A Novel Constraint Model for Parallel Planning

Roman Barták

Charles University in Prague, Faculty of Mathematics and Physics
Malostranské nám. 2/25, 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

Constraint satisfaction techniques are very successful in areas such as scheduling but they have not achieved similar success in the closely related area of planning. Constraint satisfaction was first applied in manually designed models for concrete planning domains in the system CPlan (van Beek and Chen, 1999). It seems that manual formulation of planning problems as constraint satisfaction problem is an efficient way to solve certain planning problems as was for example demonstrated for the Settlers domain in (Gregory and Rendl, 2008). Nevertheless, in this paper we focus on solving techniques that are applicable to any planning problem. It means that we are interested in fully automated reformulation of the planning problem into a constraint satisfaction problem (CSP). This research branch has also a long tradition but it mostly follows the ideas introduced in Boolean satisfiability (SAT) based planners.

One of the typical conceptual differences between the planning problems and the constraint satisfaction problems is that the length of the final plan is unknown in advance while the constraint satisfaction problem is formulated with a fixed number of variables. Kautz and Selman (1992) showed that 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. Constraint-based planning follows this idea and we will use it in our approach as well. The first automated constraint models for planning were based on another successful planning approach – a planning graph introduced in the Graphplan algorithm (Blum and Furst, 1997). Do and Kambhampati (2000) showed that constraint satisfaction techniques can be applied to plan extraction from the planning graph. Their system GP-CSP automatically encoded the planning graph as a CSP. They used the idea that the decision variable in the CSP model describes which action makes a given predicate valid (or invalid). Hence there was a CSP variable for each predicate in each layer. Lopez and Bacchus (2003) suggested a more efficient constraint formulation of the planning graph – CSP-PLAN – that used Boolean variables and successor-state constraints (Reiter, 2001). These constraints merge the constraints describing actions effects and frame axioms together. In some sense in our novel model we combine both these approaches in the way that we use a decision variable for each predicate (in fact we use the multi-valued state variables, see below) that describes the action deciding the validity of this predicate.

Following the Graphplan origins, GP-CSP and CSP-PLAN are looking for parallel plans consisting of a sequence of sets of actions, where the actions in the set can be ordered in any way. In (Barták and Toropila, 2008) we reformulated these models in several ways. 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 formulation should be more suited for constraint satisfaction techniques as it leads to fewer variables with larger domains where domain filtering pays off. Second, we encapsulated 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. Finally, we formulated the models for sequential planning which makes it easier to formulate the constraints for multi-valued state variables.

Sequential planning brings some disadvantages over parallel planning. For example, it is more liable to exploring symmetrical plans where some actions can be swapped without changing the overall effect. This is called *plan-permutation symmetry* (Long and Fox, 2003) and in (Barták and Toropila, 2009b) we suggested symmetry breaking constraints to remove some of these symmetries in the constraint models from (Barták and Toropila, 2008). A similar approach was also used (Grandcolas and Pain-

Barre, 2007). Partial order planners use a different modelling approach and they are not liable to this problem. For example, CPT (Vidal and Geffner, 2004) is probably the most successful (in terms of International Planning Competition) constraint-based planner that does partial-order planning. In (Barták and Toropila, 2009a) we integrated some ideas of partial-ordering planning into our sequential planner SeP which brought some efficiency improvement especially for harder problems. This dual model also brought some initial ideas how to do parallel planning with multi-valued state variables.

In this paper we propose a completely new constraint model for parallel planning. This model is motivated by recent success of another SAT-based planner that exploits the multi-valued state variables (Huang, Chen, and Zhang, 2010). It seems that from the modelling perspective, the success of such SAT based planners is based (among others) on two aspects – parallel planning and exploiting mutex relations. In our current model, we are going back to parallel planning. Though it looks like a return to the planning graph ideas, the novel model is much closer to the modern timeline-based approach to planning (Pralet and Verfaillie, 2009). Nevertheless, we still assume classical planning with instantaneous actions which makes the model simpler than the Constraint Network on Timelines from (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. This synchronization basically means that if some action is selected to change the value of one state variable and the same action is changing other state variables (effect) or requires them to have some particular value (precondition) then the evolution of these other state variables must assume this action. The proposed model uses only two core types of the constraints: one describing the evolution of the state variables (sequencing) and the other one for synchronization of these evolutions. There are additional constraints that bridge the sequencing and synchronization constraint to improve the inference. Despite the simplicity of the constraint model and the traditional search strategy, the novel planer, which we call PaP (Parallel Planer), finds plans much faster than the other modern constraint-based planners such as SeP and Constance (Gregory, Long, and Fox, 2010) for some domains. Note that these planners use some advanced techniques from planning and constraint satisfaction such as no-good learning, singleton consistency, lifting, search based on relevant actions (SeP) or macro-actions (Constance).

The paper is organized as follows. First, we will formally introduce the planning problems to be solved and define the parallel plans. 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 preliminary experiments comparing PaP and SeP.

The Problem

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.

For describing the 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.

The actions change the values of the state variables. An 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, the *action precondition* is a set (a conjunction) of expressions in the form either *state-variable = value* or *state-variable* $\in$ *set-of-values* (not in SAS+) such that each state variable appears at most once in this set. The action effect describes what the values of certain state variables will be after performing the action. Formally, the action effect is a set (a conjunction) of expressions in the form *state-variable* $\leftarrow$ *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* $\in$ *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.

Domain

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

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 (for simplicity, we will be talking about values) 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 ($v = b$) 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. To represent a particular planning problem, we need to identify the initial state of the FSA which is the initial value of the state variable. 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.

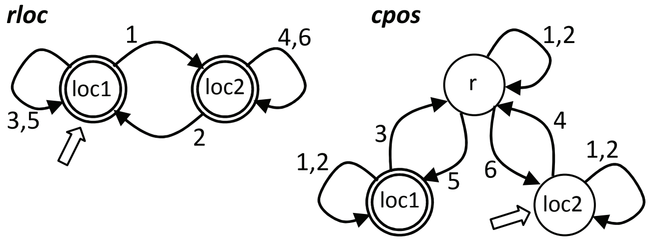

Figure 2. Representation of the planning domain (Figure 1) and problem using finite state automata.

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.

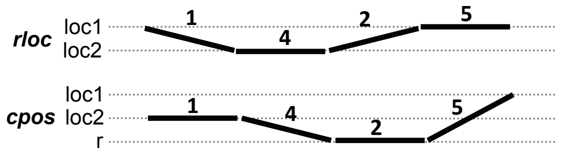

Figure 3. Timelines describing the synchronized evolution of the state variables (the action numbers are taken from Figure 1).

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. The FSA states still represent the values of the state variable. However, for a given state variable, the automaton 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 traditional no-op actions used for example in the planning-graph model (Blum and Furst, 1997) 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.

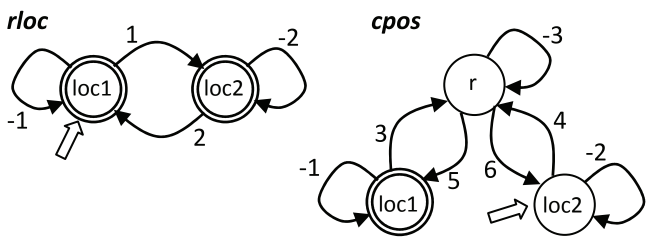

Figure 4. Representation of the planning domain (Figure 1) and problem using finite state automata with no-op actions.

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.

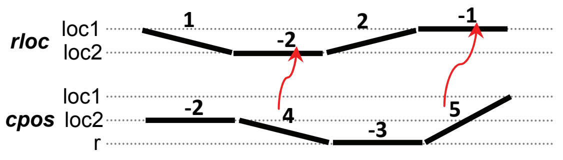

Figure 5. Timelines describing the evolution of the state variables with no-op actions (the arrows indicate synchronization).

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

As surveyed in the introduction there exist several approaches how to model the planning problem as a constraint satisfaction problem. There are ad-hoc techniques such as (Gregory and Rendl, 2008) or CPlan (van Beek and Chen, 1999) and techniques based on the planning graph such as GP-CSP (Do and Kambhampati, 2000) and CSP-Plan (Lopez and Bacchus, 2003). The SeP (Barták and Toropila, 2010) and Constance (Gregory, Long, and Fox, 2010) planners use in some form the natural FSA representation as showed in Figure 2. CPT (Vidal and Geffner, 2004) uses a specific constraint model for partial-order planning with some additional restrictions on the number of appearances of each action. In this paper we suggest a constraint model based on the FSA representation with no-op actions as shown at Figure 4.

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 ($i = 1,\dots,k$, $j = 0,\dots,n$). 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. Many constraint solvers provide the *regular* constraint (Pesant, 2004) to encode a finite state automaton, but we decided for traditional ad-hoc constraints modelling each transition separately. Note that this model does not hinder propagation in comparison with the *regular* constraints and it gives us a direct access to the state variables that are hidden in the *regular* constraint. Our *sequencing constraint* is a ternary 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 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 *p*-th state variable then $A^p_j = A$,
- if A has a precondition (but not effect) in the *p*-th state variable and B is the no-op action corresponding to the value of the precondition then $A^p_j = B$.

If A is a no-op action then we assume that it is compatible with all actions in other action variables to make to 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. Thought 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 is one more constraint connecting all action variables in the layer and requesting at least one action to be the real action.

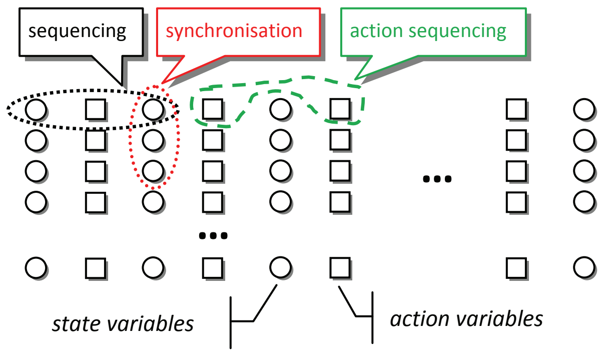

Figure 6. The structure of the constraint model, circles represent the state variables, squares represent the action variables.

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. Just note that the action sequencing constraints do not bring additional inference for this particular problem and are purely redundant. One can also 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

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

action sequencing

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

A^{cpos}_i	A^{cpos}_{i+1}
3	{-3,5,6}
4	{-3,5,6}
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 must always be accompanied by a 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 (prefer the variable whose instantiation with fail with the highest chance). 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 (prefer the value belonging to some solution), 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. Note that the instantiation of the action variable is done in two steps and hence the same action variable appears in two branching nodes. First, the domain of the variable is split into two parts. Second, if we are exploring the branch where only the real actions are left in the domain then the particular action is assigned to the variable.

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 planning problems were translated from the STRIPS representation to SAS^+ representation using Malte Helmert's translator (we were not able to translate some problems in the openstack and airport domains). The experiments ran on Intel Xeon CPU E5335 2.0 GHz processor with 8GB RAM under Ubuntu Linux 8.04.2 (Hardy Heron). Both planners run with the 30 minutes timeout. The reported runtime is in milliseconds and it includes the time to generate the constraint model (prepare the tables) and the time to find the shortest plan. Time to convert the STRIPS representation to SAS^+ representation is not included. At the beginning, we must highlight that SeP generates the shortest sequential plans while PaP generates the shortest parallel plans so both planners are solving a slightly different task. In the detailed comparison (Table 2) we report the size of the plans both in terms of the number of actions (the length of the sequential plan) and the number of layers (the length of the parallel plan).

Table 1 reports the number of solved problem instances in selected domains. SeP planner is significantly better in the openstack domain and it is also slightly better in the elevator domain. In the blockword and depots domains, both planners are comparable, though the runtimes for the blockworld (Table 2) are usually much better for SeP, while the runtimes for the depots instances are better for PaP. In other tested domains, PaP is better than SeP. For most of these domains we report the detailed results in Table 2, where we compare both the runtimes and the length of the found plans. The time efficiency of PaP is significantly better than SeP. Recall, that SeP finds the shortest sequential plans while PaP generates the shortest parallel plans. Nevertheless, as Table 2 shows, the sequential plans generated by PaP (from the parallel plans) have the same length or are only slightly longer than the plans generated by SeP.

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***		
airport-p03	17	9	17	8780	**810**
airport-p06	41	21	41	1736 250	**13 670**
airport-p07	≥38	21	41	-	**13 720**
airport-p12	39	21	39	1459 030	**19 560**
airport-p13	37	21	37	1548 350	**20 440**
airport-p15	≥31	22	58	-	**104 720**
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**	-
depots-p01	10	5	11	710	**270**
depots-p02	15	8	16	149 520	**3 870**
driverlog-p01	7	6	8	110	**40**
driverlog-p02	19	9	21	1017 570	**100**
driverlog-p03	12	7	12	11 650	**110**
driverlog-p04	≥14	7	18	-	**250**
driverlog-p05	≥13	8	21	-	**190**
driverlog-p06	11	5	11	83 940	**160**
driverlog-p07	≥11	6	18	-	**190**
driverlog-p08	≥13	7	25	-	**2 690**
driverlog-p09	≥16	10	24	-	**12 680**
driverlog-p10	≥12	7	20	-	**446 580**
driverlog-p11	≥13	9	21	-	**13 130**
freecell-p01	8	5	11	63 210	**3 300**
freecell-p02	≥9	8	20	-	**462 890**
freecell-p03	≥10	7	21	-	**43 250**
rovers-p01	10	5	10	940	**60**
rovers-p02	8	4	8	370	**20**
rovers-p03	11	7	12	2 190	**120**
rovers-p04	8	4	8	440	**60**
rovers-p05	≥13	5	22	-	**150**
rovers-p06	≥15	≥8		-	-
rovers-p07	≥11	5	18	-	**120**
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 constrained-based planner PaP for parallel planning with multi-valued state variables. This model uses only two types of base constraints: one for describing changes of a particular state variable and one for synchronization of these changes between the different state variables. There are additional sequencing constraints that assume several state variables together to strengthen inference. The used search strategy is based on traditional search strategies used in constraint satisfaction.

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 several planning domains. We are currently analyzing the experimental results to identify the weak parts of PaP in comparison with SeP. As PaP is not using any advanced planning techniques, there is still an open area for further improvements.

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 suggestions for future work.

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), AAAI Press, 525-530.

Barták, R., Toropila, D. 2009a. Integrated Constraint Models for Sequential and Partial-Order Planning. *Proceedings of the Eighth Symposium on Abstraction, Reformulation and Approximation* (SARA), AAAI Press, 18-25.

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

Barták, R., Toropila, D. 2010. Solving Sequential Planning Problems via Constraint Satisfaction. *Fundamenta Informaticae*, Volume 99, Number 2, IOS Press, 125-145.

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-2000), AAAI Press, 82-91.

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.

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

Gregory, P., Rendl, A., 2008. A Constraint Model for the Settlers Planning Domain. *Proceedings of the 27th Workshop of the UK Planning and Scheduling Special Interest Group* (PlanSIG), Heriot-Watt University.

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 Intelligenc*e (AAAI-10), AAAI Press, 89-94.

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

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.

Pesant, G., 2004. A Regular Language Membership Constraint for Finite Sequences of Variables. *Principles and Practice of Constraint Programming*, LNCS 3285, Springer, 482-495.

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

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.