Planning with Soft Regular Constraints

Alessandro Zanarini and **Gilles Pesant**
Département de génie informatique
École Polytechnique de Montréal
C.P. 6079, succ. Centre-ville, Montreal, Canada H3C 3A7
{azanarini,pesant}@crt.umontreal.ca

Michela Milano
D.E.I.S., Università di Bologna
Viale Risorgimento 2, 40136 Bologna, Italy
mmilano@deis.unibo.it

Abstract

We introduce a new approach for encoding STRIPS planning problems as Constraint Satisfaction Problems: the encoding makes use of automata for modelling the dynamics of the objects involved in the domain. We describe a total order planner based on Constraint Programming that takes advantage of this encoding, using global constraints to model the automata and to reduce the search space significantly. The planner can be easily extended, using soft global constraints, in order to deal with preferences among the goals in infeasible problems. The soft planning infrastructure can also be exploited to build an effective search heuristic and to approximate the plan length.

Introduction

Constraint Programming (CP) is considered an efficient and effective paradigm for solving classical planning problems (see (Nareyek *et al.* 2005)). Several optimal parallel planners based on CP have been proposed by the research community over the years such as CPlan (Van Beek & Chen 1999), GP-CSP (Do & Kambhampati 2001), CSP-Plan (Lopez & Bacchus 2003) and they have shown their efficiency compared to other optimal parallel planners based on SAT or planning graph encodings. The common approach for CP-based planners is to encode the planning problem as a Constraint Satisfaction Problem (CSP) and to use well-known CP techniques to solve the CSP like generalized arc consistency (GAC) or conflict-based backjumping (CBJ) (see (Dechter 2003)). Most of the CP based planners encode the CSP starting from a planning graph representation and take advantage of this in order to add mutex constraints to the CSP model.

We propose a new kind of encoding that exploits automata for modelling the dynamics of the objects that are involved in the planning problem. We show how easily this model can be extended in order to take into account preferences on the goals. We show some experimental results on a preliminary version of the planner.

The remainder of the paper is organized as follow: in Section 2 we give a brief background on Constraint Programming and the regular constraint. In Section 3 we introduce the new encoding and the planner. In Section 4 we show the soft version of the planner that can handle preferences on the goals and Section 5 shows the benefits that the soft planner can bring also to speed up the search. In Section 6 we give some experimental results. Finally, in Section 7 conclusions are given.

Background

Constraint Programming

Let $X = x_1, x_2, \ldots, x_k$ be a sequence of variables with respective domains $D_1, D_2, \ldots, D_k$. We denote $D_X = \bigcup_{1 \leq i \leq n} D_i$. A *constraint* C on X is defined as a subset of the Cartesian product of the domains of the variables in X, i.e. $C \subseteq D_1 \times D_2 \times \cdots \times D_k$. A tuple $(d_1, \ldots, d_k) \in C$ is called a *solution* to C. A value $d \in D_i$ for some $i = 1, \ldots, k$ is *inconsistent* with respect to C if it does not belong to a tuple of C, otherwise it is *consistent*. C is *inconsistent* if it does not contain a solution. Otherwise, C is called *consistent*. A constraint is called a *binary constraint* if it is defined on two variables. If it is defined on more than two variables, we call C a *global constraint*.

The solution process of constraint programming interleaves *constraint propagation* and *search*. The search process essentially consists of enumerating all possible variable-value combinations, until we find a solution or prove that none exists. We say that this process constructs a *search tree*. To reduce the exponential number of combinations, *constraint propagation* is applied to each node of the search tree: Given the current domains and a constraint C, remove domain values that do not belong to a solution to C. This is repeated for all constraints until no more domain value can be removed. Very efficient algorithms have been developped to remove such values from constraints, exploiting their structure (see (Apt 2003) and (Milano & Trick 2003) for further explanation).

Automaton and Regular Constraint

A deterministic finite automaton (DFA) may be described by a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where Q is a finite set of states, Σ is an alphabet, $\delta : Q \times \Sigma \rightarrow Q$ is a partial transition function, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of final (or accepting) states. Given an input string, the automaton starts in the initial state q_0 and processes the string

one symbol at a time, applying the transition function δ at each step to update the current state. The string is accepted if and only if the last state reached belongs to the set of final states F. The languages recognized by DFA's are precisely regular languages.

A $Regular$ constraint (Pesant 2004) is specified using a deterministic finite automaton that describes the regular language to which the sequence of values taken by the sequence of variables must belong. That automaton is then unfolded into a layered directed graph where vertices of a layer correspond to states of the automaton and arcs represent variable-value pairs. This graph has the property that paths from the first layer to the last are in one-to-one correspondence with solutions of the constraint. The existence of a path through a given arc thus constitutes a support for the corresponding variable-value pair.

As the cost-variant of the $Regular$ constraint, $CostRegular(X, \Pi, z, C)$ holds if the values taken by the sequence of finite domain variables X spell out a word belonging to the regular language associated to the deterministic finite automaton Π, and if z, a bounded-domain continuous variable, is equal to the sum of the variable-value assignment costs given by cost matrix C. Instead of simply maintaining paths, the filtering algorithm for $CostRegular$ must consider the length of these paths, defined as the sum of the costs of individual arcs, representing $\langle$ variable, value, state $\rangle$ tuples, whose costs are given by cost matrix C. Supports do not come from just any path but rather from a path whose length falls within the domain of z. To check this efficiently, it is sufficient to compute and maintain shortest and longest paths from the first layer to every vertex and from every vertex to the last layer: if the shortest way to build a path through a given arc is larger than the upper limit of the interval for z, the arc cannot participate in a solution and can thus be removed; if the longest way to build a path through a given arc is smaller than the lower limit of that interval, the arc can again be removed. In this way, domain consistency is achieved for the variables of X. The domain of z can also be trimmed using the shortest and longest paths from the first to the last layer.

Modelling planning problems with automata

The basic idea is to model the planning problem as a set of automata. Each automaton describes the dynamics of a single object (or entity) involved in the planning problem; the states of the automata denote the states of the objects and transitions between the automata states represent actions. Intuitively, if you look at the set of current states of the automata as a whole, this corresponds to the current world state. The initial state of the world and the goals are represented by the initial state and the final state of the automata. A valid (total order) plan is a sequence of actions that is recognized by each automaton; in other words it is a sequence of actions that brings every automaton (object) from the initial state to a final state.

Consider, for example, a simple instance of the blocks world domain in which you have as the initial state $\texttt{onTable}(\texttt{a}), \texttt{onTable}(\texttt{b}), \texttt{on}(\texttt{c}, \texttt{b})$ and as the goal $\texttt{on}(\texttt{a}, \texttt{b}), \texttt{on}(\texttt{b}, \texttt{c}), \texttt{onTable}(\texttt{c})$. The problem is modelled as three automata that describe the state of the blocks. Figure a, b, c shows respectively the automata for block a, b and c (note that the automata slightly differ from the following formal definition for a reason that will be clearer later).

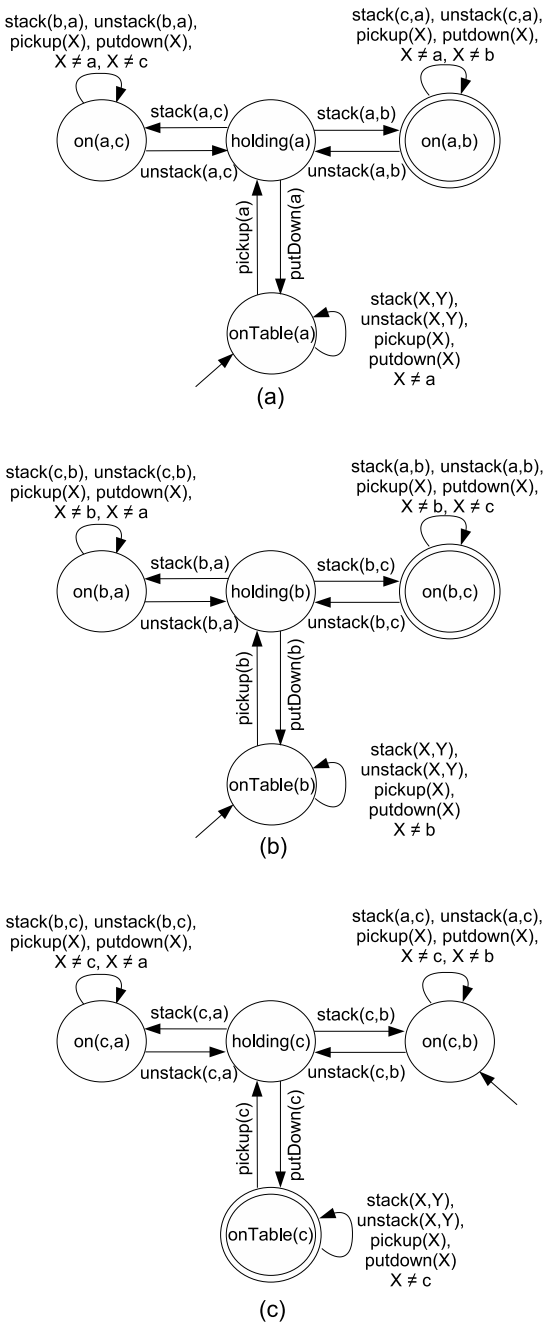

Figure 1: Automata of blocks world example.

Formally, given a plan of length L, the definition of the $CSP = (X, D, C)$ is:

- $X = (X_1, \ldots, X_L)$: a sequence of variables that repre-

sents the total order plan;

- $D = (D_1, \ldots, D_L)$: the variable domains, each domain initially contains all the possible actions.
- $C = (Regular_1, \ldots, Regular_k)$: the set of constraints defined on the set of variables; each regular constraint represents an automaton (i.e. object dynamics in the planning problem).

Given a variable X_i, the instantiation $X_i = a_j$ denotes that the action a_j should be performed in the time step i.

Given an action a_j with preconditions $Pre(a_j)$ and effects $Eff(a_j)$, we denote by $Pre_o(a_j)$ the subset of preconditions that contain the literal (object) o and analogously with $Eff_o(a_j)$ the subset of effects that contain o. We use O to denote the set of objects involved in the planning problem. Given an object $o \in O$, we write $P(o)$ for the set of all the possible propositions that involve o (i.e. the propositions that contain the literal o).

Let o be an object of the planning problem: the related automaton A_o contains one state for each possible combination of the propositions in $P(o)$; in order to simplify the notation, given a state s_k of A_o, we use s_k to denote also the conjunction of propositions represented by the state itself. A transition (action a_{s_k,s_q}) is present between two states s_k and s_q iff $Pre_o(a_{s_k,s_q}) \subseteq s_k$ and $s_q = s_k \oplus Eff_o(a_{s_k,s_q})$ (where $A \oplus B$ is defined as the operation that adds to A all the positive effects of B and deletes from A all the negative effects of B).
The global constraint $Regular_o$ is used to model the automaton A_o. Note that the variable set is constrained by several Regular constraints; this implies that an action a_j for a given time step can be performed iff it is consistent for each regular constraint i.e. the preconditions of a_j are met in each automaton; formally, $\bigcup_{o \in O} Pre_o(a_j) = Pre(a)$.
Intuitively, the regular constraints filter the domains in such a way that only the actions for which the preconditions are met, are kept in the domains. Moreover the global nature of the regular constraints allows to filter also the actions that can be hypothetically instantiated in a given time step (i.e. the action precondition are met in that time step) but that do not lead to the final states within the given plan length horizon. This kind of reasoning restricts the search space and effectively guides the search towards the goals.

Generally, the automata built in such a way, have a high number of states; since the regular constraint propagation algorithm has a complexity that is proportional to the number of the state of the underlying automaton hence it is worth minimizing the number of states of the automata.

The example presented above about blocks world shows the automata already minimized. Following the example, in the first time step actions like $\texttt{stack}(\texttt{b}, \texttt{X})$ are not allowed because they are pruned by the automaton c. Suppose now that $X_1 = unstack(c, b)$: the automaton c will reach the state $\texttt{holding}(\texttt{c})$ while the automata a and b will remain in the initial state. After this instantiation, the automaton c will filter from the domain of the variable X_2 all the actions except $\texttt{stack}(\texttt{c}, \texttt{X})$ and $\texttt{putdown}(\texttt{c})$. As you can note, the sequence $\texttt{unstack}(\texttt{c}, \texttt{b}), \texttt{putdown}(\texttt{c}), \texttt{pickup}(\texttt{b}), \texttt{stack}(\texttt{b}, \texttt{c})$, $\texttt{pickup}(\texttt{a}), \texttt{stack}(\texttt{a}, \texttt{b})$ is correctly recognized by the three automata.

Description of the algorithm Given a lower bound (eventually equal to 1) and an upper bound on the plan length, the search for a valid plan is performed, following these basic steps:

- set the plan length L to the associated lower bound;
- solve the related CSP problem with a plan length equal to L;
  - if a solution is found then stop and return the optimal total order plan
  - if no solution is found then increase the plan length and solve the new CSP problem; the iteration is stopped when no valid plan is found with a length equal to the upper bound.

Softening the planner

Soft constraints (see (Petit, Régin, & Bessière 2001) for further explanation) are a convenient modeling feature to find plans that can lead us "close" to the goal, to express preferences among the goals, or in general to deal with unsatisfiable planning problems. The presented model and planner can be easily extended in order to introduce soft constraints.

In each automaton we introduce a set of transitions $t = (s_i, s_f) \in T_{fake}$ that go from each state to the final state; these transitions represent fake actions and are exploited to compute the violations. To do that, we introduce a cost function $f : T_{fake} \rightarrow \mathbb{R}^+$. In order to deal with the quantitative approach proposed with the planning description language PDDL 3.0, in which a goal is either satisfied or unsatisfied, we can use the following function:

$$\forall\, t \in T_{fake} : f(t) = \begin{cases} 0 & \text{if the transition starts from a final state} \\ 1 & \text{if the transition starts from a non final state} \end{cases}$$

Another interesting violation function can be the distance to a goal expressed as the number of remaining actions we should perform to reach it without considering the interaction with the other goals; this is equivalent to the number of states that are present in the automaton between a state and the final state. Given the function $d : S \rightarrow \mathbb{N}$ that represents the shortest sequence of action to achieve the goal then

$$\forall\, t = (s_i, s_f) \in T_{fake} : f_d(t) = d(s_i)$$

The CSP model and the planner are adapted in the following way:

- *Violation variables*: for each automaton (regular constraint) we associate a cost variable that represents the violation. A total violation variable $TotalViolation$ is added to the model as a function of the previously defined variables.
- *Cost Regular Constraints*: we use cost regular constraints to deal with automata in which there is the notion of cost associated to the transitions.

- *Objective*: a minimization objective is added to the model for the total violation variable.
- *Slack Variable*: given a plan of length L we add a variable X_{L+1} instantiated to the fake action. Clearly if all the goals are achieved within L time steps, then all the final transitions will be from final states to final states so the violation will be null. In the case in which there is at least one unachieved goal, the fake action will lead to the final states but with a corresponding violation cost.

Note that the function that relates $TotalViolation$ to the violations of single automata can be seen as a way to express preferences among the goals. Obviously if we want to give more importance to a given goal we should give it more weight in the function. The underlying CP framework also allows us to define more complex relationships between violation variables: for example, assuming we are using the distance violation function f_d, we can use a constraint that states that the absolute value of the difference between each pair of variables must be less than a given threshold: this can be seen as a way to express fairness (we do not allow a goal to be reached while another is very far from being achieved).

Consider again the example about the blocks world domain. Figure shows the automata with the associated fake actions; on the transition T_{fake} the two different violation costs are shown. Suppose that we are searching for a plan of length 4 and X_1, X_2, X_3, X_4 are instantiated respectively to $\texttt{unstack(c, b)}, \texttt{putdown(c)}, \texttt{pickup(b)}, \texttt{stack(b, c)}$. The automaton of the block a will be in the state $\texttt{onTable(a)}$, the one for the block b in the state $\texttt{on(b, c)}$ and the one for the block c in $\texttt{onTable(c)}$. The final fake action will not change the state for the automata b and c but it will allow to find a solution since it will lead the automaton a to the final state. According to the two proposed violation functions we will get a violation equal to 1 (for PDDL-like violation) or equal to 2 if we consider the distance (we have to perform two more actions to reach the final state).
Once the violation functions are defined, then it is possible to define preferences among the goals using the objective function and/or posting constraints directly on the single violation variables. In the example, let V_a, V_b and V_c be the three violation variables and let the upper bound on the plan length be equal to 5; if we define the total violation TV as $TV = V_a + 2 * V_b + 2 * V_c$ (it is preferable to have $\texttt{on(b, c)}, \texttt{onTable(c)}$ satisfied than $\texttt{on(a, b)}$) then we will get as the final plan $\texttt{unstack(c, b)}, \texttt{putdown(c)}, \texttt{pickup(b)}, \texttt{stack(b, c)}, \texttt{pickup(a)}$. Note that $V_a = 1$, $V_b = 0$ and $V_c = 0$ hence $TV = 1$; not satisfying the goals for the block c or b will bring an increase of the total violation by at least 2 hence the solution found is optimal.

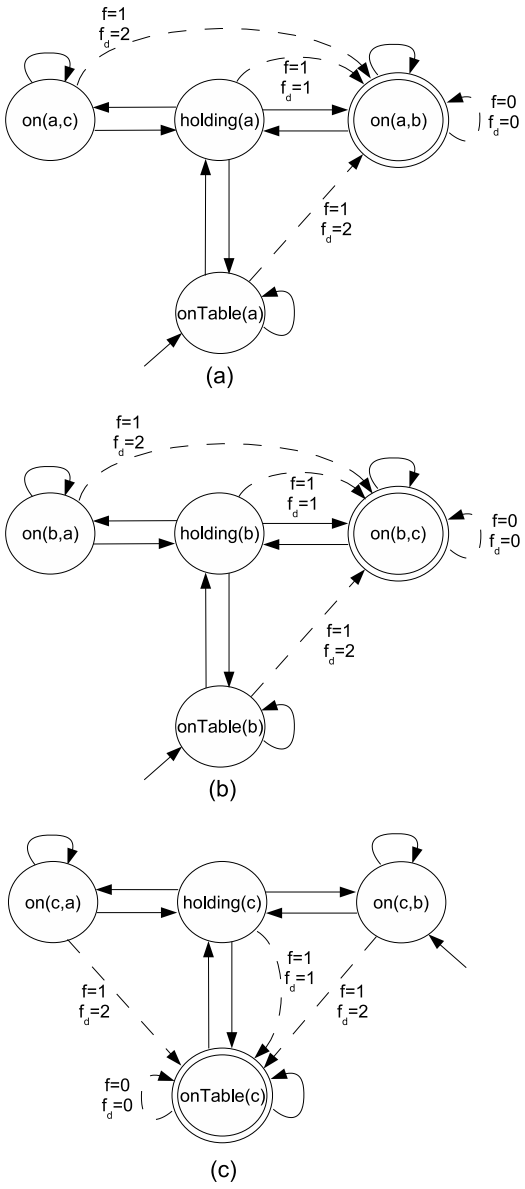

Figure 2: Soft automata for blocks world. The fake actions are shown with dashed lines.

Further Advantages of this (soft) planner

Building a search heuristic from the soft planner

In many traditional planners every time a valid plan is not found, the plan length is increased and the search is restarted. Clearly, with this approach, we revisit a large part of the search space at each iteration. However, we can exploit soft planning infrastructure to build a heuristic to speed up the search in soft and also traditional planning problems.

We use the distance based violation function and we search for a plan that is as close as possible to the goals, that is it minimizes the total violation variable. For this variable we propose two functions: $TotalViolation_{sum} = \sum_i Violation_i$, the sum of the violations from individual automata, and $TotalViolation_{max} = max_i(Violation_i)$, the greatest individual violation. Once we prove that there is no valid plan of length L (i.e. $TotalViolation > 0$) we store the best solution found and exploit it for the next itera-

tion, in two ways:

- *heuristic*: we branch first on the same values as in the stored solution, in order to quickly arrive to a promising region of the search space. Note that the proposed heuristic slightly differs from the one proposed in (Bonet & Geffner 2001); in that approach the chosen action is the one that leads to a state that is as close as possible to the goal but with the strong assumption of considering the goals independent and without considering the interference among the actions. In our approach the stored solution brings us as close as possible to the goal considering the goal interations and the interference among the actions. Even though we have not compared experimentally the two heuristics with this planner, we believe that our solution should be more effective.
- *violation bound*: the violation cost of the best solution found in the previous iteration is a valid upper bound on the violation variable. Clearly this will help to prune the search space better during the current iteration.

Plan length increase approximation

In the basic algorithm, when no solution is found for a given plan length, the plan length is increased by 1. Again, the soft planner gives us some information that we can use to get a closer approximation of the plan length.

Consider the soft planner with the distance based violation function and total violation equal to the max of the automata violations. Since we are minimizing the total violation, the cost of the best solution found indicates the minimum number of additional actions we should perform to achieve the goals. Hence, given a plan length L_i in iteration i and the best total violation found $TotalViolation_i^*$ at iteration i, we can set the plan length of the following iteration to $L_{i+1} = L_i + TotalViolation_i^*$.

Proposition 1. *The planner with the plan length increase approximation is optimal.*

Proof. Let L_i be the length of the plan at iteration i, P_i^* the best plan at iteration i with $TotalViolation_i^* > 0$ where the total violation is computed using the max function over the distance based violations. Suppose that there exists an optimal valid plan P^* of length $L^* < L_i + TotalViolation_i^*$. Consider then the partial plan $P_{partial}^*$ in which the first L_i actions are equal to the plan P^*. With $P_{partial}^*$ all the goals can be achieved with a number of actions at most equal to $L^* - L_i < TotalViolation_i^*$. So $P_{partial}^*$ has a violation strictly less than P_i^*, hence P_i^* is not the best plan with length L_i. □

Experimental results

The planner was implemented in ILOG Solver 6.1. To illustrate its behavior, we report preliminary experiments on reduced instances of the Zeno Travel problem (see (ICAPS06 2006)) in which two airplanes (A1 and A2), two persons (P1 and P2) and four cities (C1, C2, C3 and C4) are present. The instance has been modelled with four automata representing the two airplanes and the two persons. The violation function used is f_d that considers the distance to the final state of the automata. We consider different goals in order to test the soft and hard planners; the following table shows the different instances in terms of initial state and final state:

	A1	A2	P1	P2
Instance 1	C3→C2	C3→C3	C3→C1	C3→C2
Instance 2	C3→C4	C3→C3	C3→C1	C3→C2
Instance 3	C3→C2	C1→C3	C2→C1	C3→C2

The tests were performed on a Pentium-M 1.6GHz with 1GB RAM; the following table shows the results (plan length and the time expressed in seconds for finding a feasible plan) for solving the instances with the soft planner and the hard (traditional) planner; some basic techniques for breaking the symmetries have been introduced in the planners.

	Plan Length	Hard Planner	Soft Planner
Instance 1	6	1.6	1.5
Instance 2	7	23.4	3.9
Instance 3	8	23.2	5.2

We tested the impact of the different features of the soft planner on an instance with a feasible plan of length 9. The following table show the results (B: violation bound, P: plan length increase approximation, S: basic symmetry breaking techniques, H: search heuristic, Back: backward search):

Planner features	Time	Planner Features	Time
Basic	153.3	+B+P+S+H	34.7
+B	160.0	+B+P+H+Back	28.9
+B+P	141.3	+P+S+H+Back	25.3
+B+P+S	123.8	+B+P+S+H+Back	24.3

In these preliminary tests, we can see that the search heuristic allows an interesting performance boost and in general each proposed feature brings some performance increase.

In order to experiment with preferences, we defined some preferences on the goals of the first 3 instances: particularly, we expressed the total violation as $TV = V_{A1} + V_{A2} + 2 * V_{P1} + 2 * V_{P2}$ where V_{A1}, V_{A2}, V_{P1} and V_{P2} are the violations of the airplanes and of the persons; with this objective function the preference is clearly given to the persons. We used the distance based violation function for the single violations and we searched for the best plan with a tighter plan length upper bound:

With a tighter upper bound on the plan length, it was not possible to satisfy all the goals hence the plan with the minimum violation has been found. The time for solving the instances with preferences is comparable to the time for solving instances without preferences (with the same plan length). Note that the flexibility of the framework allowed us to introduce preferences among the goals simply by adding an ad hoc objective function.

	Plan Length	Time	Violations
Instance 1	4	2.2	4
Instance 1	5	4.4	2
Instance 2	5	8.8	3
Instance 2	6	6.1	1
Instance 3	6	2.0	2
Instance 3	7	3.9	1

Discussion and open issues

The presented encoding raises one main challenge: it is not always obvious how to choose the set of objects to fully and correctly model the problem (actually we could choose each entity of the planning problem but then the number of automata would become intractable). Furthermore, the termination condition for the soft planner is actually given by achieving all the goals or reaching the upper bound on the plan length. Both issues are currently under investigation.

An interesting aspect that we are currently studying, is the introduction of no-goods recording in the planner. Most of the current state-of-the-art planners showed that it is a very powerful method to improve the performance of the planner.

Another aspect to investigate in future studies is the introduction of stronger symmetry breaking techniques. It is well known that total order plans present a lot of symmetries (two or more actions can be executed in whatever order) and this can degrade the performance in cases where there is no valid plan for a given plan length. In order to prove the infeasibility of a problem for a given plan length, the actual planner explores all the search space while symmetry breaking methods can help to reduce it significantly.

To the best of our knowledge, no proposed CP-based planner exploits global constraints that are commonly known as a powerful tool to speed up the search. The contributions of this paper are:

- a new encoding for the planning problem;
- use of global constraints for solving planning problems;
- a new violation measure for the soft regular constraint;
- a CP-based planner that provides tools to express preferences on goals;
- exploitation of the soft planning infrastructure for building an effective heuristic.

The implementation of the planner in ILOG Solver (probably the best CP framework commonly used by the research community and by industry) will allow us to introduce and exploit several of the sophisticated techniques that the CP community has proposed. Even if the proposed planner has some limitations, the actual implementation and the possible improvements that can be introduced to speed up the search seem promising.

References

Apt, K. 2003. *Principles of Constraint Programming*. Cambridge University Press.

Bonet, B., and Geffner, H. 2001. Planning as Heuristic search. *Artificial Intelligence* 129:5–33.

Dechter, R. 2003. *Constraint Processing*. Morgan Kaufmann Publishers.

Do, M. B., and Kambhampati, S. 2001. Planning as constraint satisfaction: Solving the planning garph by compiling it into CSP. *Artificial Intelligence* 132:151–182.

ICAPS06. 2006. Workshop on Preferences and Soft Constraint in Planning. *http://www.cis.strath.ac.uk/derek/PSCinP.html*.

Lopez, A., and Bacchus, F. 2003. Generalizing GraphPlan by Formulating Planning as a CSP. *International Joint Conference on Artificial Intelligence IJCAI-2003* 954–960.

Milano, M., and Trick, M. 2003. *Constraint and Integer Programming - Toward a Unified Methodology*. Kluwer Academic Publishers.

Nareyek, A.; Freuder, E. C.; Fourer, R.; Giunchiglia, E.; Goldman, R. P.; Kautz, H.; Rintanen, J.; and Tate, A. 2005. Constraints and AI Planning. *IEEE Intelligent Systems* 20:62–72.

Pesant, G. 2004. A Regular Language Membership Constraint for Finite Sequences of Variables. In *Principles and Practice of Constraint Programming – CP-2004: Proceedings of the Tenth International Conference*. Springer-Verlag LNCS 3258. 482–495.

Petit, T.; Régin, J.-C.; and Bessière, C. 2001. Specific Filtering Algorithms for Over Constrained Problems. In *Principles and Practice of Constraint Programming – CP-2001: Proceedings of the Seventh International Conference*. Springer-Verlag LNCS 2239.

Van Beek, P., and Chen, X. 1999. CPlan: A Constraint Programming Approach to Planning. *Proceedings of the 16th National Conference on Artificial Intelligence* 585–590.