Casting Project Scheduling with Time Windows as a DTP

Angelo Oddi, Riccardo Rasconi and **Amedeo Cesta**

ISTC-CNR, Italian National Research Council, Italy

Abstract

This paper extends existing work on constraint-based scheduling for solving complex Resource Constrained Project Scheduling Problems. The main result of the paper is the reduction of the RCPSP/max problem to a Disjunctive Temporal Problem that allow customization of specific properties within a backtracking search procedure for makespan optimization where decision variables are *Minimal Critical Sets* (MCSs). In particular an algorithm is proposed whose branching strategy is able to deduce new constraints which explicitly represent infeasible or useless search paths, such additional information allows early pruning of alternatives and strongly improves the efficiency of the overall search procedure. The paper includes an experimental evaluation on a set of standard, quite challenging, benchmark problems giving an empirical validation of the effectiveness of the proposed ideas.

Introduction

In this paper we present a constraint-based procedure to solve instances of the Resource Constrained Project Scheduling Problem with Generalized Precedence Relations (RCPSP/max). This problem derives from a project management environment in which activities represent steps that must be performed to achieve completion of the project and are subject to partial order constraints that reflect dependencies on project progression. In particular, a *time window* constraint is imposed between a generic pair of activities which bounds the difference of the start times of the activities to be included in a interval of possible values. The presence of this kind of constraint makes the problem hard to solve; as demonstrated in (Bartusch, Mohring, and Radermacher 1988), both the optimization and the feasibility versions of the problem are *NP-hard*.

This paper draws on ideas from two existing research lines: (a) contraint-based resource-constrained scheduling (Cesta, Oddi, and Smith 1998; 2002; Laborie 2005) and (b) disjunctive temporal reasoning (Oddi and Cesta 2000; Armando, Castellini, and Giunchiglia 1999; Dechter, Meiri, and Pearl 1991; Stergiou and Koubarakis 2000; Tsamardinos and Pollack 2003) In particular, we elaborate from a basic backtracking search procedure for *makespan optimization* where decision variables are *Minimal Critical Sets* (MCSs). The algorithm proposes a new *branching strategy* able to deduce new constraints which explicitly represent infeasible or useless search paths; such additional information allows early pruning of alternatives and strongly improves the efficiency of the overall search procedure. The proposed procedure iteratively selects decision variables (MCSs) which represent minimal sets of activities requiring more of the resource capacity. Given the minimality of the conflicts (each proper subset is not a resource conflict) a capacity violation is resolved by posting a single precedence constraint between any two activities of the MCS. To solve the RCPSP/max problem, we repeatedly apply this core CSP by adding/retracting precedence constraint along a systematic backtracking procedure which stops when either an optimal solution is found or another stop condition is met.

The guiding idea of the current work is based on the observation that a single $MCS = \{a_1, a_2, \ldots, a_k\}$ represents disjunction of literals each representing a simple temporal problem among the start/end times of the activities $a_i \in MCS$ of the form $start(a_j) - end(a_i) \geq 0$ ($start(a_i) - end(a_j) \geq 0$). A solution of the problem is a set of precedence constraints which induce an ordering of the problem activities where all the MCSs are explicitly or implicitly solved. In other words, a RCPSP/max can be seen as a conjunction of disjuntive temporal formulas corresponding to MCSs of the form $x_1 - y_1 \leq r_1 \vee x_2 - y_2 \leq r_2 \vee \ldots \vee x_k - y_k \leq r_k$. Such problem is known as the Disjunctive Temporal Problem (DTP) (Stergiou and Koubarakis 1998; 2000). Given the reduction of RCPSP/max to DTP, many of the techniques developed for solving DTP instances can be reused within MCS-based search procedures like those described in (Cesta, Oddi, and Smith 2002; Laborie 2005).

The paper is organized as follows. It first defines the reference RCPSP/max problem and its representation. The central section describes the reduction of RCPSP/max to DTP and the core constraint-based search procedure. Then, the performance of the proposed algorithm is described and the most interesting experimental results explained. Some conclusions and a discussion on the future work end the paper.

The Scheduling Problem

The Resource Constrained Project Scheduling Problem (RCPSP) has been widely studied in Operations Research literature (see (Brucker et al. 1999) for a survey). RCPSP/max is a specific formulation of the basic problem underlying a number of scheduling applications (Neumann and Schwindt 1997) which is considered particularly difficult, due to the presence of temporal separation constraints (in particular maximum time lags) between project activities.

The RCPSP/max can be formalized as follows:

- a set V of n activities must be executed, where each activity j has a fixed duration d_j. Each activity has a start-time S_j and a completion-time C_j that satisfies the constraint $S_j + d_j = C_j$.
- a set E of temporal constraints exists between various activity pairs $\langle i, j\rangle$ of the form $S_j - S_i \in [T_{ij}^{min}, T_{ij}^{max}]$, called start-to-start constraints (time lags or *generalized precedence relations* between activities). [1]
- a set R of renewable resources are available, where each resource r_k has a integer capacity $c_k \geq 1$.
- execution of an activity j requires capacity from one or more resources. For each resource r_k the integer $rc_{j,k}$ represents the required capacity (or *size*) of activity j.

A schedule S is an assignment of values to the start-times of all activities in V ($S = (S_1, \ldots, S_n)$). A schedule is *time-feasible* if all temporal constraints are satisfied (all constraints $S_j - S_i \in [T_{ij}^{min}, T_{ij}^{max}]$ and $S_j + d_j = C_j$ hold). A schedule is *resource-feasible* if all resource constraints are satisfied (let $A(S,t) = \{i \in V | S_i \leq t < S_i + d_i\}$ be the set of activities which are in progress at time t and $r_k(S,t) = \sum_{j \in A(S,t)} rc_{j,k}$ the usage of resource r_k at that same time; for each t the constraint $r_k(S,t) \leq c_k$ must hold). A schedule is *feasible* if both sets of constraints are satisfied. The RCPSP/max optimization problem, then, is to find a feasible schedule with *minimum makespan* MK, where $MK(S) = max_{i \in V}\{C_i\}$.

RCPSP/max as a CSP

There are different ways to formulate the RCPSP/max problem as a *Constraint Satisfaction Problem* (CSP) (Montanari 1974). In particular, see (Cesta, Oddi, and Smith 1998) and many others, the problem can be treated as the one of establishing *precedence constraints* between set of activities that require the same resource, so as to eliminate all possible conflicts in resource use. Given a resource r, we call *conflict* each set of activities requiring r, which can mutually overlap and whose combined resource requirement is more than the resource capacity c,

[1]Note that since activity durations are constant values, end-to-end, end-to-start, and start-to-end constraints between activities can all be represented in start-to-start form.

Decision variables are called Minimal Critical Sets $MCS = \{a_1, a_2, \ldots, a_k\}$ or *forbidden minimal sets* (Bartusch, Mohring, and Radermacher 1988; Laborie and Ghallab 1995). An MCS represents resource conflics of minimal size (each subsets is not a resource conflict), which can be *resolved* by posting a single precedence constraint between any two of the competing activities in the MCS. In CSP terms, a decision variable is defined for each $MCS = \{a_1, a_2, \ldots, a_k\}$; which can take as values all the possible feasible precedence constraints $a_i \preceq a_j$ that can be imposed between each pair of activities in MCS. It is however well known that some of such values are superfluous. In particular, we can use the idea of *simplification* for a set of precedence constraints (or *resolvers*) for a given decision variable (MCS), as described in (Laborie 2005): the set of resolvers $Res(MCS) = \{pc_1, pc_2, \ldots, pc_k\}$ of a MCS can be simplified so as to remove those resolvers $pc^{\times} \in Res(MCS)$ for which there exists another resolver $pc^{+} \in Res(MCS)$ such that $pc^{\times} \Rightarrow pc^{+}$ given the current temporal network. In other words, if a precedence constraint $pc^{\times} = a_i \preceq a_j$ imposed between the pair of activities (a_i, a_j) induces also a precedence constraint $pc^{+} = a_i \preceq a_k$ or $pc^{+} = a_k \preceq a_j$, then $pc^{\times}$ can be removed.

To support the search for a consistent assignment to the set of decision variables (MCSs), for any RCPSP/max we define the *distance graph* G_d. The set of nodes V represents time points, that is, the origin point tp_0 (the reference point of the problem) together with the start and end time points, s_i and e_i, of each activity a_i. The set of edges E represents all the imposed temporal constraints, i.e., precedences and durations. All the constraints have the form $a \leq tp_j - tp_i \leq b$ and for each constraint specified in the RCPSP/max, there are two weighted edges in the graph $G_d(V,E)$. The first one is directed from tp_i to tp_j with weight b and the second one is directed from tp_j to tp_i with weight $-a$. The graph $G_d(V,E)$ corresponds to a *Simple Temporal Problem* and its consistency can be efficiently determined via shortest path computations (see (Dechter, Meiri, and Pearl 1991) for more details on the STP). Thus, a search for a solution of a RCPSP/max instance *can proceed by repeatedly adding new precedence constraints into* $G_d(V,E)$ and recomputing shortest path lengths to confirm that $G_d(V,E)$ remains consistent. Given a Simple Temporal Problem, the problem is consistent if and only if no closed paths with negative length (or negative cycles) are contained in the graph G_d.

Let $d(tp_i, tp_j)$ ($d(tp_j, tp_i)$) designate the shortest path length in graph $G_d(V,E)$ from node tp_i to node tp_j (node tp_j to node tp_i), the following constraint $-d(tp_j, tp_i) \leq tp_j - tp_i \leq d(tp_i, tp_j)$ holds (Dechter, Meiri, and Pearl 1991). Hence, the *minimal* allowed distance between tp_j and tp_i is $-d(tp_j, tp_i)$ and the maximal distance is $d(tp_i, tp_j)$. Given that d_{i0} is the length of the shortest path on G_d from the time point tp_i to the origin point tp_0 and d_{0i} is the length of the shortest path from the origin point tp_0 to the time point tp_i, the interval

$[lb_i, ub_i]$ of time values associated to the generic time variable tp_i is computed on the graph G_d as the interval $[-d(tp_i, tp_0), d(tp_0, tp_i)]$ (see (Dechter, Meiri, and Pearl 1991)). In particular, the two set of assignment values $S_{lb} = \{-d(tp_1, tp_0), -d(tp_2, tp_0), \ldots, -d(tp_n, tp_0)\}$ and $S_{ub} = \{d(tp_0, tp_1), d(tp_0, tp_2), \ldots, d(tp_0, tp_n)\}$ to the variables tp_i respectively represent the so-called *earliest-time solution* and *latest-time solution* for the given STP.

We observe that when each MCS is resolved (i.e., at least a precedence constraint is posted between a pair of activities $a_i, a_j \in MCS$ or it is a consequence of other imposed constraints) and the corresponding STP is consistent, then we can easily find a schedule S, i.e, an assignment of values to the start-times of all activities in V ($S = (S_1, \ldots, S_n)$), which satisfies both all the temporal and the resource constraints of the problem. In fact, the temporal constraints are satisfied, because the corresponding STP is consistent, and also the resource constraints are satisfied. To prove this last claim, let us suppose by contradiction that at least one resource r and an instant of time t exist, such that a set of activities mutually overlaps and the total requirement is greater the resource capacity c. This circumstance would correspond to a conflict and would contradict that all the minimal conflicts are indeed removed. Hence, when all the Minimal Critical Sets are removed then all the temporal feasible assignments to the start times represents a full solution of the problem. In the following, we will focus on the *earliest-time solution* of the problem.

Reducing an RCPSP/max to a DTP

The definition of the RCPSP/max as a CSP problem allows a reduction to the Disjunctive Temporal Problem (Stergiou and Koubarakis 2000). The Disjunctive Temporal Problem (DTP) involves a finite set of temporal variables $x_1, y_1, x_2, y_2 \ldots x_n, y_n$ ranging over the integers [2] and a finite set of constraints $C = \{c_1, c_2, \ldots c_m\}$ of the form $x_1 - y_1 \leq r_1 \vee x_2 - y_2 \leq r_2 \vee \ldots \vee x_k - y_k \leq r_k$, where r_i are integer numbers. A DTP is consistent if an assignment to the variables exists such that in each constraints $c_i \in C$ at least one disjunct $x_{ij} - y_{ij} \leq r_{ij}$ is satisfied. One way to check for consistency of a DTP (Oddi and Cesta 2000) consists of choosing one disjunct for each constraint c_i and see if the conjunction of the chosen disjuncts is consistent. It is worth observing that this is equivalent to extracting a "particular" STP (the Simple Temporal Problem defined in (Dechter, Meiri, and Pearl 1991)) from the DTP and checking its consistency. If the STP is not consistent another one is selected, and so on. Again, as observed in (Oddi and Cesta 2000), a DTP can be represented as a CSP problem, where each DTP constraint $c \in C$ represents a (meta) variable and the set of disjuncts represents variable's domain values $D_c = \{\delta_1, \delta_2, \ldots \delta_k\}$. A *meta*-CSP problem is consistent if exists at least an element S (solution) of the set $D_1 \times D_2 \times \ldots \times D_m$ such that the corresponding set of disjuncts $S = \{\delta_1, \delta_2, \ldots \delta_m\}$ $\delta_i \in D_i$ is temporally consistent. Each value $\delta_i \in D_i$ represents an inequality of the form $x_i - y_i \leq r_i$ and a solution S can be represented as a labeled graph $G_d(V_S, E_S)$ called "distance graph" (Dechter, Meiri, and Pearl 1991). The set of nodes V_S coincides with the set of DTP variables $x_1, y_1, x_2, y_2 \ldots x_n, y_n$ and each disjunct $x_i - y_i \leq r_i$ is represented by a direct edge (y_i, x_i) from y_i to x_i labeled with r_i. Again the graph $G_d(V_S, E_S)$ represents a Simple Temporal Problem and we can refer to the properties described in the above section.

A RCPSP/max can be reduced to a DTP as it follows.

– The set of time points of the RCPSP/max, that is, the origin point tp_0 (the reference point of the problem) together with the start and end time points, s_i and e_i, of each activity a_i) represent the variables of the DTP.

– For each RCPSP/max constraint of the form $a \leq x_i - y_i \leq b$ (representing duration of activities or start-start windows constraints), there are two constraints without disjunction, $x_i - y_i \leq b$ and $y_i - x_i \leq -a$, added to the set C.

– for each resource r_k, each Minimal Critical Set $MCS = \{a_1, a_2, \ldots, a_k\}$ corresponds to a DTP constraint $c = \{pc_1, pc_2, \ldots, pc_m\}$, such that pc_k is a feasible precedence constraints of the form $S_i + d_i \leq S_j$ between a generic pair of activities $a_i, a_j \in MCS$.

On the basis of the above reduction, MCS-based algorithms for solving RCPSP/max instances, like the ones described in (Cesta, Oddi, and Smith 2002; Laborie 2005), can be seen as a meta-CSP based search procedure, a la (Oddi and Cesta 2000), which uses the same heuristics and pruning techniques for the original DTP search procedure. In the next section we see how to include within a complete MCS-based search algorithm for makespan optimization the idea of *semantic branching* as used in (Armando, Castellini, and Giunchiglia 1999).

The MCS-based Search

Figure 1 shows a non-deterministic version of a complete MCS-based procedure (similarly to (Laborie 2005)) which starts from a scheduling problem P and an empty solution S, and basically executes three steps: (a) the current partial solution is checked for consistency (Step 1) by the function *CheckConsistency*. If the partial solution is a complete solution (Step 2), that is an earliest-time assignment of values to the start-times of all activities ($S = (S_1, \ldots, S_n)$) – which satisfies both all the temporal and the resource constraints of the problem exits, then the algorithm exits. If the solution is still incomplete the following two steps are executed; (b) a (meta) variable (a Minimal Conflict Set MCS) is selected at Step 5 with a variable ordering heuristic; (c) a precedence constraint pc is chosen (Step 6) and added to S (represented at the lower level as a G_d graph). Hence the solver is recursively called (Step 7) on the partial updated solution $S \cup \{pc\}$.

[2] In the original definition of the DTP variables ranges over real values, however we make this restrictive hypothesis because RCPSP/max variables range over integers

```
MCS-Search($P$, $S$)
1.  if CheckConsistency($S$)
2.    then if IsaSolution($S$)
3.           then return($S$)
4.         else begin
5.           $MCS \leftarrow$ SelectVariable($P$)
6.           $pc \leftarrow$ ChooseValue($P$, $MCS$)
7.           MCS-Search($P$, $S \cup \{pc\}$)
8.         end
9.  else return(Fail)
10. end
```

Figure 1: A MCS-based solver for RCPSP/max

The *CheckConsistency* function is the core of the CSP algorithm as it keeps the set of distances $d(x_i, y_j)$ constantly updated performing the necessary temporal propagations; every time a new precedence constraint pc is added to the G_d graph, the set of distances $d(x_i, x_j)$ are updated through a $O(n^2)$ algorithm, where n is the number of time points. The distances $d(x_i, x_j)$ are used to discover *unsolvable* MCSs; a MCS is unsolvable when none of the possible resolvers can be consistently posted. In particular, a generic precedence constraints $pc = a_i \preceq a_j$ can be posted in a partial solution when $d(e_i, s_j) \geq 0$, where e_i and s_j are respectively the end-time of the activity a_i and the start-time of a_j. Hence, as soon as an unsolvable MCS is discovered, the search stops and returns a failure. It is worth noting that in order to maintain a polynomial complexity, only a subset of the possible MCSs are analyzed, following the same sampling strategy used within the *SelectVariable* step (Steps 5 of Figure 1) explained below. This procedure, together with the other step *ChooseValue* (Steps 6 of Figure 1) is used to guide the search according to heuristic estimators, as briefly explained below (for further details the reader can refer to (Cesta, Oddi, and Smith 2002)).

SelectVariable. The MCS selection is a polynomial procedure which works as follows. Given a generic resource r_k, it relies on the notion of conflict or *peak*. A peak is a set of activities requiring r_k, which can mutually overlap and the combined resource requirement is more than the resource capacity c_k. For each peak, a set of MCSs related to the given peak is sampled; then, the candidate MCSs are ranked according to the temporal flexibility they contain (a function of the degree to which constituent activities can be reciprocally shifted in time), and finally, one MCS is chosen according to its degree of criticality (the less flexibility a MCS has, the more critical it is). The previous steps are better clarified below.

The MCS sampling procedure employed in this work attempts to sample MCSs with the smallest possible cardinality and it is driven by two parameters, namely δ and s_f, which respectively control the cardinality (number of activities) in any sampled MCS and the overall number of MCSs that are sampled and returned. In particular, given a peak P, let M_P be the set of all MCSs in P and m_P the minimal cardinality of any MCS in M_P. M_P can be at most partitioned in $(c_k + 1) - m_P + 1$ subsets of MCSs with identical cardinality. $M_P = M_0 \cup M_1 \cup \ldots \cup M_{(c_k+1)-m_P}$, where M_0 is the set of MCSs with the minimum cardinality, and in general M_δ, with $(0 \leq \delta \leq (c_k + 1) - m_P)$, is the set of MCSs with cardinality minimal plus δ. The MCSs in $M_P = M_0 \cup M_1 \cup \ldots \cup M_{(c_k+1)-m_P}$ that satisfy the δ parameter are sampled lexicographically.

For example, let r_k be a resource with capacity $c_k = 7$, and $P = \{a_1[5], a_2[3], a_3[3], a_4[2], a_5[1], a_6[1], a_7[1]\}$ be a sorted peak on resource r_k (values in square brackets represent resource requirements). The total order imposed on P by $\prec$ also induces a lexicographical order on the set M_P (the set of all the MCSs in P). This order is generated in a manner equivalent to sorting a set of English words into alphabetical order by the alphabetical order of their constituent letters[3].

If we choose a value $\delta = 1$ (implying that we want to sample MCSs with maximal cardinality $m_P + 1$; in this case since $m_P = 2$, maximal cardinality $2 + 1 = 3$), the order imposed on the activities in P induces the following lexicographical order within the set of MCSs contained in $M_0 \cup M_1$: $\{a_1[5], a_2[3]\} \prec \{a_1[5], a_3[3]\} \prec \{a_1[5], a_4[2], a_5[1]\} \prec \{a_1[5], a_4[2], a_6[1]\} \prec \{a_1[5], a_4[2], a_7[1]\} \prec \{a_2[3], a_3[3], a_4[2]\}$. Observe that the first elements in the sorted order are those MCSs which contain the fewest activities, and hence are likely to be good candidates as critical conflicts.

To lexicographically sample MCSs within a given peak P, a Depth-First Search procedure is iteratively applied until either all elements in the set $M_0 \cup M_1 \cup \ldots \cup M_\delta$ have been sampled or a maximum number of $s_f\,|P|$ elements have been collected, where $|P|$ is the number of activities that participate to the conflict P.

The sampled MCSs are therefore ordered according to their criticality by employing two different heuristic estimators. The first one is based on the evaluation of the temporal flexibility of each MCS; the second one combines the temporal flexibility with the evaluation of the *distance* between the MCS and the current partial solution. The former is called *TimeFlex* heuristic, the latter *Clustering* heuristic.

TimeFlex: the sampled MCSs are ordered according to their criticality by employing the heuristic estimator K suggested in (Laborie and Ghallab 1995) to assess the temporal flexibility of each MCS.

Given a candidate MCS and a set $\{pc_1 \ldots pc_k\}$ of precedence constraints that could be posted be-

[3]Specifically, assume that two generic MCSs $mcs_i = \{a_{i1}, a_{i2}, \ldots, a_{in}\}$ and $mcs_l = \{a_{l1}, a_{l2}, \ldots, a_{lm}\}$ are each represented as a string of elements a_i which are totally sorted according to $\prec$. Then the relation $mcs_i \prec mcs_l$ holds if an index k exists such that $a_{ip} = a_{lp}$ for $p = 1..(k-1)$ and $a_{ik} \prec a_{lk}$.

tween pairs of activities in the MCS, $K(\text{MCS})$ is defined as $K(\text{MCS})^{-1} = \sum_{i=1}^{k}(1 + commit(pc_i) - commit(pc_{min}))^{-1}$ where $commit(pc_i)$ ranges from 0 to 1 and estimates the loss in temporal flexibility as a result of posting constraint pc_i, and pc_{min} is the precedence constraint with the minimum value of $commit(pc)$. Note that $K(\text{MCS})$ ($K(MCS) \in (0,1]$) takes on its highest value of 1 in those cases where only one specific precedence constraint can be feasibly posted to resolve the conflict. In general, the closer an MCS is to being unresolvable, the higher the value of $K(\text{MCS})$. It is worth noting that an MCS that is "close to a unresolvable state" is one for which very few consistent activity start-time assignments remain (relative to other MCSs), so it represents a critical case to solve first. When choosing which MCS to solve next, we focus on the area closest to failure (i.e., the conflict selection heuristic *SelectVariable* chooses the MCS with the highest K value). The rationale here is that if the solution becomes further constrained by resolving a more temporally flexible MCS, the probability increases that less temporally flexible MCSs will eventually reach an unresolvable state.

Clustering: a generic decision variable MCS is evaluated as $h(MCS) = a_k K(MCS) + ds(MCS)$, a linear combination of the temporal flexibility $K(\text{MCS})$ and the distance $ds(MCS)$, where $a_k \in \mathbb{R}$ is a weight used to bias the contribution of the temporal flexibility in the overall evaluation of the MCS. The function $ds(MCS)$ returns an estimation of the *distance* between the current partial solution $S_p = \{p_1, pc_2, \ldots, pc_m\}$[4] and the MCS. In particular, if we consider the *last* δ_{CL} posted precedence constraints $S_p(\delta_{CL}) = \{pc_{m-\delta_{CL}+1}, \ldots, pc_m\}$, $1 \leq \delta \leq m$, let $A_p(\delta_{CL}) = \{a_i \in V : \exists p_k \in S_p(\delta_{CL}) | a_i \in p_k$[5]$\}$, then $ds(MCS) = |A_p(\delta_{CL}) \cap MCS|/|MCS|$; hence $ds(MCS) = 1$ when all the activities in MCS are in common with the already posted precedence constraints $S_p(\delta_{CL})$, while $ds(MCS) = 0$ when no activity is in common. The use of the value $ds(MCS)$ drives the selection process of the decision variables in a way that they tend to have activities in common, in this sense they form a *cluster*. Hence, precedence constraints tend to be posted within a cluster of critical MCSs, with the goal of promoting early discovery of infeasible orderings between the activities and reducing the branching factor in the search.

ChooseValue. The opposite reasoning applies in the case of value selection (which pair of activities to order and how within the selected MCS). In this case we attempt to retain as much temporal flexibility as possible. The conflict resolution heuristic implemented in the *ChooseValue* function simply chooses the activity pair whose separation constraint, when posted, preserves the highest amount of temporal flexibility (pc_{min} precedence constraint), by ordering the activities in a way similar to min-slack heuristics proposed in (Smith and Cheng 1993).

[4]A solution can be represented as the set of posted precedence constraints pc_i

[5]each $p_k = a_i \preceq a_j$ can be seen as the set of two activities $\{a_i, a_j\}$

Horizon constraint update. The non-deterministic procedure shown in Figure 1 is implemented as a depth-first search, which dynamically updates the horizon constraint $C_i \leq H$ imposed on the completion times of all the problem activities, $i = 1, 2, \ldots, n$. In particular, when the search start $H = a_H mk_0$, where a_H is a numeric coefficient (usually taking the value 3-5) and mk_0 is the makespan of the problem where the resource constraint are relaxed. During the search H is dynamically updated, as soon as a new solution S of the problem is discovered with makespan $mk(S)$, then H is set to the value $mk(S) - 1$, as soon as this constraints became feasible along the depth-first search process. Such additional constraint allows early pruning of alternatives and strongly improves the efficiency of the overall search procedure.

Extending MCS Search

Our extended version of CSP solver (called MCSS$^+$) integrates the *semantic branching* idea. This is a feature that in the SAT approach (Armando, Castellini, and Giunchiglia 1999) comes for free and that in the CSP temporal representation has to be explicitly inserted. It avoids to test again certain conditions previously proved inconsistent. The idea behind semantic branching is the following: let us suppose that the MCSS$^+$ algorithm builds a partial solution $S_k = \{pc_1, pc_2, \ldots, pc_k\}$ and a new decision variable is selected which has a disjunct set (precedence constraints) of two elements $\{pc^{'}, pc^{''}\}$. Let us suppose that the disjunct $pc^{'}$ is selected first and no feasible solution exists from the partial solution $S_k \cup \{pc^{'}\}$. In other words, each search path from the node $S_k \cup \{pc^{'}\}$ arrives to an infeasible state. In this case, the depth-first search process removes the decision $pc^{'}$ from the current solution and tries the other one $pc^{''}$. However, even if the previous computation is not able to find a solution, it demonstrates that with regard to the partial solution S_k no solution can contain the disjunct $pc^{'}$. If we simply try $pc^{''}$, we lose the previous information, hence, before trying $pc^{''}$, we add the condition $\neg pc^{'}$. That is, if $pc^{'} = a_i \preceq a_j$, we add the constraint $e_i - s_j \geq 1$ (e_i and s_j are respectively the end-time of a_i and the start-time of the activity a_j).

In addition, as we are performing an optimization procedure, we can extend the cases where we can impose the constraint $\neg pc^{'}$. In fact, let us suppose that from the partial solution $S_k \cup \{pc^{'}\}$ the best solution found has makespan $mk^{'}$. We observe that when we know a solution with makespan $mk^{'}$, we are only interested in finding a solution with makespan $mk^{''} < mk^{'}$. Hence, since from $S_k \cup \{pc^{'}\}$

the best solution found has makespan $mk^{'}$ and all the imposed constraints are monotonic, the only way to improve the makespan is to impose the constraint $\neg pc^{'}$ together with $pc^{''}$.

Experimental Evaluation

To evaluate the effectiveness of the MCS-based algorithms, the RCPSP/max benchmarks that have been chosen for the present investigation are taken from well known test sets available at `http://www.wior.uni-karlsruhe.de/LS_Neumann/Forschung/ProGenMax/rcpspmax.html` namely the `J30` generated with the project generator ProGen/max ((Kolisch, Schwindt, and Sprecher 1998)). The `J30` set is composed of 270 problem instances, and represents a rather challenging benchmark despite the relatively small size of each instance (30 activities and 5 multi-capacity resources). For the problem sets lower bounds on makespan are known for each instance (Heilmann and Schwindt 1997), providing a common reference point for measuring performance. In the repository, the currently best known makespan for each problem instance is also available.

On the `J30` we have run an intensive search test, targeted at evaluating the pruning capability of the proposed branching schema. Consequently, among the proposed evaluation criteria, the algorithms are compared with respect to the number of proven optimal solutions within a given amount of cpu time. In all the experiments reported in this paper, the initial maximum horizon `MaxH` is set to 5× $\texttt{mk}_0$, which is sufficiently large to quickly find a first solution. The experimental setup requires also the setting of the two parameters the δ and s_f of the MCS sampling strategy (see previous section on MCS Search). Following the conclusions in (Cesta, Oddi, and Smith 2002), we set the previous parameters with the following values: $\delta = 0$, $s_f = 1$. The algorithms have been implemented in CMU Common Lisp Ver. 20a and run on a AMD Phenom II X4 Quad 3.5 Ghz under Linux Ubuntu 9.0

Evaluation Criteria. Using the data available from the repository, we consider the following performance measures for the purposes of the entire comparative analysis:

– N_o – the number of optimal solutions found (i.e., solutions are proved to be optimal by means of a complete search).

– N_f – the number of problems solved to feasibility.

– N_i – the number of solutions that improve the current best-known makespans. This value highlights the cases where the algorithm performs as best.

– $\Delta_{LB}\%$ – the average relative percentage deviation from the known lower bound (infinite capacity solution).

– $\overline{N_{bk}}$ – the average number of backtracking steps performed after a first solution is found, or a first failure is encountered during the search.

Table 1: Results on J30 benchmark (270 instances), the number of feasible solutions $N_f = 185$ for all the strategies

Strategy	δ_{CL}	a_k	N_o	N_i	$\Delta_{LB}\%$	$\overline{N_{bk}}$	*cpu*
MCSS	-	-	123	7	31.00	41627.3	184.3
MCSS$_{CL}$	6	1	115	6	30.31	45661.1	204.7
	6	2	117	6	30.11	44949.5	198.7
	6	3	123	7	30.34	44134.6	192.4
	12	1	115	6	30.31	45557.3	204.7
	12	2	116	6	30.11	44652.3	198.9
	12	3	123	7	30.34	43793.2	193.0
MCSS$^+$	-	-	**149**	15	29.74	**10240.8**	**72.8**
MCSS$^+_{CL}$	6	1	144	15	29.12	14368.4	88.9
	6	2	148	15	29.04	13110.9	83.9
	6	3	148	**16**	28.96	12220.8	79.7
	12	1	144	15	29.12	14395.1	88.9
	12	2	148	15	29.04	13055.4	83.8
	12	3	148	**16**	**28.88**	12593.9	79.4
REF J30	-	-	120	-	28.18	-	-

– *cpu* – it yields a measurement of the average computational time used to find the solutions, expressed in seconds, from the loading operation of the problem instances to the output of the solutions.

Results. We compare four different variants of the procedure proposed MCS-based procedure, MCSS, MCSS$^+$, MCSS$_{CL}$ and MCSS$^+_{CL}$. MCSS is the basic and reference search procedure, which can be see as equivalent to the *Complete MCS-Based Search* described in (Laborie 2005) but does not include the resource propagation functionality of the ILOG-based implementation used in that paper. MCSS$^+$ is the first variant of the basic procedure. It basically includes the *semantic branching* technique. We have introduced two further variants of the above procedures, namely MCSS$_{CL}$ and MCSS$^+_{CL}$, which use the *Clustering* heuristic evaluator at Step 6 of the MCSS procedure in Figure 1. In particular, in the basic MCSS a generic decision variable MCS is evaluated on the basis of the K evaluator ($K(MCS) \in (0, 1]$), whereas in MCSS$_{CL}$ and MCSS$^+_{CL}$ we use the MCS evaluator $h(MCS) = a_k K(MCS) + ds(MCS)$ using the two parameters δ_{CL} (clustering locality) and a_k (time flexibility weight).

We propose a two-step empirical evaluation. A first set of results (see Table1) propose a broader analysis, where we evaluate the effects of the semantic branching and the use of the clustering heuristic evaluator. In a second set of empirical analyses (see Table2) we propose an extended and more selective analysis on the more effective strategies.

Table1 shows the experimental results on the four proposed MCS-based search procedures. As shown in Table 1 the best performances are obtained with the procedures that implement the semantic branching schema (MCSS$^+$ and MCSS$^+_{CL}$). In particular, we see how the semantic branching seems very effective in the MCSS$^+$ procedure, as it improves the highest number of proven optimal solutions without using any reference lower bound (except the infi-

Table 2: Extended results on J30 benchmark (270 instances) for the MCSS$^+$ and MCSS$^+_{CL}$ strategies

Strategy	δ_{CL}	a_k	N_o	N_f	N_i	$\Delta_{LB}\%$	$\overline{N_{bk}}$	cpu
MCSS$^+$	-	-	149	185	15	29.74	**10240.8**	**72.8**
MCSS$^+_{CL}$	6	1	144	185	15	29.12	14368.4	88.9
	6	2	148	185	15	29.04	13110.9	83.9
	6	3	148	185	16	28.96	12220.8	79.7
	6	4	**150**	184	**18**	29.11	11369.4	76.7
	6	5	**150**	185	16	29.45	10763.4	76.0
	6	6	149	185	16	29.39	10870.3	75.0
MCSS$^+_{CL}$	12	1	144	185	15	29.12	14395.1	88.9
	12	2	148	185	15	29.04	13055.4	83.8
	12	3	148	185	16	**28.88**	12593.9	79.4
	12	4	**150**	184	**18**	29.11	11518.0	76.6
	12	5	**150**	185	16	29.46	10807.1	75.9
	12	6	**150**	185	16	29.39	11012.8	74.9
MCSS$^+_{CL}$	24	1	144	185	15	29.12	14423.5	88.8
	24	2	148	185	15	29.04	13082.5	83.9
	24	3	148	185	16	28.92	12509.2	79.5
	24	4	**150**	184	**18**	29.11	11503.3	76.6
	24	5	**150**	185	16	29.45	10812.6	75.9
	24	6	149	185	16	29.39	10892.5	75.0
BESTS	-	-	**152**	185	**19**	28.31	-	-
REF J30	-	-	120	185	-	28.18	-	-

nite capacity solution) to 149 (MCSS$^+$) Vs. 123 (MCSS), strongly reducing both the number of backtracking steps (10240.8 Vs. 41627.3) and the average cpu time (72.8 Vs. 184.3 seconds). It is worth underscoring how all the experimental runs were performed imposing a time-out limit of 500 seconds. In the case the procedure exits before, it starts a new search from scratch imposing the best makespan found so far. The restart from scratch continues until a better makespan is found or the fixed time-out is reached.

The shown results are quite remarkable also in comparison with the best results published on the official web site, where it can be calculated a value $N_o = 120$ and $\Delta_{LB}\% = 28.18$ (reported in the last row of Table 1 with label REF J30). So, our procedure, without considering any reference lower bound (except the infinite capacity solution) is able to increase of 29 instances the number of optimal solutions. We also observe that the idea of clustering implemented through the evaluator $h(MCS) = a_k K(MCS) + ds(MCS)$ is effective. In fact, the combined use of the weighted heuristic evaluator $a_k K(MCS)$ and the distance value $ds(MCS)$ drives the selection process of the decision variables in a way that they tend to have activities in common (in this sense they form a *cluster*). Hence, precedence constraints tend to be posted within a cluster of *critical* MCSs, with respect to the temporal flexibility, with the effect of further promoting early discovery of infeasible orderings between the activities and reducing the branching factor in the search. In Table 2 we observe that the MCSS$^+_{CL}$ procedure yields the best Δ_{LB} value 28.88 Vs. the value 29.74 of MCSS, despite the slightly lower number of optimal solutions returned w.r.t. MCSS$^+$ (148 Vs. 149).

In Table 2 each problem is solved multiple times using different δ_{CL} and a_k values. Two types of results are reported: (1) the average result on each problem over different runs; (2) the best result obtained, computed using the best result on any single problem (the row with label *BESTS*). In addition, we report the current best results, available at `http://www.wior.uni-karlsruhe.de/LS_Neumann/Forschung/ProGenMax/rcpspmax.html` summarized in the last row with label REF J30 as in Table 1. We report these results as they are available on the above pointed web site, none of the referred algorithms was reimplemented.

The conclusion that can be drawn from Table 2 is that the MCSS$^+_{CL}$ procedure performs rather satisfactorily, as the best $\Delta_{LB}\%$ value 28.88 ($\delta_{CL} = 12$, $a_k = 3$) is quite close to the overall known best 28.18, and the best number of optimal solutions is $N_o = 150$ (e.g., $\delta_{CL} = 12$, $a_k = 4$)). Moreover, the overall number of improved solutions (see row with label BESTS) with respect to the best published is remarkable, as 19 solutions have been improved over a total of 185 feasible ones, and the overall number of optimal solutions found is 152.

Conclusions

This paper extends previous work on constraint-based scheduling for solving Resource Constrained Project Scheduling Problems with *time windows* (RCPSP/max) (Cesta, Oddi, and Smith 2002). It proposes a backtracking search procedure for makespan optimization where the decision variables are *Minimal Critical Sets* (MCSs), i.e., sets of activities participating to a resource conflict, such that any proper subsets no longer constitutes a conflict.

The main result of the paper is the reduction of the RCPSP/max problem to a Disjunctive Temporal Problem. This reduction allows the integration of techniques like the *Semantic Branching* in the currently employed CSP solver. The aim is to improve the search by avoiding to test again conditions that proved to be inconsistent at earlier stages of the search process.

The performance of the proposed procedure has been assessed on a well-known RCPSP/max benchmark sets, the J30 set used also by others. In particular, four different versions of the backtracking procedure have been tested, with and without the semantic branching integration, and by introducing two further variants of the above procedures that use a modified version (clustering) of the heuristic evaluator used to steer the selection of the variables of the CSP.

The experimental results are twofold: besides the fact that the proposed backtracking procedure yields results that are competitive with the current bests, the semantic branching integration shows to perform much better than the pure backtracking; In addition, the introduction of the clustering heuristic evaluator provides a further boosting to the algorithm's effectiveness. This last result is currently being the object of further investigation.

Acknowledgments

Authors are partially supported by CNR under project RSTL (funds 2007), EU under the ULISSE project (Contract FP7.218815), and MIUR under the PRIN project 20089M932N (funds 2008).

References

Armando, A.; Castellini, C.; and Giunchiglia, E. 1999. SAT-based Procedures for Temporal Reasoning. In *Proceedings 5th European Conference on Planning (ECP-99)*. (available at `http://www.mrg.dist.unige.it/~drwho/Tsat`).

Bartusch, M.; Mohring, R. H.; and Radermacher, F. J. 1988. Scheduling Project Networks with Resource Constraints and Time Windows. *Annals of Operations Research* 16:201–240.

Brucker, P.; Drexl, A.; Mohring, R.; Neumann, K.; and Pesch, E. 1999. Resource-Constrained Project Scheduling: Notation, Classification, Models, and Methods. *European Journal of Operations Research* 112(1):3–41.

Cesta, A.; Oddi, A.; and Smith, S. 1998. Profile Based Algorithms to Solve Multiple Capacitated Metric Scheduling Problems. In *AIPS-98. Proceedings of the* 4^{th} *International Conference on Artificial Intelligence Planning Systems*, 214–223.

Cesta, A.; Oddi, A.; and Smith, S. F. 2002. A constraint-based method for project scheduling with time windows. *J. Heuristics* 8(1):109–136.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49:61–95.

Heilmann, R., and Schwindt, C. 1997. Lower Bounds for RCPSP/max. Technical Report WIOR-511, Universität Karlsruhe.

Kolisch, R.; Schwindt, C.; and Sprecher, A. 1998. Benchmark Instances for Project Scheduling Problems. In Weglarz, J., ed., *Handbook on Recent Advances in Project Scheduling*. Kluwer.

Laborie, P., and Ghallab, M. 1995. Planning with Sharable Resource Constraints. In *Proceedings of the* 14^{th} *Int. Joint Conference on Artificial Intelligence (IJCAI-95)*.

Laborie, P. 2005. Complete mcs-based search: Application to resource constrained project scheduling. In *IJCAI*, 181–186.

Montanari, U. 1974. Networks of Constraints: Fundamental Properties and Applications to Picture Processing. *Information Sciences* 7:95–132.

Neumann, K., and Schwindt, C. 1997. Activity-on-Node Networks with Minimal and Maximal Time Lags and Their Application to Make-to-Order Production. *Operation Research Spektrum* 19:205–217.

Oddi, A., and Cesta, A. 2000. Incremental Forward Checking for the Disjunctive Temporal Problem. In Horn, W., ed., *ECAI2000. 14th European Conference on Artificial Intelligence*, 108–111. IOS Press.

Smith, S., and Cheng, C. 1993. Slack-Based Heuristics for Constraint Satisfaction Scheduling. In *Proceedings 11th National Conference on AI (AAAI-93)*.

Stergiou, K., and Koubarakis, M. 1998. Backtracking Algorithms for Disjunctions of Temporal Constraints. In *Proceedings 15th National Conference on AI (AAAI-98)*.

Stergiou, K., and Koubarakis, M. 2000. Backtracking Algorithms for Disjunctions of Temporal Constraints. *Artificial Intelligence* 120(1):81–117.

Tsamardinos, I., and Pollack, M. E. 2003. Efficient solution techniques for disjunctive temporal reasoning problems. *Artif. Intell.* 151(1-2):43–89.