Constraint Based Reasoning over Mutex Relations in Planning Graphs during Search

Pavel Surynek

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

Abstract. We deal with the search process of the GraphPlan algorithm in this paper. We concentrate on the 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 or singleton 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.

1 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 [20]. The research in planning is also motivated by needs of researches in other areas. One of the most spectacular examples of the use of planning in science is the space exploration by autonomous spacecrafts [5] and vehicles [1]. However there is a great deal of other situations both in science and real-life where autonomous devices are used. The autonomous behavior is controlled by planning techniques and algorithms in many of these cases.

From the traditional view of planning, the planning problem is posed as finding of a sequence of actions which transform a specified initial state of the planning world into a desired goal state of the world [2,10]. 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 [7]. 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 domain 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. We study the problem of how to make the planning graph stronger for answering question of state reachability. We exploit *consistency techniques* [4, 18] known from constraint programming to enforce additional consistencies in planning graph during the search. This method allows us to deduce a required answer from planning graph more quickly than it is allowed by standard approach.

Specifically, we use consistency techniques for solving sub-problem of finding supports for a *sub-goal*. This kind of 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 constraint model for solving the sub-goal sub-problem. The sub-goal sub-problem is then solved as a *constraint satisfaction problem* [8] which significantly improves the search since constraint programming techniques can be used. We maintain a certain type of consistency during the search over the sub-goal model to obtain further speedup.

The paper is organized as follows. First we put our work into relation with other works. Then we introduce some basic definitions and facts from constraint programming and planning methodology. Next we describe planning graphs and GraphPlan algorithm. The main part of the paper is about our enrichment of GraphPlan solving process with sub-goal model and its consistencies. 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 part.

2 Related works

Several techniques for solving planning problems are trying to directly translate the 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 [13,15,16]. 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 [21].

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 [3,9,11,12,14,17]. Constraint programming represents a technique which is intensively used in this way [19]. This kind utilization of constraint programming in planning is more typical than the direct translation of the problem from one formalism into another. In this paper we will use constraint programming just in this way.

More specifically 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 [9, 11].

3 CSP and Planning Basics

A constraint satisfaction problem (CSP) is a triple *(X,D,C)* [8], 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-hard in general [8].

In the description of a *planning problem* we are using the similar notations and definitions as it is used in [10]. Consider that we have a first-order language L with finitely many predicate and con-

stant symbols. There are no function symbols or variable symbols in the language *L*. Thus all constructs built over the language *L* are ground. We will use following notations. Let l be a set of literals, then l^+ denotes a set of all atoms occurring in positive literals of the set l and l^- denotes a set of all atoms occurring in negative literals of the set l. Notions by which the planning problem is described, i.e planning domain, its states, goals and actions changing the planning domain are formalized using constructs built over the language *L*. A *planning domain* is an abstraction of real environment where planning task take place. For example, if we want create plans for manufacturing operations in a factory [20] our planning domain is that factory or only the single manufacturing-line where operations are carried out. Real environment around the factory or around the manufacturing-line is irrelevant with respect to the planning task and it is possible to omit that in the formalization. Changes of the planning domain are captured by the notion of a *state*. For example, the planning domain of the manufacturing-line is changed if a product is moved by a robot from one machine to another machine. The state before the move operation is different from the state after the operation. Transitions of this type are formalized by following definitions.

Definition 1 (State). *A state s is a finite set of atoms. Semantically it is a set of propositions that are true in a certain planning domain.*

Definition 2 (Action). *An action a is a pair (precond(a), effects(a)), where precond(a) is a finite set of atoms and effects(a) is a finite set of literals for which a condition* $effects^+(a) \cap effects^-(a) = \emptyset$ *holds. Semantically the action determines how the state can be changed (the change is specified by effects(a)) provided that the state allow the use of the action (the allowance is specified by precond(a)).*

Definition 3 (Applicability). *An action a is applicable to the state s if and only if* $precond(a) \subseteq s$. *The result of the application of the action a to the state s, where a is applicable to s, is a new state* $\gamma(s,a)$, *where* $\gamma(s,a) = (s - effects^-(a)) \cup effects^+(a)$.

Example 1. A planning domain of the manufacturing line contains three machines: a drilling-machine, a turning-machine and a packing-machine. There is also a product which is processed by these machines and a transportation robot which moves the processed product between machines. A set of actions in the planning domain of the manufacturing line consists of actions representing processing of the product on a machine and transportation of the product between machines by the robot.

Actions: *process(product, drilling-machine) = ({in(product, drilling-machine); unprocessed(product, drilling-machine)}; {processed(product,drilling-machine); ¬unprocessed(product, drilling-machine)})*
load(robot, product, drilling-machine) = ({in(product, drilling-machine); at(robot, drilling-machine)}, {loaded(robot, product); ¬in(product, drilling-machine)}), ...

The action *process(product, drilling-machine)* is applicable to the state *{in(product, drilling-machine); unprocessed(product, drilling-machine)}*, but not applicable to the state *{in(product, drilling-machine); processed(product, drilling-machine)}*.

A no-operation action (*noop*) is also considered to be a valid action. It does not change anything when it is applied on the state. The no-operation *noop(p)* action is associated with every possible

atom p. Formally we have an action $noop(p) = (\{p\}, \{p\})$ for every atom p. Normally, this type of action is useless, but is important for the Graphplan algorithm.

Definition 4 (Goal). *A goal g is a finite set of literals. The goal g is satisfied in a state s if and only if $g^+ \subseteq s$ and $g^- \cap s = \emptyset$. Semantically the goal is a set of propositions we want to be true in a certain state.*

Given a set of actions and a goal the task is to find out how to reach a state satisfying the given goal by using the allowed actions only. The whole process of finding of how to satisfy the goal starts in a specified initial state of the planning domain. This notion is described in the following definitions more formally.

Definition 5 (Problem). *A planning problem P is a triple (s_0, g, A), where s_0 is an initial state, g is a goal and A is a finite set of allowed actions. Semantically the initial state describes the planning domain state at the beginning and g represents a condition which a state we want to reach must satisfy. The required goal can be satisfied by using the allowed actions from the set A only.*

Definition 6 (Solution). *We inductively define the application of a sequence of actions $\theta = (a_1, a_2, ..., a_n)$ to a state s_0 in the following way: a_1 must be applicable to the state s_0, let us denote $s_i = \gamma(s_{i-1}, a_i)$, then a_i must be applicable to s_i for all $i = 2, ..., n$. The result of the application of the sequence of actions θ to the state s_0 is the state s_n. We denote $s_n = \gamma(s_0, \theta)$. The sequence $sol = (a_1, a_2, ..., a_n)$ is a solution of the planning problem $P = (s_0, g, A)$ if and only if the sequence sol is applicable to the initial state s_0 and g is satisfied in the result of the application of the sequence sol and $a_i \in A$ for $i = 1, 2, ..., n$.*

Example 2. The planning task over the planning domain of manufacturing line consists of an initial state, where the product is unprocessed, of the goal, where product is processed by all the machines in the line and of a finite set of actions.

Initial state: *{processed(product, drilling-machine); processed(product, turning-machine); processed(product, packing-machine); loaded(robot, product); at(robot, drilling-machine)}*
Goal: *{processed(product, drilling-machine); processed(product, turning-machine); processed(product, packing-machine)}*
Actions: *{process(product, drilling-machine); process(product, turning-machine); process(packing-machine); move(robot, drilling-machine, turning-machine); load(robot, product, drilling-machine); unload(robot, product, drilling-machine); ...}*
Solution: *<unload(robot, product, drilling-machine); process(product, drilling-machine); load(robot, product, drilling-machine); move(robot, drilling-machine, turning-machine); unload(robot, product, turning-machine), process(product, ...>.*

---

4 GraphPlan Algorithm

The GraphPlan algorithm [7] relies on the idea of state reachability analysis. The standard formulation of the GraphPlan algorithm puts an additional restriction on goals. Negative literals are not al-

lowed in a goal. The goal has to be a finite set of atoms. Since the preconditions of actions are also a finite set of atoms (definition 2), the preconditions of actions are goals as well in the standard GraphPlan formulation.

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 extended again.

The planning graph for a planning problem $P = (s_0, g, A)$ is defined as follows. It consists of two alternating structures called *proposition layer* and *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 the additional condition. This additional condition requires that no two propositions of the action are *mutually excluded* (we briefly say that they are *mutex*). The mutual exclusion relation will be defined inductively using the following definitions.

Definition 7 (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. A set of actions π is independent if and only if every pair of actions {a, b} from π is independent.

Definition 8 (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 9 (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 A_i for which $q \in effects^+(b)$ and layer A_i does not contain any action c for which $\{p, q\} \subseteq effects^+(c)$.*

Example 3. The planning problem of the manufacturing contains actions that are dependent.

For example actions *process(product, drilling-machine)* and *load(robot, product, drilling-machine)* are dependent since $effects^-(load(robot, product, drilling\text{-}machine)) \cap (precond(process(product, drilling\text{-}machine)) \cup effects^+(process(product, drilling\text{-}machine))) = \{in(product, drilling\text{-}machine)\} \cap$ $\cap (\{in(product, drilling\text{-}machine); unprocessed(product, drilling\text{-}machine)\} \cup \{processed(product, drilling\text{-}machine)\}) = \{in(product, drilling\text{-}machine)\} \neq \emptyset$.

Action and proposition mutexes are represented in the planning graph as special links between nodes representing actions and propositions. We have just defined the *(k+1)*th action layer A_{k+1}.

The *(k+1)*th proposition layer contains all the propositions that appear as the effect of some action in the *(k+1)*th proposition layer.

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 $A_1, A_2, ..., A_i$.*

We omit the proof of the theorem since it is given in details in [7]. The theorem gives the necessary condition for the existence of a solution of the planning problem. In other words, a valid plan can be extracted from the planning graph only if atoms of g are contained in some proposition layer P_j and there is no mutex between any two atoms from g in the layer P_j. The GraphPlan algorithm utilizes the result of the proposition for reduction of the search space that is necessary to be explored during the search for a solution.

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 a 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.

We supposed that we know how long the planning graph should be in this explanation. This is not true for real implementation. In real implementation some top most procedure is iteratively trying to extract plan for increasing lengths of planning graph (if the attempt is unsuccessful the planning graph is prolonged).

The very weak point of this version of the GraphPlan algorithm is the search for a set of non-mutex supporting actions. From the constraint programming point of view the search for supporting actions is a satisfaction process over the network of mutex constrains. Such interpretation allows us to use constraint programming techniques to solve the sub-problem of finding a set of supports.

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    return (P.p)
6  else
7    select q ∈ g
8    supports ← {a| a∈pG/Actions[l] & q∈effects+(a)}
9    if supports = ∅ then return (<failure>)
10   for each s∈supports do
11     if CheckSupport(pG,s,p,l) then
12       g2 ← g-effects+(s)
13       p2 ← p∪{s}
14       return ExtractPlanFromLayer(pG,l,g2,p2)
15   return (<failure>)

function CheckSupport(pG,s,p,l):boolean
1  for each r∈p do
2    if (r,s) ∈ pG/AMutexes[l] then return (False)
3  return (True)
```

5 Planning with State Variables

It is possible to use another approach for representation of a planning domain which is more suitable with respect to constraint programming. Such approach is for example a so called *state variable representation*. Instead of saying that some proposition holds in a certain situation we say that a certain property takes a certain value in that situation. The planning domain in state variable representation consists of a finite set of *state variable functions* $f_1, f_2, ..., f_m$, where $f_i: S \rightarrow D(f_i)$ for $i \in \{1, 2, ..., m\}$. S denotes the set of possible states of the planning domain and $D(f_i)$ is the domain of the state variable function f_i. An individual state variable function represents a single property of the object residing in the planning domain (for example we can have a state variable function representing the location of a robot, the domain of such function would be all the possible locations where the robot can go). A state variable F_i is associated with every state variable function f_i for $i \in \{1, 2, ..., m\}$.

Definition 10 (State in state variable representation). *A state s in state variable representation is a finite set of assignments of the form $F_j = d_k$, where $d_k \in D(f_j)$.*

Definition 11 (Action in state variable representation). *An action a in state variable representation is a pair (precond(a), effects(a)), where precond(a) and effects(a) are finite set of assignments of the form $F_j = d_k$, where $d_k \in D(f_j)$.*

Definition 12 (Goal in state variable representation). *A goal g in state variable representation is a finite set of propositions of the form $F_j R\ d_k$, where $d_k \in D(f_j)$ and $R \in \{=, \neq\}$.*

Example 5. Consider the planning domain of the manufacturing line again. This example shows state variable representation of this planning domain. The planning domain is described by state variable functions. States, actions and goals are modified slightly modified in this representation.

State variables: *in(product)∈{drilling-machine, turning-machine, packing-machine, robot},*
at(robot)∈{drilling-machine, turning-machine, packing-machine},...

States: *{in(product)=robot; at(robot)=drilling-machine);*
processed(product, drilling-machine)=true},...

Actions: *process(product, drilling-machine) = ({in(product)=drilling-machine;*
processed(product, drilling-machine)=false}; {processed(product,
drilling-machine)=true})
load(robot, product, drilling-machine) = ({in(product)=drilling-machine;
at(robot)=drilling-machine}, {in(product)=robot}),...

Goal: *{processed(product, drilling-machine)=true; processed(product, turning-machine)=true;*
processed(product, packing-machine)=true}

We omit the translation of *applicability*, *planning problem* and *solution* into the state variable representation since it is almost obvious. The planning graphs can be easily defined over the state variable representation.

The relation of independence of the actions as well as the relation of mutual exclusion and its propagation can be defined within the state variable formulation of the planning problem. The im-

portant property of the state variable representation is that it allows strengthening the independence relation. Stronger relation of independence results into higher number of mutexes in the planning graph. More mutexes means more pruning of the search space.

Definition 13 (Independence in state variable representation). *A pair of actions {a, b} is independent in state variable representation if and only if:*

(i) *effects(a) and (precond(b) ∪ effects(b)) does not compete and*

(ii) *effects(b) and (precond(a) ∪ effects(a)) does not compete.*

Otherwise {a, b} is a pair of dependent actions in state variable representation. A set of actions π is independent in state variable representation if and only if every pair of actions {a, b} from π is independent. Two sets of propositions A and B compete if and only if $A \cup B$ contains propositions F=a and F=b, for some state variable F and $a \neq b$.

The stronger independence relation (it is less probable that two actions are independent) is due to the fact that we preserve more information in the formalism about the original planning domain. Our experiments confirmed this conjecture.

Notice that, as in the standard GraphPlan algorithm we restrict the definition of goals. A goal is a finite set of assignments (negative assignments are not allowed in the goal). Thus the goal is a state within our GraphPlan formulation as well.

6 Goal Resolution Constraint Model

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* [18] and s*ingleton arc-consistency* [4,6] 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)* (*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 line 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 (some kind of 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 actual domain or to the support variable with the singleton set $\{a\}$ as its actual 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 the way how consistency is enforced in the model. We maintain consistency (arc-consistency or singleton 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 and singleton arc-consistency is used in the model. Let us recall the definitions.

Definition 14 (Arc-consistency and singleton 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.*

The value d of the variable x is singleton arc-consistent if and only if the constraint satisfaction problem restricted to x = d is arc-consistent. The constraint satisfaction problem (X, C, D) is singleton arc-consistent if and only if every value of every variable is singleton 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. And the last 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 actual 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.
- **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 actual 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 not always the best choice.

The arc-consistency was chosen for its simplicity and performance. The motivation for choosing singleton arc-consistency for our framework is that it provides strong consistency. This type of consistency can discover forbidden situations for which the standard version of the GraphPlan requires search. The following observation formalizes this idea.

Observation 1 (Constraint model provides stronger propagation). *Singleton arc-consistency can discover that some action cannot be preformed within a certain planning graph layer and the same fact cannot be discovered by the standard Graphplan without search.*

Demonstration. Consider the following situation. Let us have state variable functions x,y and z and corresponding state variables $X \in \{x_1,...,x_{nx}\}$; $Y \in \{y_1,...,y_{ny}\}$ and $Z \in \{z_1,...,z_{nz}\}$. Next we have several actions. Since the particular preconditions of these actions are not interesting in this demonstration we omit them. The actions are $a_{11}=(_,X=x_1)$; $a_{12}=(_,Y=y_2)$; $a_{13}=(_,Z=z_3)$; $a_{21}=(_,X=x_1)$; $a_{22}=(_,Y=y_2)$; $a_{23}=(_,Z=z_3)$. Consider now that actions a_{11}, a_{12} and a_{13} are pair wise mutex. And the same holds for actions a_{21}, a_{22} and a_{23}. A part of the constraint model of the described situation is depicted in figure 1. Now consider an action b that requires the assignment $[X=x_1, Y=y_2, Z=z_3]$ as its precondition. The action b is supported by actions a_{11}, a_{12}, a_{13}, a_{21}, a_{22} and a_{23}. It it obvious that the version of the Graphplan algorithm shown here as algorithm 1 requires some search to discover that action b cannot be performed. By contrast the action b is not supported with respect to our model since the part of the constraint model represented by the variables $a(X,x_1)$; $a(Y,y_2)$ and $a(Z,z_3)$ is not singleton arc-consistent. Hence the action b cannot be performed. n

7 Experimental Results and Concluding Remarks

We made several experiments with simple planning domain[1]. The planning domain we have used consists of traffic network. Within this traffic network transporters of various capacities can move. There are ordinary places within the traffic network called locations and special places where packages can be loaded and unloaded. These special places are called sites. 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.

Table 1. Plan lengths (planning graph length / plan length)

Problem no.	*01*	*02*	*03*	*05*	*07*	*08*	*09*	*10*	*11*
Plan length	6/9	6/8	2/4	14/24	16/36	8/8	8/16	8/24	8/32

Table 2. Number of backtracks. The type of consistency is denoted by SAC (singleton arc-consistency) of by AC (arc-consistency). Variant of propagation is denoted by the letter in parenthesis.

Problem	*Standard*	*SAC(A)*	*SAC(B)*	*SAC(C)*	*AC(A)*	*AC(B)*	*AC(C)*
problem_01	18238	229	91	91	575	281	110
problem_02	2920	145	49	49	178	95	72
problem_03	32	32	3	3	32	5	3
problem_05	590245	1125	592	592	7180	5840	549
problem_07	N/A	N/A	N/A	N/A	145976	109708	4322
problem_08	251	248	53	53	248	76	55
problem_09	246	240	43	43	240	67	45
problem_10	241	232	33	33	232	58	35
problem_11	236	224	23	23	224	50	25

Table 3. Number of actions considered.

Problem	*Standard*	*SAC(A)*	*SAC(B)*	*SAC(C)*	*AC(A)*	*AC(B)*	*AC(C)*
problem_01	1717	229	229	229	446	394	266
problem_02	389	131	131	131	138	130	141
problem_03	32	32	32	32	32	32	32
problem_05	61695	1029	1029	1029	3157	3151	943
problem_07	N/A	N/A	N/A	N/A	73997	73659	7997
problem_08	248	248	248	248	248	248	248
problem_09	240	240	240	240	240	240	240
problem_10	232	232	232	232	232	232	232
problem_11	224	224	224	224	224	224	224

[1] http://ktiml.mff.cuni.cz/~surynek/research/csclp2006/

Table 4. Number of mutex checks.

Problem	*Standard*	*SAC(A)*	*SAC(B)*	*SAC(C)*	*AC(A)*	*AC(B)*	*AC(C)*
problem_01	133324	160633	128450	151014	20015	19774	38134
problem_02	23458	177562	153606	162683	9208	9208	31553
problem_03	240	9377	2785	3532	690	690	785
problem_05	5879590	4109180	3834881	3867564	733457	733457	939520
problem_07	N/A	N/A	N/A	N/A	12824724	12824376	4263763
problem_08	3745	417062	209018	239667	13759	13759	33938
problem_09	3537	376862	183012	205265	12657	12657	32227
problem_10	3344	340571	161501	179091	11788	11788	30865
problem_11	3166	307809	143791	158023	11345	11345	29389

Table 5. Time in seconds (planning graph building time/plan extraction time)

Problem	*Standard*	*SAC(A)*	*SAC(B)*	*SAC(C)*	*AC(A)*	*AC(B)*	*AC(C)*
problem_01	7.4/5.7	7.2/7.0	7.5/6.2	7.1/6.6	7.1/0.9	7.4/0.9	7.4/1.5
problem_02	7.3/1.1	7.5/8.4	7.3/7.7	7.5/8.1	7.6/0.4	7.5/0.4	7.2/1.1
problem_03	0.3/0.0	0.3/0.2	0.3/0.1	0.3/0.1	0.3/0.0	0.3/0.0	0.3/0.0
problem_05	86.5/ /252.5	86.5/ /241.0	86.8/ /234.3	87.3/ 233.7	87.0/ /38.4	87.3/ /38.5	88.3/ /35.4
problem_07	> 2 hours	> 2 hours	> 2 hours	> 2 hours	145.2/ /726.4	145.6/ /689.1	143.4/ /157.4
problem_08	24.5/0.9	25.2/15.1	24.8/10.0	24.8/10.8	25.3/0.5	25.1/0.5	25.1/1.1
problem_09	24.4/0.8	24.9/14.2	24.5/9.2	24.7/10.1	25.5/0.5	24.2/0.5	25.1/1.0
problem_10	24.7/0.8	25.0/13.7	24.5/8.9	24.8/9.6	25.1/0.5	25.4/0.5	24.9/1.0
problem_11	24.5/0.8	24.3/12.7	24.6/8.6	24.8/8.9	25.3/0.5	24.7/0.5	24.7/0.9

The experiments showed that maintaining arc-consistency is better choice than singleton arc-consistency. However the number of backtracks is significantly lower for singleton arc-consistency with variant B or C. This result showed that it worth doing further research on singleton arc-consistency in this context. Arc-consistency as well as singleton arc-consistency in these experiments was based on simple AC-3 algorithm. This may be an important handicap for performance of the algorithm using singleton arc-consistency. Both types of consistencies bring significant improvements in number of backtracks, number of considered actions in comparison with standard GraphPlan. Arc-consistency is better in overall time and in number of 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. We have also performed some tests with classical representation of our planning domain (according to definitions 1 to 6). Although we do not present exact result about this experimentation let us say that our experiments showed that standard version of the GraphPlan algorithm with classical representation was about twice slower than the same algorithm base on state variable representation.

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 and singleton 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. When singleton arc-consistency is used the results are not so good. It is caused mainly by the choice of inefficient algorithm for enforcing the consistency. We plan to evaluate better algorithms [4] in context of our framework. The interesting result about the model and singleton arc-consistency is that this type of consistency reduces the number of backtracks very much. Therefore it seems that the structure of the sub-goal resolution sub-problem does not require extensive search and can be solved by alternative type of reasoning (with little search).

References

[1] M. Ai-Chang, J. Bresina, L. Charest, A. Chase, J. Hsu, A. Jónson, B. Kanefsky, P. Morris, K. Rajan, J. Yglesias, B. Chafin, W. Dias and P. Maldague. *MAPGEN: Mixed-Initiative Planning and Scheduling for the Mars Exploration Rover Mission.* IEEE Intelligent Systems 19(1), 8-12, IEEE Press, 2004.

[2] J. Allen, J. Hendler and A. Tate (editors). *Readings in Planning.* Morgan Kaufmann Publishers, 1990.

[3] M. Baioletti, S. Marcugini and A. Milani: *An Extension of SATPLAN for Planning with Constraints.* In Proceedings of 8th International Conference AIMSA (AIMSA-98), 39-49, LNCS 1480, Springer-Verlag, 1998.

[4] R. Barták and R. Erben. *A New Algorithm for Singleton Arc Consistency.* In Proceedings of the 17th Florida Artificial Intelligence Research Society Conference (FLAIRS-2004), AAAI Press, 2004.

[5] D. Bernard, E. Gamble, N. Rouquette, B. Smith, Y. Tung, N. Muscettola, G. Dorias, B. Kanefsky, J. Kurien, W. Millar, P. Nayak and K. Rajan. *Remote Agent Experiment. Deep Space 1 echnology Validation Report.* NASA Ames and JPL report, 1998.

[6] C. Bessière and R. Debruyne. *Optimal and Suboptimal Singleton Arc Consistency Algorithms.* In Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-2005), 54-59, Professional Book Center, 2005.

[7] A. Blum and M. L. Furst. *Fast Planning through Planning Graph Analysis.* Artificial Intelligence 90(1-2), 281-300, AAAI Press, 1997.

[8] R. Dechter. *Constraint Processing.* Morgan Kaufmann Publishers, 2003.

[9] M. B. Do and S. Kambhampati. Sol*ving Planning-graph by Compiling it into CSP.* In Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (AIPS-2000), 82-91, AAAI Press, 2000.

[10] M. Ghallab, D. S. Nau and P. Traverso. *Automated Planning: theory and practice.* Morgan Kaufmann Publishers, 2004.

[11] S. Kambhampati. *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, 2000.

[12] S. Kambhampati, E. Parker and E. Lambrecht. *Understanding and Extending Graphplan.* In Proceedings of 4th European Conference on Planning (ECP-97), 260-272, LNCS 1348, Springer-Verlag, 1997.

[13] H. A. Kautz, D. A. McAllester and B. Selman. *Encoding Plans in Propositional Logic.* In Proceedings of the 5th Conference on Principles of Knowledge Representation and Reasoning (KR-96), 374-384, Morgan Kaufmann Publishers, 1996.

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

[15] H. A. Kautz and B. Selman. *Planning as Satisfiability.* In Proceeding of 10th European Conference on Artificial Intelligence (ECAI-92), 359-363, John Wiley and Sons, 1992.

[16] H. A. Kautz and B. Selman. *Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search.* In Proceedings of the 13th National Conference on Artificial Intelligence (AAAI-96), 1194-1201, AAAI Press, 1996.

[17] A. Lopez and F. Bacchus. *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 Publishers, 2003.

[18] A. K. Mackworth. *Consistency in Networks of Relations.* Artificial Intelligence 8, 99-118, AAAI Press, 1977.

[19] A. Nareyek, E. C. Freuder, R. Fourer, E. Giunchiglia, R. P. Goldman, H. A. Kautz, J. Rintanen and A. Tate. *Constraints and AI Planning.* IEEE Intelligent Systems 20(2), 62-72, IEEE Press, 2005.

[20] D. S. Nau , W. C. Regli and K. S. Gupta. *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 Publishers, 1995.

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