On State Management in Plan-Space Planning from CP Perspective

Pavel Surynek

Charles University
Faculty of Mathematics and Physics
Malostranské náměstí 2/25, 118 00 Praha 1, Czech Republic
surynek@ktiml.mff.cuni.cz

Abstract

In this paper we address the problem of encoding of plan-space planning problems as a CSP. We propose a constraint model for expressing plan-space partial plans. As an enrichment of the notion of partial plans we propose a world state management within the partial plan. We incorporate the world state management into our CSP model too. Next we propose a specialized algorithm for solving the CSP model. The algorithm builds the constraint model dynamically as conflicts are resolved in the model. The performed preliminary experiments showed the usefulness of state management in the constraint model. The usage of state management allows the additional search space pruning compared to the model without states.

Introduction

Planning is a widely studied topic of artificial intelligence. The importance of studying planning arises from needs of real-life applications such as industrial automation, transportation, robotics and other braches (Nau, Regli, Gupta, 1995). The research in planning is also motivated by the needs of researches in other areas. The most prominent example from the above areas is space exploration where autonomous spacecrafts (Bernard et al., 1998) and vehicles (Ai-Chang et al., 2004) are successfully used. But autonomous devices can be used in many other situations both in science and real-life. The autonomous behavior is controlled by planning techniques and algorithms in many of these cases.

From the traditional view of planning, the planning problem is posed as finding of a sequence of actions which transform a specified initial state of the planning world into a desired goal state of the world (Allen, 1990). The limitation is that only actions from a set of allowed actions can be used. The action typically makes a small local change of the world in which the task takes place.

There are several paradigms for solving planning problems. We deal with so called *state-space* planning (Fikes, Nilsson, 1971) and *plan-space* planning (Sacerdoti, 1975) in this paper. The state-space planning represents a more traditional approach for solving planning problems while the plan-space planning is a newer method, which is currently not so spread.

Although the plan-space planning provides a flexible and intuitive paradigm for solving planning problems, it is less competitive to the recent traditional state-space planning techniques in situations where heuristics play a role. As it is evidenced by the results of the 2004 International Planning Competition, several successful heuristic based planning algorithms (*HSP*, *TP-4*) use state-space approach (Haslum, Geffner, 2001; Bonet, Geffner, 2001). The most significant drawback of plan-space planning is caused mainly by the fact that domain specific heuristics are often based on extracting information from the states of the planning world (Ghallab, Nau, Traverso, 2004). The extraction of information cannot be done in straightforward way within the plan-space framework. On the other hand, another successful planner (*CPT*) in IPC 2004 uses ideas from plan-space approach and from constraint programming (Vidal, Geffner, 2004). This fact was one of our motivations to deal with plan-space planning approach.

The classical state-space planning is based on the notion of an evolution of states of the planning world by planning operators. A given world state is changed by planning operators into new ones. A state based solving algorithm for planning problems performs this evolution in some kind of a controlled way until a required goal state of the planning world is reached. The states of the planning world are explicitly available along the whole resolution process.

Algorithms for solving planning problems within the plan-space paradigm work in a completely different way. They usually refine so called *partial plan* until a certain level of consistency is enforced. The partial plan contains a set of *planning operators* and a set of precedence and binding relations over these operators which the final plan has to satisfy. When the partial plan becomes consistent it is ensured that a valid plan (sequence of fully instantiated planning operators, i.e. *actions*) can be extracted from it. The main difference from state-space planning is the lack

of explicit planning world states during the resolution process. This fact represents the key obstacle in adaptation of domain specific heuristics for plan-space planning paradigm.

Our work is motivated by a number of successful attempts to use constraint satisfaction techniques in planning (Nareyek et al., 2005). We want to propose a framework in which the complete planning problems can be expressed. The design of the framework should allow the use of advanced constraint satisfaction techniques. Our goal is also to preserve world states to be directly accessible along the reasoning process. The explicit presence and accessibility of the world states will allow us to reduce uncertainty in the framework. Moreover, when we have explicit world states, the domain specific heuristics can be adapted more easily.

Many approaches exploiting constraints in planning are based on *Graphplan* algorithm (Blum, Furst, 1997). Graphplan-based algorithms use some kind of compression of many world states into a single structure. Another approach represents *CPlan* algorithm (Van Beek, Chen, 1999). The CPlan algorithm uses state-space based encoding of the whole planning problem up to a certain number of steps. Compared to Graphplan-based algorithms the CPlan algorithm provides an easier access to the states of the planning world along the resolution process. It is enabled by direct encoding of the world states into the model.

Our work is primarily inspired by the CPlan algorithm. But contrary to CPlan we are trying to use state encoding within the plan-space framework. More precisely, we address a problem of integration of state management into the plan-space framework. We formulate partial plans as a *dynamic constraint satisfaction problem* where states of the planning world are also modeled. We propose a specialized search algorithm for this model. Our search algorithm dynamically changes the constraint model according to the applicable refinements until a partially consistent (in our case *arc-consistent*) partial plan is obtained. Then the algorithm continues with trying to extract a valid plan from the consistent partial plan while it is still applying refinements if it is necessary.

The state management in the constraint model of the partial plan allows the search algorithm to reduce uncertainty in the model and to deduce more information about the precedence relations in the partial plan. This leads into a stronger search space reductions and the final fully consistent partial plan is found earlier. The presence of explicit planning world states in the constraint model can be utilized for an easier integration of domain specific heuristics into the search algorithm.

The paper is organized as follows. State-space and plan-space planning paradigms are introduced in the first section. Then we describe how the constraints are used in plan-space planning and the details of our constraint model. Having the constraint model, we design a specialized search algorithm for the model subsequently. Finally we show some empirical results.

State-Space and Plan-Space Planning

The classical formalism for stating planning problems is based on a first order language with finitely many predicate and constant symbols, and infinitely many symbols for variables. There are no function symbols in the language (Ghallab, Nau, Traverso, 2004).

The classical statement of a *planning problem* is a triple $P = (O, s_0, g)$, where s_0 is an *initial state* of the planning world, g is a *goal condition* and O is a finite set of *planning operators*. The initial state s_0 is expressed as a finite set of ground atoms. It is interpreted as a set of propositions that hold in the planning world at the beginning. The goal condition g is a finite set of literals. Similarly, it is interpreted as a set of proposition we want to make true (compared to initial state we allow negative propositions). Finally, O is a finite set of planning operators, through which it is possible to locally change the world states. The planning operator is a triple $o = (name(o), precond(o), effects(o))$, where $name(o)$ denotes the name of the operator, $precond(o)$ is an operator precondition and $effects(o)$ is an effect of the operator. Precondition is modeled as a finite set of atoms which must hold in the world state before the operator can be applied. The effect of the operator is modeled as a finite set of literals that hold in the state after the operator is applied. The task is to find a sequence of ground planning operators (*actions*) such that it transforms the given initial state into a state satisfying the given goal condition. We call this sequence a *plan*.

The required sequence of actions is searched directly within the state-space planning. The search process starts from the initial or from the goal state and builds the sequence of actions by adding a new action at the end of the already finished partial sequence of actions. The current state of the world corresponding to the application of the partial sequence of actions on the initial state or on the goal state is maintained along the whole search process.

In plan-space planning, we work with so called partial plans. A *partial plan* is a tuple $\pi = (A, \prec, B, L)$ (Ghallab, Nau, Traverso, 2004), where A is a finite set of partially instantiated planning operators present in the partial plan, $\prec$ is a finite set of precedence constraints on the elements of A, B is a set of binding constraints, which bind variables appearing in the planning operators in A, and L is a set of *causal links*. A causal link is a relation of the form $a_i \xrightarrow{p} a_j$, where $a_i, a_j \in A$ and a_i has an effect p, which is required as a precondition for a_j.

CSP and Plan-Space Planning

The structure of the partial plan is very close to the structure of constraint satisfaction problem. Both structures use the notion of a constraint between a set of objects.

A *constraint satisfaction problem* (CSP) is a triple (X, D, C) (Dechter, 2003), where X is a finite set of variables, D is a finite domain of values for the variables from X and C is a finite set of constraints over the vari-

ables from X. The constraint can be an arbitrary relation over the elements of the domains of its variables. Having a constraint satisfaction problem the task is to find an assignment of values to the variables such that all the constraints are satisfied.

In our framework for plan-space planning we model partial plans as constraint satisfaction problems. Actions occurring in the partial plan are modeled as CSP variables whose domains are sets of possible moments of execution of the corresponding action. In the following sections we will denote these variables as *time variables*. With time variables it is easy to translate temporal precedence relations directly into constraints of the constructed CSP model. Since we will be working only with actions (ground operators) we do not need to translate binding relations into constraints of our CSP model (note that we would also need other types of variables if we want to model partially instantiated operators). Note that this kind of constraint satisfaction problem can be solved in polynomial time (Dechter, Meiri, Pearl, 1991). Such simplified problem does not exploit the full generality of CSP which is NP-hard in general. However, our constraint model will be extended with other types of variables and constraints in the following sections.

Since the standard resolution process over the partial plan changes the structure of the plan, we also need to reflect this dynamicity in our CSP model. Typically plan-space algorithms insert new actions and relations into the partial plan as the search proceeds. We can model these changes as incremental additions of variables and constraints into and from our CSP model.

Partial Plans as an Incremental Dynamic CSP

A *dynamic constraint satisfaction problem* (DCSP) (Dechter, 1988) is a sequence of constraint satisfaction problems $P_0, P_1, P_2, \ldots, P_\alpha, \ldots$, where each problem is a result of a modification of the preceding one. The allowed modifications are constraint addition and retraction, and variable addition and removal. The evolution of the dynamic problem is controlled by outside mechanism, for example by the interaction with the user. The controlling mechanism is the solving algorithm itself in our case with partial plans. All the static problems from the sequence must be solved to solve the whole dynamic problem.

Since we are doing only incremental changes in our model we do not need the general dynamic constraint satisfaction problem as given above but only an incremental restriction of DCSP. We also do not follow the above concept exactly in the sense of finding a solution. It is sufficient to solve only the last constraint satisfaction problem from the sequence of the dynamic problem.

Constraint Model

Our constraint model is always built for a limited maximum length of the resulting sequence of actions. Let l be this length. First we will show how actions are stated within the model. Let us consider a ground operator $a = (name(a), precond(a), effects(a))$. For the action a we introduce a time variable $time_a = \{1,2,\ldots,l\}$ into the constraint model*. Semantically the value of the time variable describes the time of the execution of the corresponding action in the final plan. When we are working with non-durative actions only, the value of the corresponding time variable determines the position of the action in the resulting plan. By using time variables we can cover the presence of actions in the partial plan. We define the following bounds for each time variable:

Earliest execution time of action a: $et(a) = minimal\ element\ in\ the\ current\ domain\ of\ time_a$

Latest execution time of action a: $lt(a) = maximal\ element\ in\ the\ current\ domain\ of\ time_a$

Incorporation of States into the Model

For the purposes of more compact representation within the constraint satisfaction problem we require that preconditions and effects of the action are expressed functionally using state variables. More specifically instead of saying that some proposition about some object is true in a given world state, we say that the function describing corresponding property of the object takes a certain value in the given world state. The situation is explained in the following examples. In the example 1 a classical representation of an action is showed.

Example 1. Consider an action that moves robot_1 from location_A to location_B, where robot, location_A and location_B are constants with obvious meaning. We also use a binary predicate *at* saying that the first argument is at the place specified by the second argument. The classical representation of the move action is following:

$$(move - robot\_1 - location\_A - location\_B;\ precondition = \{at(robot\_1, location\_A)\};\ effects = \{at(robot\_1, location\_B),\ \neg at(robot\_1, location\_A)\}).$$

To express the move action functionally we need a function $location : robots \times S \longrightarrow locations$, where *robots* and *locations* are sets of constants for which $robot\_1 \in robots$ and $\{location\_A, location\_B\} \subseteq locations$, and S is the set of states. Let us call this function a *state variable function*. The following example shows the action represented using state variable function.

Example 2. Representation of the action moving robot_1 from location_A to location_B using state variable function.

$$(move - robot\_1 - location\_A - location\_B;\ precondition = \{location(robot\_1) = location\_A\};\ effects = \{location(robot\_1) = location\_B\}).$$

* We allow a certain action to appear multiple times in the partial plan.

With each action in the partial plan we associate a set of *precondition variables* and a set of *effect variables*. Together we will call these variables *world state variables*. The set of precondition variables contains one variable for each state variable function describing properties of a certain object in our planning world. Similarly for the set of effect variables, this set also contains one variable for each state variable function in the constraint model. The domains of precondition and effect variables are the same as the ranges of corresponding functions. The situation is explained in more details in the following example.

Example 3. Again consider the action that moves robot_1 from location_A to location_B. In addition we have a new constant robot_2 in this example. There is also the state variable function $location : robots \times S \longrightarrow locations$, where $\{robot\_1, robot\_2\} \subseteq robots$ and $\{location\_A, location\_B\} \subseteq locations$. We introduce following state variables into the constraint model:

$$precond^{move-robot\_1-AB}_{location(robot\_1)} = \{location\_A, location\_B, \ldots\}$$
$$precond^{move-robot\_2-AB}_{location(robot\_2)} = \{location\_A, location\_B, \ldots\}$$
$$effect^{move-robot\_1-AB}_{location(robot\_1)} = \{location\_A, location\_B, \ldots\}$$
$$effect^{move-robot\_2-AB}_{location(robot\_2)} = \{location\_A, location\_B, \ldots\}.$$

The set of precondition variables describes the world state before the corresponding action is applied. Similarly the set of effect variables describes the world state after the application of the action. The initial state and the goal condition are handled as special actions.

How the Model is Constrained

The above simple collection of variables is sufficient to describe all the objects and possible transitions that can appear within the modeled partial plan. But it is useless without the appropriate set of constraints that would forbid incompatible tuples of values in the domains of variables. A lot of constraints arise directly from the definition of partial plans.

State Transition Constraints. The first type of constraints we introduce into our model ensures that state transitions are reflected in the world state variables. Consider an action and corresponding world state variables that represents the world state before and after the execution of the action. Suppose that the action requires the state variable function f to take the value x before the action can be executed. Then the state transition constraint must ensure that the precondition world state variable corresponding to the state variable function f takes the value x. Next, suppose that the action changes the value of the state variable function f to the value y. Then the state transition constraint ensures that the effect world state variable corresponding to the state variable function f takes the value y. Since these constrains are unary, the information contained in them can be directly encoded into domains of world state variables. The following example shows encoding of state transitions into the variable domains.

Example 4. Let us have the same action and the same constants as in the example 3. We can restrict the domains the world state variables as follows:

$$precond^{move-robot\_1-AB}_{location(robot\_1)} = \{location\_A\}$$
$$effect^{move-robot\_1-AB}_{location(robot\_1)} = \{location\_B\}.$$

The world state variables, whose state variable function does not appear in preconditions or in the effects of the actions, remain unchanged.

Frame Axiom Constraints. State variable functions that do not occur in the effects of the particular action do not change its value after execution of this action. This fact is not implied by the state transition constraints and must be ensured separately. Generally for this reason so called *frame axioms* are used. In our constraint model we also use constraints for encoding frame axioms. The frame axiom constraint ensures that a certain precondition world state variable and the corresponding effect world state variable have the same values. The frame axiom constraints are added for all pairs of precondition and effect variables whose state variable function is invariant with respect to the corresponding action.

Example 5. Let us have the same action and the same constants as in the example 3. The frame axiom constraint is the following:

$$precond^{move-robot\_2-AB}_{location(robot\_2)} = effect^{move-robot\_2-AB}_{location(robot\_2)}.$$

Suppose that there is a frame axiom constraint for action a and state variable function f, then the propagator for the constraint is:

$$precond^a_f \leftarrow precond^a_f \cap effect^a_f$$
$$effect^a_f \leftarrow precond^a_f \cap effect^a_f.$$

Unique Action Time Constraints. Each action in the partial plan has to be executed at a unique time. Our constraint model is designed to produce a plan as a totally ordered sequence of actions, so this is a natural requirement. Another reason for this requirement is that we want to reduce uncertainty in the model as much as possible. It is important for the next type of constraints which are used to manage evolution of states. The unique action time constraint is modeled as *allDifferent* constraint over time variables. Suppose that there are actions $a_1, a_2, \ldots, a_k$ in the partial plan. Then we introduce a constraint $allDifferent(time_{a1}, time_{a2}, \ldots time_{ak})$ into the constraint model. The propagator for the *allDifferent* constraint is described in (Régin, 1994).

State Sequencing Constraints. As the current domains of time variables are narrowed during the search process, there arise situations when it is necessarily true that two actions must be executed consecutively. More specifically, an action must be performed right after another

action if all the other actions are scheduled before or after this pair of actions.

Consider that there are actions a_i and a_j in the partial plan and it is known that these two actions must be executed consecutively in the resulting plan. Without loss of generality suppose that a_j must be executed right after a_i. Then it is possible to post a constraint that binds precondition world state variables belonging to the action a_j to effect world state variables belonging to the action a_i. This idea can be used conversely too. Consider that it is known that effect world state variables of one action cannot be bind to the precondition world state variables of the other action. In such situation we can conclude that these two actions cannot be executed consecutively and it is possible to post a constraint into the model that ensures this fact.

Now we can propose a constraint that realizes the above statements. We will call it a *state sequencing constraint*. It comprises a condition enforcing the equality of effect world state variables and precondition world state variables when the corresponding two actions must be executed consecutively. And next it comprises a condition which does not allow the execution of one action right after the other action if its precondition and effect world state variables are not compatible. The situation is explained in the following example.

Example 6. Let us have an action that moves robot_1 from location_A to location_B and an action that moves robot_1 from location_B to location_C. Then the state sequencing constraints look like as follows:

if $time_{moveBC} = time_{moveAB} + 1$ **then**
$effect^{moveAB}_{location(robot\_1)} = precondition^{moveBC}_{location(robot\_1)}$

if $time_{moveAB} = time_{moveBC} + 1$ **then**
$effect^{moveBC}_{location(robot\_1)} = precondition^{moveAB}_{location(robot\_1)}$.

Since the right hand side of the conditional statement of the second constraint cannot be satisfied, the action *moveAB* cannot be executed right after *moveBC* action.

The propagator for state sequencing constraint is similar to that of frame axiom constraint. Suppose that there are actions a_i and a_j in the partial plan, and state variable function f. Then the propagator is:

if $time_{aj} = time_{ai} + 1$ **then**
$effect^{ai}_f \leftarrow precondition^{aj}_f \cap effect^{ai}_f$
$precondition^{aj}_f \leftarrow precondition^{aj}_f \cap effect^{ai}_f$

if $time_{ai} = time_{aj} + 1$ **then**
$effect^{aj}_f \leftarrow precondition^{ai}_f \cap effect^{aj}_f$
$precondition^{ai}_f \leftarrow precondition^{ai}_f \cap effect^{aj}_f$

if $effect^{ai}_f \neq precondition^{aj}_f$ **then**
propagate $time_{aj} \neq time_{ai} + 1$

if $effect^{aj}_f \neq precondition^{ai}_f$ **then**
propagate $time_{ai} \neq time_{aj} + 1$.

The reason for having state sequencing constraints is not for maintaining world states itself. The whole state management is intended to work reversely. The state sequencing constraint can discover that two actions cannot be consecutive. If we add the fact that the maximum plan length is usually very tight this mechanism allows a significant pruning of current domains of time variables. It is preferred to post as many as possible state sequencing constraints into the model and therefore it is also important to have the above unique action time constraints.

Precedence Constraints. Until now we did not reflect causal links in our constraint model. If there is a causal link $a_i \xrightarrow{p} a_j$ in the partial plan, it is necessary that the action a_i is executed before the action a_j. This relation is simply expressed through a *precedence constraint* between corresponding time variables:

$time_{ai} < time_{aj}$.

The propagator for precedence constraint is the following:

$lt(a_i) \leftarrow lt(a_j) - 1$
$et(a_j) \leftarrow et(a_i) + 1$.

Threat Resolving Constraints. A *threat* for a causal link $a_i \xrightarrow{p} a_j$ is an action in the partial plan that changes the value of a state variable function to the value that is not compatible with the effect p of the action a_i. To resolve the threat it is necessary to ensure that it is executed before both a_i and a_j or after both a_i and a_j. Since the actions a_i and a_j are already ordered by precedence constraint is sufficient to ensure the execution of threat action before a_i or after a_j. Suppose that a_k is a threat for causal link $a_i \xrightarrow{p} a_j$. The corresponding *threat resolving constraint* is:

$(time_{ak} < time_{ai}) \vee (time_{aj} < time_{ak})$.

A propagator for threat resolving constraint is following:

if $lt(a_i) < et(a_k)$ **then**
$lt(a_j) \leftarrow lt(a_k) - 1$, $et(a_k) \leftarrow et(a_j) + 1$

if $et(a_j) > lt(a_k)$ **then**
$lt(a_k) \leftarrow lt(a_i) - 1$, $et(a_i) \leftarrow et(a_k) + 1$.

The Utility of State Management in the Model

The proposed constraint model is built dynamically as the solving algorithm proceeds. During solving process domains of variables in the constraint model are narrowed. It is the standard process how the constraint satisfaction problems are solved.

The narrowing of variable domains results into discovering of new conflicts in the partial plan (for example a threat or unsatisfied precondition of the already present action is discovered). The conflicts are resolved by the extension of the model with new variables and constraints. This may prolong the resulting plan. The extension of the model is forbidden when the maximum allowed length l of the resulting plan is reached. The final

goal of a solving algorithm is to find the total ordering of actions in the plan.

The total ordering of actions in the plan is found when the domains of all the time variables become singleton. In order to reach this we want to prune the domains of time variables as much as possible. The state management in the constraint model helps the search algorithm to discover additional inconsistent values in the domains of time variables and to rule them out.

The situation when the state management is useful is showed in figure 1. Note the propagation of effects in the chain of actions through frame axiom constraints. If there would not be the action that loads the package X at location A in the partial plan, the remaining two actions can be executed consecutively.

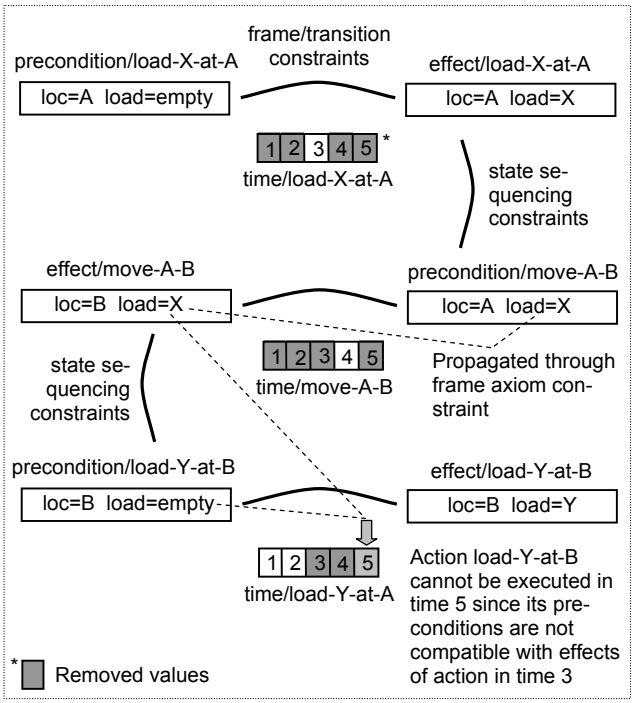

Figure 1: Constraint propagation through world state variables

Algorithm

In this section we describe an algorithm for solving the proposed constraint model. Our algorithm is a combination of standard backtracking based labeling algorithm for solving constraint satisfaction problems and a special resolution mechanism which dynamically extends the constraint model when a flaw in the partial plan is discovered. Along the whole search process the constraint model is maintained in an arc-consistent state. We are using an incremental version of a classical AC-3 algorithm (Mackworth, 1977).

In each step of the algorithm there are two possibilities what can be done. If the constraint model involves a flaw (the flaw in constraint model directly corresponds to a flaw in modeled partial plan) the algorithm can repair the flaw by extending the constraint model with new variables and constraints. The second possibility is to perform a labeling step. The labeling step consists in addition of a labeling constraint. The labeling constraint selects a value for a selected variable and distributes the original constraint satisfaction problem (Schulte, 2002) into two simpler sub-problems. The labeling constraint is positive in the first sub-problem (the selected variable must take a selected value) and negative in the second sub-problem (the selected variable must take the different value than selected). If one of the sub-problems is solved then this solution is also a solution of the original problem.

The flaw resolution step is preferred. Thus if there is a flaw in the constraint model the algorithm would try to repair it and not to label a variable.

In both cases the arc-consistency of the constraint model is restored. The maintaining arc-consistency during the search allows an earlier detection of inconsistencies.

Flaw Resolution Step

The flaw is an *open goal* or a threat. From the previous sections we already know how to resolve a threat. The treat is repaired by posting an appropriate threat constraint into the constraint model.

An open goal is a precondition of an action for that there is no causal link in the partial plan. This means it is not decided how to satisfy the precondition. Consider that the precondition p of the action a_j is an open goal. There are two possibilities how to resolve this open goal. If there is an action a_i already present in the constraint model that has the unsatisfied precondition as its effect. We can simply create a new causal link $a_i \xrightarrow{p} a_j$. After creation of a new causal link a new precedence constraint $time_{ai} < time_{aj}$ is added to the model. Although causal links are explicitly included in our constraint model via precedence constraints, it is more convenient to maintain current causal links in a separate set. Every newly created causal link has to be inserted into this set. The set of causal links can then be used during both kinds of flaw resolution.

Another way how to resolve the open goal is to add a new action into the constraint model that supports the open proposition p of a_j. The addition of a new action into the constraint model is carried out by adding a time variable and corresponding precondition and effect world state variables. It is necessary to insert appropriate set of constraints too.

Consider that we want to add an action a_i. It must have p as an effect. A time variable $time_{ai}$ and world state variables $precondition_f^{ai}$ and $effect_f^{ai}$ for all the state variable functions f are added into the model. State transition constraints are encoded directly into the domains of world state variables. Next we add frame axiom constraints for all the pairs of $precondition_f^{ai}$ and $effect_f^{ai}$ variables, where state variable function f is not affected by the action a_i. The unique action time constraint is extended over the $time_{ai}$ variable. Here a dynamic ver-

sion of *allDifferent* constraint for backtracking based environments can be used (Barták, 2003). The state sequencing constraint is added for all new pairs of actions, i.e. for all pairs of actions (a_i, a_k) where $k \neq i$ a state sequencing constraints that bind $precondition_f^{ai}$, $precondition_f^{ak}$, $effect_f^{ai}$, $effect_f^{ak}$, $time_{ai}$ and $time_{ak}$ variables for all the state variable functions f are added. Finally the precedence constraint $time_{ai} < time_{aj}$ is added.

Labeling Step

The labeling step corresponds to the standard variable labeling from CSP solving algorithms. In each labeling step a variable and a value from its current domain are selected. An ordering heuristics are used for selection of the variable and the value. Currently we use heuristics that select the variable with the smallest current domain and then the first value from its domain is selected for labeling. Let us suppose that variable x and value v is selected for labeling. Then constraints $x = v$ and $x \neq v$ are used to distribute the problem.

The deterministic implementation of the algorithm should use a special ordering heuristics to determine the most promising order of threats and open goals. The heuristics should be also used for selecting the best refinements. However, in our testing implementation we did not use any heuristic for resolution steps. Threats and open goals are tried in the order as they were discovered.

Figure 2 summarizes the whole search algorithm.

```
SolveCSPModel((V,C), I)
(V,C) ← EnforceArcConsistency((V,C))                    } arc-consistency
if exists x∈V with empty current domain then
| return FAILURE;
threats ← GetThreat((V, C))                             } threat resolution
if threats ≠ Ø then
| non-deterministically choose a
|   threat t∈ threats
| (V,C) ← ResolveThreat(t, (V,C), I)
| return SolveCSPModel((V,C), I)
else
| openGoals ← GetOpenGoals((V,C))                       } open goal resolution
| if openGoals ≠ Ø then
| | non-deterministically choose an open
| |   goal g∈openGoals
| | (V,C) ← ResolveOpenGoal(g, (V,C), I)
| | return SolveCSPModel((V,C))
| else
| | if all the variables in V are assigned then        } labeling
| | | return (V,C)
| | select a not yet assigned variable x∈V
| |   according to the variable ordering heuristic
| | select a value v from the current
| |   domain of x
| | non-deterministically choose a labeling
| | constraint c∈ {x=v, x≠v}
| | (V,C) ← (V,C∪{c})
| | return SolveCSPModel((V,C), I)
```

Figure 2: Search algorithm for solving partial plan constraint model

Preliminary Empirical Results

We have implemented the proposed constraint model and search algorithm in C++ in order to examine its properties. The goal of our evaluation is to examine the benefit of state management within the constraint model. We have compared how the algorithm works on the model with and without variables and constraints for managing states. In our preliminary experiments we have used a naive implementation of the *allDifferent* constraint. We intend to use a dynamic *allDifferent* constraint as it is proposed in (Barták, 2003) in the future.

The algorithm was tested on a simple planning problem. There were several packages and locations in the testing planning problem. There were also one or more transporters to transport packages from one location to other location. The transporter can move between locations and can load just one package. The goal was to transport packages to selected locations. We have created several testing instances of our transportation planning problem. These instances differ in the minimum plan length and in number of goals that has to be satisfied (if it is required to transport two packages there are two goals).

Our search algorithm was tested on several instances of the proposed transportation problem. In each run we have measured the number of steps of the algorithm. We have measured separately the number of resolution and labeling steps. In addition we counted the number of actions that were considered to be a part of the partial plan.

Plan length / Steps / States		4	5	6	7	8	9	10
Resolution steps	no	33	39	121	137	218	259	451
	yes	*22*	*25*	*52*	*59*	*117*	*127*	*337*
Labeling steps	no	15	19	44	55	145	163	1179
	yes	*12*	*15*	*18*	*22*	*51*	*62*	*695*
Actions considered	no	7	8	27	28	51	52	71
	yes	*4*	*5*	*10*	*11*	*21*	*21*	*49*

Figure 3: Empirical results for models with and without states

The performed experiments showed that if there is only one goal in the planning problem, the number of steps of the algorithm was almost the same no matter whether the state management was used or not. But note that if the state management is used more variables are necessary be assigned. The same situation appeared when there were more goals in the planning problem but independent on each other (this corresponds to the situation with two or more transporters and the same number of packages to transport).

We have obtained different results for problems with more than one goal which were interacting. This corresponds to the situation when there are fewer available transporters than the number of packages to transport. The state management within the constraint model seems promising in this situation. The results for problems with independent goals are listed in figure 3.

The current stage of the implementation does not allow us to confirm the corresponding improvement in time when state management is used. Our testing implementation has very different costs of the operations. So we cannot make any conclusion about the time improvement at the current stage.

Conclusion and Future Work

We studied the utility of integration of state management into plan-space planning from the constraint programming perspective. We proposed the constraint model for expressing partial plans. The world states are maintained as the actions are sequenced in the resulting plan. We propose special variables and constraints suitable for expressing such world state sequencing.

We have also designed a special solving algorithm for the proposed model. The algorithm dynamically builds the constraint model as the uncertainty is reduced. Our empirical evaluation showed that the state management within the model significantly reduces the number of steps of the algorithm in certain situations. Namely when the planning problem requires satisfying of multiple interacting goals the state management seems to be useful. We cannot say the same about the time improvement at current stage. It is a matter of further research and testing.

We plan to use the proposed state management in the constraint model for an integration of domain specific heuristics into the solving algorithm. The algorithm makes many decisions during solving the planning problem. All these decisions can be more targeted with the aid of the appropriate domain specific heuristics.

Acknowledgement

The research is supported by the Czech Science Foundation under the contract no. 201/04/1102 and by the Grant Agency of Charles University (GAUK) under the contract no. 326/2006/A-INF/MFF. I would like to thank anonymous reviewers for their useful comments.

References

Ai-Chang, M.; Bresina, J.; Charest, L.; Chase, A.; Hsu, J.; Jónson, A.; Kanefsky, B.; Morris, P.; Rajan, K.; Yglesias, J.; Chafin, B.; Dias, W.; and Maldague, P. 2004. *MAPGEN: Mixed-Initiative Planning and Scheduling for the Mars Exploration Rover Mission.* IEEE Intelligent Systems 19(1), 8-12.

Allen, J.; Hendler, J.; and Tate, A. (editors). 1990. *Readings in Planning.* Morgan Kaufmann Publishers.

Barták, R. 2003. *Dynamic Global Constraints in Backtracking Based Environments.* Annals of Operations Research, no. 118, 101-119, Kluwer.

Bernard, D.; Gamble, E.; Rouquette, N.; Smith, B.; Tung, Y.; Muscettola, N.; Dorias, G.; Kanefsky, B.; Kurien, J.; Millar, W.; Nayak, P.; and Rajan, K. 1998. *Remote Agent Experiment. Deep Space 1 Technology Validation Report.* NASA Ames and JPL report.

Blum, A. L., and Furst, M. L. 1997. *Fast planning through planning graph analysis.* Artificial Intelligence 90, 281-300, AAAI Press.

Bonet, B., and Geffner, H. 2001. *Heuristic Search Planner 2.0.* AI Magazine 22(3), 77-80, AAAI Press.

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

Dechter, R., and Dechter, A. 1988. *Belief Maintenance in Dynamic Constraint Networks*. In Proceedings AAAI-88, 37-42, AAAI Press.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. *Temporal Constraint Networks*. Artificial Intelligence 49, 61-95.

Fikes, R. E., and Nilsson, N. J. 1971. *STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving.* Artificial Intelligence 2, 189-208.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2004. *Automated Planning: theory and practice.* Morgan Kaufmann Publishers.

Haslum, P., and Geffner, H. 2001. *Heuristic planning with time and resources.* In Proceedings of ECP-2001 4/2001, Springer-Verlag.

Mackworth, A. K. 1977. *Consistency in Networks of Relations.* Artificial Intelligence 8, 99-118.

Nareyek, A.; Freuder, E.C.; Fourer, R.; Giunchiglia, E.; Goldman, R. P.; Kautz, H. A.; Rintanen, J.; and Tate, A. 2005. *Constraints and AI Planning.* IEEE Intelligent Systems 20(2), 62-72.

Nau, D. S.; Regli, W. C.; and Gupta K. S. 1995. *AI Planning versus Manufacturing Operation Planning: A Case Study.* In Proceedings of IJCAI-95, 1670-1676, Morgan Kaufmann.

Régin, J. C. 1994. *A Filtering Algorithm for Constraints of Difference*. In Proceeding of AAAI-94, 362-367, AAAI Press.

Sacerdoti, E. 1975. *The Nonlinear Nature of Plans.* In Proceedings of IJCAI-75, 206-214.

Schulte, C. 2002. *Programming Constraint Services: High-Level Programming of Standard and New Constraint Services.* LNCS 2302, Springer-Verlag.

Van Beek, P., and Chen, X. 1999. *A Constraint Programming Approach to Planning*. In Proceedings of the 16th National Conference on Artificial Intelligence (AAAI-99), 585-590, AAAI Press.

Vidal, V., and Geffner, H. 2004. *Branching and Pruning: An Optimal Temporal POCL Planner Based on Constraint Programming.* In Proceedings of AAAI-2004, 570-577, AAAI Press.