A Flexible Constraint Model for Validating Plans with Durative Actions

Roman BARTÁK
Charles University, Malostranské nám. 2/25, 118 00 Praha, The Czech Republic
roman.bartak@mff.cuni.cz

Abstract. Planning problems with durative actions represent one of the hot research topics in AI planning. Durative actions introduce numerical aspects to planning and so constraint satisfaction technology is becoming more popular in solving this new type of planning problems. The paper describes a constraint programming approach for validating and finishing partially ordered plans with durative actions. In particular, we propose a Boolean constraint model of the planning graph, a numerical constraint model of durative actions and precedence relations, and channeling constraints connecting both models. We also briefly discuss solving techniques for such integrated model, in particular using binary decision diagrams versus constraint propagation.

1. Introduction

AI 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.

Sometimes, a set of actions in encapsulated into an abstract task that is often decomposable to several sets of actions (and tasks). Then, the planning task is to find the right decomposition of given abstract tasks into primitive actions that form a valid plan in terms of the STRIPS model. This is called Hierarchical Task Network (HTN) planning [4]. We call the decomposition of the abstract task a *task network* because the actions and tasks in the decomposition are often connected, for example they are partially ordered.

The above purely logical formulation of the planning problem has been recently updated to cover numerical features [6]. In particular, durative actions and numerical preconditions and effects modeling resources are assumed. In this paper, we cover durative actions with logical preconditions that must be satisfied when the action starts, and the logical effects that become valid when the action finishes. Moreover, we allow specifying an invariant condition that must be valid during execution of the action. Naturally, when durative actions are assumed then overlapping of actions is allowed to obtain shorter plans in terms of time.

In this paper, we address the problem of validating task networks with durative actions. Such a task network, i.e., a decomposition of the abstract task into actions and other abstract tasks, is given by the user. Before using this task network by the planner, it is better to check, whether the task network can form a valid plan. If it cannot form a valid plan then the decomposition of the abstract task into such task network should be avoided during planning because it leads to a dead end. The 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. We can see this problem from a more general perspective as a plan validation problem where the plan is given by a partially ordered set of actions. In addition to checking validity of the task network, we may require some additional information about the task network that can be used during planning. For example, one may ask what the minimal makespan of the task network is, which can be used as the minimal duration of the corresponding abstract task. Moreover, it is possible to deduce what the required precondition of the task network is as well as what the necessary effect of the task network is (independently of the particular time allocation and ordering of the actions). Again, this information can be encoded in the abstract task to allow better co-ordination of the abstract tasks before they are decomposed into actions. Last but not least, the planners producing partially ordered plans may use the same tool to finish the plan by allocating the actions to time. We propose to use constraint satisfaction technology for the above described plan validation because this technology can naturally model logical features via Boolean constraints as well as durations and precedence relations via numerical constraints. Moreover, the proposed constraint model is flexible enough to allow addition of other features like numerical preconditions and effects.

The main contribution of the paper is twofold. First, an extended version of the planning graph is proposed to handle durative actions. Second, a new constraint model for this extended planning graph is designed. We also discuss various constraint satisfaction techniques to solve the proposed model, namely binary decision diagrams versus singleton consistency.

The paper is organized as follows. First, we will give more details about the plan validation problem. Then, we will survey the models supporting durative actions and constraint-based approaches to planning. The main part of the paper will be devoted to a description of the constraint model for the extended planning graph that will also be introduced there. After that we will briefly discuss the solving techniques and we will conclude with a description of possible extensions of the proposed model.

2. Plan Validation Problem

In this paper we study 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 specifying which propositions are valid. The propositions which are not valid are invalid. The *initial state* is specified by a propositional formula that is built over the propositions using conjunction, disjunction, and negation. In planning, the initial state is usually described as a list of valid propositions which is a description equivalent to a conjunction of positive (valid) and negative (invalid) propositions. We use a more general formula to describe the initial state mainly because of the task networks where the initial state is equivalent to the initial precondition of the task network so it could be a general formula. The *goal state* is described using a propositional formula too. In case of task networks, the goal is equivalent to the effect of the task network. One of the tasks that we are solving is to find out which propositions must or must not hold in the initial state and in the goal state. This information may be used to fine tune the description of the precondition and effect of the abstract task that is decomposable to a given task network. We also allow specification of another propositional formula – a so called *invariant condition* – that must be satisfied by all the states inside the task network. The invariant condition substitutes axioms from PDDL. For example, it is possible to describe axioms like “if doors 1 or doors 2 are open then the room A is accessible”. Notice also that both the initial state and the goal 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 and 4 (one of them must be valid, but the formula does not specify which one). The formula also says nothing about validity of propositions other than 1, 2, 3, and 4.

The input plan to be validated is given by a finite set of partially ordered actions. Each action has assigned a duration that could be either a positive integer or an interval starting with a positive integer. Durative actions have a start time and an end time that we call *action time points*. Action duration equals to the difference between the action time points. We expect discrete time so time points are represented by integers. Some planning problems are formulated over the continuous time but there is always some ε describing the minimal resolution of the plan so actually the time is discrete there too.

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. Formally, if *st* is the start time of the action A then the world state at time *st* must satisfy the precondition of A. The action has an *effect* that is a list of added and deleted propositions. If *p* is an add effect of some action finishing at the time *et* then *p* becomes valid at time *et*. If *p* is a delete effect of some action finishing at the time *et* then *p* becomes invalid at time *et*. Finally, it is possible to specify an *action invariant condition* that is a propositional formula that must hold when the action is executed. Formally, if an action A starts at time *st* and finishes at time *et* then the world states at times $\langle st,et\text{-}1\rangle$ must satisfy the invariant condition of A. The invariant condition of A prevents the actions that interfere with A to overlap with A.

As we already mentioned, there is a partial order specified among the actions. We allow specifying the 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 (negation is not necessary there). Thus, one may specify a shared unary resource using the familiar disjunctive relation – either A finishes before B starts or B finishes before A starts. We call all these relations *precedence constraints*.

The plan validation problem is to decide whether the plan is feasible. The feasible plan 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 the goal 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 in time so it might be useful to find the shortest plan in terms of total duration. It might be also interesting to find out which propositions must be valid or invalid in the initial and goal states. This information can be used to specify better the minimal duration, precondition, and effect of the task network.

The plan validation problem differs from the planning problem because the set of actions is known in the plan validation problem. The basic task is to find out a proper timing of the known actions which is a task closer to the scheduling problem. However, the difference from the traditional scheduling problem is that the interaction of actions is more complex via preconditions and effects.

3. 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 layer consists of nodes representing the propositions that describe the world state. The action layer consists of nodes representing the actions that change the world state. Each action is connected to its preconditions in the preceding propositional layer and to add effects in the next propositional layer. The delete effects are modeled via a so called mutex that describes activities (and propositions) that cannot be active together in the same layer. Planning is done by constructing the planning graph of a given size and extracting the plan from the graph. If no plan exists then a longer planning graph is constructed until a plan is found.

The traditional planning graph expects instantaneous actions which have no duration (or unit duration). Smith and Weld [12] proposed a temporal planning graph to handle durative actions. They use a directed graph with nodes representing propositions and actions. The arcs connect the preconditions with the action and the action with its effects. Moreover a time stamp is assigned to each node indicating the first time point at which the proposition or the action appears. The plan is constructed by backward-chaining search through the temporal planning graph. Note that even if the action appears just once in the graph, it may be added several times to the plan because of the cycles in the graph. Moreover, because there are no explicit layers in the graph, the actions may have real duration. The difficulty of this approach (from a CSP view) is that the plan is built dynamically.

Fox and Long [7] proposed a different way of handling durative actions. Their idea was to split the durative action into a set of instantaneous actions, in particular to start, invariant, and end actions. Instead of attaching duration to the actions, the duration is attached to the propositional layers. Then, the duration of the action equals to the sum of durations of the propositional layers between the start and end actions. The invariant actions are used to pass information between the start and end actions as well as to ensure the possible invariant condition of the action. The disadvantage of this approach is that more actions are necessary.

A static formulation of constraint satisfaction problems complicates usage of constraints in planning because of a dynamic character of planning (the number of planned actions is unknown). Nevertheless, as pointed out by Kautz and Selman [8] it is possible to start planning with some lower bound on the plan size and to formulate this sub-problem statically. They used a SAT formulation but it is possible to use a constraint formulation too.

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. They used variables for propositions in the propositional layers and the domain of these variables was a set of actions that have a given proposition among add effects. The constraints were used to model mutexes as well as to model preconditions of the actions. The difficulty of this approach is a large number of constraints.

Lopez and Bacchus [9] proposed a different constraint model of the planning graph that uses binary variables both for propositions and for actions. The values of the variables are *true* and *false* and they indicate whether a given node is active in the layer or not. The constraints connect actions with preconditions and effects. It is also possible to specify mutexes as constraints.

In this paper, we propose an extension of the encoding by Lopez and Bacchus [7] to handle durative actions. Actually, we use the same Boolean variables and constraints to model preconditions and effect. However, we propose an extension of the planning graph such that the action may lie in several layers. In some sense, we also use the idea of Fox and Long [7] of splitting the action into start, middle, and end part. However, the action is not really split, there will be a timetable indicating in which stage the action is. Keeping the action as a single object simplifies modeling of action duration and other relations between the actions like the precedence relations. In particular, we use numerical constraints over the actions to model these features. Note also that all the above surveyed approaches are used for planning while we are solving a plan validation problem. In particular, we expect that the actions are known and the task is to find out a proper timing of the actions respecting the logical and precedence relations between the actions.

4. 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.

4.1 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 0-1 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.

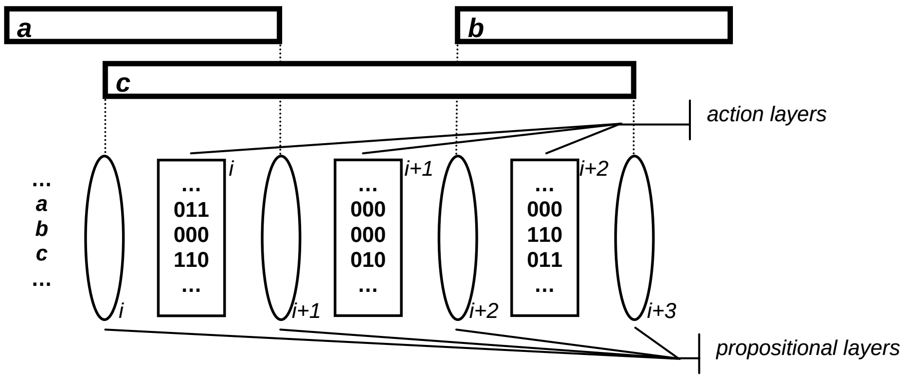

Figure 1. Extended planning graph. The 0/1 numbers in the action layers indicate the values of *Start*, *Middle*, and *End* variables for actions *a*, *b*, and *c*.

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 an 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:

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

We use propositional formulas to describe the initial state, the goal 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 goal 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 the action precondition and the action invariant condition using the following constraints:

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

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:

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

Note that the above constraint allows the proposition to be true if it is added by some action and deleted by another action at the same time. Such situation is not valid so we use the following constraint to forbid this situation:

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

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 because the actions may share some layers. 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 is a non-empty action layer then all the preceding action layers are also non-empty:

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

4.2 Numerical constraints

The propositional layer models the state at some time so a numerical time variable $Time_L$ is attached to each propositional 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:

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

We use only the simple temporal relations between the actions so we can estimate the upper bound for the plan length by 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. All these attributes could be variable so let us denote by $StartTime_a$, $EndTime_a$, 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 as equalities or inequalities (=, <, ≤) over the *StartTime* and *EndTime* variables. Note that in a constraint satisfaction framework it is possible to use more general constraints, for example to specify that the action A finishes 5 time units before the action B starts.

4.3 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 constraints:

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

Note that such constraints can be easily modeled in existing constraint satisfaction packages using the `element` constraint [11]. The semantics of the `element` constraint is as follows: `element(X,List,Y)` is true if and only if the `X`-th element of `List` is `Y`. `X`, `Y`, and the elements of `List` could be variables with finite domains.

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 unit larger index than the index of the end action layer (see Figure 1).

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

5. Solver

One of the advantages of constraint programming is a specification of the constraint model independently of a particular constraint solver. We have implemented the above described constraint model using constraint satisfaction packages in SICStus Prolog [11]. First, we 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] and it can produce a solution (for the logical constraints) 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 corresponding arithmetical operations over the 0-1 variables are used (Table 1).

Table 1. Conversion between logical and arithmetical operations.

logical operation	**arithmetical operation**
$A\vee B$	min(1,A+B)
$A\wedge B$	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 than BDDs. To achieve better pruning, we have applied singleton arc consistency [10] to the Boolean variables modeling the planning graph. There exist some studies comparing the power of BDDs with constraint propagation and search [13,14] 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. The reason could be that the domains of Boolean variables consist of two elements so if a value is removed from the domain due to inconsistency then the remaining value is assigned immediately to the variable. Note that after making the problem consistent, it is possible to deduce some information for the task network. In particular, the propositions that are known to be valid or invalid (the respective variable is instantiated) in the initial layer form a more specific precondition of the task network. Similarly, it is possible to deduce add and delete effects of the task network from the propositions in the last layer. This additional information can then be used during planning.

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 generic variable ordering heuristics, like first-fail, play a role and they can improve efficiency of search. Actually deciding a value for the *StartLayer* variable is equivalent to finding values for 2*N-1 Boolean variables *Start*. 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 to obtain a plan with the minimal makespan.

6. Example

Assume that the task network consists of three actions A, B, and C such that the actions A and B cannot overlap in time (end(A)≤start(B) ∨ end(B)≤start(A)), all actions have duration 1 and there are three predicates p, q, and r. Action A has a precondition (p ∧ r) and a delete effect {p,r}, action B has a precondition (p) and an add effect {q}, and action C has a precondition (q ∧ ¬r) and a delete effect {q}. Then the proposed plan validator deduces (using constraint propagation combined with singleton consistency, i.e., no search is used) that the ordering of actions in the network is B<<A<<C, the precondition of the network is (p ∧ r), the delete effect of the network is {p,q,r}, and the minimal duration of the network is 3 (see Figure 2).

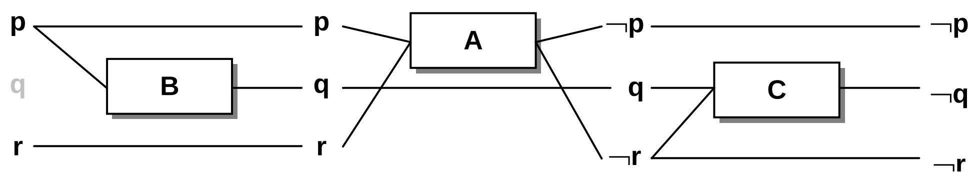

Figure 2. A simple task network.

7. 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 (note that search is still necessary to resolve the numerical part of the model). 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. We did only a few preliminary experiments that showed that using singleton consistency 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 [13,14]. Our future research will go in the direction of generalizing the proposed approach to do full planning. After this generalization we will be able to do comparison with existing approaches for temporal modeling.

Acknowledgements

Research is supported by the Czech Science Foundation under the contract no. 201/04/1102 and by the project LN00A056 of the Ministry of Education of the Czech Republic. The author has been awarded an ECCAI Travel Grant to present the paper during ECAI'04. My thanks go also to Susanne Biundo and her team, namely 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] Fox, M., Long, D.: Fast Temporal Planning in a Graphplan Framework, AIPS workshop on Temporal Planning, Toulouse, France (2002)
[8] Kautz, H. and Selman, B.: Planning as satisfiability. Proceedings of the European Conference on Artificial Intelligence (1992), 359-363.
[9] 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
[10] Prosser P., Stergiou K., Walsh T.: Singleton Consistencies. In Principles and Practice of Constraint Programming (CP). LNCS. Springer-Verlag (2000) 353–368
[11] SICStus Prolog 3.11.0 User's Manual, SICS (2003)
[12] Smith, D., Weld, D.: Temporal Graphplan with mutual exclusion reasoning, In Proceedings of International Joint Conference on Artificial Intelligence (1999)
[13] 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
[14] Walsh, T.: SAT v CSP. In Proceedings of CP-2000. LNCS, Vol. 1894. Springer-Verlag (2000) 441–456