Maintaining Arc-consistency over Mutex Relations in Planning Graphs during Search

Pavel Surynek and **Roman Barták**

Charles University
Faculty of Mathematics and Physics
Malostranské náměstí 2/25, 118 00, Praha 1, Czech Republic
{pavel.surynek, roman.bartak}@mff.cuni.cz

Abstract

We deal with the search process of the GraphPlan algorithm in this paper. We concentrate on a problem of finding supports for a sub-goal which arises during the search. We model the problem of finding supports as a constraint satisfaction problem in which arc-consistency is maintained. Contrary to other works on the similar topic, we do not model the whole planning problem as a CSP but only a small sub-problem within the standard solving process. Our model is based on dual views of the problem which are connected by channeling constraints. We performed experiments with several variants of propagation in the constraint model through channeling constraints. Experiments confirmed that the dual view of the problem enhanced with maintaining of arc-consistency is a good choice.

Introduction

Planning is an intensively studied area of artificial intelligence. The importance of studying planning arises from needs of real-life applications such as industrial automation, transportation, robotics and other braches (Nau et al., 1995). The research in planning is also motivated by needs of researches in other areas. From the traditional point of view, the planning problem is a task of finding a sequence of actions which transform a specified initial state of the planning world into a desired goal state (Allen et al., 1990; Ghallab et al., 2004). The limitation is that only actions from a set of allowed actions can be used. An individual action typically makes a small local change of the state of the world. Therefore it is necessary to carry out a set of actions in the right order to achieve the goal.

Among the most successful techniques for solving planning problems belong algorithms based on state reachability analysis. The first such algorithm was GraphPlan (Blum and Furst, 1997). The algorithm introduced a concept of so called *planning graphs*. The planning graph is a structure which makes easier answering questions whether a certain state of the planning world can be reached by using a certain set of actions. The structure of the planning graph allows discovering majority of forbidden situations quickly. This feature significantly helps to prune the search space during search for solution. Unfortunately the planning graph does not allow discovering all forbidden situations. Therefore the search is still necessary.

In this paper we concentrate on planning graphs from the constraint programming perspective. Specifically, we use consistency techniques for solving sub-problem of finding supports for a *sub-goal*. This kind of a sub-problem arises many times during the GraphPlan-style solving process. Therefore the fast answering of this problem is a key factor for the efficiency of the solving algorithm. We build a special constraint model for modeling the sub-goal sub-problem. The sub-goal sub-problem is then solved as a *constraint satisfaction problem* (Dechter, 2003) which significantly improves the search since constraint programming techniques can be used (namely constraint propagation). Our technique of choice is arc-consistency which is maintained during the search over the sub-goal model. The important feature of our model is that we model the problem by two different approaches which are combined together by a special channeling constraint. We account large portion of the overall improvement to this feature.

The paper is organized as follows. First we put our work into relation with other works. Then we describe planning graphs and GraphPlan algorithm. The main part of the paper is about our enrichment of GraphPlan solving process with our sub-goal model and its consistency. Finally we present some experimental results and discuss our contribution. Several variants of our approach are compared with the standard version of the GraphPlan algorithm in this last part.

Related Works

Lot of techniques for solving planning problems are trying to directly translate a given problem into another formalism. After translation they solve the problem in a new formalism. Many of these approaches use Boolean formula (SAT) or constraint satisfaction as the target formalism. SAT based planners are described in (Kautz et al., 1996; Kautz and Selman, 1999). The drawback of these methods is that the information induced by the original formulation is often lost during translation into the target formalism.

Some planners are trying to overcome this drawback by hand tailored encoding of a planning problem into the target formalism (Van Beek and Chen, 1999).

The significant breakthrough in planning was done when reachability analysis using planning graphs was incorporated into planners. Many of the successful existing planners use some Boolean formula or constraint satisfaction algorithms to solve models based on planning graph formulation of the planning problem (Baioletti et al., 1998; Kambhampati, 2000; Kambhampati et al., 1997; Kautz and Selman, 1999; Lopez and Bacchus, 2003). Constraint programming represents a technique which is intensively used in this way (Nareyek et al., 2005). This kind utilization of constraint programming in planning is generally more successful than the direct translation of the problem from one formalism into another (Ghallab et al., 2004).

We use constraint programming techniques to solve a small sub-problem which arises during the GraphPlan style solving process. This is in contrast to other approaches which use constraint programming formalism on the planning problem as a whole (Kambhampati, 2000). The way how we model our problem can be viewed as a synthesis of the encoding style of the planning graph as a CSP known from GP-CSP planner (Kambhampati, 2000) and the Boolean formula satisfaction approach known for example from SATPlan planner (Kautz and Selman, 1999).

GraphPlan Algorithm

The GraphPlan algorithm (Blum and Furst, 1997) relies on the idea of state reachability analysis. The state reachability analysis is done by constructing a data structure called *planning graph* in the GraphPlan algorithm. The algorithm works in two interleaved phases. In the first phase planning graph is incrementally expanded. The second phase consists of an extraction of a valid plan from the extended planning graph. If the second phase is unsuccessful the process continues with the first phase - the planning graph is expanded again.

The planning graph for a planning problem $P = (s_0, g, A)$, where s_0 is an initial state (finite set of atoms), g is a goal (finite set of literals) and A is a set of actions (each action has preconditions, positive effects and negative effects), is defined as follows. It consists of two alternating structures called a *proposition layer* and an *action layer*. The initial state s_0 represents the *0*th proposition layer P_0. The layer P_0 is just a list of atoms occurring in s_0. The rest of the planning graph is defined inductively. Consider that the planning graph with layers $P_0, A_1, P_1, A_2, P_2, \ldots, A_k, P_k$ has been already constructed (A_i denotes the *i*th action layer, P_i denotes the *i*th proposition layer). The next action layer A_{k+1} consists of actions whose preconditions are included in the *k*th proposition layer P_k and which satisfy an additional condition. This additional condition requires that no two propositions of the action are *mutually excluded* (we briefly say that they are *mutex*).

Definition 1 (Independence). *A pair of actions {a, b} is independent if and only if:*
(i) $effects^-(a) \cap (precond(b) \cup effects^+(b)) = \emptyset$ *and*
(ii) $effects^-(b) \cap (precond(a) \cup effects^+(a)) = \emptyset$.
Otherwise {a, b} is a pair of dependent actions.

Definition 2 (Action mutex / mutex propagation). *We call the two actions a and b within the action layer A_i a mutex if and only if either the pair {a, b} is dependent or an atom of the precondition of a is mutex with an atom of the precondition of b (defined in the following definition).*

Definition 3 (Proposition mutex / mutex propagation). *We call the two atoms p and q within the proposition layer P_i a mutex if and only if every action a within the layer A_i where $p \in effects^+(a)$ is mutex with every action b within the layer the A_i for which $q \in effects^+(b)$ and layer A_i does not contain any action c for which $\{p, q\} \subseteq effects^+(c)$.*

Theorem 1 (Necessary condition on state reachability). *Consider a state s containing atoms p and q that are mutex in layer P_i. Then the state s cannot be reached from the initial state s_0 by any sequence of actions determined by the action layers $A_1, A_2, \ldots, A_i$.*

We omit the proof of the theorem since it is given in details in (Blum and Furst, 1997). The theorem gives the necessary condition for the existence of a solution of the planning problem.

The key elements of the standard GraphPlan algorithm are shown here as algorithm 1. The program consists of functions for extraction of a plan from the planning graph. The program supposes that the planning graph is built for a certain length (i.e. action and proposition layers are constructed and action and proposition mutexes are propagated according to the defined rules). Then the plan is extracted recursively using backtracking search. The algorithm is trying to satisfy a goal by finding a set of actions which have this goal as their effect. Preconditions of actions from this resolving set form a new goal which is recursively satisfied in the same way.

Let us describe the process in more details. Suppose we have a goal for which we are trying to find a plan starting in the initial state. Next suppose that we know how long the planning graph should be. The process starts by construction of the planning graph of a given length. After the construction of the planning graph the algorithm starts to satisfy the goal in the last action layer by finding a set of non-mutex actions that satisfy the goal (we say these actions support the goal). The set of supporting actions have preconditions which also have to be satisfied. Preconditions of supporting actions form a new goal for the previous layer of the planning graph. The plan extracting procedure is recursively called at this point with parameters specifying the new goal and the intention to extract this goal in the previous layer. If the recursive call of the procedure is unsuccessful the algorithm continues with further attempts to find another set of non-mutex actions supporting the original goal.

Algorithm 1: Basic procedures of the GraphPlan algorithm as a pseudo-code. We use a special notation for the planning graph structure. It is denoted as pG in the code. pG/Propositions[i] denotes a set of propositions in the *i*th proposition layer (P_i), pG/Actions[i] denotes a set of action in the *i*th action layer (A_i), pG/PMutexes[i] denotes a set of proposition mutexes between propositions in the *i*th proposition layer, pG/AMutexes[i] denotes a set of action mutexes between actions in the *i*th action layer and pG/Nogoods[i] denotes a set of nogoods for the *i*th proposition layer. The resulting plan is a sequence of sets of actions. A concatenation operation is denoted by '.' (dot).

Function *ExtractPlan* gets parameters *pG* - planning graph of a certain length, *l* - layer in which the specified goal has to be satisfied and *g* - the goal. The result of the function is plan consisting of actions from action layers 1 to *l* of *pG* (an element of the resulting sequence is a set of actions from a single action layer) satisfying the specified goal *g* or *failure* if no such plan exists. Function *ExtractPlanFromLayer* gets parameters *pG* - planning graph, *l* - layer in which the specified goal has to be satisfied, *g* - the goal and *p* - plan consisting of action from the specified layer. The main purpose of this function is to find supports for the goal in the specified layer.

```
function ExtractPlan(pG,l,g):sequence
1  if l = 0 then
2    if g ⊆ pG/Proposition[0] then return (<>)
3    else return (<failure>)
4  if g ∈ pG/Nogoods[l] then return (<failure>)
5  p ← ExtractPlanFromLayer(pG,l,g,∅)
6  if p = <failure> then
7    pG/Nogoods[l] ← pG/Nogoods[l]∪{g}
8    return (<failure>)
9  else return (p)
function
ExtractPlanFromLayer(pG,l,g,p):sequence
1  if g = ∅ then
2    g1 ← {precond(a)| a∈p}
3    P ← ExtractPlan(g1,PG,l-1)
4    if P = <failure> return (<failure>)
5    else return (P.p)
6  else
7    select q ∈ g
8    supports ← {a| a∈pG/Actions[l] &
9                   & q∈effects+(a)}
10   if supports = ∅ then return (<failure>)
11   for each s∈supports do
12     if CheckSupport(pG,s,p,l) then
13      g2 ← g-effects+(s)
14      p2 ← p∪{s}
15      return ExtractPlanFromLayer(pG,l,g2,p2)
16   return (<failure>)
function CheckSupport(pG,s,p,l):boolean
1  for each r∈p do
2    if (r,s) ∈ pG/AMutexes[l] then
3      return (False)
4  return (True)
```

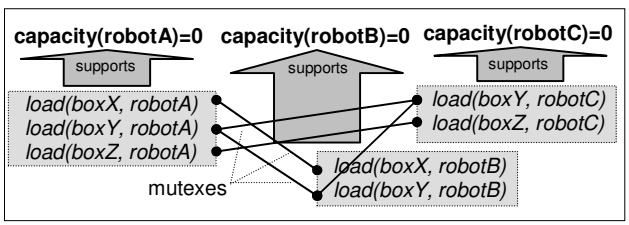

Figure 1: Example of a problem of finding supports for a sub-goal. Three robots must be loaded. Each robot has a capacity of one box. Mutexes are depicted as connections between actions.

Goal Resolution Constraint Model

A constraint satisfaction problem (CSP) is a triple *(X,D,C)* (Dechter, 2003), where *X* is a finite set of variables, *D* is a finite domain of values for the variables from *X* and *C* is a finite set of constraints over the variables from *X*. The constraint is an arbitrary relation over the elements of the domains of its variables. Having a constraint satisfaction problem the task is to find an assignment of values from *D* to all the variables from *X* such that all the constraints from *C* are satisfied. The problem of finding a solution of the constraint satisfaction problem is *NP*-complete in general.

We designed a simple constraint model for finding supports for goals arising during the search by the GraphPlan algorithm (let us call these goals sub-goals to distinguish them from the major goal). This formulation of the sub-goal sub-problem allows us to use constraint programming techniques to improve the solving process. Namely we are using *arc-consistency* (Mackworth, 1977) for pruning the search space during the search for supporting actions. The constraint model is built whenever a sub-goal arises in some layer of the planning graph. Suppose that the sub-goal *g* appeared in the *i*th level of the planning graph. We use two types of variables to model the problem of finding supports.

Activity variables: A Boolean variable *active(a)* is included into the model for every action *a* from the *i*th action layer of the planning graph which supports some proposition in the sub-goal *g*.

Support variables: A variable *support(p)* is included into the model for every proposition $p \in g$. The domain of the variable *support(p)* are all the actions from the *i*th action layer of the planning graph which support proposition *p* (i.e. the action in the domain of *support(p)* have *p* as one of its effects).

Constraints in the model are accumulated in two clusters. The first cluster is formed by constraints between Boolean activity variables and the second cluster is represented by constraints between support variables. There is one special channeling constraint between these two clusters.

Activity mutex constraint: A binary constraint forbidding assignment of value *true* to the pair of Boolean activity variables *active(a)* and *active(b)* (that is *active(a)=true & active(b)=true* is forbidden) is included into the model if and only if actions *a* and *b* are mutex in the *i*th layer of the planning graph.

Support mutex constraint: A binary constraint between variables *support(p)* and *support(q)* is refined by adding a new forbidden assignment *support(p)=a & support(q)=b* if and only if actions s *a* and *b* are mutex in the *i*th layer of the planning graph.

Having this model the sub-goal resolution process on lines 7 to 14 of the function *ExtractPlanFromLayer* of the algorithm 1 can be replaced by solving of the proposed constraint model. Labeling is done by selecting a proposition with fewest supports from the current sub-goal (standard type of a simple variable ordering heuristic) and by

selecting a support for this proposition. The support selection for the proposition is done over the support variables.

The propagation in the model is ensured by several ways. Whenever the algorithm gets to know that an action must be performed to provide the sub-goal with supports, the sub-goal is refined by deleting all the propositions which are effects of the action. This situation corresponds to activity variable with singleton set $\{true\}$ as its current domain or to the support variable with the singleton set $\{a\}$ as its current domain. The latter case means that a is the only supporting action for some proposition. The model is also refined in this case. All the propositions satisfied by the selected action are removed from the model (i.e. corresponding support variables are removed from the model and constraint graph is appropriately modified).

The most important propagation is done through the special channeling constraint which connects the cluster of activity variables and the cluster of support variables. We proposed three variants of propagation through both clusters. The method of propagation through the channeling constraint strongly relate to the way how consistency is enforced in the model. We maintain arc-consistency along the whole solving process. Every time when the labeling step is performed the consistency is enforced in the model (or more precisely, consistency is enforced in a selected part of the model). As we mentioned, arc-consistency is used in the model. Let us recall the definition first.

Definition 4 (Arc-consistency). *The value d of the variable x is arc-consistent if and only if for every variable y connected to x by the constraint c there exists a value e in the domain of y such that the assignment x = d & y = e is allowed by the constraint c. The constraint satisfaction problem (X, C, D) is arc consistent if and only if every value of every variable is arc-consistent.*

Propagation of variant A: When a supporting action is selected to satisfy a proposition in the sub-goal, the corresponding activity variable is set to be *true*. Then consistency is enforced in the cluster of activity variables. The next step consists of propagation of the changes in the cluster of activity variables into the cluster of support variables through the channeling constraint. The channeling constraint is defined as follows in this variant. If an activity variable is definitely *false*, then the corresponding action is removed from current domains of all the supporting variables. If an activity variable is definitely *true*, then the current sub-goal is updated and corresponding support variables are removed from the model. This variant is equivalent functionally to forward checking (Dechter, 2003) in the cluster of supporting variables (however this version is faster thanks to simplicity of the activity cluster).

Propagation of variant B: We proceed similarly as in the variant A. When a supporting action is selected to satisfy some proposition the corresponding activity variable is set to be *true*. Then consistency is enforced in the cluster of activity variables and changes are propagated into the cluster of support variables. This propagation is done in the same way as in the variant A. In addition to the variant A, changes in the cluster of support variables are propagated back to the cluster of activity variables. It is done in the following way. When a support variable has a singleton set as its current domain (the proposition has the only support) the corresponding activity variable is set to be *true* and consistency is enforced again in the cluster of activity variables. The process is repeated until changes are made.

Propagation of variant C: This variant further evolves the previous variant. Now consistency is enforced in both clusters. After selecting the action to support the given proposition a corresponding activity variable is set to be *true* and consistency is enforced in the cluster of activity variables. Then changes are propagated into the cluster of support variables where same type of consistency is enforced too. The last step of the iteration consists of propagation of changes from the cluster of support variables into the cluster of activity variables. Propagation in both direction between variable clusters through channeling constraint is done in the same way as in previous variants. The whole process is again repeated until the model in changing.

It is expectable that the constraint model with maintained consistency would provide better search space pruning than the approach used within the standard GraphPlan. The question is which variant performs best and what type of consistency is better. Experiments showed that variant C is the best choice on problems with higher number of interacting objects and with high action parallelism. However on problems with law object interaction the simple variant A is the best which is expectable on the whole. Let us note the cluster of action variables provides faster constraint propagation compared to the cluster of support variables since it is structurally simpler.

Experimental Results

We made several experiments with simple planning domains. All the planning problems which were used for experiments are available at the web site: http://ktiml.mff.cuni.cz/~surynek/research/flairs2007/.

Dock Worker Robots planning domain. This planning domain consists of a traffic network, transportation robots and of cranes (Ghallab et al., 2004). Each transportation robot has a certain capacity of packages and can move within the traffic network. There are two types of places within the traffic network called locations and sites. A location is an ordinary place which represents a node in the traffic network. A site is a special place where packages can be loaded and unloaded to and from the transportation robot. Each site has certain number of cranes and certain number of piles of packages (packages in pile behave like a stack - LIFO). Each crane can load and unload a package to and from a transporter. Typically, not all piles within a site are reachable by a single crane so the cooperation among cranes on the site is necessary.

The task within this planning domain is usually to transport some packages from one site to another site and to put them on piles in the right order.

Refueling Planes planning domain. Consider that we need to plan how to refuel planes in order to get to far destinations. For simplicity we have several airports in the line and several planes with certain fuel capacities. Planes can travel between the airports. A plane consumes certain amount of fuel to travel a unit of distance. Some extra fuel is also necessary for landing and taking-off. Each airport has an unlimited source of fuel and planes can refuel at the airport. The important ability of planes is to transfer fuel from one plane to another plane in-flight.

The task is typically to get a fleet of planes from one airport to some distant one. The task is especially interesting when planes need an intermediate landing on some middle airport or in-flight refueling.

Towers of Hanoi planning domain. This planning domain is a generalization of the well known puzzle. The original game consists of three pegs and a number of discs of different sizes stacked on pegs. It is possible to move a disc on the top of one peg to another peg in each turn. The condition that a smaller disc is always on larger disc must be preserved throughout the game. Our generalization is that we use arbitrary number of pegs and more than one disc can be moved in each turn. We can pick for example two discs and then place them in a different order than they were picked. The original game starts with all disc stacked on the first peg. However we allow arbitrary configuration (satisfying the condition on disc sizes) as the starting point in our generalization. Originally, the objective is to move all discs to the last third peg. Again we allow arbitrary valid configuration as a goal.

Table 1: Plan lengths and experimental results for Dock Worker Robots domain (planning graph length / plan length)

Problem	*dwr_01*	*dwr_05*	*dwr_07*	*dwr_11*	*dwr_16*
Plan length	6/9	14/24	16/36	8/32	18/34

Problem	***Standard***	***AC (A)***	***AC (B)***	***AC (C)***
Backtracks				
dwr_01	18238	575	281	110
dwr_05	590245	7180	5840	549
dwr_07	N/A	145976	109708	4322
dwr_11	236	224	50	25
dwr_16	N/A	442053	143138	23667
Actions				
dwr_01	1717	446	394	266
dwr_05	61695	3157	3151	943
dwr_07	N/A	73997	73659	7997
dwr_11	224	224	224	224
dwr_16	N/A	232797	232797	38694
Mutex checks				
dwr_01	133324	20015	19774	38134
dwr_05	5879590	733457	733457	939520
dwr_07	N/A	12824724	12824376	4263763
dwr_11	3166	11345	11345	29389
dwr_16	N/A	43009600	43009574	11857488
Time (planning graph building/plan extraction)[seconds]				
dwr_01	7.4/5.7	7.1/0.9	7.4/0.9	7.4/1.5
dwr_05	86.5/252.5	87.0/38.4	87.3/38.5	88.3/35.4
dwr_07	> 2 hours	145/726	146/689	143/157
dwr_11	24.5/0.8	25.3/0.5	24.7/0.5	24.7/0.9
dwr_16	> 2 hours	196.9/4701	192.5/4888	207.8/714

For our experiments we used our own implementation of the described techniques in C++ language. The tests were performed on a machine with AMD AthlonXP-M 3000+ (1600MHz) and 512 MB of memory running Mandrake Linux 10.0. The implementation was compiled with gcc compiler version 3.3.2 with maximum optimization for the target machine (-O9 -mtune=athlon). Our results are shown in tables 1-3. For each of our three resting domains we selected five problems of various difficulties. For each problem we counted number of backtracks of the algorithm, number of actions which were tried to be part of the resulting plan, number of mutex checks and the overall time. Since our current implementation consumes (relatively) lot of time by construction of the planning graph we measured planning graph construction time and plan extraction time separately.

Table 2: Plan lengths and experimental results for Refueling Planes domain (planning graph length / plan length)

Problem	*pln_01*	*pln_04*	*pln_05*	*pln_06*	*pln_10*
Plan length	5/9	5/9	6/14	9/14	10/15

Problem	***Standard***	***AC (A)***	***AC (B)***	***AC (C)***
Backtracks				
pln_01	777	52	52	42
pln_04	565	59	41	31
pln_05	564657	5577	5048	1883
pln_06	7958435	36165	32631	8960
pln_10	1688906	21209	15522	9310
Actions				
pln_01	125	62	62	56
pln_04	91	84	56	44
pln_05	51585	6059	4882	3220
pln_06	192217	28550	24911	23874
pln_10	83136	14195	14195	10613
Mutex checks				
pln_01	1669	2646	1646	1983
pln_04	1163	1047	1047	1180
pln_05	1757954	271254	271254	229287
pln_06	17162441	2399500	2399490	3198055
pln_10	3439254	1151049	1151049	1084931
Time (planning graph building/plan extraction)[seconds]				
pln_01	20.4/0.1	20.7/0.1	20.5/0.1	20.3/0.1
pln_04	5.4/0.0	5.3/0.0	5.3/0.0	5.5/0.0
pln_05	69.2/57.1	69.9/10.8	69.2/10.9	70.0/6.8
pln_06	80.3/460.6	80.8/91.8	80.7/94.5	80.0/86.0
pln_10	6.4/165.8	6.0/48.2	6.2/50.6	6.4/36.9

Our experiments showed that maintaining arc-consistency bring significant improvement in number of backtracks, number of considered actions in comparison with standard GraphPlan. Maintaining arc-consistency brings also significant improvement in overall time and mutex checks. Experiments also showed that the dual view of the problem is useful. Constraint propagation in both clusters of variables gives different results. If these results are combined together by a channeling constraint the obtained information is stronger than the information from individual cluster itself. Namely the variant C of propagation between clusters is most successful on hardest problems from our set of planning problems. However the variant C of propagation is not always the best choice. For example on problems with Hanoi towers sometimes the variant C has higher number of mutex checks than the standard version of the GraphPlan algorithm. The reason is that these problems do not use parallel actions and therefore the sub-goal sub-problems are almost trivial. Maintaining arc-consistency according to the variant C to solve easy sub-problems represents a not very useful overhead in such case compared to the simple variant A.

Table 3: Plan lengths and experimental results for Towers of Hanoi domain (planning graph length / plan length)

Problem	*han_01*	*han_02*	*han_03*	*han_04*	*han_07*
Plan length	6/6	14/14	30/30	10/12	14/20

Problem	***Standard***	***AC (A)***	***AC (B)***	***AC (C)***
Backtracks				
han_01	79	37	20	11
han_02	4496	848	492	298
han_03	98308	13161	7983	4116
han_04	48558	2128	1530	660
han_07	896283	24996	16783	8683
Actions				
han_01	40	87	62	48
han_02	1189	2471	1781	1665
han_03	20816	42491	29182	27799
han_04	5551	5697	4239	3970
han_07	86094	19095	19055	8683
Mutex checks				
han_01	162	255	255	491
han_02	9566	9905	9899	20461
han_03	249137	213507	213474	437933
han_04	121174	55660	55662	97213
han_07	2540413	841018	841018	1421706
Time (planning graph building/plan extraction)[seconds]				
han_01	0.1/0.0	0.1/0.0	0.1/0.0	0.1/0.0
han_02	1.2/0.4	1.2/0.4	1.2/0.4	1.2/0.6
han_03	7.5/10.5	7.4/8.0	7.5/8.1	7.5/12.2
han_04	5.9/5.0	6.0/2.5	6.0/2.6	6.0/3.1
han_07	14.0/142.2	13.5/44.71	13.3/47.8	13.6/54.5

Conclusions and Future Work

To summarize our contribution, we proposed a constraint model for solving sub-goal resolution sub-problem which arises in the GraphPlan-style solving process of planning problems. We experimented with maintaining of arc-consistency in the model. The modified GraphPlan algorithm enhanced with the proposed model and arc-consistency is better in terms of number of backtracks as well as in terms of overall time.

Finally let us note that our experimental planner is not competitive at the current stage to the state-of-the-art planners from well known IPC competition (Gerevini et al., 2006). It is partially due to not well optimized implementation and partially due to the fact that we do not use any domain specific heuristics. However our work should be regarded as a step towards competitive planner.

For future work we plan to design a technique which would allow some type of a global reasoning over the studied problem (arc-consistency is local technique). The also interesting question is how to reduce the size of planning graphs (for example reducing the high numbers of no operation actions may be interesting). We are also interested in an extension of the planning algorithm which would handle time and resources in numerical form. In such environment consistency techniques seems to be promising.

Acknowledgement

This work is supported by the Czech Science Foundation under the contract no. 201/07/0205 and by the Doctoral Project of Charles University under the contract no. 201/05/H014. We would like to thank anonymous reviewers for their useful comments and corrections.

References

Allen, J. et al. (editors), 1990. *Readings in Planning.* Morgan Kaufmann.

Baioletti, M. et al., 1998. *An Extension of SATPLAN for Planning with Constraints.* In Proceedings of the 8th International Conference AIMSA (AIMSA-98), 39-49, LNCS 1480, Springer-Verlag.

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

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

Gerevini, A. et al. (editors), 2006. Fifth International Planning Competition. Event in the context of ICAPS 2006 conference, http://ipc5.ing.unibs.it, University of Brescia, Italy, June 2006.

Ghallab, M. et al., 2004. *Automated Planning: theory and practice.* Morgan Kaufmann.

Kambhampati, S., 2000. *Planning Graph as a (Dynamic) CSP: Exploiting EBL, DDB and other CSP Search Techniques in GraphPlan.* Journal of Artificial Intelligence Research 12 (JAIR 12), 1-34, AAAI Press.

Kambhampati, S. et al., 1997. *Understanding and Extending GraphPlan.* In Proceedings of the 4th European Conference on Planning (ECP-97), 260-272, LNCS 1348, Springer-Verlag.

Kautz, H. A., et al., 1996. *Encoding Plans in Propositional Logic.* In Proceedings of the 5th Conference on Principles of Knowledge Representation and Reasoning (KR-96), 374-384, Morgan Kaufmann.

Kautz, H. A. and Selman, B., 1999. *Unifying SAT-based and Graph-based Planning.* In Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99), 318-325, Morgan Kaufmann.

Lopez, A. and Bacchus, F., 2003. *Generalizing GraphPlan by Formulating Planning as a CSP.* In Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-2003), 954-960, Morgan Kaufmann.

Mackworth, A. K., 1977. *Consistency in Networks of Relations.* Artificial Intelligence 8, 99-118.

Nareyek, A. et al., 2005. *Constraints and AI Planning.* IEEE Intelligent Systems 20(2), 62-72.

Nau, D. S. et al., 1995. *AI Planning versus Manufacturing Operation Planning: A Case Study.* In Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95), 1670-1676, Morgan Kaufmann.

Van Beek, P. and Chen, X., 1999. *A Constraint Programming Approach to Planning.* In Proceedings of the 16th National Conference on Artificial Intelligence (AAAI-99), 585-590, AAAI Press.