Soft Arc Consistency Applied to Optimal Planning

Martin Cooper, Sylvain Cussat-Blanc, Marie de Roquemaurel, and Pierre Régnier

IRIT, 118 route de Narbonne, 31062 Toulouse cedex 9, France
{cooper, cussat, deroquemaurel, regnier}@irit.fr

Abstract. We show in this article[1] how the Weighted CSP framework can be used to solve an optimisation version of numerical planning. The WCSP finds an optimal plan in the planning graph containing all solution plans of minimum length. Experimental trials were performed to study the impact of soft arc consistency techniques (FDAC and EDAC) on the efficiency of the search for an optimal plan in this graph. We conclude by giving a possible theoretical explanation for the fact that we were able to solve optimisation problems involving several hundred variables.

1 Introduction

In the field of planning, one of today's challenges is the solution of numerical problems to optimality. Some numerical planners perform heuristic choices, aiming simply to produce a good quality plan [5] while others merely compute *a posteriori* the cost of the solution-plan [6]. Some planners already use CSP to encode the planning graph [4], but never in a numerical approch to planning. We use the WCSP [7] framework to find a minimum cost plan, allowing the representation of strict constraints and an optimisation criterion expressed as the aggregation of cost functions.

2 WCSP and Numerical Planning

2.1 Numerical Planning Graph

A numerical planning problem is a triple $\langle A, I, G\rangle$ such that the initial state I is a finite set of propositional and numerical variables (or fluents) with their initial assignments, A is a set of actions, i.e. triples $\langle prec(a), effect(a), cost(a)\rangle$, where $prec(a)$ is the set of preconditions of action a, $effect(a)$ is the set of effects of a (Adds, Deletes and Modifiers of fluents), $cost(a)$ is the cost of applying a. An action a is applicable in a state S iff its preconditions are satisfied. A proposition p is satisfied in S iff $p \in S$; a numerical condition c is satisfied in state S iff the numerical variables of c are defined in S and verify c. G is the set of propositional and numerical goals to be satisfied. One of the most efficient and

[1] An extended version of this paper is available at http://www.irit.fr/recherches/RPDMP/persos/Regnier/PLANIF/index.html

influential algorithms in the field of planning is that of GRAPHPLAN [1]. We had to adapt the construction of the planning graph in order to solve numerical planning problems. The numerical planning graph G_p is a disjunctive graph which can be considered as a compact summary of all solution-plans up to a given maximal length. Actions can be applied in parallel but with three restrictions (called *interference*): (1) none deletes a precondition or an add-effect of another, (2) they have no numerical variable in common and (3) for each pair of distinct preconditions of actions at level i, there is at least one pair of non-interfering actions at the level $i-1$ which produce them.

In GRAPHPLAN, the planning graph G_p is extended level by level until either the extraction of a solution-plan is successful or the planning graph levels off. Level-off occurs when the sets of actions, fluents and mutexes (mutual exclusion constraints) are identical at levels i and $i+1$. It can be shown that if no solution has been found in a planning graph that has levelled off, then no solution exists. The first level of the graph consists simply of fluents corresponding to the initial state I. The next levels $i > 0$ are developed using the following algorithm:

1. We find all actions whose preconditions are satisfied in level $i-1$. To maintain at level i a fluent f present at level $i-1$, there is a noop action which has f as its only precondition and its only effect.
2. Next, we have to calculate the mutual exclusion relations (or mutexes) between actions. Two actions are mutex iff one interferes with the other.
3. We can now add to the graph all the fluents produced by these level i actions. As for the actions, we have to search for interferences between fluents: Two fluents are mutex at a given level i if there is no couple of non-mutex actions at the same level i that add these fluents.
4. The final step of level construction is to check if the goal is satisfied in the current state. If this is the case, the algorithm halts and we extract a solution plan from the graph. Otherwise, we go back to step 1.

The quality of a numerical plan can be estimated through a function known as a plan metric. We consider only the problem of minimising a linear additive metric, the sum of the costs of the actions in a plan P.

After having constructed the numerical planning graph, we reduce it by eliminating actions which cannot possibly be part of a solution-plan, ie all actions and fluents that do not have a path to any goal fluent. In preparation for coding the numerical planning graph as a WCSP (see Section 2.2), we rename the fluents $f_1, f_2, \ldots$ and renumber the actions 1,2,..., starting at the last level.

2.2 Coding the Planning Graph as a WCSP

Once the numerical planning graph has been constructed and reduced, we code it as a WCSP as follows:

1. *Creation of variables and domains:* for each fluent (not present in the initial state), we create a variable whose domain is the set of actions which produce this fluent. For each fluent (not present in the goal state) we add the value -1 to represent its non-activation.

2. *Translation of mutexes between fluents:* for all mutex fluents f_i and f_j, we code the fact that f_i et f_j cannot both be activated: $(f_i = -1) \vee (f_j = -1)$.
3. *Translation of mutexes between actions:* for all mutex actions a and b with (respective) effects f_i and f_j $(f_i \neq f_j)$, this mutual exclusion is coded by a constraint which states that f_i and f_j cannot simultaneously be triggered by the actions a and b: $\neg((f_i = a) \wedge (f_j = b))$.
4. *Translation of activity arcs:* the activation of a fluent f_i produced by an action a implies the activation of the preconditions of this action. This is coded by activity constraints: $\forall f_j \in prec(a)$, $(f_i = a) \Rightarrow (f_j \neq -1)$.
5. *Translation of the cost of actions:* For each value a, we add a unary constraint for $f = a$ corresponding to the cost of the action. No-ops have cost 0.
6. *Actions with multiple effects:* when an action a produces several fluents $f_i \in effect(a)$, the cost of this action could be counted several times. To avoid this problem, we create an intermediary fluent f^{int} with domain $\{a, -1\}$. Furthermore, the action a is replaced by a false action a^{int} (cost 0) in the domains of each f_i. We add the activity constraint $(f_i = a^{int}) \Rightarrow (f^{int} = a)$ between the fluent f^{int} and each of the fluents $f_i \in effect(a)$.

A limitation of our approach is that we only search for the optimal plan among parallel plans of length L, where L is the length of a shortest parallel solution-plan. Once this optimal plan among shortest-length plans has been discovered, it could, of course, be used as a good lower bound in the search for an optimal plan of any fixed length. Note, however, that the search space of all numerical plans is, in the worst case, infinite.

2.3 Search for an Optimal Solution to the WCSP

To simplify the WCSP [7] before solving it, we can apply soft arc consistency algorithms. Node consistency (NC) corresponds to taking the sum of minimum unary costs at each variable as a lower bound.

Propagating all infinite costs (between unary and binary constraints) and projecting binary costs until convergence establishes (soft) arc consistency (SAC). In order to have non-zero costs available as soon as possible during search, directional arc consistency (DAC) always chooses to send costs (via project and extend operations) towards variables which occur earlier in the instantation order. Full directional arc consistency (FDAC) [2] is the combination of directional arc consistency and arc consistency.

FDAC has recently been extended to existential directional arc consistency (EDAC) [3] which also performs the following operation: if for *each* value a in the domain of variable X_i, there exists a neighbouring variable X_j such that it is possible to increase $c_i(a)$ by sending costs from c_j and c_{ij}, then perform all these operations and then establish NC at X_i.

3 Experimental Trials

To test the utility of this approach, we carried out a large number of trials on different benchmark problems from IPC (International Planning Competition)

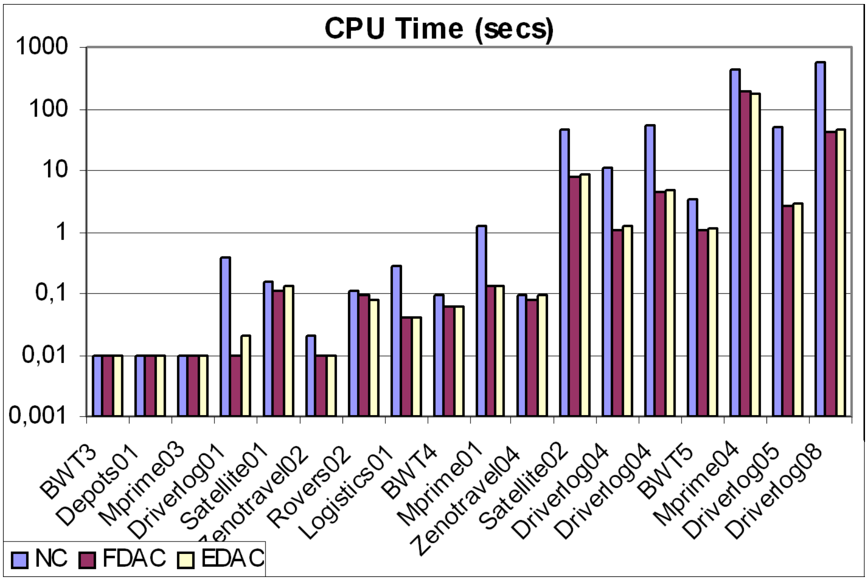

Fig. 1. Results of the trials using NC, FDAC and EDAC

covering diverse application areas (Blocksworld (bwt), Depots, Driverlog, Logistics, Mprime, Rover, Satellite and Zenotravel). We used the Toolbar library[2] [3] to solve the WCSP derived from the numerical planning graph[3]. The IPC benchmarks are derived from real problems and are highly structured. For each different domain, there is a series of problems of increasing difficulty, this being a function of the number of available actions and the number of fluents in the initial and goal states. The solution-plans for the hardest problems consist of thousands of actions. These benchmarks can be found at the IPC website[4].

We tested different soft arc consistency algorithms (NC, FDAC and EDAC) on different problems from the benchmark domains. NC, FDAC or EDAC was established at each node of the branch-and-bound search. Fig 1 compares the performances of these three soft arc consistency algorithms in terms of CPU time.

Firstly, comparing NC and FDAC (Figure 1), we observed that the number of nodes visited was always less with FDAC and that, on average, FDAC visited 17 times less nodes than NC. As for CPU time, FDAC is on average 7 times faster than NC. We can conclude that the use of FDAC significantly improves the time to extract an optimal solution from the numerical planning graph coded as a WCSP. Figure 1 also allows us to compare FDAC and EDAC. We found no significant difference between the two techniques. On average, EDAC visited

[2] http://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/ToolBarIntro
[3] wcsp files are available here: http://mulcyber.toulouse.inra.fr/plugins/scmcvs/cvsweb.php/benchs/planning/?cvsroot=toolbar
[4] http://ipc.icaps-conference.org/

5% less nodes but used 6% more CPU time. We can therefore conclude that EDAC, unlike on random problems of similar density [3], does not provide an improvement compared with FDAC on the problems tested.

The most striking result of our experimental trials is that we were able to solve real optimal planning problems coded as WCSPs with several hundred variables (maximum search space size about 10^{376}), even though previous trials on random WCSPs had indicated a practical upper limit of about 50 variables (search space size 10^{45}) [3]. The good performance of intelligent branch and bound search on planning problems can probably be explained by the existence of many crisp constraints and the structure of the constraint graph: the variables can be divided into levels (corresponding to the levels of the planning graph) and binary constraints only exist between variables at the same or adjacent levels. If there are L levels, then at level $\frac{L}{2}$, i.e. after instantiating half of the problem variables, we can already apply approximately half of the constraints. In a random problem, when half of the variables have been instantiated, we can only apply approximately one quarter of the constraints. We formalize this idea in the following definition.

Definition 1. *A WCSP is* linearly incremental *under an ordering $X_1, \dots, X_n$ of its variables if, for all $p \in \{1, \dots, n\}$, the number c_p of constraints whose scopes are subsets of $\{X_1, \dots, X_p\}$ satisfies $c_p = \frac{c}{n}(p + \circ(n))$, where c is the total number of constraints.*

A random problem is not linearly incremental, since in this case $c_p = \frac{cp(p-1)/2}{n(n-1)/2}$ $= O(\frac{cp^2}{n^2})$. In the optimal planning problem under consideration in this paper, assuming for simplicity that there are the same number of variables in each of L levels and that L, n/L are both $\circ(n)$, we have, for p a multiple of n/L: $c_p = \frac{c(\frac{pL}{n}-1)}{L-1} = \frac{c}{n}(p + \circ(n))$ and hence the problem is linearly incremental.

References

1. A. Blum & M. Furst, “Fast planning through planning graph analysis”, *AI* 90 p.281-300 (1997).
2. M.C. Cooper, “Reduction operations in fuzzy and valued constraint satisfaction”, *Fuzzy Sets and Systems* 134, p.311-342 (2003).
3. S. de Givry, F. Heras, M. Zytnicki & J. Larrosa, “Existential arc consistency: Getting closer to full arc consistency in weighted CSPs”, *IJCAI 2005* p.84-89.
4. M. Do & S. Kambhampati, “Planning as constraint satisfaction: Solving the planning graph by compiling it into CSP”, *AI* 132 p.151-182 (2001)
5. P. Haslum & H. Geffner, “Heuristic Planning with Time and Resources”, *European Conference on Planning* (2001).
6. J. Hoffmann, “The Metric-FF Planning System: Translating “Ignoring Delete Lists” to Numeric State Variables”, *JAIR* 20 p.291-341 (2003).
7. J. Larrosa & T. Schiex, “In the quest of the best form of local consistency for Weighted CSP”, *IJCAI 2003* p.239-244.