Solution and Reasoning Reuse in Space Planning and Scheduling Applications

Gérard Verfaillie and **Thomas Schiex**
ONERA-CERT *
2 avenue Edouard Belin, BP 4025
31055 Toulouse Cedex, France
phone: (33) 61 25 25 25
fax: (33) 61 25 25 50
{verfail,schiex}@cert.fr

Key words and phrases

Artificial Intelligence, Dynamic Constraint Satisfaction Problems, Planning, Scheduling.

Abstract

In the space domain, as in other domains, the CSP (*Constraint Satisfaction Problems*) techniques are increasingly used to represent and solve planning and scheduling problems. But these techniques have been developed to solve CSPs which are composed of fixed sets of variables and constraints, whereas many planning and scheduling problems are *dynamic*. It is therefore important to develop methods which allow a new solution to be rapidly found, as close as possible to the previous one, when some variables or constraints are added or removed.

After presenting some existing approaches, this paper proposes a simple and efficient method, which has been developed on the basis of the *dynamic backtracking* algorithm [1]. This method allows previous solution and reasoning to be reused in the framework of a CSP which is close to the previous one. Some experimental results on general random CSPs and on operation scheduling problems for remote sensing satellites are given.

Space planning and scheduling applications and CSP

In the space domain, as in other domains, the *Constraint* based approach is increasingly used to represent and solve planning and scheduling problems. The CSP (*Constraint Satisfaction Problems*) framework offers a general formalism for constrained problems (any kind of constraint is allowed) and powerful solving methods [2]. Various *constraint programming* languages and tools have been developed these last years on this basis and are now available. Using them avoids long and useless software developments.

Let us recall that a CSP is defined by two sets: a set V of *variables* and a set C of *constraints*. Each variable has a finite set of possible values: its *domain*. Each constraint links a subset V' of the CSP variables and defines the set of the possible combinations of values for the variables in V'.

The usual problem is to find a solution, *i.e* a value for each variable such that all the constraints are satisfied. The most used methods are combinations of a *backtrack* search, using a *depth-first* strategy and some *heuristics* along with a *filtering* method (*forward-checking*, *arc-consistency*, *path-consistency* ...) which allows the search space to be pruned.

Dynamic problems: origin

A strong limitation of these techniques lies in the fact that they have been developed under the assumption that the sets of variables and constraints are given once and for all. In many real applications, and particularly in space applications, this assumption is not valid [3]. The reasons are numerous:

- before a mission, in the phase of specification and analysis, engineers may want to explore several alternatives and their implications; they may also want to derive a new specification from a previous one;
- during a mission, there is always a great difference between execution and forecast: operation results, durations, resource consumptions, possible breakdowns, ...
- according to new requirements or decisions of the people in charge of the mission, some new operations may have to be performed and others already planned and scheduled may have to be removed.

Dynamic problems: requirements

According to the computing point of view, all these situations are very similar: a previous CSP has been solved; a new one, which is close to the previous one (just some variables and constraints have been added or removed), has now to be solved. It is obviously

*This work was done at the University of New Hampshire (USA) and was supported the French Ministry of Defence (DGA-DRET).

possible to solve it from scratch, as it has been done for the first one, but this naive method may be very inefficient and lead to an instability of the successive solutions.

During the mission, if the time available to find a new plan or a new schedule is limited, *efficiency* can become a very important requirement. Before or during the mission, if some work (training, organization, orders ...) has been started on the basis of the previous solution, *stability* of the successive solutions can also be important.

Therefore one needs methods which, starting from the previous solution and the previous reasoning, allow a new solution to be rapidly found, as close as possible to the previous one.

Existing approaches

The existing approaches can be classified into three groups:

- *heuristic* approaches, which consist of using any previously consistent assignment (complete or not) as a heuristic in the framework of the current CSP [4];
- *local repair* approaches, which consist of starting from any previously consistent assignment (complete or not) and of repairing it, using a sequence of local modifications [5, 6, 7, 8, 9];
- *constraint recording* approaches, which consist of recording any kind of constraint which can be deduced in the framework of a CSP and its justification, in order to reuse it in the framework of any new CSP which includes this justification[4, 10].

Dynamic backtracking

In spite of its name, the *dynamic backtracking* algorithm [1] does not deal with *dynamic* CSPs. The term *dynamic* means here that its *backtracking* mechanism allows the variables to be unassigned in an order which is different from the one which has been used to assign them. It can be described as follows:

- let *val* be a value which can not be assigned to a variable v, because of a constraint c which links v to previously assigned variables and would be unsatisfied; let V' be the set of variables linked by c; the set $V' - \{v\}$ is recorded as an *eliminating explanation* for *val*; the *conflict set* of a variable is the union of the *eliminating explanations* of all its eliminated values;
- let v be a variable whose current domain is empty, let V' be its *conflict set*; let v' be the last variable in V' according to the assignment order and val' be its current value; v' is unassigned; then all the *eliminating explanations* where v' is involved are removed (they are no more valid) and the set $V' - \{v'\}$ is recorded as an *eliminating explanation* for val'.

Termination, correctness and completeness of this algorithm have been proven.

Note the difference between such a mechanism and the usual *chronological backtracking* and *conflict directed backjumping* [11] mechanisms:

- *chronological backtracking* does not backtrack to v', but systematically to the variable which immediatly precedes v according the assignment order;
- *conflict directed backjumping* backtracks (backjumps) to v', but, doing that, it unassigns all the variables which are between v' and v according to the assignment order;
- *dynamic backtracking* also backtracks to v', but it only unassigns v'.

This allows us to say that the *dynamic backtracking* mechanism is more pertinent and less destructive than both other ones.

Extended Dynamic Backtracking

Such features are very interesting in the framework of dynamic CSPs, when constraints and variables are added or removed in any order. For that, the notion of *eliminating explanation* has first to be extended in order to take into account constraints and variable domains as assumptions, as previously done with variable assignments. An extended *eliminating explanation* involves previously assigned variables (assignment constraints), variable domains (unary constraints) and usual constraints, which are together responsible for the value elimination. The previous description has just to be slightly modified to take into account this extension:

- let *val* be a value which can not be assigned to a variable v, because of a constraint c which links v to previously assigned variables and would be unsatisfied; let V' be the set of variables linked by c; the set $V' - \{v\} \cup \{c\}$ is recorded as an *eliminating explanation* for *val*; the *conflict set* of a variable is the union ...
- let v be a variable whose current domain is empty, let V' be its *conflict set* and $d(v)$ be its initial domain; let v' be the last variable in V' according to the assignment order and val' be its current value; v' is unassigned; then all the *eliminating explanations* where v' is involved are removed and the set $V' - \{v'\} \cup \{d(v)\}$ is recorded as an *eliminating explanation* for val'.

And the previous algorithm can be extended as follows to deal with dynamic CSPs:

- let c be a constraint which is added or restricted (this includes the case of restricted variable domains); if the current assignment does not violate c, there is nothing to do; else, let V' be the set of the variables linked by c; let v be the last variable involved in V' according the assignment order and *val* be its current value; v is unassigned; then all the *eliminating explanations* where v is involved are removed and the set $V' - \{v\} \cup \{c\}$ is recorded as an *eliminating explanation* for *val*.

- let c be a constraint which is removed or relaxed (this includes the case of relaxed variable domains); all the *eliminating explanations* where c is involved are removed;

Such an algorithm has very interesting properties:

- all the possible changes to a CSP (variable and constraint addition, removal and modification) are covered;
- previous solution and reasoning (*eliminating eliminations* previously recorded) are systematically reused; just the variable assignments which are no more consistent and the *eliminating explanations* which are no more valid are removed; in that sense, this extended *dynamic backtracking* algorithm combines the advantages of the *local repair* and *constraint recording* approaches and should provide goods results in terms of both efficiency and stability;
- changes can be taken into account at any time, either after or during the search;
- in case of inconsistency, the user can be provided with an explanation: a subset of the CSP constraints and domains which are together responsible for this inconsistency;
- computing *eliminating explanations* and *conflict sets* is a very simple task (only union operations are required) and the space required to record them is polynomially bounded (it is $O(nd(n+m))$, where n is the number of variables, m the number of constraints and d the maximum domain size);

Experiments, results and analysis

This algorithm (called *ddbt* for *dynamic dynamic backtracking*) has been experimented on *dynamic* CSPs and compared with others, like *conflict directed backjumping* (*cbj* [11]), *dynamic backtracking* (*dbt* [1]), *heuristic repair* (*hrp* [6]) and *local changes* (*lc* [9]), with *backward* and *forward-checking*.

A first set of general and binary CSPs has been used for these experiments. These CSPs have been randomly generated using fixed values for the number of variables (16) and the domain size (13) and various values for the constraint tightness (from 0.1 to 0.9), the graph connectivity (from 0.2 to 0.9) and the change size (ratio between the number of added or removed constraints and the number of constraints, from 0.01 to 0.16).

The results, which have been obtained by using *forward-checking* with each algorithm, are summed up in the four following set of curves. The three first ones show efficiency results (number of constraint checks) on underconstrained, intermediate and overconstrained problems. The last one shows stability results (distance between successive solutions, *i.e.* the number of variables which are differently assigned in both solutions) on underconstrained problems:

- the first and the third sets of curves show that *ddbt* is the most efficient on underconstrained (always consistent) and overconstrained (always inconsistent) problems;
- the second one shows that *cbj* remains the most efficient on the intermediate problems (the hardest ones to be solved; sometimes consistent, sometimes not), but that *ddbt* is not far worse;
- the fourth one shows that the algorithms which reuse the previous solution such as *hrp*, *lc* and *ddbt* provide a better stability than the others do.

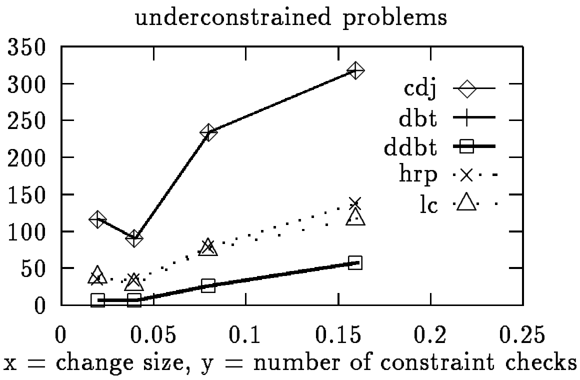
The same algorithms have been applied with the same kind of results on randomly generated operation scheduling problems for remote sensing satellites. These problems, whose definition comes from previous studies for the French Space Agency (CNES), in the framework of the SPOT program, are composed of a set of remote sensing satellites and a set of user observation requirements:

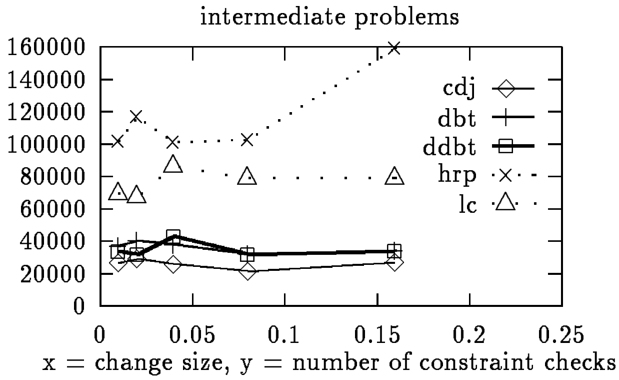
- each user requirement is defined by an aera to observe and some constraints related to the mode, the quality and the period of the observation;
- each satellite is defined by its trajectory, its observation capabilities, its possible modes and minimal transition times between modes.

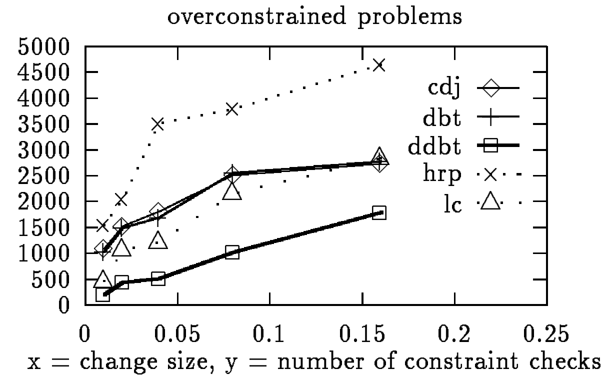
One assumes that these data allow a finite set of pairs (satellite, time slot) to be computed for each user requirement. In these conditions, the problem becomes a CSP where the only constraints are related to the minimal transition time between two time slots corresponding to the same satellite. But the lack of real data considerably limited the interest of these experiments.

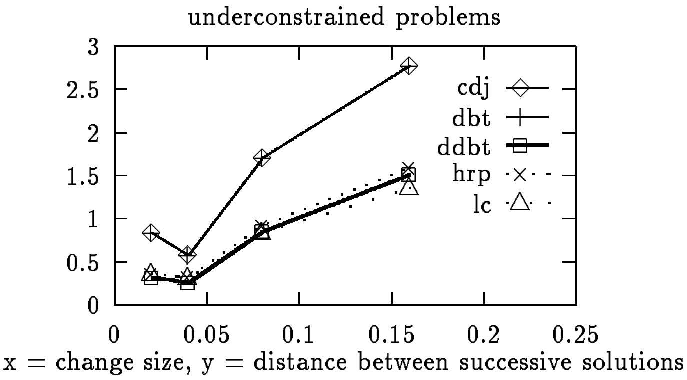
Conclusion

With this extension, the *dynamic backtracking* algorithm offers the opportunity to reuse previous solution and reasoning, when the problem changes, during or after the search. First experiments on small problems are promising. It should allow *dynamic* and *on-line planning and scheduling problems* to be efficiently dealt with. But these results have to be confirmed by larger experiments on various real problems.









References

[1] M. Ginsberg, "Dynamic Backtracking", *Journal of Artificial Intelligence Research*, vol. 1, pp. 25–46, 1993.

[2] E. Tsang, *Foundations of Constraint Satisfaction*, Academic Press Ltd., London, 1993.

[3] C. Bastien-Thiry and G. Verfaillie, "Space Mission Plan : Robustness, Flexibility, Replanning and Explanations", *in Proc. of the ESTEC Workshop on Artificial Intelligence and Knowledge-Based Systems for Space*, 1993.

[4] P. Van Hentenryck and T. Le Provost, "Incremental Search in Constraint Logic Programming", *New Generation Computing*, vol. 9, pp. 257–275, 1991.

[5] B. Freeman-Benson, J. Maloney, and A. Borning, "An Incremental Constraint Solver", *Communications of the ACM*, vol. 33, pp. 54–63, 1990.

[6] S. Minton, M. Johnston, A. Philips, and P. Laird, "Minimizing Conflicts: a Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems", *Artificial Intelligence*, vol. 58, pp. 160–205, 1992.

[7] B. Selman, H. Levesque, and D. Mitchell, "A New Method for Solving Hard Satisfiability Problems", *in Proc. of AAAI-92*, pp. 440–446, San Jose, CA, 1992.

[8] K. Ghedira, *MASC : une Approche Multi-Agent des Problèmes de Satisfaction de Contraintes*, Thèse de doctorat, ENSAE, Toulouse, France, 1993.

[9] G. Verfaillie and T. Schiex, "Solution Reuse in Dynamic Constraint Satisfaction Problems", *in Proc. of AAAI-94*, Seattle, WA, 1994.

[10] T. Schiex and G. Verfaillie, "Nogood Recording for Static and Dynamic CSP", *in Proc. of the 5^{th} IEEE International Conference on Tools with Artificial Intelligence*, Boston, MA, 1993.

[11] P. Prosser, "Hybrid Algorithms for the Constraint Satisfaction Problems", *Computational Intelligence*, vol. 9, pp. 268–299, 1993.