A Novel Constraint Model for Parallel Planning

Roman Barták
Charles University in Prague, Faculty of Mathematics and Physics
Malostranské nám. 25, 118 00 Praha 1, Czech Republic
bartak@ktiml.mff.cuni.cz

Abstract

A parallel plan is a sequence of sets of actions such that any ordering of actions in the sets gives a traditional sequential plan. Parallel planning was popularized by the Graphplan algorithm and it is one of the key components of successful SAT-based planers. SAT-based planners have recently begun to exploit multi-valued state variables – an area which seems traditionally more suited for constraint-based planners – and they improved their performance further. In this paper we propose a novel view of constraint-based planning that uses parallel plans and multi-valued state variables. Rather than starting with the planning graph structure like other parallel planners, this novel approach is based on the idea of timelines and their synchronization.

Introduction

Classical AI planning deals with finding a sequence of actions that transfer the world from some initial state to a desired state. We assume a *fully observable* (we know precisely the state of the world), *deterministic* (the state after performing the action is known), and *static* (only the entity for which we plan changes the world) *world* with a finite (though possibly large) number of states. We also assume actions to be instantaneous so we only deal with action sequencing. Actions are usually described by a set of *preconditions* – features that must hold in a state to make the action applicable to that state – and a set of *effects* – changes that the action makes to the state. Action sequencing is naturally restricted by *causal relations* between the actions – the effect of certain action gives a precondition of another action.

Traditional *sequential planning* algorithms explore directly the sequences of actions. One of the disadvantages of this approach is liability to exploring symmetrical plans where some actions can be swapped without changing the overall effect. Hence if some sequence of actions does not lead to a goal then the algorithm may explore a similar sequence of actions where certain actions are swapped though this sequence leads to exactly same non-goal state. This is called *plan-permutation symmetry* (Long and Fox 2003). It is possible to remove some of these symmetries by symmetry breaking constraints as suggested in (Grandcolas and Pain-Barre 2007) or (Barták and Toropila 2009). Another way to resolve this problem is *partial-order planning* where the plans are kept as partially ordered sets of actions (the partial order respects the causal relations). CPT planner (Vidal and Geffner 2004) is probably the most successful (in terms of International Planning Competition) constraint-based planner that does partial-order planning. A half way between partial-order and sequential planning is *parallel planning*, where the plan is represented as a sequence of sets of actions such that any ordering of actions in the sets gives a traditional sequential plan. This concept was popularised by the Graphplan algorithm (Blum and Furst 1997) that introduced a so called *planning graph* to efficiently represent causal relations between the actions. Planning graph became a popular representation of parallel plans for approaches that translate the planning problem to other formalisms such as Boolean satisfiability or constraint satisfaction (Do and Kambhampati 2000; Lopez and Bacchus 2003).

In this paper we propose a novel constraint model for parallel planning. This model is motivated by recent success of SAT-based planner SASE (Huang, Chen, and Zhang 2010) that exploits multi-valued state variables (Helmert 2006). Rather than following the planning graph translation to a constraint satisfaction problem, the novel model is much closer to the modern timeline-based approach to planning (Pralet and Verfaillie 2009). The model is proposed for the multi-valued state variable representation of planning problems and it is based on idea of describing the evolution of each state variable and synchronizing the changes between the different state variables. In the paper, we will first formally introduce the planning problems to be solved and give necessary background and related works. Then we will describe the core concept of the planner and after that we will formally define the constraint model and describe the used search strategy. We will conclude with the experimental evaluation of the new planner.

Copyright © 2011, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

The Problem

AI planning task is to find a sequence of actions that transfer the initial world state to a state satisfying a given goal condition. For describing world states and actions we use a so called SAS+ formalism (Bäckström and Nebel 1995) that is based on *multi-valued state variables* (Helmert 2006). For each feature of the world, there is a state variable describing this feature, for example rloc(r1,S) describes the location of robot r1 at state S. This state variable may acquire one of finitely many values. Each world state is described by a complete instantiation of the state variables. The advantage of this representation over the classical propositional representation is that it naturally expresses some facts, such as that the robot cannot be simultaneously at two locations which is not directly expressed when the logical propositions are used.

Action is applicable to world states satisfying the action precondition. Briefly speaking the action precondition expresses which values of the state variables are required by the action. Formally, *action precondition* is a set (a conjunction) of expressions in the form either *state-variable = value* or *state-variable* ∈ *set-of-values* (not in SAS+) such that each state variable appears at most once in this set. Actions change the values of the state variables, which is captured by actions' effects. Formally, an action effect is a set (a conjunction) of expressions in the form *state-variable* ← *value* such that each state variable appears at most once in the set. After performing the action, the state variables that do not appear in the action effect will not change their value (the frame axiom) while the state variables appearing in the effect will take the value specified in the effect. Figure 1 gives an example of such representation.

The planning task can be formulated as follows. Given a complete specification of the initial state (the values of all state variables) and a description of the goal condition as a set of expressions in the form either *state-variable = value* or *state-variable* ∈ *set-of-values* find a sequence of actions that transfer the world from its initial state to the state satisfying the goal condition. This is called *sequential planning*. In this paper we deal with *parallel planning* where we are looking for a sequence of sets of independent actions such that the sequential plan is obtained by arbitrary ordering of actions in the sets. Two actions are independent if no state variable appearing in the effect of one action appears in the precondition or effect of the other action. This condition ensures that if both actions are applicable to a given state then they can be applied in any order and the states after application of both actions will be identical. Hence for a given state *s* and a set *AS* of pair wise independent actions applicable to this state we can define a state after application of these actions as the state *s* modified by the effects of actions in *SA*. Note that this is possible because the actions in *SA* are setting different state variables thanks to their independence. Obviously, a sequential plan is a special case of the parallel plan; the advantage of using parallel plans is removing some plan-permutation symmetries.

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

State Variables
rloc ∈ {loc1,loc2} ;; robot's location
cpos ∈ {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

Figure 1. Example of planning domain represented using multi-valued state variables.

Background and Related Works

Constraint satisfaction problem (CSP) is defined by a set of variables, each variable has a finite set of possible values, and constraints specify allowed combinations of values assigned to the variables. The task is to find an instantiation of the variables satisfying all the constraints. CSPs are solved by the combination of search and inference realized via maintaining constraint consistency. Designing the proper constraint model is the key step in problem solving as it defines the level of inference – how much the search space is pruned.

Constraint satisfaction techniques were first applied in AI planning via manually designed constraint models for concrete planning domains (van Beek and Chen 1999). General models applicable to any planning problem were based on translation of the planning graph to a CSP (Do and Kambhampati 2000; Lopez and Bacchus 2003). Barták and Toropila (2008) reformulated these models to use multi-valued state variables and ad-hoc tabular constraints. These ideas are used in planning systems SeP (Barták and Toropila 2009) and Constance (Gregory, Long, and Fox 2010).

The Concept

There are many ways to describe planning problems in a form appropriate for problem solving. A natural representation for multi-valued state variables is a *state transition diagram* (also known as a *domain transition graph*) which is basically a finite state automaton (FSA). Each planning state variable is represented using a single automaton whose states correspond to the values of the variable and the arcs describe how the actions are changing the value of the state variable. In particular, if a given state variable v is both in the precondition ($v = a$) and effect ($v \leftarrow b$) of the action then the arc(s) connects the value(s) from the precondition with the value in the effect ($a \rightarrow b$). If the state variable is only in the effect then there are arcs from all values to the value in the effect. If the state variable is only in the precondition then there is a loop in the corresponding value. Finally, if the state variable is not used by a given action then there are loops in all values. The initial state in FSA describes the initial value in the planning problem; the final states are defined by the goal condition (all states are final, if the state variable does not appear in the goal condition). Figure 2 gives an example of the FSA representation for the planning domain from Figure 1, where we assume the robot to be initially at location loc1 and the container at location loc2 and the goal is to have the container at location loc1.

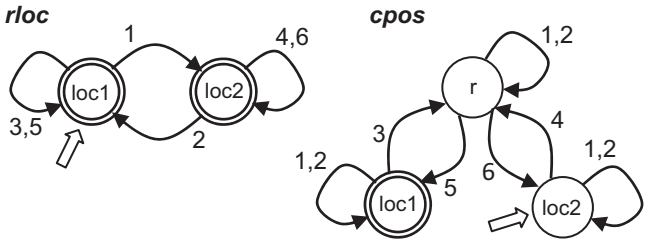

Figure 2. Representation of the planning domain (Figure 1) and problem using finite state automata.

Each finite state automaton defines a regular language describing the sequences of actions transferring the automaton from the initial state to the goal state. Hence, any plan belongs to the intersection of the regular languages defined by the automata for the state variables. In other words, to solve the planning problem we need to find a path in each FSA and the paths must be synchronized between the automata. We can describe the evolution of the state variable as a timeline as Figure 3 shows. Notice that the sequences of actions are identical for all the state variables.

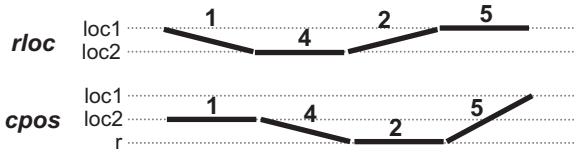

Figure 3. Timelines describing the synchronized evolution of the state variables (the action numbers are taken from Figure 1).

The above representation using FSA is appropriate for sequential planning as each action changes the states in all automata. In parallel planning we allow different actions to appear at a single planning step provided that the actions are independent. In other words, it is possible to change the states in different automata using different actions at the same planning step if these changes are not in conflict. To support parallel planning we modify the FSA representation in the following way. Each FSA contains two sorts of arcs: the arcs defining the effects of the actions for actions changing the state variable and the arcs defining the no-op actions indicating that the state variable is not changed. The difference from the no-op actions used in the planning-graph is that we use a dedicated no-op action for each value of the state variable. Figure 3 gives an example of this modified representation where the no-op actions are indicated by negative numbers.

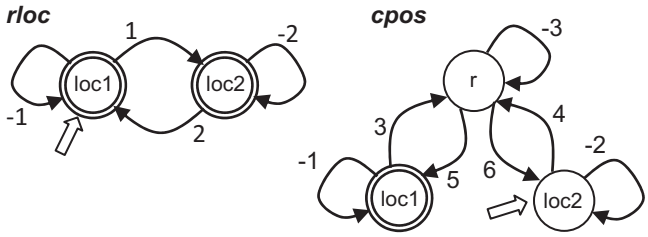

Figure 4. Representation of the planning domain (Figure 1) and problem using finite state automata with no-op actions.

The reason for having more no-op actions per FSA is that we still need to synchronize the automata, in particular to model the preconditions of actions. If some action requires a particular value of the state variable but the action is not changing that state variable then we require the corresponding FSA to move along the arc annotated by the no-op action representing the value in the action precondition. Notice that this model allows different actions in a single step to have the same precondition exactly in accordance with the definition of independent actions. Figure 5 shows the evolution of the state variables in this modified model. We also show there how the real actions are forcing the presence of some no-op actions.

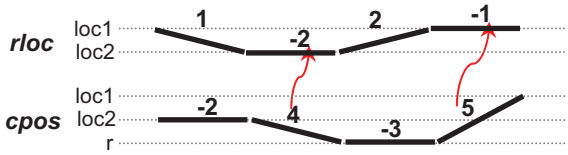

Figure 5. Timelines describing the evolution of the state variables with no-op actions (the arrows indicate synchronization).

The FSA model with no-op actions supports parallel independent actions, but the synchronization of automata is more complex and must be done explicitly as different actions may change the values of different state variables at the same time. We shall now describe how to encode this model as a constraint satisfaction problem.

The Constraint Model

We are using the traditional approach of converting the planning problem where the number of actions in the plan is unknown in advance to a static constraint satisfaction problem as suggested in (Kautz and Selman 1992). In particular, we formulate the problem of finding a parallel plan of length n as a CSP and we solve the original planning problem by starting with $n = 0$ and incrementing n by 1 if no plan is found. Let k be the number of state variables then we introduce $k(n+1)$ state variables S^i_j in the constraint model describing all the states "visited" by the parallel plan ($i = 1,\ldots,k$, $j = 0,\ldots,n$). The domain of variable S^i_j consists of values of the i-th state variable. The state variables S^i_0 are instantiated using the values from the initial state while the state variables S^i_n are restricted based on the goal condition. We also introduce kn action variables A^i_j in the constraint model describing the actions changing the state variables. The domain of variable A^i_j consists of actions that have the i-th state variable among the effects and the no-op actions for that state variable. For example, the domain of A^{cpos}_j is {-3,-2,-1,3,4,5,6} for the problem from Figure 4.

There are two core types of constraints in our model. First, we need to model the state transitions based on the FSA representation. We use a ternary *sequencing constraint* connecting variables S^i_{j-1}, A^i_j, S^i_j. Basically, this constraint describes the arcs in the FSA representation. The triple (p,a,q) satisfies the constraint for the i-the state variable if one of the following conditions hold:

- a is a real action such that q is the value of its effect in the i-th state variable and value p is compatible with the precondition of a (a is using value q as its precondition or a has no precondition in the i-th state variable),
- a is a no-op action for value p of the i-th state variable and $p = q$.

In the terms of FSA, we can say that a annotates the arc connecting states p and q in the automaton. Hence, the sequencing constraint describes the evolution of the corresponding state variable as shown in Figure 5.

The second type of constraint describes the synchronisation between the evolutions of the state variables. In particular, if the action is changing several state variables then the action must be assigned to the action variables for all these states in a given layer (we call the variables A^i_j with identical j a layer). Moreover, if the action has a precondition in the state variable that is not among its effects (for example actions 3-6 in Figure 1 have precondition in the state variable *rloc* while changing only the variable *cpos*) then we must ensure that the corresponding state variable is assigned to the requested value. This is done indirectly by requesting the action variable for that state variable to be assigned to the specific no-op action. It would be possible to describe the synchronisation constraint as a single k-ary constraint between the variables $A^1_j,\ldots, A^k_j$. However, the extensional representation of this constraint would be too large as in general it must describe all possible subsets of independent actions. Hence, we decided to use k k-ary *synchronisation constraints* each describing the requirements of some state transition. Let i be the index of certain state variable. Then for each action from the FSA representation of the i-th state variable we define which actions are compatible in other action variables of the same layer. If a is a real action assigned to variable A^i_j then the constraint requires the following assignment:

- if a affects the l-th state variable then $A^l_j = a$,
- if a has a precondition (but not effect) in the l-th state variable and b is the no-op action corresponding to the value of the precondition then $A^l_j = b$,
- if a does not use the l-th state variable then A^l_j can be assigned to any action independent of a or no-op.

If a is a no-op action then we assume that it is compatible with all actions in other action variables to make the extensional representation compact. Note that the synchronisation constraints for other state variables may connect this no-op action with real action as described above. Though we described the synchronisation constraints as k-ary constraints, in fact we can cut-off some variables from the constraint, if these variables are not really constrained. This significantly reduces the arity of the constraint as the actions usually use a small number of state variables in preconditions and effects.

Figure 6 sketches the structure of the base constraint model. In addition to above constraints there are also *active layer constraints* connecting all action variables in each layer and requesting at least one action to be a real action.

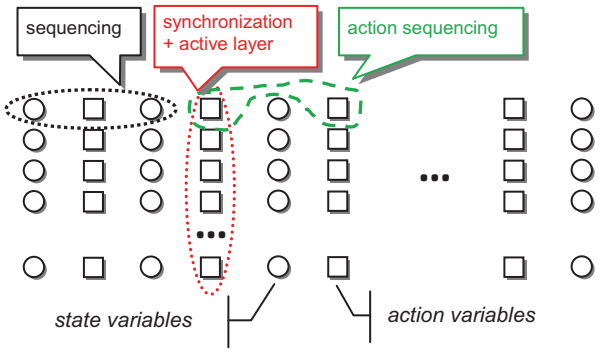

Figure 6. The structure of the constraint model, circles represent the state variables, squares represent the action variables.

To strengthen inference we add one more sequencing constraint. Notice that action b can immediately follow action a in the timeline for a given state variable if the effect of a restricted to that state variable is compatible with the precondition of b on the same state variable. However, there might be another state variable where the effect of a is in conflict with the precondition of b and hence a cannot directly precede b. To discover this conflict using inference, we include a binary *action sequencing constraint* between all subsequent pairs of action variables in each timeline (see Figure 6). Let a and b be two actions from the FSA describing a certain timeline then the pair

(*a*,*b*) satisfies the action sequencing constraint if one of the following conditions hold:

- *a* and *b* are identical no-op actions,
- *a* is a no-op action compatible with the precondition of real action *b*,
- *b* is a no-op action that corresponds to the effect of real action *a*,
- all effects of real action *a* are compatible with all preconditions of real action *b*.

Figure 7 gives an example of all above mentioned constraints for the planning problem described by the finite state automata from Figure 4. One can easily check that the instantiation of action variables shown in Figure 5 satisfies all the constraints.

sequencing

S^{rloc}_{i-1}	A^{rloc}_{i}	S^{rloc}_{i}
loc1	1	loc2
loc2	2	loc1
loc1	-1	loc1
loc2	-2	loc2

action sequencing

A^{rloc}_{i}	A^{rloc}_{i+1}
1	{-2,2}
2	{-1,1}
-1	{-1,1}
-2	{-2,2}

S^{cpos}_{i-1}	A^{cpos}_{i}	S^{cpos}_{i}
loc1	3	r
loc2	4	r
r	5	loc1
r	6	loc2
loc1	-1	loc1
loc2	-2	loc2
r	-3	r

A^{cpos}_{i}	A^{cpos}_{i+1}
3	{-3,5}
4	{-3,6}
5	{-1,3}
6	{-2,4}
-1	{-1,3}
-2	{-2,4}
-3	{-3,5,6}

synchronisation

A^{rloc}_{i}	A^{cpos}_{i}
1	{-1,-2,-3}
2	{-1,-2,-3}
{-2,-1}	{-3,…,6}

A^{cpos}_{i}	A^{rloc}_{i}
3	-1
4	-2
5	-1
6	-2
{-3,-2,-1}	{-2,…,2}

Figure 7. The compact representation of ad-hoc constraints for the planning problem from Figure 4.

Search Strategy

The constraint model can be accompanied by a specific search strategy that guides the search algorithm exploring the possible instantiations of the variables. This search strategy is composed of two types of heuristics: the variable ordering heuristic that recommends which variable is instantiated first and the value ordering heuristic that suggests which value is tried first.

In our constraint model, we use only the action variables as the decision variables participating in the search procedure. The state variables are instantiated by means of constraint propagation (local inference). There exist several generally applicable *variable ordering heuristics* typically based on the first-fail principle. Dom heuristic (Golomb and Baumert 1965) that prefers instantiation of the variables with the smallest domain is among the most widely used. We slightly modified this heuristic in the following way. We select only among the variables whose domains contain at least one real action. The action variables that can be instantiated only to some no-op action are ignored during search. These variables are instantiated by means of constraint propagation. Note that this decision is done dynamically during search as constraint propagation can remove some actions from the domain of action variables. The *value ordering heuristics* are based on the succeed-first principle, but there are no widely accepted general value ordering heuristics. Obviously, it is not clear in advance which value (action) belongs to the solution. We used the following simple heuristic. First, the domain of the selected variable is split into two parts: the no-op actions and the real actions. This leads to binary branching; the branch where the no-op actions remain in the variable domain is explored first. The motivation was that this will minimize the number of used actions (see the experiments). If a variable whose domain contains only the real actions is being instantiated then we simply try the actions in the order in which the actions appear in the problem description. Though the search strategy may seem important for problem solving, it is the constraint model that influences most the efficiency.

Experimental Results

We implemented the PaP constraint model using the clpfd library of SICStus Prolog 4.1.2 and compared it with the SeP planner built on top of the same constraint library. We used selected planning domains from past International Planning Competitions (STRIPS versions) for the comparison. The experiments ran on 2.0 GHz Intel Xeon CPU with 8GB RAM under Ubuntu Linux 8.04.2 (Hardy Heron). Both planners run with the 30 minutes timeout.

Table 1 reports the number of solved problem instances in selected planning domains. PaP is better in all domains but openstacks and elevator. For openstacks, it seems that no-good recording helped SeP to solve the problems. For elevator, we identified some overhead in the PaP constraint model that caused longer runtime.

Table 2 gives a more detailed view for three domains with actual runtimes and plan lengths. PaP is slower in blockworld problems because the plans cannot be parallelized there and more constraints (compared to SeP) cause overhead. In other domains the results are similar to zenotravel and tpp domains. An interesting observation is that PaP finds plans with identical or only slightly larger number of actions in comparison to the shortest plans found by SeP. We think that this is due to used search strategy that prefers the no-op actions over the real actions. We did not perform a direct comparison with the Constance planner that beat SeP in domains such as driverlog, zenotravel, and tpp, but the detailed results reported in (Gregory, Long, and Fox 2010) show that PaP achieves better performance at these domains.

domain	*SeP*	*PaP*
airport (15)	4	**6**
blocks (16)	**7**	**7**
depots (10)	**2**	**2**
driverlog (15)	4	**12**
elevator (30)	**30**	27
freecell (10)	1	**3**
openstacks (7)	**5**	0
rovers (10)	4	**6**
tpp (15)	4	**8**
zenotravel (15)	6	**11**

Table 1. The number of solved problems in each domain (the numbers in parenthesis indicate the number of tried problems).

problem	*plan length*			*runtime (ms)*	
	SeP	*PaP*		*SeP*	*PaP*
		par	*seq*		
blocks-p-4-1	10	10	10	**160**	190
blocks-p-5-0	12	12	12	**1 670**	4 600
blocks-p-5-1	10	10	10	**1 050**	4 790
blocks-p-5-2	16	16	16	37 420	**34 160**
blocks-p-6-0	12	12	12	**8 720**	59 370
blocks-p-6-1	10	10	10	**9 760**	74 450
blocks-p-7-0	20	20	20	**926 820**	-
tpp-p01	5	5	5	10	**0**
tpp-p02	8	5	8	20	**10**
tpp-p03	11	5	11	160	**30**
tpp-p04	14	5	14	2 110	**20**
tpp-p05	≥17	7	23	-	**100**
tpp-p06	≥15	9	29	-	**4 110**
tpp-p07	≥14	9	38	-	**3 170**
tpp-p08	≥14	9	44	-	**5 930**
zenotravel-p01	1	1	1	**10**	20
zenotravel-p02	6	5	6	60	**50**
zenotravel-p03	6	5	9	300	**130**
zenotravel-p04	8	5	11	970	**130**
zenotravel-p05	11	5	14	153 990	**240**
zenotravel-p06	11	5	12	530 390	**510**
zenotravel-p07	≥12	6	16	-	**560**
zenotravel-p08	≥10	5	15	-	**1 690**
zenotravel-p09	≥11	6	24	-	**145 760**
zenotravel-p10	≥12	6	24	-	**252 040**
zenotravel-p11	≥9	6	16	-	**41 780**

Table 2. The length of found plans and the runtime (in milliseconds) for selected planning problems.

Conclusions

The paper describes a novel constraint-based planner PaP for parallel planning with multi-valued state variables. Despite the simplicity of the constraint model and the search strategy, the new planner beats significantly existing constraint-based optimal sequential planners SeP and Constance in most planning domains. We expect that performance of PaP can be further improved by removing some redundancy in constraints and by integrating advanced techniques such as nogood recording.

Acknowledgements

The research is supported by the Czech Science Foundation under the contract P103/10/1287. I would like to thank Daniel Toropila for performing the experiments and anonymous reviewers for valuable comments.

References

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

Barták, R., Toropila, D. 2008. Reformulating Constraint Models for Classical Planning. *Proceedings of The Twenty-First International Florida Artificial Intelligence Research Society Conference (FLAIRS-21)*, 525-530. AAAI Press.

Barták, R., Toropila, D. 2009. Revisiting Constraint Models for Planning Problems. *Proceedings of the Eighteenth International Symposium on Foundations of Intelligent Systems (ISMIS '09)*, 582-591, Springer-Verlag.

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

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)*, 82-91. AAAI Press.

Golomb, S., Baumert, L. 1965. Backtrack programming. *Journal of the ACM* 12: 516-524.

Grandcolas, S., Pain-Barre, C. 2007. Filtering, Decomposition and Search Space Reduction for Optimal Sequential Planning. *Proceedings of AAAI-2007*, 993-998, AAAI Press.

Gregory, P., Long, F., Fox. M. 2010. Constraint Based Planning with Composable Substate Graphs. *Proceedings of 19th European Conference on Artificial Intelligence (ECAI)*, 453-458, IOS Press.

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

Huang, R., Chen, Y., Zhang, W. 2010. A Novel Transition Based Encoding Scheme for Planning as Satisfiability. *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)*, 89-94. AAAI Press,

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

Long, D., Fox, M., 2003. Plan Permutation Symmetries as a Source of Planner Inefficiency, *Proceedings of Workshop of the UK Planning and Scheduling Special Interest Group* (PlanSIG).

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

Pralet, C., Verfaillie, G. 2009. Forward Constraint-Based Algorithms for Anytime Planning. *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS)*, 265-272, AAAI Press.

van Beek, P. and Chen, X. 1999. CPlan: A Constraint Programming Approach to Planning. *Proceedings of AAAI 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. *Proceedings of the AAAI Conference on Artificial Intelligence* (*AAAI-04)*, 570-577, AAAI Press.