

Conflict-based repair techniques for solving dynamic scheduling problems

Abdallah Elkhyari¹, Christelle Guéret^{1,2}, and Narendra Jussien¹

¹ École des Mines de Nantes – 4, rue Alfred Kastler
F-44307 Nantes Cedex 3 – France
{aelkhyar, gueret, jussien}@emn.fr

² IRCCyN – Institut de Recherche en Communications et Cybernétique de Nantes

1 Introduction

Scheduling problems have been studied a lot over the last decade. Due to the complexity and the variety of such problems, most works consider static problems in which activities are known in advance and constraints are fixed. However, every schedule is subject to unexpected events (consider for example a new activity to schedule, or a machine breakdown). In these cases, a new solution taking these events into account is needed in a preferably short time and as close as possible to the current solution.

Constraint Satisfaction Problems (CSP) are increasingly used for solving scheduling problems. Many global temporal and resource constraints have been developed for solving such problems [1–3]. However, dynamic CSP (an extension of the CSP framework where the set of variables or/and constraints evolves throughout computation [4]) are not as spread. Notice that dynamic scheduling problems are seldom studied [5]. Two classical methods used to solve such problem are: recomputing a new schedule from scratch each time an event occurs (a quite time consuming technique) and constructing a partial schedule and completing it progressively as time goes by (like in on-line scheduling problems – this is not compatible with planning purposes).

Extending work on dynamic arc-consistency [6], the use of *explanations* (a set of constraint justifying solver actions) have been introduced for solving dynamic problems [7]. However, no application of such a technique have been made to scheduling problems.

In this paper, we introduce the integration of explanation capabilities within scheduling-related global constraints and show how using such techniques speeds up the solving of dynamic problems (compared to solving a series of static problems).

2 Solving dynamic problems using explanations

Solving dynamic problems requires incremental addition and retraction of constraints. Even though incremental constraint addition is naturally handled by modern constraint solver, incremental retraction of constraints is often helped

with recording trace/undo information [6, 7]. Such an information is used to determine past effects of removed constraint that need to be undone.

Explanations [7] are a generalization of that information. An explanation is a set of constraints that justifies an action of the solver (classically value removals) *i.e.* as long as each constraint that appears in the explanation remains active (not removed) the value removal (in our example) is valid (no valid solution can be build from this partial assignment). An explanation for the removal of value a from variable x is noted $\text{expl}(x \neq a)$.

Several explanations may exist for a given removal. Providing precise and short explanations for solver actions is a key issue to the interest of using explanation-based techniques within constraint programming: upon undoing past effects of a constraints only the necessary modifications are to be done. A good compromise between precision and ease of computation is to use the solver-embedded knowledge to provide interesting explanations [7].

Explanations can also be used to efficiently guide search. Indeed, classical backtracking-based searches only proceed by backtracking to the last choice point when encountering failures. Explanations can be used to improve standard backtracking and to exploit information gathered to improve the search: to provide intelligent backtracking, to replace standard backtracking with a jump-based approach *à la Dynamic Backtracking* [8], or even to develop new local searches on partial instantiations [9]. The www.e-constraints.net web site contains several pointers about explanations.

When considering global constraints, an easy way of providing explanations is to merely consider the whole set of variables involved in that constraint. However, efficient explanations need a more precise study of the algorithms used to propagate those constraints.

3 Explanations for scheduling-related global constraints

The Resource Constrained Project Scheduling Problem (RCPSP) is a general scheduling problem. It consists of a set of activities $A = \{1, 2, \dots, n\}$ and a set of renewable resources $R = \{1, \dots, r\}$. Each resource k is available in a given constant amount R_k . Each activity i has a duration p_i and requires a constant amount a_{ik} of resource k to be processed. Preemption is not considered. Activities are related by two sets of constraints: temporal constraints modelled through precedence constraints (*i.e.* mathematical binary relations), and resource constraints that state that for each time period and for each resource, the total demand cannot exceed the resource capacity. The objective considered here is the minimization of the makespan (total duration) of the project. This problem is *NP-hard* [10].

Providing explanation for temporal binary constraints is straightforward (consider the associated mathematical relations – see [7]). Explanations for resource management constraints are not that easy. It is necessary to study the algorithms used for propagation. Classical techniques for maintaining resource limitations

for the scheduling problems are: *core-times* [3], *task-interval* [1, 2] and *resource-histogram* [1, 2].

3.1 Resource-histogram constraints

The principle of the *resource-histogram* technique [1, 2] is to associate to each resource k an array $\text{level}(k)$ in order to keep a timetable of the resource requirements. This histogram is used for detecting a contradiction and reducing the time-window of activities.

The *core-times* technique [3] is used for computing lower bounds using a destructive method. A *core-time* $\text{CT}(i)$ is associated to each activity i . It is defined as the interval of time during which a portion of an activity is always executed whether it starts at its earliest or latest starting time. The lower bound of the schedule is obtained when considering only the *core-time* of each activity. If a resource-conflict is detected then some lower bounds can be upgraded.

Combining these two techniques provides an efficient resource-conflict detecting constraint. A timetable for each resource is computed as follows: for each activity i , its *core-time* $\text{CT}(i) = [f_i, r_i + p_i)$ (f_i is the latest starting time of i , r_i its earliest starting time) is computed and the amount a_{ik} of resource k for each time interval $[f_i, f_i + 1), \dots, [r_i + p_i - 1, r_i + p_i)$ is reserved. We associate to each *timetable constraint* two histograms (see Fig. 1):

- a *level histogram* which contains the amount of resource required at each time interval $[t - 1, t)$.
- an *activity histogram* which contains the sets S_t of activities which require any amount of resource for each time period $[t - 1, t)$. It will essentially be used for providing explanations.

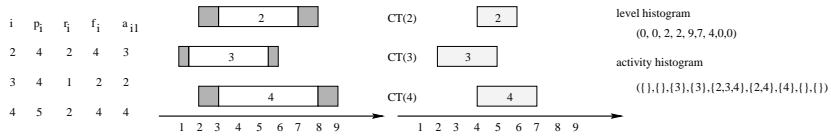


Fig. 1. Example of timetable.

These histograms are used in the following ways:

- detecting resource conflicts when the required level of a resource k at one time period t exceeds the resource capacity. The conflict set associated to this contradictory situation is constituted from the set of activities (S_t) stored in slot t of the activity histogram. We use the following equation (c being the histogram constraint itself): $\left(\bigwedge_{v \in S_t} \left(\bigwedge_{a \in d(v)} \text{expl}(v \neq a) \right) \right) \wedge c$.

- tightening the time-window of an activity. It may occur that the current bounds of the time-window of an activity are not compatible with the other activities. In that situation, the tightening of the time-window will be explained by the set S_t of activities requiring the resources during the incompatible time-period $[t - 1, t)$. The following equation is used (c being the histogram constraint itself): $\left(\bigwedge_{v \in S_t} \left(\bigwedge_{a \in d(v)} \text{expl}(v \neq a)\right)\right) \wedge c$.

3.2 Task-interval constraints

Using *task-intervals* for managing cumulative resources provides different services [1, 2]: conflict detection, precedence deduction and time-window tightening.

A *task-interval* $T = [i, j]$ is associated to each pair of activities (i, j) which require the same resource k . It is defined as the set of activities ℓ which share the same resource and such that $r_i \leq r_\ell$ and $d_\ell \leq d_j$ (r_i being the earliest starting time of the activity i and d_i its due date) *i.e.* that need to be scheduled between tasks i and j . The set $\text{inside}(TI)$ represents the set of activities constituting the *task-interval*, while $\text{outside}(TI)$ contains the remaining activities. Let define $\text{energy}(TI) = \sum_{\ell \in \text{inside}(TI)} (d_\ell - r_\ell) \times a_{\ell k}$ the *energy* required by a *task-interval*.

Several propagations rules can be defined upon *task-interval* [1, 2]. We consider here the following two:

Integrity rule If $\text{energy}(TI)$ is greater than the total *energy* available in the task-interval $[i, j]$ (*i.e.* $(d_j - r_i) \times R_k$) then a conflict is detected. The conflict explanation set is built from the set of activities *inside* the *task-interval* *i.e.* (c being the constraint associated to this rule) $\left(\bigwedge_{v \in \text{inside}(TI)} \text{expl}(v \neq a)\right) \wedge c$.

Throw rule This rule consists in tightening the time-window of an activity intersecting a given *task-interval* TI . This is done by comparing the necessary energy shared with the activities in TI and energy available during the *task-interval*. The explanation of this update is built from the set of activities *inside* the *task-interval* (c being the constraint associated to this rule): $\left(\bigwedge_{v \in \text{inside}(TI)} \text{expl}(v \neq a) \wedge c\right)$.

4 Experiments on RCPSP

We developed a branch and bound search using the presented explanation-based techniques and based upon a branch and bound algorithm from [11]. This interactive system accepts several types of modification on the scheduling problem: temporal events (adding/removing precedence/overlapping/disjunctive relations, modifying time-windows), activity related events (adding/removing), resource related events (adding/removing/modifying).

Table 1 presents some first results on dynamic RCPSP. In this table, we report the relative time speed-up obtained using explanation-based dynamic constraint solving compared to solving each problem from scratch. The table reports results considering 4 consecutive modifications from an original problem. As we can see, those results are quite promising. Even bad results (instance 4) get better in

the long run. However, notice that some results (not reported here) show that dynamic handling is not always the panacea and from scratch rescheduling can be very quick.

Table 1. Some Kolish, Sprecher and Drexel instances (4 consecutive dynamic events). (www.wior.uni-karlsruhe.de/RCPSP/ProGen.html). Relative speed-up (in %)

12 act./4 res.	Modif. 1	Modif. 2	Modif. 3	Modif. 4
# 1	46.24	54.92	62.68	63.13
# 2	3.81	26.38	46.62	55.92
# 3	22.13	10.64	21.08	22.21
# 4	-29.21	-0.35	15.74	30.27
# 5	32.01	67.90	75.15	75.65
# 6	46.72	52.58	47.75	49.27
# 7	12.76	26.09	35.84	35.58
# 8	35.88	44.12	45.82	45.87

5 Conclusion

In this paper, we presented the integration of explanations within scheduling-related global constraint and its interest for solving dynamic scheduling problems. We presented first experimental results of a system that we developed using those techniques. These results demonstrate that incremental constraint solving for scheduling problem is useful.

We are currently improving our system with user-interaction capabilities still using explanations. Moreover, we are conducting high scale experiments (larger problems, easy/medium/hard problems, ...) to validate our approach.

A complete version of this paper is available at: www.emn.fr/jussien/publications/elkhyari-dynamic-rcpsp.pdf.

References

1. Caseau, Laburthe: Improving clp scheduling with task intervals. In Proc. of ICLP'94, MIT Press (1994) 369–383
2. Caseau, Laburthe: Cumulative scheduling with task-intervals. In: JICSLP'96.
3. Klein, Scholl: Computing lower bounds by destructive improvement: an application to Resource-Constrained Project Scheduling Problem. *EJOR* **112** (1999) 322–345
4. Dechter, Dechter: Belief maintenance in dynamic constraint networks. In: *AAAI'88*, St Paul, MN (1988) 37–42
5. Artigues, Roubellat: A polynomial activity insertion algorithm in a multi-resource schedule with cumulative constraints and multiple modes. *EJOR* **127** (2000) 297–316
6. Bessière: Arc consistency in dynamic constraint satisfaction problems. In: *Proceedings AAAI'91*. (1991)
7. Jussien: e-constraints: explanation-based constraint programming. In: *CP01 Workshop on User-Interaction in Constraint Satisfaction*, Paphos, Cyprus (2001)
8. Jussien, Debruyne, Boizumault: Maintaining arc-consistency within dynamic backtracking. In: *Proc. CP 2000*. LNCS 1894, Springer Verlag (2000) 249–261
9. Jussien, Lhomme: Local search with constraint propagation and conflict-based heuristics. *Artificial Intelligence* **139**(1) (2002) 21–45.
10. Blazewicz, Lenstra, Rinnoy Kan: Scheduling projects subject to resource constraints: classification and complexity. *Discrete Applied Mathematics* **5** (1983) 11–24
11. Brucker, Knust, Schoo, Thiele: A branch and bound algorithm for the Resource-Constrained Project Scheduling Problem. *EJOR* **107** (1998) 272–288