

CONSTRAINT PROGRAMMING FOR DYNAMIC SCHEDULING PROBLEMS

Abdallah Elkhyari, Christelle Guéret[†], and Narendra Jussien

École des Mines de Nantes
4, rue Alfred Kastler – BP 20722
F-44307 Nantes Cedex 3 – France
{aelkhyar,gueret,jussien}@emn.fr
[†] IRCCyN CNRS UMR 6597

Abstract

Scheduling problems considered in the literature are often static (activities are known in advance and constraints are fixed). However, every real-life schedule is subject to unexpected events. In these cases, a new solution is needed in a preferably short time and as close as possible to the current solution. In this paper, we present an exact approach for solving dynamic Resource-Constrained Project Scheduling Problems or RCPSP. This approach combines explanation-based constraint programming and operational research techniques. We present our first experimental results that show impressive improvements in both computation time and stability when comparing our approach to a re-execution from scratch.

Keywords: scheduling, constraint programming, dynamic problems, stability.

1. DYNAMIC RESOURCE CONSTRAINED PROJECT SCHEDULING PROBLEMS

The Resource Constrained Project Scheduling Problem (RCPSP) is a general scheduling problem which consists in scheduling a set of activities taking into account temporal and resource constraints. Preemption is not allowed. The objective considered here is the minimization of the makespan (total duration) of the project. This problem is *NP-hard* (Blazewicz et al., 1983).

Most work about RCPSP consider static problems in which activities are known in advance and constraints are fixed (Klein, 1999). However, every schedule is subject to unexpected events (consider for example a new activity to schedule, or a resource failure – *eg.* machine breakdown). When such a situation arises, a new solution taking these events into account is needed generally in a short time. Furthermore, this new solution must preferably be not too far from the previous one.

Several works concern dynamic scheduling problems. But generally, they deal with very specific problems like one-machine problems (Mehta and Uzsoy, 1999; Aloulou and Portmann, 2002) or m-processors (Moukrim et al. 1999), and the number of types of events taken into account are limited to one or two (Vieira et al., 2000; Daniels and

Carillo, 1997). Furthermore, to our knowledge, no optimal approach has been proposed for any dynamic scheduling problems. The only approach concerning the dynamic RCPSP has been recently proposed by (Artigues et al., 2000). The authors developed a heuristic based on a flow network model to update an initial static schedule when considering the insertion of an unexpected activity.

2. CONSTRAINT PROGRAMMING FOR DYNAMIC SCHEDULING PROBLEMS

Constraint programming provides software tools to solve constraint satisfaction problems (a set of variables each with a given domain and upon which relations – constraints – need to be verified). Classical constraint programming solving interleaves filtering (removing from the current domain of the variables a set of values which can be proved not to appear in any solution) and enumeration (making choices – instantiating variables – when no more filtering is possible). Constraint programming is increasingly used for solving scheduling problems as its flexibility is well suited for real-life scheduling problems (Caseau and Laburthe, 1996; Baptiste and Le Pape, 1997; Guéret et al., 2000). Moreover, solving dynamic constraint satisfaction problems (DCSP) is a vivid research topic in the constraint programming community. A DCSP (Dechter and Dechter, 1988) is a constraint satisfaction problem whose set of variables or/and constraints evolves throughout computation leading to a series of problems differing one from the other by the addition/retraction of single variable/constraint. However, no technique, as far as we know, has been proposed yet to handle dynamic scheduling problems.

Several techniques to handle dynamic problem exist: *pro-active* techniques that use a model of the potential perturbations of the initial problem and *re-active* methods that handle the perturbation as it appears. As far as the latter set of techniques is concerned, a recent extension of constraint programming can be used: explanation-based constraint programming (Debruyne et al., 2003).

We introduce here a constraint-based technique to handle dynamic RCPSP instances. This technique which provides optimal solutions is able to handle a large number of differ-

ent unexpected events and computes a new solution quicker than when solving the new problem from scratch. Moreover, the successive computed solutions are much more stable than solutions computed from scratch.

3. EXPLANATION-BASED CONSTRAINT PROGRAMMING TECHNIQUES

Efficiently solving dynamic constraint satisfaction problems requires incremental addition and retraction of constraints. Even though incremental constraint addition is naturally handled by modern constraint solvers, incremental retraction of constraints is often performed with recording trace/undo information. Such an information is used to determine past effects of removed constraint that need to be undone. *Explanations* for constraint programming 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 is valid; thus, no valid solution can be build from this partial assignment.

Although incremental constraint addition is usually built in modern constraint solvers, incremental constraint retraction is still an issue (Debruyne et al., 2003). Explanations are a useful tool regarding that topic. Determining the past effects of a constraint to be retracted from the constraints system is quite easy: it is the set of actions whose explanation contains the retracted constraint. Therefore, those past effects can be undone at once and a limited re-propagation is performed in order to obtain a consistent state of the problem (Debruyne et al., 2003).

Explanation-based constraint programming techniques need two prerequisites: all handled constraints need to be enhanced with explanation capabilities; the branching scheme used during the enumeration phase need to make decisions represented as a single constraint and whose negation is also a single constraint. To meet those two requisites for solving dynamic RCPSP problems, we enhanced existing resource management constraints (*core-times* (Klein and Scholl, 1999), *task-intervals* (Caseau and Laburthe, 1996)) with explanation capabilities (Elkhyari et al., 2002). Moreover, we introduced a notion of distance between activities that is used to define branching decisions with simple constraints whose negations are also simple.

4. AN ENVIRONMENT FOR SOLVING DYNAMIC RCPSP

Our dynamic RCPSP solving system is based on a branch and bound algorithm inspired from (Brucker et al., 1998). This branch and bound algorithm has been enhanced with explanation capabilities as in (Guéret et al., 2000). We are capable of dynamically handling a various set of events:

- **temporal events:** addition, retraction, or modification of classical and generalized precedence, disjunction or

overlapping constraints between two activities which is useful to handle, for example, new constraints arising from the arrival of a new urgent order, or, new technical constraints, etc.

- **activity related events:** addition, retraction, or modification of an activity (new orders, order removals, duration or resource needs modifications, etc.
- **resource related events:** addition, retraction, or modification of a resource which is useful to handle machine breakdowns, newly repaired machines, etc.

The process used to solve dynamic RCPSP instances is the following: an optimal solution for the original problem is computed; unexpected events are handled incrementally from this first solution:

- upon the **addition** of a new information: if a conflict with the current solution is identified, a re-optimization is asked (the conflict is analyzed using recorded explanations, some past decisions are undone, some new decisions are made until a new optimal solution is obtained); otherwise, nothing happens because the current solution remains optimal;
- upon the **removal** of an existing information: the related constraints are incrementally removed using explanations and re-optimization is performed (starting from the current solution and stored information about past search);
- **modifications** are considered as: first, an information removal and, second, an information addition.

One of the interests of our technique is that although some re-optimisation is needed in order to provide optimal solutions, explanations act as a learning mechanism and are used to discard at once many parts of the search space that has been explored before and that are still valid in the current context (Jussien and Lhomme, 2002).

Notice that adding, removing, and modifying constraints can quite quickly lead to an unfeasible instance. Thanks to the explanations, our system is able to provide some information to the user in order to let him/her know what is the subset of modifications that leads to this situation letting him/her modify things in order to get back to a feasible instance (Debruyne et al., 2003).

5. EXPERIMENTAL RESULTS

We give here results obtained in terms of stability of successive solutions when comparing our approach with a re-execution from scratch after each modification. Stability can be measured by several parameters. We report here the number of relative positions of activities that change between consecutive solutions (the POSI measure). As no benchmarks exist regarding dynamic RCPSP, we created our own set using the SMCP (*Single-Mode Project Scheduling*) series (Kolish et al, 1995): 200 problems broken into 20 series of 10 problems.

We report here results obtained considering activity additions and retractions¹:

- Considering **activity additions**, for each instance in the SMCP series, we randomly chose a set of activities (between 2 and 4 in tables 1 and 3) to be removed from the original problem (along with precedence constraints related to them) leading to what is denoted problem P_0 . Then, problem P_i is built from problem P_{i-1} by adding a randomly chosen activity from the set of removed activity. Precedence constraints are added relating the added activity to activities in problem P_{i-1} . P_{end} is the last obtained problem (either P_2 , P_3 or P_4 depending on the size of the problem). Notice that P_{end} is exactly the original problem from the SMCP series.
- Considering **activity retractions**, the initial problem is a problem from the SMCP series. P_i is computed from P_{i-1} by randomly removing an activity and the precedence constraints relating this activity to the other activities.

Table 1 reports improvement results (average improvement in percent compared to from scratch rescheduling) obtained when considering activities additions on series of 10 problems with 12, 22, and 32 activities. These results are given for the successive problems and the last column reports the overall improvement (or loss – negative values). We can see that the average improvement lies between -51.8% and 81%. What is to be noted is that the more important the size of the problem, the more improvement our technique provides.

Table 2 reports improvement results (average improvement in percent compared to from scratch rescheduling) obtained when considering activities retractions on problems with 12, 22, and 32 activities. These results are given for the successive problems and the last column reports the overall improvement (or loss – negative values). We can see that the average improvement lies between -27% and 43.6%. Very impressive improvements are encountered for problems considering 22 and 32 activities.

Notice that the way we compute the improvement it cannot be more than 100%. The results presented here show that using explanation-based techniques provides impressive results regarding stability of successive solutions. The larger the instance, the more improvement is brought by our technique.

Tables 3 and 4 report computation time needed to perform those experiments. Table 3 shows that the average computation improvement relies between 10.7% and 78.2%. Moreover, the general improvement (considering time spent from the initial problem to the last considered one) is quite important: from 43.7% to 90.5%. The greater the number of activities, the more important the general improvement is. Table 4 reports the same kind of results even if considering 12 activities problems leads to some general loss. As soon as 22 activities are considered, using our technique leads to

impressive general improvements.

We also conducted experiments considering addition and inversion of precedence constraints. We obtained very similar results that are not reported here. A synthesis of our experiments led to us to the following: our technique provides very important improvements both in computation time and stability (we also considered other stability measures such as, for example, the number of activities whose starting date has changed). However, our technique does not perform well in the following situations:

- retracting activities in small size instances (12 activities). Computation time loss can be explained by the fact that those instances are quite easy to solve and removing activities makes them even more easier. Therefore, repairing time is greater than recomputing from scratch. However, stability improvement is observed for these instances;
- inverting precedence constraints in average size instances (22 activities). Inversions highly disturb the current solution leading to highly time consuming repair and highly modified solutions.

6. CONCLUSION

We introduced here a new approach for solving dynamic RCPSP instances. Our proposal is based on new constraint programming techniques: explanations. We provided a complete system able to handle both dynamic and over-constrained scheduling problems. A wide set of perturbations is available: activity addition/retraction/modification, resource addition/retraction/modification, generalized precedence/overlapping/disjunctive constraints addition/retraction/modification.

We performed a wide set of experimental results validating our approach compared to classical from-scratch recomputation. Our results show that our technique provides improvements in terms of both computation time and stability of consecutive solutions.

Our current topics following this work include:

- improving the handling of resource constraints by integrating more recent propagation techniques;
- providing real interaction tools within our system to make our approach useable by non-specialists;
- generalizing our technique to other kind of scheduling problems (*eg.* timetabling problems).

Moreover, a theoretical study about the stability of our approach would be quite interesting.

¹Complete results are available in (Elkhyari, 2003).

Table 1 Stability (using the POSI measure) improvements when considering activity additions. $X \times Y$ represents problems with X activities and Y resources.

	$P_0 \rightarrow P_1$	$P_1 \rightarrow P_2$	$P_2 \rightarrow P_3$	$P_3 \rightarrow P_4$	$P_0 \rightarrow P_{end}$
12×4	-24%	-51.8%	–	–	-58.3%
22×4	18.2%	12.5%	-14.6%	–	-19%
32×1	73.1%	79%	81%	-27.6%	11.2%
32×2	53.7%	37.4%	38.2%	63.5%	37.5%
32×3	54.3%	45.8%	76.5%	33.3%	21.1%
32×4	36.7%	62.1%	34.1%	49.7%	27.3%

Table 2 Stability (using the POSI measure) improvements when considering activity additions. $X \times Y$ represents problems with X activities and Y resources.

	$P_0 \rightarrow P_1$	$P_1 \rightarrow P_2$	$P_2 \rightarrow P_3$	$P_3 \rightarrow P_4$	$P_0 \rightarrow P_{end}$
12×4	-27%	1.7%	–	–	28.9%
22×4	17%	25.4%	-7.9%	–	25%
32×1	14.3%	7.2%	15.6%	30%	27.5%
32×2	35.7%	17.9%	-0.1%	39.1%	38.5%
32×3	-0.1%	27.7%	14.9%	43.6%	32.1%
32×4	12.9%	36.5%	20.2%	18.9%	0.9%

Table 3 Computation time improvements when considering activity additions. Minimum and maximum (along with average) improvements are reported for each modification as well as the general improvement (from the starting problem to the last dynamic event – GAvg). $X \times Y$ represents problems with X activities and Y resources.

	$P_0 \rightarrow P_1$		$P_1 \rightarrow P_2$		$P_2 \rightarrow P_3$		$P_3 \rightarrow P_4$		GAvg
	Min/Max	Avg	Min/Max	Avg	Min/Max	Avg	Min/Max	Avg	
12×4	-0.9%	37.7%	-63.8%	32.2%	–	–	–	–	43.7%
	64.4%		69.6%		–		–		
22×4	-34.6%	29.7%	-208%	5.8%	-155.2%	34.4%	–	–	78.9%
	94.4%		94.4%		96%		–		
32×1	30%	58.3%	18.2%	68.5%	40.7%	77%	-66.1%	46.7%	83.8%
	98.7%		99.3%		98.7%		99.6%		
32×2	-280.9%	10.7%	-140.1%	57.8%	-64.2%	68.7%	-78.9%	62.8%	84.9%
	91.2%		97.1%		96.9%		97.6%		
32×3	12.7%	61.4%	15.9%	66%	16.8%	74.6%	30.6%	78.2%	90.5%
	95.2%		98.2%		98.8%		96.3%		
32×4	33.3%	57%	-18.1%	63.5%	-19.6%	60.7%	26.7%	78.2%	90%
	75.9%		97.4%		98.7%		99.2%		

Table 4 Computation time improvements when considering activity retractions. Minimum and maximum (along with average) improvements are reported for each modification as well as the general improvements (from the starting problem to the last dynamic event – GAvg). X×Y represents problems with X activities and Y resources.

	$P_0 \rightarrow P_1$		$P_1 \rightarrow P_2$		$P_2 \rightarrow P_3$		$P_3 \rightarrow P_4$		GAvg
	Min/Max	Avg	Min/Max	Avg	Min/Max	Avg	Min/Max	Avg	
12×4	-212.2%	-51%	-195.5%	-65.3%	–	–	–	–	-19.1%
	21.7%		19.5%		–		–		
22×4	-637.8%	-22.9%	-242.8%	20.5%	-232.1%	21.5%	–	–	44.7%
	91.5%		94.3%		88.8%		–		
32×1	-185%	13.4%	-119%	20.6%	-102.6%	0.9%	-151.5%	-0.5%	47.7%
	97.1%		92.7%		91.1%		91.5%		
32×2	-148.4%	24.2%	-219.4%	14.4%	-161.2%	38.9%	-164.8%	34%	59.4%
	98%		98.7%		97.4%		97.5%		
32×3	0%	65.5%	-34.8%	52.3%	-38.3%	48.1%	-43.1%	49.2%	51.8%
	94.1%		94%		91.7%		90.3%		
32×4	-280.4%	31%	-259.5%	32.5%	-180.2%	28.3%	-148.1%	26.3%	59.4%
	92.4%		95.7%		92.3%		91.3%		

References

- ALLOULOU, M.A. and PORTMANN, M.C. (2002): A genetic algorithm to achieve scheduling flexibility for a single machine problem. submitted to RAIRO-OR.
- ARTIGUES, C. and ROUBELLAT, F. (2000): A polynomial activity insertion algorithm in a multi-resource schedule with cumulative constraints and multiple modes. In: *European Journal of Operational Research*, 127(2):179–198.
- BAPTISTE, P. and LE PAPE, C. (1997): Constraint propagation and decomposition techniques for highly disjunctive and highly cumulative project scheduling problems. In: *Proceedings CP'97*, Lecture Notes in Computer Science, 1330, Springer-Verlag.
- BESSIÈRE, C. (1991): Arc consistency in dynamic constraint satisfaction problems. In *Proceedings AAAI'91*.
- BLAZEWICZ, J., LENSTRA, J.K. and RINNOY KAN, A.H.G. (1983): Scheduling projects subject to resource constraints: classification and complexity. In: *Discrete Applied Mathematics*, 5:11–24.
- BRUCKER, P., KNUST, S., SCHOO A., and THIELE, O. (1998): A branch and bound algorithm for the Resource-Constrained Project Scheduling Problem. In: *European Journal of Operational Research*, 107:272–288.
- CASEAU, Y. and LABURTHE, F. (1996): Cumulative scheduling with task-intervals. In: *Joint International Conference and Symposium on Logic Programming (JICSLP)*.
- DANIELS R.L. and CARILLO, J.E. (1997): Beta-robust scheduling for single-machine systems with uncertain processing times. In: *IIE Transactions*, 29:977–985.
- DEBRUYNE, R., FERRAND, G., JUSSIEN, N., LESAIN, W., OUIS, S. and TESSIER, A. (2003): Correctness of Constraint Retraction Algorithms. In: *Proceedings FLAIRS'03*, AAAI press, pages 172–176.
- DECHTER, R. and DECHTER, A. (1988): Belief maintenance in dynamic constraint networks. In: *Proceedings AAAI'88*, pages 37–42, St Paul, MN.
- ELKHYARI, A. (2003): Outils d'aide à la décision pour des problèmes d'ordonnancement dynamiques. PhD Thesis. École des Mines de Nantes – Université de Nantes, France. Nov 27. In French.
- ELKHYARI, A., GUÉRET, C. and JUSSIEN, N. (2002): Conflict-based repair techniques for solving dynamic scheduling problems. In: *Proceedings CP 2002*, LNCS 2470, Springer-Verlag, pages 702–707.
- GUÉRET, C., JUSSIEN, N., and PRINS, C. (2000): Using intelligent backtracking to improve branch and bound methods: an application to open-shop problems. In: *European Journal of Operational Research*, 127(2):344–354.
- JUSSIEN, N., and LHOMME, O., (2002): Local search with constraint propagation and conflict-based heuristics. In: *Artificial Intelligence*, 139(1):21–45.
- KLEIN, R. and SCHOLL, A. (1999): Computing lower bounds by destructive improvement: an application to Resource-Constrained Project Scheduling Problem. In: *European Journal of Operational Research*, 112:322–345.
- KLEIN, R. (1999): Scheduling of Resource Constraints Projects. In: *Kluwer Academi, Boston*.
- KOLISCH, R. SPRECHER, A. and DREXL, A. (1995): Characterization and generation of a general class of resource-constrained project scheduling problems. In: *Management Science*, volume 41, pages 1693–1703.
- METHA, S.V. and UZSOY R. (1999): Predictable scheduling of a single machine subject to breakdowns. In: *International Journal of Computer Integrated Manufacturing*, 12:15–38.

- MOUKRIM, A. and SANLAVILLE, E. and GUINAND, F. (1999): Scheduling with Communication Delays and On-Line Disturbances. In: *Proceedings of Euro-Par'99 - Parallel Processing: 5th International Euro-Par Conference*, LNCS ISSN: 0302-9743.
- VIEIRA, G.E., HERRMANN, J.W. and LIN, E. (2000): Analytical models to predict the performance of a single-machine system under periodic and event-driven rescheduling strategies. In: *International Journal of Production Research*, 38(8):1899-1915.