed result tables are omitted due to lack of space, but are available on-line[1]; as a first remark, we observed the introduction of uncertain du-

[1] At http://www.lia.deis.unibo.it/Staff/MicheleLombardi/

rations makes the problem considerably more difficult; we identify the following main reasons: (1) the increased number of possible overlapping yields many more possible conflicts; (2) uncertain durations and the lack of precise start time assignments tend to result in large time windows and make resource filtering less effective. As a general trend, on the J30 benchmarks the ENM approach has better performance; in particular, the method is often slower, but reports fewer time-outs; this tend to confirm the effectiveness of the preserved space heuristics on the PSPLIB instances. Conversely, the number of conflicts in the modified J60 instances sets a severe challenge to the enumeration procedure, often preventing the ENM solver from finding any solution.

6 Conclusion

We proposed an efficient complete solver for facing Resource Constraint Project Scheduling with minimum and maximum time lags and variable durations. The main contributions are: an effective time model inherited by STNU, an efficient algorithm for conflict set detection and its encapsulation in a sophisticated search strategy. Current research is aimed at introducing objective functions, and in taking into account run time policies.

Acknowledgement

Many thanks to Valentina Cacchiani for the useful advices about algorithms for finding stable sets.

References

Brucker, P.; Drexl, A.; Mhring, R.; Neumann, K.; and Pesch, E. 1999. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research* 112(1):3 – 41.

F. Stork. 2000. Branch-and-bound algorithms for stochastic resource-constrained project scheduling. Technical Report Research Report No. 702/2000, Technische Universitat Berlin.

F. Stork. 2001. *Stochastic resource-constrained project scheduling*. Ph.D. Dissertation, Technische Universitat Berlin.

G. Igelmund, and F. J. Radermacher. 1983a. Algorithmic approaches to preselective strategies for stochastic scheduling problems. *Networks* 13(1):29–48.

G. Igelmund, and F. J. Radermacher. 1983b. Preselective strategies for the optimization of stochastic project networks under resource constraints. *Networks* 13(1):1–28.

Herroelen, W., and Leus, R. 2005. Project scheduling under uncertainty: Survey and research potentials. *European Journal of Operational Research* 165(2):289 – 306. Project Management and Scheduling.

J.C. Beck, and A.J. Davenport. 2002. A survey of techniques for scheduling with uncertainty. Available from http://www.eil.utoronto.ca/profiles/chris/gz/uncertainty-survey.ps.

Kolisch, R. 1997. PSPLIB - A project scheduling problem library OR Software - ORSEP Operations Research Software Exchange Program. *European Journal of Operational Research* 96(1):205–216.

Laborie, P., and Rogerie, J. 2008. Reasoning with conditional time-intervals. In *Proc. 21th International FLAIRS Conference (FLAIRS 2008)*, 555–560.

Laborie, P. 2005. Complete MCS-Based Search: Application to Resource Constrained Project Scheduling. In *IJCAI*, 181–186.

Luca Benini; Michele Lombardi; and Michela Milano. 2009. Robust non-preemptive hard real-time scheduling for clustered multicore platforms. In *Proc. of DATE09*.

Martin Golumbic. 2004. *Algorithmic Graph Theory And Perfect Graphs*. Elsevier, second edition edition.

Muscettola, N. 2002. Computing the envelope for stepwise-constant resource allocations. In *CP*, 139–154.

Policella, N.; Oddi, A.; Smith, S. F.; and Cesta, A. 2004a. Generating robust partial order schedules. In *CP*, 496–511.

Policella, N.; Smith, S. F.; Cesta, A.; and Oddi, A. 2004b. Generating Robust Schedules through Temporal Flexibility. In *Proc. of ICAPS*, 209–218.

Policella, N.; Cesta, A.; Oddi, A.; and Smith, S. F. 2007. From precedence constraint posting to partial order schedules: a csp approach to robust scheduling. *AI Commun.* 20(3):163–180.

R. H. Möhring; F. J. Radermacher; and G. Weiss. 1984. Stochastic scheduling problems I - General strategies. *Mathematical Methods of Operations Research* 28(7):193–260.

R. H. Möhring; F. J. Radermacher; and G. Weiss. 1985. Stochastic scheduling problems II - set strategies. *Mathematical Methods of Operations Research* 29(3):65–104.

Reyck, B. D., and Herroelen, W. 1998. A branch-and-bound procedure for the resource-constrained project scheduling problem with generalized precedence relations. *European Journal of Operational Research* 111(1):152 – 174.

Rolf H. Möhring, and Frederik Stork. 2000. Linear preselective policies for stochastic project scheduling. *Mathematical Methods of Operations Research* 52(3):501–515.

Vidal, T., and Fargier, H. 1999. Handling contingency in temporal constraint networks: from consistency to controllabilities. *J. Exp. Theor. Artif. Intell.* 11(1):23–45.