Reformulating Constraint Models for Classical Planning

Roman Barták, Daniel Toropila

Charles University, Faculty of Mathematics and Physics
Malostranské nám. 2/25, 118 00 Praha 1, Czech Republic
roman.bartak@mff.cuni.cz, daniel.toropila@matfyz.cz

Abstract

Constraint satisfaction techniques are commonly used for solving scheduling problems, still they are rare in AI planning. Although there are several attempts to apply constraint satisfaction for solving AI planning problems, these techniques never became predominant in planning; and they never reached the success of, for example, SAT-based planners. In this paper we argue that existing constraint models for classical AI planning are not fully using the power of constraint satisfaction; thus we propose a reformulation, which significantly improves their efficiency.

Introduction

Classical AI planning deals with finding a sequence of actions that transfer the world from some initial state to a desired state. Typically, the world *state* is described as a set of predicates that hold in the state, such as `location(robot1, city23)` saying that `robot1` is located in `city23`. *Actions* are described using a triple (Prec, Eff^+, Eff^-), where Prec is a set of predicates that must hold for the action to be applicable (preconditions), Eff^+ is a set of predicates that will hold after performing the action (positive effects), and finally Eff^- is a set of predicates that will not hold after performing the action (negative effects). Formally, action a is applicable to state s if $\text{Prec}(a) \subseteq s$. The result of applying action a to state s is a new state $\gamma(s, a) = (s - \text{Eff}^-(a)) \cup \text{Eff}^+(a)$. The set of predicates together with the set of actions is called a *planning domain*. We assume both sets to be finite. The *goal* is specified as a set of predicates that must hold in the goal state, that is, if g is a goal then any state s such that $g \subseteq s$ is a goal state. The *planning problem* is defined by the planning domain, the initial state s_0 and the goal g and the task of planning is to find a sequence of actions $\langle a_1, a_2, \ldots, a_n\rangle$ called a *plan* such that a_1 is applicable to the initial state s_0, a_2 is applicable to state $\gamma(s_0, a_1)$ etc. and $g \subseteq \gamma(\ldots\gamma(\gamma(s_0,a_1),a_2)\ldots,a_n)$.

One of the difficulties of planning is that the length of the plan, that is, the set of used actions, is unknown in advance so some dynamic technique which can produce plans of "unrestricted" length is required. Usually, the shortest plan is being looked for, which is a form of *optimal planning*. As it has been shown in (Kautz and Selman, 1992), the problem of shortest-plan planning can be translated to a series of SAT problems, where each SAT instance encodes the problem of finding a plan of a given length. First, we start with finding a plan of length 1 and if it does not exist then we continue with a plan of length 2 etc. until the plan is found. There exist criteria to stop these extensions if the plan does not exist, but for simplicity reasons in this paper we assume that a plan always exists.

Now, the problem of finding a plan of length n can be encoded as a constraint satisfaction problem (Dechter, 2003). A straightforward constraint model of this type has been described in (Ghallab, Nau, Traverso, 2004). Other constraint models that exploit a structure of a planning graph (Blum and Furst, 1997) have been proposed in (Do and Kambhampati, 2000) and (Lopez and Bacchus 2003). Though, these two models can explore several plans in parallel, their ideas can be applied to classical state-space planning as we shall show later in the paper.

All above mentioned constraint models have the origins in SAT and they are using Boolean variables and constraints in the form of logical formulas. As we argue in this paper, such formulation is less appropriate for constraint satisfaction techniques and therefore we propose a different formulation that is much more efficient. First, instead of predicates describing the states we use a fully instantiated multi-valued representation called *multi-valued planning tasks* (Helmert, 2006) based on the state variable formalism SAS^+ (Bäckström and Nebel, 1995). This leads to fewer variables with larger domains where domain filtering pays off. The second improvement is encapsulating the set of logical constraints from the original models into combinatorial constraints with an extensionally defined set of admissible tuples. These constraints filter out more inconsistencies than the original logical constraints and the propagation loop is faster which significantly reduces runtime.

The paper is organized as follows. First, we will explain the important details of current constraint satisfaction techniques. Then, after describing the multi-valued representation, we will present the existing constraint models formulated for classical planning with state variables. After that, we will reformulate the constraints and finally, we will give an experimental comparison of all presented models that justifies our proposal.

Constraint Satisfaction in a Nutshell

A *constraint satisfaction problem* (CSP) is a triple (X, D, C), where X is a finite set of decision variables, for each $x_i \in$ X, $\mathrm{D_i} \in$ D is a finite set of possible values for the variable x_i (the domain), and C is a finite set of constraints (Dechter, 2003). A constraint is a relation over a subset of variables (its *scope*) that restricts possible combinations of values to be assigned to the variables. A *solution to a CSP* is a complete instantiation of variables such that the values are taken from respective domains and all constraints are satisfied. CSPs are usually solved by combination of inference techniques and search. *Arc consistency* is the main inference technique used in current constraint solvers (Dechter, 2003). Briefly speaking, arc consistency algorithms filter values (remove them from domains) violating the individual constraints. As these techniques usually do not remove all inconsistencies they need to be combined with search that resolves remaining alternatives.

Efficiency of constraint solving is highly influenced by the choice of decision variables and constraints – a so called *constraint modeling*. For example, Boolean variables are not assumed to be a good modeling style because arc consistency cannot infer a lot for them (filtering corresponds to instantiation of the variable which is rare in hard problems). Similarly, constraints in the form of disjunction are not recommended because constraint solvers do not achieve full arc consistency for them (due to efficiency issues). As we shall see later, the existing constraint models for planning problems fall in this category of non-recommended modeling style.

Multi-Valued Representation

In order to better utilize the power of today's constraint solvers we have decided to employ multi-valued representation of planning problems called *multi-valued planning tasks* (MPTs) (Helmert, 2006) as the base input. One of the fundamentals of such encoding is preventing mutually exclusive predicates from appearing in the same state description, for example, that a given truck cannot be at two different locations simultaneously. That is, instead of plain enumeration of facts that hold in a given state, several *state variables* denoting different fragments of the world state are created with domains of values indicating exclusive options. Considering our previous example, for each truck the variable denoting truck's location would be created.

Whole state description is hence divided into variables with multi-valued domains, assignment of which expresses the exact state of the world. It can be seen that if confronted with purely propositional fact-based encoding, multi-valued representation results in encodings with a much smaller number of variables associated with larger domains. Thus, as this is generally the recommendation for constraint modeling, we believe that it leads to considerable performance improvement of constraint solvers over models utilizing such (or similar) formalism.

In addition to variables, multi-valued representation also includes enumeration of instantiated actions operating over variables in the same manner as operators in propositional encoding.

Since the similarity between multi-valued variables and CSP variables in constraint modeling is more than evident here, multi-valued variables have served us as the base in all of the constraint models described below.

Existing Constraint Models

We use the following iterative technique to find optimal solutions to planning problems. A specific constraint model is first used to encode the problem of finding a plan of length n (starting with n=1). Then the search for plan is performed. In case of success, the plan found is returned, otherwise encoding for the problem of finding a plan of length n+1 is constructed (by extending the previous encoding with the new layer of variables and constraints, not building it from scratch). Whole process is repeated until the plan is found or computation runs out of time or another termination condition applies.

Based on several successful approaches to *planning as a CSP* published recently in the literature, we have derived three different constraint models for state-space planning, adapted to employ multi-valued formalism:

- *Straightforward Model* (Ghallab, Nau, Traverso, 2004)
- *Model à la GP-CSP* (Do and Kambhampati, 2000)
- *Model à la CSP-PLAN* (Lopez and Bacchus, 2003)

The most important steps when designing a constraint model are surely the selections of variables, their domains and finally constraints defining consistent tuples of the variables. We will describe further the three above-mentioned constraint models together with the analysis of the expected number of constraints. For the purpose of analysis, suppose that we have a instantiated actions, v multi-valued variables denoting the state, and the average number of preconditions and effects of each action are p, and e respectively. Let t indicates the average number of actions affecting a given variable (notice that t is related to e by the equality $vt = ae$), and d indicates the average domain size of variables (typically $d << t$, or at least $d < t$).

Straightforward Model

The world state is described using v multi-valued variables, instantiation of which exactly specifies a particular state. A CSP denoting the problem of finding a plan of length n consists of n+1 sets of above mentioned multi-valued variables, having 1st set denoting the initial state and kth set denoting the state after performing k-1 actions, for $k \in \langle 2, n+1 \rangle$, and of n variables indicating the selected actions. Hence, we have $v(n+1)$ state variables V_i^s and n action variables A^j, where i ranges from 0 to v-1, j ranges from 0 to n-1, and s ranges from 0 to n (Figure 1).

Next to the variables we need logical constraints that connect two adjacent sets of state variables through the corresponding action variable between them, i.e. for given s we connect state variable layers V_i^s and V_i^{s+1}, $i \in \langle 0, v\text{-}1\rangle$, through the action variable A^s:

$$A^s = act \rightarrow \text{Pre}(act)^s\ , \forall act \in \text{Dom}(A^s), \quad (1)$$

$$A^s = act \rightarrow \text{Eff}(act)^{s+1}, \forall act \in \text{Dom}(A^s), \quad (2)$$

where $\text{Pre}(act)^s$ and $\text{Eff}(act)^{s+1}$ are conjunctions of equalities setting the values for required state variables corresponding to preconditions of action *act* in layer s, and its effects in layer s+1 respectively. We also need constraints representing the *frame axioms*, i.e. constraints that would enforce equalities between those state variables V_i^s and V_i^{s+1}, which are not affected by selected action A^s (the frame assumption is implicit in classical planning representations):

$$A^s \in \text{NonAffAct}(V_i) \rightarrow V_i^s = V_i^{s+1}, \forall i \in \langle 0, v\text{-}1\rangle, \quad (3)$$

where $\text{NonAffAct}(V_i)$ is the set of actions that do not have state variable V_i among its effects. Please note that this set depends purely on actions' definition and thus can be pre-computed in advance.

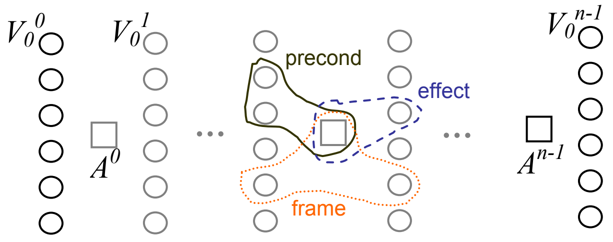

Figure 1. Base decision variables and constraints modeling plans.

The above model consists of $n(ap + ae + v)$ constraints (having each of preconditions and effects counted as a separate constraint).

Model à la GP-CSP

Planning graph's idea of *supporting actions* has been taken into consideration while constructing our next constraint model. Similarly as in (Do and Kambhampati, 2000) where for each proposition in a given layer a supporting action is being searched for, we have also employed the idea of support: having the same set of CSP variables as in the *Straightforward Model*, for each state variable V_i^s we create yet another variable S_i^s indicating the action that is actual cause of the value for V_i^s. The initial domain of each variable S_i^s is therefore preset to include only those actions that affect corresponding state variable V_i^s, plus one special value denoting the *no-op* action. Despite the introduction of newly added action (support) variables we do preserve the main action variable A^s, linking all the related variables and enforcing the serial manner of plans using the following logical constraints.

To encode action preconditions we use constraints (1) as described for the *Straightforward Model*. Then we use a slightly different version of frame axioms. For each s and each support variable S_i^{s+1} we introduce the constraint:

$$S_i^{s+1} = no\text{-}op \rightarrow V_i^s = V_i^{s+1}. \quad (4)$$

Support variables are also used for encoding action effects by introducing constraints

$$S_i^s = act \rightarrow V_i^s = val, \forall act \in \text{Dom}(S_i^s), \quad (5)$$

where *val* denotes the effect of action *act* for the state variable V_i^s. Finally, we need to specify the relation between support and action variables. For each s and each support variable S_i^{s+1} we put two constraints in the form of equivalences:

$$A^s \in \text{AffAct}(V_i) \leftrightarrow S_i^{s+1} = A^s, \text{ and}$$
$$A^s \in \text{NonAffAct}(V_i) \leftrightarrow S_i^{s+1} = no\text{-}op, \quad (6)$$

where $\text{NonAffAct}(V_i)$ and $\text{AffAct}(V_i)$ denote the same set as in the previous section, and its complement respectively.

For such a model, the total number of constraints is $n(ap + v + vt + 2v)$ which is equal to $n(ap + ae + 3v)$.

Model à la CSP-PLAN

The last of the models we considered is dominantly inspired by the way how logical equivalences are used in (Lopez and Bacchus, 2003) to describe *successor state constraints*, originally described in (Reiter, 2001). From the perspective of our previous models, successor state constraints can also be viewed as expressions that merge effect constraints and frame axioms together.

In this constraint model we start again with the same variable set as for the *Straightforward Model*, this time without any additional variables. The encoding of action preconditions is again using constraints of type (1). However, in contrary to the models above we use the successor state axioms. In particular, for each possible assignment of state variable $V_i^s = val$, $val \in \text{Dom}(V_i^s)$, we have a constraint between it and the same state variable assignment $V_i^{s\text{-}1} = val$ in the previous layer. The constraint says that state variable V_i^s takes value *val* if and only if some action assigned this value to the variable V_i^s, or equation $V_i^{s\text{-}1} = val$ held in the previous layer and no action changed the assignment of variable V_i. Formally:

$$V_i^s = val \leftrightarrow A^{s\text{-}1} \in C(i,val) \vee (V_i^{s\text{-}1} = val \wedge A^{s\text{-}1} \in N(i)), \quad (7)$$

where $C(i,val)$ denotes the set of actions containing $V_i = val$ among their effects, and $N(i)$ denotes the set $\text{NonAffAct}(V_i)$ as described within the previous models.

Regarding the number of constraints, this model uses $n(ap + vd)$ constraints, which is the least amount when comparing to other two models (to ease the comparison with previous models let us kindly recall here a pair of relations that hold for typical planning problems: $d << t$, and $vt = ae$). As we shall see in the next section, this amount can be further reduced by reformulation of some logical formulae using a combinatorial approach.

Reformulated Constraint Models

Most of the logical expressions used in above models take the form of implications that in turn are equivalent to disjunctions (in *propositional calculus*, a disjunction $P \vee Q$ is commonly viewed as a so called syntactic sugar or a shortcut for the implication $\neg P \rightarrow Q$). However, as we already mentioned the disjunctive formulae are infamous in the constraint modeling methodology. This, together with the fact that CSP based planners have not (yet) reached the success of other techniques which compile planning problems into a combinatorial substrate, makes us strongly feel that existing constraint models for classical AI planning are indeed not exploiting fully the power of constraint satisfaction. This observation has played an important role as a motivation for our endeavor in constraint reformulation.

One can observe that above logical constraints of the same type, say (1), share several variables (at least A^s is present there) and differ in which value is assigned to these variables. Hence we need a lot of constraints with a similar scope which, first, slows down the arc consistency procedure (it needs to check more constraints) and makes domain filtering weaker (each constraint is processed separately). In this paper we adopt the approach of substituting some of the above mentioned propositional formulae using the constraints with extensionally defined set of admissible tuples (sometimes also called *combinatorial constraints*). The idea is to union the scope of "similar" constraints and to define the admissible tuples in extension rather than using a formula. Such types of constraints are frequently called *table constraints*, because the set of admissible tuples is given in a table-like structure. There also exist compressed formats for describing the admissible tuples, for example using a DAG that resembles decision trees. In SICStus Prolog, such a constraint is called *case constraint* (Carlsson *et al*. 1997).

We shall now describe how to reformulate constraints from the above models using combinatorial constraints. We abstract from a particular form of the constraint, either table or case, the combinatorial constraint will be specified by its scope (variables) and a set of admissible tuples.

Straightforward Model: Reformulated

In the first of our models, formulae enforcing action preconditions (1) and effects (2) (all in the form of implications) comprise the majority of constraints ($ap + ae$) essential to associate adjacent layers of variables (layers with indices s and s+1). All of these propositional formulae can be however replaced by one combinatorial constraint in the following way. The scope of the constraint consists of action variable A^s, state variables V_i^s, and state variables V_i^{s+1}, that is, in total 2v+1 variables. Each action *act* is represented by a single row in the table of admissible tuples, which basically describes a restriction imposed by constraints (1) and (2) for $A^s = act$. The leftmost table in Figure 2 gives an example of compatible tuples according to these constraints. Values of preconditions and effects are filled there. The remaining cells are supposed to contain sets of values $\{0,\ldots,d_i\text{-}1\}$, where d_i is the domain size of variable V_i^s (or V_i^{s+1}) so each row actually describes many admissible tuples in a compact form. Note also that the table can be computed just using information about the planning domain.

Basically, the above constraint defines a state transition at layers s and s+1, but it does not subsume the frame axiom. It is possible to encode the frame axioms in the table, but then the above compact form is impossible as each row would describe a single tuple which leads to a huge table. So we rather keep all of the frame axiom constraints (3) from the original model obtaining together $n(1+v)$ constraints.

Model à la GP-CSP: Reformulated

We can utilize the approach described above to reformulate action preconditions for our second model using one combinatorial constraint instead of ap propositional constraints of type (1). The scope of the constraint consists of action variable A^s and state variables V_i^s only. Likewise we preserve the frame axioms (4) as defined in the original version of this model. The rest of the formulae can be encoded using another combinatorial constraint.

Firstly, we use a table to define action effects (5) as a relationship between support variable S_i^s and its corresponding state variable V_i^s (there is a row for each

Domain

DWR domain with two locations (loc1, loc2),
a robot capable of loading and unloading by itself (r),
and one container (c)

State Variables

$rloc \in$ {loc1, loc2} ;; robot's location
$cpos \in$ {loc1, loc2, r} ;; container's position

Actions

```
1 : move(r, loc1, loc2)
    ;; robot r at location loc1 moves to location loc2
    precond:   rloc = loc1
    effects:   rloc ← loc2

2 : move(r, loc2, loc1)
    ;; robot r at location loc2 moves to location loc1
    precond:   rloc = loc2
    effects:   rloc ← loc1

3 : load(r, c, loc1)
    ;; robot r loads container c at location loc1
    precond:   rloc = loc1, cpos = loc1
    effects:   cpos ← r

4 : load(r, c, loc2)
    ;; robot r loads container c at location loc2
    precond:   rloc = loc2, cpos = loc2
    effects:   cpos ← r

5 : unload(r, c, loc1)
    ;; robot r unloads container c at location loc1
    precond:   rloc = loc1, cpos = r
    effects:   cpos ← loc1

6 : unload(r, c, loc2)
    ;; robot r unloads container c at location loc2
    precond:   rloc = loc2, cpos = r
    effects:   cpos ← loc2
```

Table for Straightforward Model: Refor.

A^s	$rloc^s$	$cpos^s$	$rloc^{s+1}$	$cpos^{s+1}$
1	loc1	{..}	loc2	{..}
2	loc2	{..}	loc1	{..}
3	loc1	loc1	{..}	r
4	loc2	loc2	{..}	r
5	loc1	r	{..}	loc1
6	loc2	r	{..}	loc2

Tables for Model a la GP-CSP: Refor.

S^s_{rloc}	$rloc^s$
no-op	{..}
1	loc2
2	loc1

S^s_{cpos}	$cpos^s$
no-op	{..}
3	r
4	r
5	loc1
6	loc2

S^s_{rloc}	A^s
no-op	{3,4,5,6}
1	1
2	2

S^s_{cpos}	A^s
no-op	{1,2}
3	3
4	4
5	5
6	6

Tables for Model a la CSP-PLAN: Refor.

A^{s-1}	$rloc^{s-1}$	$rloc^s$
2	{..}	loc1
1	{..}	loc2
{3,4,5,6}	loc1	loc1
{3,4,5,6}	loc2	loc2

A^{s-1}	$cpos^{s-1}$	$cpos^s$
5	{..}	loc1
6	{..}	loc2
{3,4}	{..}	r
{1,2}	loc1	loc1
{1,2}	loc2	loc2
{1,2}	r	r

Figure 2. Example of constraint reformulation for presented models using table constraints.

action having V_i among its effects – see the first two tables in the middle part of Figure 2). In each layer *s* there are *v* such tables.

Secondly, a table is also used to make connections between support variables S_i^s and action variables A^s as in (6) (there is a row for each action affecting state variable V_i – see the last two tables in the middle part of Figure 2). Again, in each layer *s* there are *v* such tables.

Thanks to above reformulation of propositional formulae we obtain $1+v+v+v = 1+3v$ constraints per layer, which is $n(1+3v)$ constraints in total.

Model à la CSP-PLAN: Reformulated

For the sake of revision, in the last of our models we originally have just two types of constraints: *ap* constraints for action preconditions (1), and *vd* constraints for successor state axioms (7), earlier of which we can substitute in the same way as in the previous two models using one combinatorial constraint per layer. Moreover, the later set of formulae can also be replaced by combinatorial constraints in the following way.

The scope of the constraint consists of action variable A^{s-1}, state variable V_i^{s-1}, and state variable V_i^s so it is a ternary constraint. Briefly speaking, each such constraint describes how to set a value of variable V_i^s depending on action A^{s-1} and value of V_i^{s-1} (hence there are *v* such ternary constraints in each layer). The two rightmost tables in Figure 2 give an example of the admissible triples. The upper rows describe the actions with effects related to variable V_i. The lower rows correspond to frame axioms for the same variable (all values are listed there). Clearly, there are at most $2d_i$ rows in the table (compare to the reformulated Straightforward Model, where the frame axioms would lead to a huge table).

As proposed earlier, this model needs only $1+v$ combinatorial constraints per layer (and no extra propositional formulae), which is $n(1+v)$ constraints in total – the least number of constraints among all six constraint models presented.

Search Strategy

Designing a constraint model is just the first step to solve the problem using constraint satisfaction technology. The next step is defining the search strategy, that is, the order in which the variables are instantiated and the order in which values are tried for the variables. This is called *labeling* (there exist other search strategies, for example based on domain splitting). One can use generic labeling techniques, for example based on first-fail principle, however, we decided to utilize a labeling strategy derived from the nature of the problem (we used the same labeling procedure for all models to evaluate the efficiency of the model rather than efficiency of the search procedure).

First, one should realize that it is enough to instantiate just the action variables A^s because when their values are known then the values of remaining variables, in particular the state variables, are set by means of constraint propagation. Of course, we assume that the values for state variables V_i^0 modeling the initial state were set and similarly the state variables V_i^n in the final layer were set according to the goal (the final state is just partially specified so some state variables in the final layer remain un-instantiated).

We utilized a regression planning approach in the search strategy meaning that we instantiate the state variables in the decreasing order from A^{n-1} to A^0. This is called a fixed variable ordering in constraint satisfaction. The values in each state variable were enumerated increasingly starting from the first action in the domain (fixed value ordering). Moreover, only values corresponding to actions that contribute to the goal are explored. More formally, if variable A^s is being instantiated then we explore only the actions (values) that have some effect $V_i^{s+1} = b$ and the variable V_i^{s+1} is already instantiated to *b*. This corresponds (roughly) to the relevant actions selected during regression planning (Ghallab, Nau, Traverso, 2004).

Experimental Comparison

We implemented all above models in SICStus Prolog 4.0.2 and compared them using selected domains from International Planning Competitions (STRIPS versions), namely Gripper, Logistics, Mystery (IPC1), Blocks, Elevator (IPC2), Depots, Zenotravel, DriverLog (IPC3), Airport, PSR (IPC4), and Pipesworld, Rovers, TPP (IPC5). In total we used 37 problems. The experiments run on Pentium M 730 1.6 GHz processor with 1GB RAM under Windows XP. We used a timeout 300 seconds to explore plans of a given length. If no plan was found within the time limit, a next layer was added until a plan is found or a time limit is exceeded. Because some methods may stop earlier, say at layer M, due to timeout while the optimal length was N, we added time penalty $300^*(\mathrm{M}-\mathrm{N})^2/2$ to underestimate total runtime for finding a plan of length N. The reason is that according to our observations the runtime increases at least quadratically for exploring the problem extended by one layer so using the above penalty gives us a more accurate comparison of methods even if some methods did not find a solution within the time limit.

Figure 3 compares runtimes for all methods on problems from above domains – we sorted the domains increasingly using the runtime of reformulated model à la CSP-PLAN which solved all selected problems. The results clearly demonstrate that the models based on combinatorial constraints significantly outperform their logical counterparts (up to 2 orders of magnitude). Also, the straightforward model and the model à la GP-CSP have similar performance which is not surprising due to similarity of constraints. The reformulated model à la CSP-PLAN is a clear winner – constraint propagation for this model is strongest (thanks to equivalence constraints) and fastest (the smallest number of constraints) among the compared models.

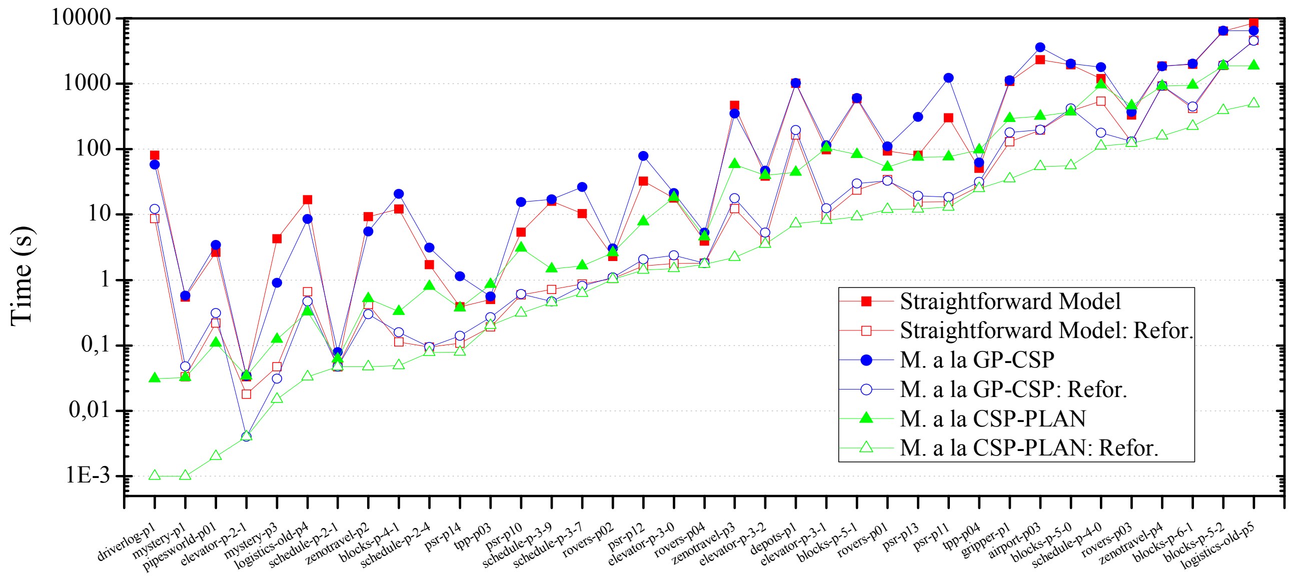

Figure 3. Comparison of runtimes (logarithmic scale) for selected problems from IPC 1-5

Conclusions

The paper shows that a more careful design of the constraint model that takes into account how constraint satisfaction works may significantly improve efficiency of planning via constraint satisfaction. Basically, we proposed reformulation of sets of logical constraints using extensionally defined (combinatorial) constraints. The extensional representation may seem memory consuming, but the experiments showed that it is viable. Still, a further study with larger problems is necessary. To highlight the difference between the constraint models, we focused on pure description of classical planning problem without implied constraints (for example modeling mutex-like relations) and without sophisticated search strategies (with carefully chosen heuristics). We are aware that these advanced techniques may significantly influence the overall efficiency and hence the presented approaches do not aspire (yet) to be the fastest planning system so far. Nevertheless, the paper shows a promising way for using constraints in state-space planning.

We also did not compare to hand-crafted constraint models à la CPlan by van Beek and Chen (1999) as we are looking for a fully automated solver. We also did not compare to CPT planner by Vidal and Geffner (2004) which is a partial order planner. Nevertheless, we believe that our observations may improve efficiency of other constraint-based planners too.

Acknowledgments

We thank anonymous reviewers for valuable comments and Malte Helmert for providing us with the implementation of translator from PDDL to MPT planning problem specification. The research is supported by the Czech Science Foundation under the project 201/07/0205.

References

Bäckström, Ch., Nebel, B. 1995. Complexity results for SAS$^+$ planning. *Computational Intelligence* 11(4), 625-655.

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

Carlsson M., Ottosson G., Carlson B. 1997. An Open-Ended Finite Domain Constraint Solver. *Programming Languages: Implementations, Logics, and Programs*.

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

Do, M.B. and Kambhampati, S. 2000. Solving planning-graph by compiling it into CSP. *Proceedings of the Fifth International Conference on Artificial Planning and Scheduling* (AIPS-2000), AAAI Press, 82-91.

Ghallab, M., Nau, D., Traverso P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann.

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research 26*, 191-246.

Kautz, H. and Selman, B. 1992. Planning as satisfiability. *Proceedings of ECAI*, 359-363.

Lopez, A. and Bacchus, F. 2003. Generalizing GraphPlan by Formulating Planning as a CSP. *Proceedings of IJCAI*, 954-960.

Reiter, R. 2001. *Knowledge in Action: Logical Foundation for Specifying and Implementing Dynamic Systems*. MIT Press.

van Beek, P. and Chen, X. 1999. CPlan: A Constraint Programming Approach to Planning. *Proceedings of AAAI-99*, 585-590.

Vidal, V. and Geffner, H. 2004. Branching and Pruning: An Optimal Temporal POCL Planner based on Constraint Programming. *Proceedings of AAAI-04*, 570-577.