

Project Scheduling as a Disjunctive Temporal Problem

Angelo Oddi and Riccardo Rasconi and Amedeo Cesta¹

Abstract. The main result of the paper is the reduction of the RCPSP/max problem to a Disjunctive Temporal Problem that allows customization of specific properties within a backtracking search procedure for makespan optimization. In addition, a branching strategy is proposed able to deduce new constraints which explicitly represent infeasible or useless search paths. A new variable ordering heuristic (called *clustering*) is also used which provides a further boosting to the algorithm's effectiveness.

1 INTRODUCTION

In this paper we present a constraint-based procedure to solve instances of the Resource Constrained Project Scheduling Problem with Time Windows (RCPSP/max). The problem is formalized as follows. A set V of n activities, where each activity j has a fixed duration d_j , a start-time S_j and a completion-time C_j that satisfy 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 $S_j - S_i \in [T_{ij}^{min}, T_{ij}^{max}]$ (start-to-start constraints). There are a set R of renewable resources, where each resource r_k has a integer capacity $c_k \geq 1$. The 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 of activity j . A *feasible* schedule S is an assignment of values to the start-times of all activities in V ($S = (S_1, \dots, S_n)$) that is both *time-feasible* (all temporal constraints are satisfied) and *resource-feasible* (each resource's c_k is exceeded at no time). The RCPSP/max optimization problem, then, is to find a feasible schedule with *minimum makespan* MK . Both the optimization and the feasibility versions of the problem are *NP-hard* [2]. This paper draws on ideas from two existing research lines: (a) constraint-based resource-constrained scheduling [3, 4] and (b) disjunctive temporal reasoning [5, 1, 6, 7]. It is organized as follows. It first describes the reduction of RCPSP/max to DTP. Next, it proposes an effective *branching strategy* able to deduce new constraints, which explicitly represent infeasible or useless search paths, and a new variable ordering heuristic, called *clustering*, which provides a further boosting to the algorithm's effectiveness. An experimental section ends the paper.

2 CASTING SCHEDULING AS A DTP

The RCPSP/max problem is formulated as a *Constraint Satisfaction Problem* (CSP) such that the problem can be treated as one of establishing *precedence constraints* between set of activities that require the same resource (e.g., see [3]), so as to eliminate all possible conflicts in resource use. Given a resource r , a *conflict* is a set of activities requiring r , which can mutually overlap and whose combined resource requirement is in excess of the resource capacity c . Decision variables are called Minimal Critical Sets $MCS = \{a_1, a_2, \dots, a_k\}$

or *forbidden minimal sets* [2, 4]. An MCS represents resource conflicts of minimal size (each subsets is not a resource conflict), which can be *resolved* by posting a single precedence constraint between two of the competing activities in the set MCS. In CSP terms, a decision variable is defined for each subset of activities $MCS = \{a_1, a_2, \dots, a_k\}$ which is a Minimal Critical Set and can take as values all the possible feasible precedence constraints $a_i \preceq a_j$ which can be imposed between each pair of activities in MCS. As it is well-known, given a MCS some of the values (or *resolvers*) are superfluous, in the following we use the idea of *simplification* (see [4] for more details).

The Disjunctive Temporal Problem (DTP) [6] involves a finite set of temporal variables $x_1, y_1, x_2, y_2 \dots x_n, y_n$ and a finite set of constraints $C = \{c_1, c_2, \dots c_m\}$ of the form $x_1 - y_1 \leq r_1 \vee x_2 - y_2 \leq r_2 \vee \dots \vee x_k - y_k \leq r_k$. 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. A RCPSP/max can be reduced to a DTP as it follows. (1) The set of time points of the RCPSP/max, that is, the origin point tp_0 (the reference point of the problem) together the start and end time points, s_i and e_i , of each activity a_i represent the variables of the DTP. (2) For each constraint of the form $a \leq x_i - y_i \leq b$ (duration of activities or start-start 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 $MCS = \{a_1, a_2, \dots, a_k\}$ corresponds to a DTP constraint $c = \{pc_1, pc_2, \dots, pc_m\}$, 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 [3, 4], can be seen (and enhanced) as a CSP based search procedure, a la [5], which uses the heuristics and pruning techniques for the original DTP search procedure.

3 EXTENDED MCS-SEARCH

Figure 1 shows a non-deterministic version of a complete MCS-based procedure (similarly to [4]) which starts from a scheduling problem P and basically executes three steps: (a) the current partial solution is checked for consistency (Step 1). If the partial solution S is a complete solution (Step 2), 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 (for further details on Steps 5 and 6 see [3]). Hence the solver is recursively called (Step 7) on the partial updated solution $S \cup \{pc\}$. Based of the reduction to DTP, we enhance the procedure of Figure 1 with two contributions:

Semantic branching. Exploiting this well-known technique in the search process of our MCS-based procedure basically entails the explicit addition of a negated constraint $\neg pc'$ to each partial solution

¹ ISTC-CNR, Via San Martino della Battaglia 44, 00185 Rome, Italy. Email: {angelo.odd, riccardo.rasconi, amedeo.cesta}@istc.cnr.it

```

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

```

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

S_k in all cases where every search path from the node $S_k \cup \{pc'\}$ is known to reach an infeasible state (see [1, 5] for details).

Clustering. The rationale behind this heuristic strategy is suggested by the structure of the DTP temporal constraints and is targeted on reducing the branching factor in the search. It drives the selection of the decision variables (MCSS) in a way that they tend to have activities in common, in this sense they form a *cluster*. Hence, the set of posted precedence constraints (resolvers) are more constrained among one another, and it is more likely to have other decision variables (MCSS) with a smaller number of resolvers. A decision variable MCS is evaluated as $h(MCS) = a_k K(MCS) + ds(MCS)$ and is chosen the decision variable with the maximal value h . Where a_k is a weight used to bias the contribution of the temporal flexibility estimator $K(MCS)^2$ and given the current partial solution $S_p = \{p_1, pc_2, \dots, pc_{m-\delta_{CL}+1}, \dots, pc_m\}^3$, $ds(MCS) \in [0, 1]$ is the fraction of activities contained in the last δ_{CL} posted resolvers (each $p_k = a_i \preceq a_j$ can be seen as the set $\{a_i, a_j\}$) respect to the set MCS . Hence, $ds(MCS) = 1$ when all the activities in MCS are in common with the last δ_{CL} already posted precedence constraints, while $ds(MCS) = 0$ when no activity is in common.

4 EXPERIMENTAL RESULTS

The RCPSP/max J30 benchmark that have been chosen for the present investigation is taken from the well known test sets available at http://www.wior.uni-karlsruhe.de/LS_Neumann/Forschung/ProGenMax/rcpspmax.html. It 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). In our experiments we consider the following performance measures: N_o – the number of proved optimal solutions found; N_f – the number of problems solved to feasibility; N_i – the number of solutions that improve the current best-known makespans; $\Delta_{LB}\%$ – 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; cpu – it yields a measurement of the average computational time (seconds) used to find the solutions. We compare the performances of three different variants of the MCSS basic reference search procedure⁴, $MCSS^+$, $MCSS_{CL}$ and $MCSS_{CL}^+$, respectively employing *semantic branching*, the *clustering* heuristic, and both. A preliminary analysis, not inserted here for the sake of space, shows how the best performances are obtained with the procedures that implement the semantic branching schema ($MCSS^+$ and $MCSS_{CL}^+$). In particular, $MCSS^+$ improves the highest number of proven optimal solu-

tions without using any reference lower bound (except the infinite capacity solution) to 149 ($MCSS^+$) Vs. 123 ($MCSS$), strongly reduces both the number of backtracking steps (10240.8 Vs. 41627.3) and the average cpu time (72.8 Vs. 184.3 seconds). Each problem

Table 1. Performance on the J30 set for the $MCSS^+$ and $MCSS_{CL}^+$

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	-	-

is solved multiple times by tuning the clustering factor (δ_{CL}) and the bias of the flexibility-based MCS selection heuristic K estimator (a_k). In Table 1 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 the above pointed URL summarized in the last row with label REF J30. The results in Table 1 show 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). Moreover, the $MCSS_{CL}^+$ procedure performs rather satisfactorily as the best $\Delta_{LB}\%$ value 28.88 ($\delta_{CL} = 12$, $a_k = 4$) is quite close to the overall known best 28.18, and the best number of optimal solutions is $N_o = 150$ (e.g., $\delta_{CL} = 12$, $a_k = 4$). Lastly, 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.

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

- [1] A. Armando, C. Castellini, and E. Giunchiglia, ‘SAT-based Procedures for Temporal Reasoning’, in *ECP-99*, (1999).
- [2] M. Bartusch, R. H. Mohring, and F. J. Radermacher, ‘Scheduling Project Networks with Resource Constraints and Time Windows’, *Annals of Operations Research*, **16**, 201–240, (1988).
- [3] A. Cesta, A. Oddi, and S.F. Smith, ‘A Constraint-Based Method for Project Scheduling with Time Windows’, *J. Heuristics*, **8**(1), 109–136, (2002).
- [4] P. Laborie, ‘Complete MCS-Based Search: Application to Resource Constrained Project Scheduling’, in *IJCAI-05*, (2005).
- [5] A. Oddi and A. Cesta, ‘Incremental Forward Checking for the Disjunctive Temporal Problem’, in *ECAI-00*, (2000).
- [6] K. Stergiou and M. Koubarakis, ‘Backtracking Algorithms for Disjunctions of Temporal Constraints’, *Artif. Intell.*, **120**(1), 81–117, (2000).
- [7] I. Tsamardinos and M. E. Pollack, ‘Efficient Solution Techniques for Disjunctive Temporal Reasoning Problems’, *Artif. Intell.*, **151**(1-2), 43–89, (2003).

² The closer an MCS is to being unresolvable, the higher the value of $K(MCS) \in [0, 1]$ (see [5] for details)

³ A solution is represented as a set of posted precedence constraints pc_i

⁴ $MCSS$ is equivalent to the *MCS-Based Search* described in [4], without the resource propagation functionality.