Validating Plans with Durative Actions via Integrating Boolean and Numerical Constraints

Roman Barták

Charles University in Prague, Faculty of Mathematics and Physics
Institute for Theoretical Computer Science
Malostranské nám. 2/25, 118 00 Praha 1, Czech Republic
bartak@kti.mff.cuni.cz

Abstract. The paper describes a constraint programming approach for validating and finishing partially ordered plans with durative actions. We propose a Boolean constraint model of planning graphs, a numerical constraint model of durative actions and precedence relations, and channeling constraints connecting both models. We also discuss solving techniques, in particular using binary decision diagrams versus constraint propagation.

Introduction

Artificial Intelligence Planning is an area dealing with finding plans that convert some initial state of the world into a desired state. Probably the most widely used formulation of the planning problem is a STRIPS model [5]. The *state of the world* is described there as a conjunction of propositions that are either valid – *positive propositions* – or invalid – *negative propositions*. The state can be changed by *actions* that make some propositions valid – an *add effect* – and other propositions invalid – a *delete effect*. The action can be applied to a given state only if an action precondition is satisfied. The *action precondition* is expressed as a propositional formula over the state propositions. Together, the standard STRIPS formulation of the planning problem consists of a finite set of actions, a finite set of propositions describing the initial state, and a finite set of propositions describing the desired state. The planning task is to find a sequence of actions converting the initial state into a desired state.

An extension of the above formulation unites the sets of actions into a so called task networks. The *task network* is a partially ordered set of actions achieving some goal; usually alternative decompositions of the network to actions exist. We can see the task network as a meta-action and so it is also possible to nest the task networks. In a so called Hierarchical Task Network (HTN) planning [4] the planning task is solved by decomposing the task networks into actions until a valid plan is obtained.

The above purely logical formulation of the planning problem has been recently updated to cover numerical features, in particular durative actions and numerical preconditions and effects modeling resources. Dealing with durative actions and numerical resources is one of the hottest topics in planning [6].

In this paper we address the problem of validating task networks with durative actions. We can see this problem from a general view as a plan validation problem where the plan is given by a partially ordered set of actions. Plan validation consists of finding a time allocation for the actions respecting the precedence relations and the logical dependencies between the actions. By logical dependencies we understand the relations between the actions expressed in preconditions and effects.

The plan validation problem appears in early and final stages of planning. The HTN planners require such validation to check whether the task networks specified by the user are feasible and to derive additional information about the network, for example what is its minimal duration. The planners producing partially ordered plans may use the validation tool to finish the plan by allocating the actions to time.

We propose to use constraint satisfaction technology for plan validation. In particular, we integrate the Boolean constraints modeling the logical features of the plan with the numerical constraints modeling durations and precedence relations. We also discuss various constraint satisfaction techniques to solve the proposed model, namely binary decision diagrams and singleton consistency.

The paper is organized as follows. First, we will give more details about the problem to be addressed by this paper. Then, we will survey the existing results related to our problem area. The main part of the paper is dedicated to a description of the constraint models and solving techniques. We will conclude with a discussion of the results and possible extensions of presented work.

Problem Area – Plan Validation Problem

We solve the problem of validating partially ordered plans with durative actions. We use a propositional representation of the world states there. It means that a finite set of propositions is given and we describe the state by telling which propositions are valid. The propositions which are not valid are invalid. The *initial state* is specified by a propositional formula over the propositions - the formula is built over the propositions using conjunction, disjunction, and negation. The *desired state* is also described using a propositional formula and it is also possible to specify another propositional formula – a so called *invariant condition* – that must be satisfied by all the states. Notice that both the initial state and the desired state can be specified incompletely meaning that the propositional formula does not force validity or invalidity of all the propositions. For example, the formula *(1 and (not 2) and (3 or 4))* sets the proposition 1 to be valid and the proposition 2 to be invalid but it does not force validity of 3 or 4 and it says nothing about validity of propositions other than 1,2,3, or 4.

The plan is given by a finite set of actions. Each action has assigned a duration that could be either a positive number or an interval starting with a positive number. The action has a *precondition* that is a propositional formula built from the propositions using conjunction, disjunction, and negation. The precondition must be satisfied when the action starts. The action has an *effect* that is a list of added and deleted propositions. It is also possible to specify an *action invariant condition* that is a proposition formula that must hold when the action is executed.

Last but not least, there is partial order specified among the actions. Durative actions have a start time and an end time that we call *action time points*. We allow specifying order of action time points, for example to say that the action A starts before the action B finishes or that the action A starts at the same time as the action B. Moreover these simple ordering constraints can be connected via disjunction and conjunction. Thus, one may specify simple unary resource sharing using the familiar disjunctive relation – either A finishes before B starts or B finishes before A starts. We call all these formulas *precedence constraints*.

The basic task is to decide whether the plan is feasible. It means that there exist times for the activity time points in such a way that the precedence constraints are satisfied and the plan is valid with respect to all the logical relations. In particular, the given plan transfers the initial state into a desired state, all intermediate states satisfy the invariant condition, and the actions are applied correctly. For example an action deleting some proposition cannot precede directly another action that uses this proposition as its precondition. Recall that the activities may overlap so it might be also useful to find the shortest plan in terms of total duration.

Related Works

Probably the most widely used approach to planning is based on a so called *planning graph* by Blum and Furst [1]. The planning graph is a layered graph starting with a propositional layer, continuing with an action layer, followed by another propositional layer and so on until the final propositional layer. The propositional layers describe the world state. The action layers specify the actions changing the state described by the preceding propositional layer into a state described by the next propositional layer. Planning is done by constructing the planning graph of a given size and extracting the plan. If no plan exists then a longer planning graph is constructed until a plan is found. Do and Kambhampati [3] proposed a constraint encoding of the planning graph so the plan extraction stage can be done using a constraint satisfaction technology. Lopez and Bacchus [7] improved the encoding to be more efficient. We will further extend this encoding to handle durative actions and precedence relations.

Constraint Model

To solve the plan validation problem, we use an extended version of the planning graph covering durative actions. In this section we will present this extension together with its constraint representation. Recall that the constraint representation consists of the set of variables, their domains, and the set of constraints

Because the actions may overlap, we do not know the number of propositional layers in the planning graph in advance. However, we know the number of actions in the plan, say N. Each action requires the state when the action starts and the state when the action ends so we can deduce that the maximal number of states in the plan and hence the maximal number of propositional layers in the planning graph is 2*N.

Logical constraints

The propositional layer consists of a set of binary variables – one variable per proposition. Let us use a 0-1 variable $Prop_{L,i}$ to denote the validity of proposition *i* in the layer *L*. Recall, that the actions may overlap so before an action finishes another action may start. Thus, an action may spread through several consecutive layers because we need to capture the states inside the duration of the action when another action starts or stops. To model this situation, we use three types of the action layers: start, middle, and stop layer. The action layer consists of a set of binary variables – one variable per action – that indicate when the action starts, runs, and ends. Let us denote these variables $Start_{L,a}$, $Middle_{L,a}$, and $End_{L,a}$ for the layer *L* and the action *a*. Figure 1 shows how these variables are used to describe the position of an action.

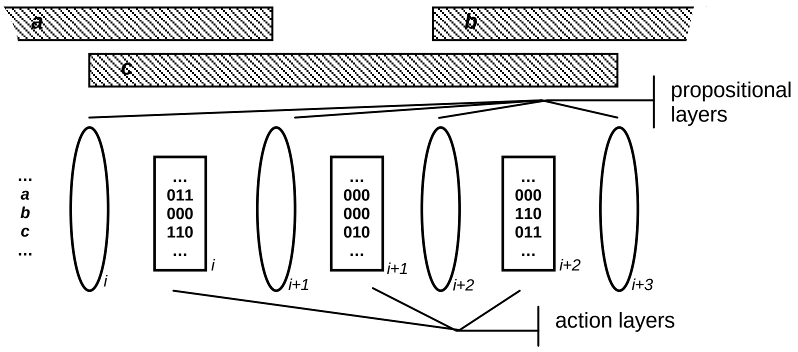

Fig. 1. Extended planning graph. The numbers in action layers indicate the values of *Start*, *Middle*, and *End* variables for actions *a*, *b*, and *c*.

Notice that an action starts in exactly one layer and stops in exactly one layer. This can be modeled using the following constraints:

$$\forall a\text{:}\ \ (\Sigma_{L=1,..,2*N-1}\ Start_{L,a} = 1)$$
$$\forall a\text{:}\ \ (\Sigma_{L=1,..,2*N-1}\ End_{L,a} =1)$$

We can see the 0-1 variables as Boolean variables where 1 indicates *true* and 0 is *false*. This view simplifies the notation of the following logical constraints.

An action is processed in all layers between the start layer and the end layer. It means that the action is being processed in a given action layer if it starts here or if it has been processed in the previous layer and it did not stop here:

$$\forall a\ \forall L\in\{2,..,2*N-1\}\text{:}\ \ Middle_{L,a} = (Start_{L,a} \vee (Middle_{L-1,a}\ \&\ (\neg\ End_{L-1,a})))$$
$$\forall a\text{:}\ \ Middle_{1,a} = Start_{1,a}$$

We use propositional formulas to describe the initial state, the desired state, and the invariant condition. These formulas can be directly converted to the constraints over the first propositional layer for the initial state, over the final propositional layer for the desired state, and over all the layers for the invariant condition. We also use the propositional formulas for the action preconditions and for the action invariant conditions. Let us denote $Precondition_{L,a}$ the precondition for the action *a* defined over the propositions in the layer L and $Invariant_{L,a}$ the action invariant condition for the action a over the propositional layer L. Then we can describe satisfaction of action precondition and action invariant condition using the following constraints:

$$\forall a\ \forall L \in \{1,..,2{*}N\text{-}1\}:\ \ Start_{L,a} \Rightarrow Precondition_{L,a}$$
$$\forall a\ \forall L \in \{1,..,2{*}N\text{-}1\}:\ \ Middle_{L,a} \Rightarrow Invariant_{L,a}$$

The actions have add and delete effects that change validity of propositions. Because the validity of propositions is not changing between the propositional layers, the only possibility for change is when some action finishes in the preceding action layer. Assume that *Add(i)* is a set of actions that make the proposition *i* valid – the proposition *i* is an add effect of these actions – and *Del(i)* is a set of actions that make the proposition *i* invalid – the proposition *i* is a delete effect of these actions. The proposition is valid in some layer if it is an add effect of some action finishing in the preceding action layer or if the proposition is already valid in the preceding propositional layer and there is no action deleting it and finishing in the preceding action layer. Formally:

$$\forall i\ \forall L \in \{1,..,2{*}N\text{-}1\}: Prop_{L+1,i} = (\vee_{a\in \mathrm{Add(i)}} End_{L,a}\ \vee (\&_{a\in \mathrm{Del(i)}} (\neg\ End_{L,a})\ \&\ Prop_{L,i})).$$

Note that the above constraint allows the proposition to be true if it is added by some action and deleted by another action. This is not allowed and so we use the following constraint to remove such situation:

$$\forall i\ \forall L \in \{1,..,2{*}N\text{-}1\}: Prop_{L+1,i}\ \Rightarrow (\&_{a\in \mathrm{Del(i)}} (\neg\ End_{L,a})).$$

The above constraint model fully describes the logical relations in the plan. It means that we have a valid plan if all the variables are assigned and all the constraints are satisfied. Recall that we use the upper estimate on the number of layers so it is possible that some of the layers are not necessary. Basically, action layers may exist such that no action is starting or finishing in them – let us call them *empty action layers*. These empty action layers may appear anywhere in the planning graph which increases the number of valid but equivalent solutions and also increase the size of the search space to be explored when solving the problem. We propose to collect these empty layers to the end of the planning graph. The following constraint ensures that if there in non-empty action layer then all preceding action layers are also non-empty:

$$\forall L \in \{1,..,2{*}N\text{-}1\}:\ \vee_a (Start_{L+1,a} \vee End_{L+1,a}) \Rightarrow \&_{K=1,..,L} (\vee_a (Start_{K,a} \vee End_{K,a}).$$

Numerical constraints

The propositional layer models the state at some time so a numerical time variable $Time_L$ is attached to each layer L. Assume that we measure time from zero so the initial state is in time zero hence $Time_1 = 0$. Moreover, the layers describe how the state evolves in time. We do not know the actual time distance between the layers so we can post only the following constraints:

$$\forall L \in \{1,..,2{*}N\text{-}1\}:\ \ Time_L < Time_{L+1}.$$

We have only simple temporal relations between the actions so we can estimate the upper bound for the plan length as a sum of durations of all the actions. This upper bound defines the upper bound for domains of the variables T_L.

Each action starts at some time, stops at another time, and has a duration specified by the user. Let us denote $StartTime_a$, $EndTime_a$, and $Duration_a$ the start time, the end time and the duration of the action a. There is a constraint connecting these variables:

$$StartTime_a + Duration_a = EndTime_a.$$

The precedence constraints from the problem specification can now be directly expressed over the *StartTime* and *EndTime* variables.

Channeling constraints

To connect the logical and numerical parts of the constraint model we need to identify the layers where the action starts and stops. Let us use the following two variables to identify the action layers where the action starts – $StartLayer_a$ – and stops – $EndLayer_a$. Visibly, the following constraint must hold: $StartLayer_a \leq EndLayer_a$.

The connection between the logical variables describing the position of the action in the planning graph and *StartLayer* and *EndLayer* variables is established using the following constraints:

$$\forall a:\ Start_{StartLayer_a,a} = 1$$
$$\forall a:\ End_{EndLayer_a,a} = 1$$

Note that such constraints can be easily modeled in existing constraint satisfaction packages using the `element` constraint [9].

Finally, it is necessary to connect the action time points with the times of the propositional layers. This connection can be realized in the same way as above. Just note that the end propositional layer for the action has a one point larger index then the index of the end action layer (see Figure 1).

$$\forall a:\ Time_{StartLayer_a,a} = StartTime_a$$
$$\forall a:\ Time_{EndLayer_a+1,a} = EndTime_a$$

Solver

One of the advantages of constraint programming is the possibility to specify a constraint model independently of the particular constraint solver. We have implemented the above described constraint model using constraint satisfaction packages in SICStus Prolog [9]. In the first step, we have tried a Boolean constraint solver to implement the logical constraints and a finite domain solver to implement the numerical constraints. Note that the integration of the solvers is natural via shared variables. The Boolean solver is based on Binary Decision Diagrams (BDDs) [1] so it can produce a solution without search. However, this solver requires a lot of memory and it crashed even for small problems. Therefore, we decided to use a finite domain solver for the logical constraints as well. Instead of logical operations we use corresponding arithmetical operations over the 0-1 variables as Table 1 shows.

Table 1. Conversion between logical and arithmetical operations.

logical operation	**arithmetical operation**
$A \vee B$	min(1,A+B)
A&B	A*B
$\neg A$	1-A
$A \Rightarrow B$	$A \leq B$

The finite domain constraint solver uses the techniques of constraint propagation, in particular generalized arc consistency, to remove inconsistent values from variables' domains. Not surprisingly, constraint propagation is weaker then BDDs. To achieve better pruning, we have applied singleton consistency [8] to the Boolean variables modeling the planning graph. There exist some studies comparing the power of BDDs with constraint propagation and search [10,11] but we are not aware about any work comparing BDDs to singleton arc consistency. Anyway, in our test models, we have achieved the same pruning as BDDs without the memory consumption of BDDs. Thus, singleton consistency seems to be an appropriate method for initial domain pruning of Boolean variables modeling the planning graph.

Constraint propagation can prune the domains but it does not guarantee existence of the solution. Thus, it is usually combined with search that attempts to assign values to the variables – this is often called labeling. We have decided that only the numerical variables will participate in labeling, namely *StartLayer*, *EndLayer*, *StartTime*, and *EndTime*. If these variables are assigned, constraint propagation ensures that the relevant Boolean variables in the action layers are assigned as well. In our models, we use action preconditions in the form of conjunction only (which is the case of most planning problems) so the Boolean variables in the proposition layers are decided as well. If this is not the case, these variables should be labeled as well. We decided to use numerical variables in labeling because then the standard variable ordering heuristics, like first-fail, play role and they can improve efficiency of search. We first label the layer variables *StartLayer* and *EndLayer*. This ensures that the actions are located to layers so all the logical relations between them are valid. In the second round, we label the time variables *StartTime* and *EndTime*. The labeling procedure is wrapped into a branch-and-bound algorithm that minimizes the completion time of the last action so we get a plan with the minimal makespan.

Conclusions

In this paper, we proposed a new constraint-based approach for validating task networks and finishing plans with durative actions. Its main advantages are clarity and extendibility. It is easy and fast to implement the model in existing constraint packages like SICStus Prolog. Moreover, the constraint satisfaction technology allows painless extensions of the model. We have already integrated Boolean and numerical constraints there and it is possible to add other constraints for example modeling numerical effects of the actions or resource consumption and production. Also, the solving technology can be changed without necessity to modify the constraint model. We have tried a Boolean solver based on Binary Decision Diagrams (BBDs) to solve the Boolean part of the model. It has the advantage of providing solutions without search but it consumes a lot of memory. Therefore, we have converted the Boolean constraints into numerical ones and we used standard generalized arc consistency for the whole model improved by singleton consistency for the original Boolean part. The experiments showed that this approach achieves the same domain pruning as BBDs without horrible memory consumption. However, a further theoretical study is necessary there which could be based on works [10,11]. Our future research will go in the direction of generalizing the proposed approach to do full planning. We are also going to perform extensive tests of the proposed constraint model.

Acknowledgements

Research is supported by the Czech Science Foundation under the contract no. 201/04/1102. I would like to thank Susanne Biundo and her team, in particular to Hartmut Jungholt, for introducing me to the problem of task network validation and for providing data sets for experiments.

References

1. Blum, A., Furst, M.: Fast planning through planning graph analysis. Artificial Intelligence 90 (1997) 281–300
2. Bryant, R.E.: Graph-Based Algorithms for Boolean Function Manipulation. IEEE Transactions on Computers C-38-8 (1986) 677–691
3. Do, M.B., Kambhampati, S.: Planning as Constraint Satisfaction: Solving the planning graph by compiling it into CSP. Artificial Intelligence 132 (2001), 151–182
4. Erol K., Hendler J., and Nau, D.: UMCP: A Sound and Complete Procedure for Hierarchical Task-Network Planning. In Proceedings of Second International Conference on AI Planning Systems (1994) 249–254
5. Fikes, R. E. and Nilsson, N. J.: STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. Artificial Intelligence 3–4 (1971) 189–208
6. Fox, M., Long, D.: PDDL 2.1: An extension to PDDL for expressing temporal planning domains. Journal of Artificial Intelligence Research (2003)
7. Lopez, A., Bacchus, F.: Generalizing GraphPlan by Reformulating Planning as a CSP. In Gottlob, G., Walsh, T. (eds.): Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence. Morgan Kaufmann Publishers (2003), 954–960
8. Prosser P., Stergiou K., Walsh T.: Singleton Consistencies. In Principles and Practice of Constraint Programming (CP). LNCS. Springer-Verlag (2000) 353–368
9. SICStus Prolog 3.11.0 User's Manual, SICS (2003)
10. Uribe, T., Stickel, M.E.: Ordered Binary Decision Diagrams and the Davis-Putnam Procedure. In Jouannaud J.P. (ed.): Proceedings of the First International Conference on Constraints in Computational Logics. LNCS, Vol. 845. Springer-Verlag (1994), 34–49
11. Walsh, T.: SAT v CSP. In Proceedings of CP-2000. LNCS, Vol. 1894. Springer-Verlag (2000) 441–456