Projection Global Consistency: An Application in AI Planning

Pavel Surynek

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

Abstract. We are dealing with solving planning problems by the GraphPlan algorithm. We concentrate on solving a problem of finding supporting actions for a goal. This problem arises as a sub-problem many times during search for a solution. We showed in the paper that the supports problem is NP-complete. In order to improve the solving process of the supports problems we proposed a new global consistency technique which we call projection consistency. We present a polynomial algorithm for enforcing projection consistency. The projection consistency was implemented within our experimental planning system which we used for empirical evaluation. The empirical tests showed improvements in order of magnitudes compared to the standard GraphPlan (both in time and number of constraint checks). A significant improvement was also reached compared to the recent similar technique based on maintaining of arc-consistency.

Keywords: GraphPlan, Planning graphs, CSPs, Maintaining Consistency, Arc-consistency, Projection consistency, Constraint propagation

1 Introduction

In this paper, we would like to explain our new contribution to solving AI planning problems. We called our new concept a projection global consistency and it is designed to be used to help with pruning of the search space during solving planning problems over planning graphs.

Planning as a task of finding a sequence of actions resulting in achieving some goal is one of the most challenging problems of artificial intelligence [2]. It is necessary to solve planning problems almost every time when a complex autonomous behavior of a certain agent is required. It is the case of spacecrafts and vehicles for distant space and planetary exploration [1,3] as well as the case of unmanned military devices [5]. There are many successful approaches how to solve planning problems. One of them is usage of so called *planning graphs* on which we concentrate in this paper. The concept of planning graphs introduced by Blum and Furst [4] brought a substantial break-through in solving of planning problems. Many of the consequent achievements in planning are based on ideas of planning graphs. In this paper we are studying planning graph from the perspective of constraint programming [6]. We analyzed the original Blum's and Furst's GraphPlan algorithm [4] as well as other approaches based on that [10, 11, 15, 17]. Our conclusion was that there is a room for exploiting some type of a global reasoning which is in constraint programming known as *global constraints* [20]. However an evident antagonism is that global constraints often work over numeric domains (with defined operations and orderings) while planning problems in its basic form are rather of a symbolic character. We found that even in a domain such as planning problems are it is possible to develop some kind of global reasoning.

The paper is organized as follows. First we introduce basic definition of planning problems and also some definitions from constraint programming. Then we briefly explain planning graphs and related GraphPlan algorithm. The next section is devoted to a sub-problem which arises during the

search using GraphPlan algorithm. The main part of the paper describes the projection consistency and constraint which is designed to help to solve the mentioned sub-problem. Finally we present some empirical tests of the proposed concept and discuss our contribution in relation to other works.

2 AI Planning and Constraint Programming Essentials

For purposes of clarity we are using a simple language for expressing planning problems in this paper. A language is associated with a planning domain (in other words we consider different languages for different planning problems). To describe problem over a certain planning domain we use language L with finitely many predicate and constant symbols. The set of predicates will be denoted as P_L and the set of constants as C_L. Constants represent objects appearing in the planning world and predicate symbols are used to express relations over objects. Let us note that simplicity of the language is not at the expense of expressivity (in [8] Ghallab et al. show more approaches for describing planning problems). The following definitions assume a fixed language L.

Definition 1 (Atom, Literal). An *atomic formula* is a construct of the form $p(c_1,c_2,\ldots,c_n)$, where $p \in P_L$ and $c_i \in C_L$ for $i = 1,2,\ldots,n$. Atomic formulas are called atoms in short. A *literal* is an atom or its negation.

Definition 2 (State, Goal, Goal satisfaction). A *state* is a finite set of atoms. A *goal* is also a finite set of atoms. The goal g is *satisfied* in the state s if $g \subseteq s$.

States provide a formal description of a situation in the planning world (snapshot of the planning world at a moment). A goal is a formal description of a situation which we want to establish. The situation in the planning world is changed by actions. Actions formally define possible transitions between states. Action applied to the state results into a new state.

Definition 3 (Action, Applicability, Application). An *action* a is a triple $(p(a),e^+(a),e^-(a))$, where $p(a)$ is a *precondition* of the action, $e^+(a)$ is a *positive effect* of the action and $e^-(a)$ is a *negative effect* of the action. All the action tree components are finite sets of atoms. An action a is *applicable* to the state s if $p(a) \subseteq s$. The *result of the application* of the action a to the state s is a new state $\gamma(s,a)$, where $\gamma(s,a) = (s - e^-) \cup e^+$.

For purposes of planning graphs there are also assumed so called *no-op actions* which represent no operation. For every atom t we assume a no-op action $noop_t = (t,t,\varnothing)$. Briefly said a no-op action preserves an atom into the next state.

Given a set of allowed actions and a goal the objective is to transform a given initial state into a state satisfying the goal. State transitions to achieve the objective are carried out by applying actions from the set of allowed actions. We suppose that the number of preconditions, the number of positive and the number of negative effects are bounded by a constant. In other words we do not allow actions of arbitrary size.

Definition 4 (Planning 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.

Definition 5 (Application of sequence of actions, Solution). We inductively define *application of a sequence of actions* $\phi = (a_1, a_2, \ldots, a_n)$ to a state s_0 in the following way: a_1 must be applicable to s_0, let us inductively denote the result of application of the action a_i to the state s_{i-1} as s_i for all $i = 1,2,\ldots,n$; the condition that a_i is applicable to the state s_{i-1} for all $i = 2,3,\ldots,n$ must hold. The *result of application of the sequence of actions* ϕ to the state s_0 is the state s_n. Sequence $\xi = (a_1, a_2, \ldots, a_n)$ is a *solution* of the planning problem $P = (s_0, g, A)$ if the sequence ξ is applicable to the initial state s_0 and the goal g is satisfied in the result of application of the sequence ξ and $a_i \in A$ for all $i = 1,2,\ldots,n$.

A method developed in this paper regards some problems from the constraint programming perspective. Although the method is described in self containing style some basic definitions from constraint programming are necessary. The key concept is a *constraint satisfaction problem*.

Definition 6 (Constraint satisfaction problem) [6]. A *constraint satisfaction problem* (CSP) is a triple (X, D, C), where X is a finite set of variables, D is a finite domain of values for variables from the set X and C is a finite set of constraints. A *constraint* is an arbitrary relation over the domains of its variables. A set of variables constrained by a constraint c is denoted as X_c.

Definition 7 (Solution of CSP) [6]. A *solution* of a constraint satisfaction problem (X, D, C) is an assignment of values to the variables $\psi : X \to D$ such that all the constrains are satisfied for ψ, that is $(\forall c \in C)[x_1, x_2, \ldots, x_k] = X_c \Rightarrow [\psi(x_1), \psi(x_2), \ldots, \psi(x_k)] \in C$.

CSPs are often solved using so called *constraint propagation*. If a domain of a variable is changed then this change of information is propagated into the domains of other variables through constraints. Often the quality of the solving algorithm is tightly connected with the quality of propagation algorithms for individual constraints. It is especially true for so called global constraints which bind large number of variables and perform fine grained and effective propagation throughout the whole problem (for example Regin's *allDifferent* constraint [20]). Our method is trying to follow the concept of such global propagation.

3 Planning Graphs and GraphPlan Algorithm

The *GraphPlan* algorithm is due to Blum and Furst [4]. It relies on the idea of state reachability analysis. The state reachability analysis is done by constructing a special data structure called *planning graph*. The algorithm itself works in two interleaved phases. In the first phase planning graph is incrementally expanded. Then in the second phase an attempt to extract a valid plan from the ex-

tended planning graph is performed. If the second phase is unsuccessful the process continues with the first phase, that is the planning graph is extended again.

Algorithm 1: Plan extraction - a part of the GraphPlan algorithm

```
function extractPlan(pG, l, g) : sequence
1:  if l = 0 then
2:  |  if g ∈ pG / Propositions[0] then return []
3:  |  else return [failure]
4:  if g is subsumed in pG / Nogoods[l] then return [failure]
5:  ξ ← extractPlanFromLayer(pG, l, g, ∅)
6:  if ξ = [failure] then
7:  |  pG / Nogoods[l] ← pG / Nogoods[l] ∪ {g}
8:  |  return [failure]
9:  else return ξ

function extractPlanFromLayer(pG, l, g, ζ) : sequence
10: if g = ∅ then
11: |  g' ← {p(a) | a ∈ ξ}
12: |  Ξ ← extractPlan(pG, l − 1, g')
13: |  if Ξ = [failure] then return [failure]
14: |  else return concatenate(Ξ, ξ)
15: else
17: |  select any t ∈ g
18: |  s_t ← {a | a ∈ pG / Actions[l] & t ∈ e+(a)}
19: |  if s_t = ∅ then return [failure]
20  |  for each a ∈ s_t do
21: |  |  if checkSupports(pG, l, ξ, a) then
22: |  |  |  g' ← g − e+(a)
23: |  |  |  ξ' ← ξ ∪ {a}
24: |  |  |  return extractPlanFromLayer(pG, l, g', ξ')
25: return [failure]

function checkSupports(pG, l, a, ζ) : boolean
26: for each b ∈ ξ do
27: |  if (a, b) ∈ pG / Mutexes[l] then return False
28: return True
```

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 plan-

ning graph is defined inductively. Consider that the planning graph with layers P_0, A_1, P_1, A_2, P_2, …, 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 that no two propositions of the action are *mutually excluded* (we briefly say that they are *mutex*).

Definition 8 (Independence of actions). A pair of actions $\{a,b\}$ is *independent* if $e^-(a) \cap (p(b) \cup e^+(b)) = \varnothing$ and $e^-(b) \cap (p(a) \cup e^+(a)) = \varnothing$. Otherwise $\{a,b\}$ is a pair of *dependent* actions.

Definition 9 (Action mutex and mutex propagation). We call a pair of actions $\{a,b\}$ within the action layer A_i a *mutex* if either the pair $\{a,b\}$ is dependent or an atom of the precondition of the action a is mutex with an atom of the precondition of the action b (defined in the following definition).

Definition 10 (Proposition mutex and mutex propagation). We call a pair of atoms $\{p,q\}$ within the proposition layer P_i a *mutex* if every action a within the layer A_i where $p \in e^+(a)$ is mutex with every action b within the action layer A_i for which $q \in e^+(b)$ and the action layer A_i does not contain any action c for which $\{p,q\} \subseteq e^+(c)$.

4 Problem of Finding Supports for a Sub-goal

A problem of finding supports for a sub-goal is definable for arbitrary action layer of the planning graph and for arbitrary goal. Consider an action layer of a given planning graph. Let A be a set of actions of the action layer and let μA be a set of mutexes between actions from A. Next let us have a goal g. For the given goal g and action layer the question is to determine a set of actions $\zeta \subseteq A$ where no two actions from ζ are mutex with respect to μA and ζ satisfies the goal g. The set of actions ζ satisfies the goal g if $g \subseteq \bigcup_{a \in \zeta} e^+(a)$ (notice that positive and negative effects of actions from the set of non-mutex actions does not interfere). The actions from the set ζ are called *supports* for the goal g in this context. The goal g is called a *sub-goal* to distinguish it from the global goal for which we are building a plan. Typically many sub-goals must be satisfied along the search for the global goal in the standard GraphPlan algorithm. The problem of finding supports for a sub-goal will be called a supports problem in short.

The effectiveness of a method for solving supports problem has a major impact on the performance of the planning algorithm as a whole. Unfortunately the supports problem is NP-complete. This claim can be easily proved by using reduction of Boolean formula satisfaction problem (SAT) to supports problem.

Theorem 1 (Complexity of supports problem). *The problem of finding supports for a sub-goal is NP-complete.*

Proof: The supports problem is obviously in *NP*. It is sufficient treat sets as lists to prove the claim. Having a set of actions ζ it is possible to check whether it is a solution of the supports problem for a goal g in $O(|\zeta|(|A|+|\mu A|+|g|))$ steps. First we need check if all the actions from ζ are also from A. It takes $|\zeta||A|$ steps to check if $\zeta \subseteq A$ holds. Next we check if no two actions from ζ are mutex. A mutex $\{a_i, a_j\}$ where $a_i, a_j \in A$ can be checked against ζ in $2|\zeta|$ steps. For all mutexes this can be done in $2|\zeta||\mu A|$ steps. Computing of the set $\bigcup_{a \in \zeta} e^+(a)$ takes $d|\zeta|$ steps where d is the action size bounding constant (that is $(\forall a \in A) \quad d \geq \max(|p(a)|, |e^+(a)|, |e^-(a)|)$). Checking whether $g \subseteq \bigcup_{a \in \zeta} e^+(a)$ takes $d|\zeta||g|$ steps. The total number of steps for verifying the solution is $|\zeta||A| + 2|\zeta||\mu A| + d|\zeta||g|$ which is $O(|\zeta|(|A|+|\mu A|+|g|))$. The resulting expression is polynomial in size of the input.

Completeness with respect to *NP* class can be proved by using polynomial reduction of Boolean formula satisfaction problem (*SAT*) to supports problem. Consider a Boolean formula B. It is possible to assume that the formula B is in the form of conjunction of disjunctions, that is $B = \wedge_{i=1}^{n} \vee_{j=1}^{m_i} x_j^i$, where x_j^i is a variable or a negation of a variable. For each clause $\vee_{j=1}^{m_i} x_j^i$ where $i = 1,2,\ldots,n$ we introduce a literal l_i into the constructed goal g. Next we introduce an action $a_j^i = (\varnothing, \{l_i\}, \varnothing)$ into the set of actions A for each x_j^i from the clause (the action has the only one positive effect and no preconditions and no negative effects). Actions are introduced in this way for all the clauses from B. If for some $i,k \in \{1,2,\ldots,n\}$; $j \in \{1,2,\ldots,m_i\}$; $l \in \{1,2,\ldots,m_k\}$ $x_j^i = \neg x_l^k$ or $\neg x_j^i = x_l^k$ holds we introduce a mutex $\{a_j^i, a_l^k\}$ into the set of mutexes μA. The constructed sets $A, \mu A$ and the goal g constitute the instance of the supports problem. The size of the resulting supports problem is $O(|B|^2)$, where $|B|$ is the number of literals appearing in B.

Having a set of actions ζ solving the constructed instance of the supports problem we can construct valuation f as follows $f(x_j^i) = true$ (that is: if $x_j^i = v$ for some variable v then $f(v) = true$, if $x_j^i = \neg v$ then $f(v) = false$) for each $a_j^i \in A$. The truth values for the remaining variables in B can be selected arbitrarily. Mutexes ensure that the valuation f is correctly defined function. Moreover we have $f(\vee_{j=1}^{m_i} x_j^i) = true$ for $i = 1,2,\ldots,n$. Thus every clause of B is positively valued. This is implied by the fact that whole the goal g is satisfied by ζ. The solution of the original Boolean formula satisfaction problem is obtained from ζ in $O(|B|)$ steps. n

5 Projection Constraint and Projection Consistency

We proposed a new global constraint to improve the search of the GraphPlan planning algorithm in the phase of plan extraction from the planning graph. We called the constraint projection constraint according to the way how propagation is done through that. We use the projection constraint to model and to improve the solving of the supports problem. The supports problem must be solved

many times along the search for a plan in plan extraction phase (as it is done by the standard GraphPlan algorithm). A part of the algorithm which solves the supports problem is responsible for majority of backtracks (this observation is evident from our experiments, see the section about experiments). That is why the efficiency of solving of this sub-problem has significant impact on the efficiency of the whole planning algorithm.

In [22] and [23] Surynek examined the effect of maintaining arc-consistency and singleton arc-consistency for solving the supports problem. He obtained substantive speedups using these types of reasoning compared to pure backtracking based method. However both arc-consistency and singleton arc-consistency provides only some type of a local reasoning over the problem. Whereas the term local means here that a small part of the problem is considered at once. This feature of the consistency technique may lead to high number of iterations of the given consistency enforcing algorithm till the consistency is established. Moreover if the cost of an iteration of such local technique is too high there remains only a little advantage against the pure backtracking (namely this is the case of maintaining singleton arc-consistency [23]).

By contrast the projection constraint introduces some type of a global reasoning over the supports problem. That is we consider the problem as a whole at once. The important requirement on the method for solving the supports problem within the plan extraction phase of the GraphPlan is its completeness. It means if there exists a solution of a given supports problem the method must guarantee finding of that solution. On the other hand if there is no solution of a given supports problem the method must prove this fact. Specifically it is necessary to enumerate solutions of the supports problem within plan extraction.

The projection constraint was motivated by the observation of mutex graphs of layers of the planning graph. These mutex graphs embody high density of edges on majority of testing planning problems (however our method works with sparse mutex graphs as well). The high density of edges is caused by various factors. Nevertheless we regard the set of actions that change states of an object or group of objects of the planning domain as the most important one. Actions from such set are pair wise mutually excluded since they change a single property (for example imagine a robot at coordinates $[3,2]$, the robot can move to coordinates in its neighborhood, so the actions are: $moveTo([2,2])$, $moveTo([2,3])$, $moveTo([3,3])$, ...). That is such set of actions induces a clique within a mutex graph.

The knowledge of clique decomposition of the mutex graph would allow us to identify the above described strong correspondence among actions from a clique (at most one action from a clique can be selected). Such knowledge can be used for some kind of advanced reasoning afterwards. This is just the first part of the idea how projection constraint works. The second part of the idea of projection constraint is to take a subset of literals of a given goal and to calculate how a certain clique of action contributes to satisfaction of the subset of literals. This reasoning can be used to discover that some actions within a certain clique do not contribute enough to the goal and therefore can be ruled out. Actions that are ruled out are no more considered along the search and hence the search speeds up since smaller number of alternatives must be considered.

5.1 Preprocessing Step: Clique Decomposition of Mutex Graph

Projection constraint assumes that a clique decomposition of a mutex graph of a given action layer of the planning graph is known. Thus we need to perform a preprocessing step in which a clique

decomposition (clique cover) of the mutex graph is constructed. Let $G = (A, \mu A)$ be a mutex graph (vertexes are represented by actions, edges are represented by mutexes). The task is to find a partition of the set of vertexes $A = C_1 \cup C_2 \cup \ldots \cup C_n$ such that $C_i \cap C_j = \emptyset$ for every $i, j \in \{1,2,\ldots,n\}$ & $i \neq j$ and C_i is a clique with respect to μA for $i = \{1,2,\ldots,n\}$. Cliques of the partitioning do not cover all the mutexes in general case. That for $mA = \mu A - (C_1^2 \cup C_2^2 \cup \ldots \cup C_n^2)$ $mA \neq \emptyset$ holds in general (where $C^2 = \{\{a,b\} \mid a,b \in C \;\&\; a \neq b\}$). Our requirement is to minimize n and $|mA|$. Unfortunately the problem of clique cover of the defined property is *NP*-complete on a graph without any restriction. The proof of this claim was provided by Golumbic in [9].

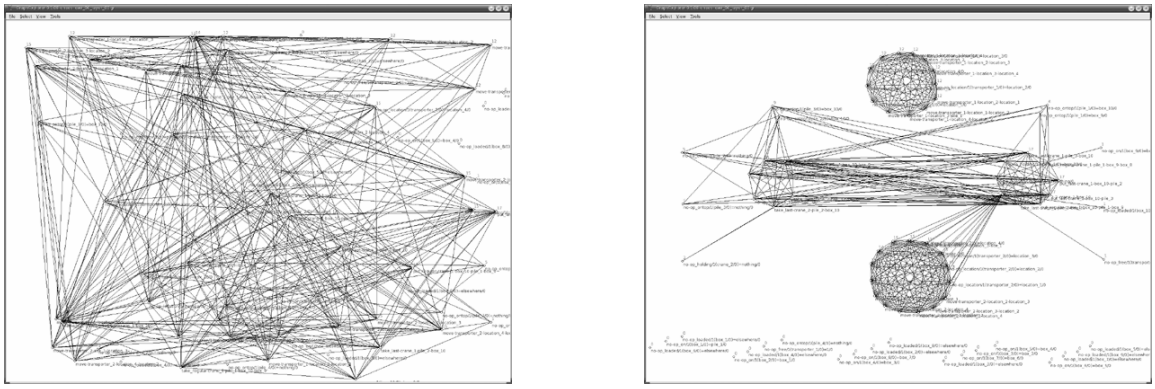

Fig. 1. Illustration of a clique decomposition of a mutex graph obtained from the action layer from the planning graph for a complex dock worker robot problem. The smaller window shows original graph which seems to be unstructured. The larger window shows the same graph decomposed into cliques by the greedy algorithm. The individual cliques of actions are depicted by grouping of vertexes into clusters.

Algorithm 2: Greedy algorithm for finding clique cover

function $cliqueCover(A, \mu A)$: **pair**

1: $n \leftarrow 1$; $mA \leftarrow \emptyset$
2: **while** $A \neq \emptyset$ **do**
3: $C_n \leftarrow \emptyset$; $A_n \leftarrow A$; $\mu A_n \leftarrow \mu A$
4: **while** $A_n \neq \emptyset$ **do**
5: $a \in A_n \mid (\forall b \in A_n) \deg_{(A_n, \mu A_n)}(a) \geq \deg_{(A_n, \mu A_n)}(b)$
6: $C_n \leftarrow C_n \cup \{a\}$
7: $\mu A_n \leftarrow \{\{a,b\} \mid \{a,b\} \in \mu A_n \;\&\; |\{a,b\} \cap C_n| \neq 0\}$
8: $A_n \leftarrow \{b \mid (\exists c \in C_n)\{b,c\} \in \mu A_n\}$
9: $mA \leftarrow mA \cup \{\{a,b\} \mid \{a,b\} \in \mu A \;\&\; |\{a,b\} \cap C_n| = 1\}$
10: $A \leftarrow A - A_n$
11: $\mu A \leftarrow \mu A - (C_n^2 \cup mA)$
12: $n \leftarrow n + 1$
13: **return** $(\{C_1, C_2, \ldots, C_n\}, mA)$

As an exponential amount of time spent in preprocessing step is unacceptable it is necessary to abandon the requirement on optimality of clique cover. It is sufficient to find some clique cover to be able to introduce projection constraint. However the better the clique cover is (with respect to n and $|mA|$) the better is the performance of projection constraint. Our experiments showed that a simple greedy algorithm provides satisfactory results. The greedy algorithm is listed as algorithm 2. Its complexity is polynomial in size of the input graph which is acceptable for preprocessing step.

Observation 1 (Complexity of the greedy clique cover algorithm). The greedy algorithm for finding clique cover (algorithm 2) for a graph $G=(A,\mu A)$ can be implemented to run in $O(|A|^2+|\mu A|)$ steps.

Proof: Suppose that all the sets appearing in the algorithm are implemented as lists. Selection of a vertex of highest degree takes $O(|A|)$ steps and must be performed $O(|A|)$ times. Each edge from μA is considered at most constant number of times. More precisely each directed edge is either included into the constructed clique or into the set of remaining edges mA . n

5.2 Projection Consistency

For the following description of projection constraint consider an action layer of the planning graph for which a clique cover $A=C_1\cup C_2\cup\ldots\cup C_n$ of the set of actions A with respect to the set of mutexes μA was computed. Further suppose we have the set mA consisting of mutexes outside the clique cover. Projection consistency is defined over the above decomposition for a goal p. The goal p is called a projection goal in this context. The fact that at most one action from a clique can be selected allows us to introduce the following definition.

Definition 11 (Clique contribution). A *contribution* of a clique $C\in\{C_1,C_2,\ldots,C_n\}$ to the projection goal p is defined as $\max(|e^+(a)\cap p| \mid a\in C)$. The contribution of a clique C to the projection goal p is denoted as $c(C,p)$.

The concept of clique contribution is helpful when we are trying to decide whether it is possible to satisfy the projection goal using the actions from the clique cover. If for instance $\sum_{i=1}^{n} c(C_i,p)<|p|$ holds then the projection goal p cannot be satisfied. Nevertheless the projection constraint can handle more general case as it is described in the following definitions.

Definition 12 (Projection consistency: supported action). An action $a\in C_i$ for $i\in\{1,2,\ldots,n\}$ is *supported* with respect to *projection consistency* with the projection goal p if $\sum_{j=1,j\neq i}^{n} c(C_j,p)\geq|p-e^+(a)|$ holds.

Definition 13 (Projection consistency of a problem). The preprocessed instance of the supports problem consisting of actions $A = C_1 \cup C_2 \cup \ldots \cup C_n$, mutexes μA and the goal g is *projection consistent* with respect to a projection goal $p \subseteq g$, $p \neq \emptyset$ if every $a \in C_i$ for $i = 1,2,\ldots,n$ is supported

If cliques of the clique cover are regarded as CSP variables and actions from the cliques are regarded as values for these variables then we can introduce a *projection constraint*. The projection constraint bounds domains of all the clique variables. That is the constraints bounds all the variables of the CSP problem. The constraint with respect to the projection goal $p \subseteq g$ is satisfied for an assignment $(C_1 = a_1, C_2 = a_2, \ldots, C_n = a_n)$ if $p \subseteq \bigcup_{i=1}^{n} e^+(a_i)$. To enforce projection consistency over the supports problem for some projection goal p we can easily remove values from the domains of variables. Specifically it is necessary to rule out actions which are not supported according to the definition 8 for the projection goal p. But notice that projection consistency is not a sufficient condition to obtain a solution. There still remain assignments for which the constraint is not satisfied. The second note is on the slight difference of the definition of a solution of the constraint satisfaction problem over the clique variables from the standard definition. We do not necessarily need to assign all the clique variables to solve the problem. The solution requires satisfaction of the projection goal only.

Proposition 1 (Correctness of projection consistency). Projection consistency is correct. That is the set of solutions of the supports problem S is the same as the set of solutions of the supports problem S' which we obtain from S by enforcing projection consistency with respect to a projection goal $p \subseteq g$.

Proof: The proposition is easy to prove by observing that an unsupported action cannot participate in any assignment satisfying the projection constraint for the goal p. Let $a \in C_i$ be an unsupported action for $i \in \{1,2,\ldots,n\}$, that is $\sum_{j=1, j \neq i}^{n} c(C_j, p) < \left| p - e^+(a) \right|$ holds. It is obvious that after the selection of the action a there is no chance to satisfy the projection goal p. n

A useful property of the projection consistency with a single projection goal p is that removal of an unsupported action does not affect any of the remaining supported actions. We call this property a *monotonicity*. The usefulness consist in the fact that it is enough check each action of the problem only once to enforce the projection consistency.

Proposition 2 (Monotonicity of projection consistency). Projection consistency with a projection goal p is *monotone*. That is if an arbitrary unsupported a action is removed from a clique C_i for $i \in \{1,2,\ldots,n\}$ the set of supported actions within the problem remains unchanged.

Proof: Let $b \in C_j$ be an unsupported action after removal of a from C_i. That is after removal of a from C_i $\sum_{k=1, k \neq j}^{n} c(C_k, p) < \left| p - e^+(b) \right|$ holds. First let us investigate the case when $i = j$. It is obvious that removal of a has no effect on the truth value of the expression

$\sum_{k=1,k\neq j}^{n} c(C_k,p) < \left|p - e^+(b)\right|$. Hence the action b was unsupported even before a was removed. For the case when $i \neq j$ the situation is similar. If $c(C_i,p) = c(C_i - \{a\},p)$ then the removal of the action a has effect on the truth value of the expression $\sum_{k=1,k\neq j}^{n} c(C_k,p) < \left|p - e^+(b)\right|$. If $c(C_i,p) > c(C_i - \{a\},p)$, then $\left|p \cap e^+(a)\right| = c(C_i,p)$. From the assumption $\sum_{j=1,j\neq i}^{n} c(C_j,p) < \left|p - e^+(a)\right|$ we have $\sum_{k=1}^{n} c(C_k,p) < \left|p\right|$. Hence also $\sum_{k=1,k\neq j}^{n} c(C_k,p) < \left|p - e^+(b)\right|$ holds. n

5.3 Propagation Algorithm

In order to be able to discuss complexity issues of our approach we have to formally define propagation algorithm for projection consistency. The propagation algorithm for projection consistency is shown as algorithm 3. The input of the algorithm is a projection goal p and the clique decomposition.

Algorithm 3: Projection consistency propagation algorithm

function *propagateProjectionConsistency*$(p,\{C_1,C_2,\ldots,C_n\})$: **set**
1: $\gamma \leftarrow 0$
2: **for** $i = 1,2,\ldots,n$ **do**
3: $\quad c_i \leftarrow calculateCliqueContribution(p,C_i)$
4: $\quad \gamma \leftarrow \gamma + c_i$
5: **for** $i = 1,2,\ldots,n$ **do**
6: $\quad$ **for each** $a \in C_i$ **do**
7: $\quad\quad$ **if** $\gamma + \left|e^+(a) \cap p\right| < \left|p - e^+(a)\right| + c_i$ **then** $C_i \leftarrow C_i - \{a\}$
8: **return** $\{C_1,C_2,\ldots,C_n\}$

function *calculateCliqueContribution*(p,C): **integer**
9: $c \leftarrow 0$
10: **for each** $a \in C$ **do**
11: $\quad c \leftarrow \max(c,\left|e^+(a) \cap p\right|)$
12: **return** c

Theorem 2 (Complexity of projection consistency). *Propagation algorithm for projection consistency with a projection goal* $p \subseteq g$ *over the supports problem consisting of actions* $A = C_1 \cup C_2 \cup \ldots \cup C_n$, *mutexes* μA *and a goal* g *runs in* $O(\left|p\right|\left|A\right|)$ *steps.*

Proof: Since the algorithm for enforcing projection consistency is quite straightforward it is easy compute its complexity. The auxiliary function *calculateCliqueContribution* performs $O(\left|p\right|\left|C\right|)$ steps (the loop on lines 9-11 performs exactly $\left|C\right|$ iterations, each iteration of the loop takes $d\left|p\right|$

steps, where d is the action size bounding constant). Hence lines 2-4 of the main function *propagateProjectionConsistency* takes $O(\sum_{i=1}^{n}|p||C|) = O(|p||A|)$. Finally lines 5-7 of the main function performs a conditional statement on line 7 $|A|$ times. Each check of the condition in the conditional statement on line 7 takes $d|p|$ steps. Hence we have $O(|p||A|)$ steps in total. n

5.4 How to Select Projection Goals

We were not concerned with the question of how to select projection goals for a problem with a goal g. The only condition on a projection goal p is that $p \subseteq g$ must hold. It is suitable to enforce projection consistency with respect to several projection goals. Each of these goals filters out different actions from cliques of the decomposition. The selection of all the sub-goals of the goal g is unaffordable since they are $2^{|g|}$ which is too many. Hence we can select only a limited number of projection goals. At the same time the selection must be done carefully in order to achieve strongest possible filtration. We provide a brief analysis of projection goal selection here. The following ideas are concentrated on comparison of projection consistency with arc-consistency of the supports problem as it was introduced in [22, 23].

Definition 14 (Arc-consistency of the supports problem) [22, 23]. Let us have a supports problem S with a goal g. For each atom $t \in g$ we introduce a so called support variable which contains all the actions that supports the atom t in its domain (an action a *supports* an atom t if $t \in e^{+}(a)$ & $t \notin e^{-}(a)$, a set $s_t = \{a \mid a \in A$ & supports atom $t\}$ is called a s*et of support*s for an atom t) Between every two support variables there is a mutex constraint. The mutex constraint is satisfied by an assignment of actions to its variables if the actions of the assignment are non-mutex. The supports problem is *arc-consistent* if every mutex constraint is arc-consistent [18].

Depending on the quality of the clique decomposition of the mutex graph of the supports problem there may be a situation in which a projection goal can be selected to simulate arc-consistency by projection consistency. Moreover there may be situations when projection consistency is stronger than arc-consistency. Both cases are formally summarized in the following observations. Experiments showed that such cases are not rare, especially projection goals are selected in order to prefer such cases.

Observation 1 (Arc-consistency by projection consistency). For a given supports problem S with a goal g there may be a projection goal $p \subseteq g$ such that if the problem S is projection consistent with the projection goal p then it is arc-consistent.

Proof: It is sufficient to investigate a case for a single constraint between two support variables. An action a in the domain of a support variable v should be removed i order to establish arc-consistency if it does not have a support with respect to the given constraint. That is all the actions in the domain of the support variable u which neighbors with v through the given constraint are mutex with a. Hence $a \notin dom(u)$ holds. Let us suppose that $\{a\} \cup dom(u)$ is a part of a single ac-

tion clique of the decomposition. Further let us suppose that action a do not support the literal corresponding to the variable u Then the projection consistency with respect to a projection goal p which contains the literal corresponding to the variable u removes action a from the action clique. n

Although situation for the projection consistency from the proof is rather artificial, our empirical experimentation gives us evidence that it is not a rare case. Moreover there are a lot of other situations when projection consistency gives the same results as arc-consistency. However these situations are difficult to be theoretically classified. Our last note to arc-consistency is that enforcing arc-consistency by the standard *AC-3* algorithm [18] takes $O(|g||A|^3)$ steps for the supports problem consisting of actions A. In contrast the projection consistency requires only $O(|p||A|)$ steps.

Observation 2 (Strength of projection consistency). For a given supports problem S with a goal g there may be a projection goal $p \subseteq g$ such that the problem S is arc-consistent but it is not projection consistent with the projection goal p.

Proof: We will prove the observation by constructing an instance of the supports problem. Let us have a goal $g = \{t_1, t_2, t_3\}$ where t_i for $i = 1,2,3$ are atoms and actions $a_1^1 = (_, \{t_1\}, \{t_2, t_3\})$, $a_2^1 = (_, \{t_2\}, \{t_1, t_3\})$, $a_3^1 = (_, \{t_3\}, \{t_1, t_2\})$, $a_1^2 = (_, \{t_1\}, \{t_2, t_3\})$, $a_2^2 = (_, \{t_2\}, \{t_1, t_3\})$ and $a_3^2 = (_, \{t_3\}, \{t_1, t_2\})$ ($_$ denotes anonymous variable, that is we do not care about that). It is obvious that the supports problem consisting of actions $\{a_1^1, a_2^1, a_3^1, a_1^2, a_2^2, a_3^2\}$ and the goal g cannot be solved. Actions a_1^1, a_2^1 and a_3^1 are pair-wise mutex as well as actions a_1^2, a_2^2 and a_3^2. The domain of a support variable for the atom t_1 is $\{a_1^1, a_1^2\}$, for the atom t_2 it is $\{a_2^1, a_2^2\}$ and for the atom t_3 it is $\{a_3^1, a_3^2\}$. The arc-consistency does not remove any action from the domains of supports variables. On the other hand, projection consistency is more successful. The preprocessing step finds cliques $\{a_1^1, a_2^1, a_3^1\}$ and $\{a_1^2, a_2^2, a_3^2\}$. The positive contributions of both cliques is 1. Hence any of the actions is supported with respect to projection consistency. So the projection consistency removes all the actions and detects insolvability of the problem. n

Our preliminary experimentations showed that propagation of a good quality can be obtained using projection goals which have the constant number of supports for its literals. That is the projection consistency is enforced for a projection goals $p \subseteq g$ which contains all the literals for which the number of satisfying actions (see definition 10) is the same. More formally let $p_i = \{t \mid t \in g \;\&\; |s_t| = i\}$, then projection consistency is enforced for every $i = \{1,2,\ldots\}$ for which $p_i \neq \emptyset$. This approach prefers cases from observations 5 and 6. Nevertheless we do not know whether there is a set of projection goals which provides better results.

It takes $O(\sum_{i=1,2,\ldots \;\&\; p_i \neq \emptyset} |p_i||A|) = O(|g||A|)$ steps to enforce projection consistency with respect to all projection goals as defined above. If the projection consistency is enforced with respect to one projection goal it may happen that it becomes inconsistent with respect to another projection goal.

Therefore the consistency should be enforced repeatedly in *AC-1* style until cliques of actions are changing. This takes $O(|g||A|^2)$ which is still better than $O(|g||A|^3)$ steps of *AC-3*. However empirical tests showed that such repeating does not provide any significant extra filtration. Hence we use the only iteration of projection consistency with respect to projection goals p_i for $i = \{1,2,\ldots\}$ where $p_i \neq \varnothing$.

6 Experimental Results

We have implemented the proposed projection consistency propagation algorithm within our experimental planning system written in C++. The projection consistency is used to improve solving of the supports problems within backtracking based plan extraction of the GraphPlan algorithm. We exactly follow the algorithm 1 except the part for solving the supports problem. The difference is that projection consistency (with respect to projection goals discussed in section 5.4) is maintained along the search for a solution of a supports problem. Whenever the backtracking algorithm makes a decision (supporting action for an atom is selected) the projection consistency is enforced in order to prune the remaining search space. Our approach is similar to that of Surynek used in [22, 23], but instead of using arc-consistency or singleton arc-consistency we use projection consistency.

We have made several experiments with our algorithm on simple planning domains. We were comparing the standard GraphPlan algorithm, which exactly corresponds to the algorithm 1, and the version of GraphPlan which maintains arc-consistency for solving supports problems with our new version which is maintaining projection consistency for solving supports problems.

Table 1. Time statistics of solving process over several planning problems (part 1). The line *Length* shows planning graph length / solution plan length. The line *PlanGraph* shows time spent by building planning graphs, the line *Extraction* shows time spent by extracting plans from planning graphs, the line *Cliques* shows time spent by building clique covers and the line *Total* shows the total time necessary to find a solution.

Problem	**han02**	**pln04**	**dwr02**	**dwr01**	**han04**	**pln01**	**han03**	**pln10**	**han07**
Length	14/14	5/9	6/10	6/12	10/12	5/9	30/30	10/15	14/20
***Standard* GraphPlan**									
PlanGraph	0.90	4.35	5.13	5.23	4.30	15.07	5.47	6.37	14.03
Extraction	0.43	0.28	2.69	8.53	6.75	0.51	12.03	165.82	142.15
Total	1.33	4.63	7.82	13.76	11.05	15.58	17.50	172.19	156.18
GraphPlan with maintaining *arc-consistency* for supports problems									
PlanGraph	0.91	4.11	4.99	4.97	4.25	15.27	5.34	6.37	13.55
Extraction	0.54	0.15	1.59	1.77	3.41	0.36	12.09	36.86	54.57
Total	1.45	4.26	6.58	6.74	7.66	15.63	17.43	43.23	68.12
GraphPlan with maintaining *projection consistency* for supports problems									
PlanGraph	***0.92***	***4.35***	***5.09***	***5.14***	***4.35***	***15.29***	***5.30***	***6.42***	***13.70***
Cliques	***0.06***	***0.33***	***0.16***	***0.15***	***0.24***	***1.18***	***0.24***	***0.45***	***0.86***
Extraction	***0.33***	***0.1***	***0.29***	***0.51***	***1.68***	***0.22***	***5.32***	***9.3***	***21.62***
Total	***1.31***	***4.78***	***5.54***	***5.80***	***6.27***	***16.69***	***10.86***	***16.17***	***36.18***

All the planning problems which were used for our experiments are available at the web site: http://ktiml.mff.cuni.cz/~surynek/research/cpaior2007/. The planning domains are the same as that used for empirical tests in [22]. They are Dock Worker Robots planning domain, Refueling Planes planning domain and Towers of Hanoi planning domain. The used planning domains are described in details in [22]. Several problems of varying difficulty of each planning domain were used for our experiments. The planning problems were selected to cover the range from easy problems to relatively hard problems. The lengths of solutions varied from 9 to 38 actions. Performance results on some of these problems are shown in tables 1 and 2 and in figures 2, 3 and 4.

Table 2. Time statistics of solving process over several planning problems (part 2).

Problem	**pln05**	**dwr05**	**pln06**	**pln11**	**dwr07**	**han08**	**dwr16**	**pln13**	**dwr17**
Length	6/14	14/28	9/14	10/14	16/36	20/26	18/34	10/16	20/38
***Standard* GraphPlan**									
PlanGraph	38.6	57.9	44.0	54.8	N/A	39.9	N/A	N/A	N/A
Extraction	460.3	554.3	2660.2	3441.3	N/A	2056.2	N/A	N/A	N/A
Total	499.0	612.2	2704.2	3496.1	N/A	2096.1	N/A	N/A	N/A
GraphPlan with maintaining *arc-consistency* for supports problems									
PlanGraph	38.1	57.2	42.2	53.7	100.50	41.5	207.8	82.3	378.0
Extraction	31.8	60.4	221.2	311.5	279.12	549.8	714.3	1052.5	6148.6
Total	69.9	117.7	263.4	365.2	379.62	591.4	922.2	1134.8	6526.6
GraphPlan with maintaining *projection consistency* for supports problems									
PlanGraph	***40.0***	***57.5***	***44.1***	***56.1***	***99.08***	***40.9***	***204.9***	***86.3***	***369.8***
Cliques	***2.87***	***2.10***	***3.05***	***4.94***	***3.43***	***2.68***	***6.78***	***6.1***	***13.21***
Extraction	***18.92***	***6.13***	***29.45***	***37.16***	***107.74***	***184.02***	***288.61***	***103.81***	***2182.23***
Total	***61.88***	***65.76***	***76.67***	***98.20***	***210.25***	***227.63***	***500.32***	***196.24***	***2565.24***

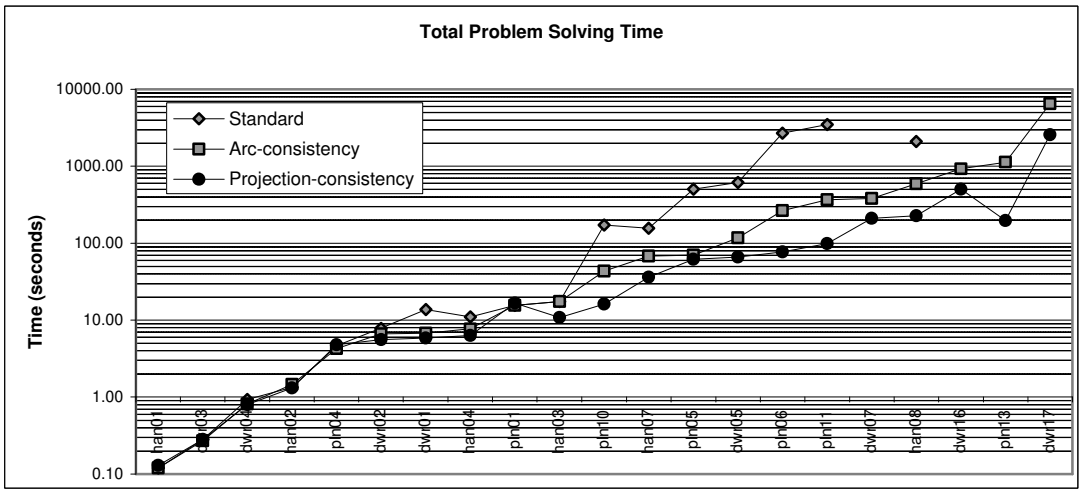

Fig. 2. Problem solving time in total (planning graphs building time + clique cover time + plan extraction time) of several planning problems. Time range uses logarithmic scale. Several results of standard GraphPlan for hard problems are missing due to timeout (2 hours).

The tests were performed on a machine with two AMD Opteron 242 processors (2×1600MHz) and 1 GB of memory running Mandriva Linux 10.2. The implementation was compiled with gcc compiler version 3.4.3 with maximum optimization for the target machine (-O9 -mtune=opteron). No parallel execution was used. The two processors were used only for running two tests simultaneously.

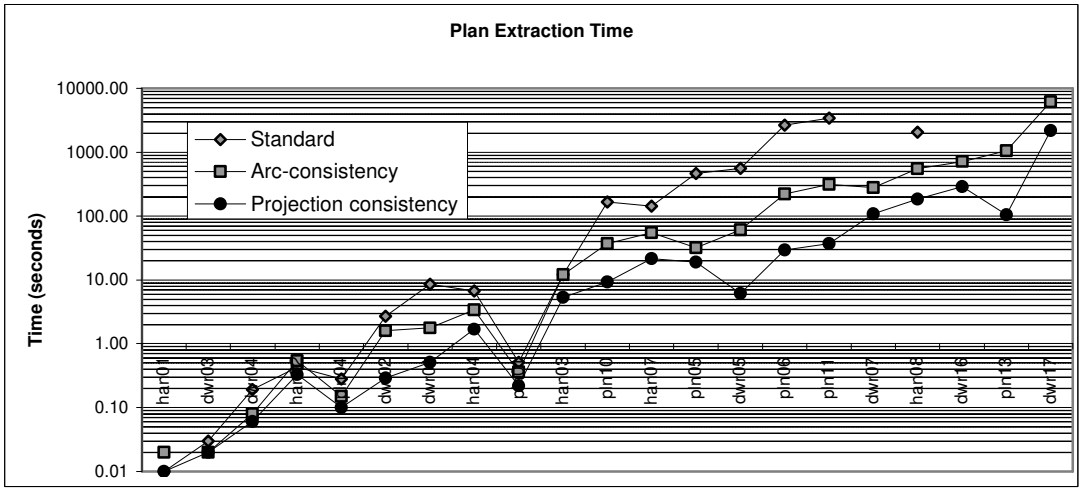

Fig. 3. Plan extraction time of several planning problems. Time range uses logarithmic scale. Again, several results of standard GraphPlan for hard problems are missing due to timeout (2 hours).

6.1 Analysis of Experimental Results

The proposed method for solving supports problems based on maintaining of projection consistency brings significant improvement in terms of time as well as in terms of number of constraint checks on hard problems compared to the version which uses maintaining of arc-consistency. Notice that the version of the algorithm with projection consistency must perform clique decompositions before the supports problem is solved. Although the clique decompositions represent an overhead on easy problems the improvement in plan extraction with projection consistency overrides this disadvantage on hard problems. The improvement of the plan extraction phase is up to about 1000% . Moreover we can say that the larger the portion of time is spent by search the better the improvement by use of projection consistency is.

If we compare the plan extraction which uses projection consistency with the standard version the improvement is up to about 1000% in overall problem solving and up to about 10000% in plan extraction phase.

It is also possible to observe that the projection consistency is especially successful on problems with many interacting objects in the planning world and high parallelism of actions (for example planes problems 11 and 13 and dock worker robots problem 07). On the other hand if the interaction between objects in the planning world is low and if there is a low parallelism, the advanced reasoning over supports problems does not represent any significant improvement (for example Hanoi tower problem 07 and 08).

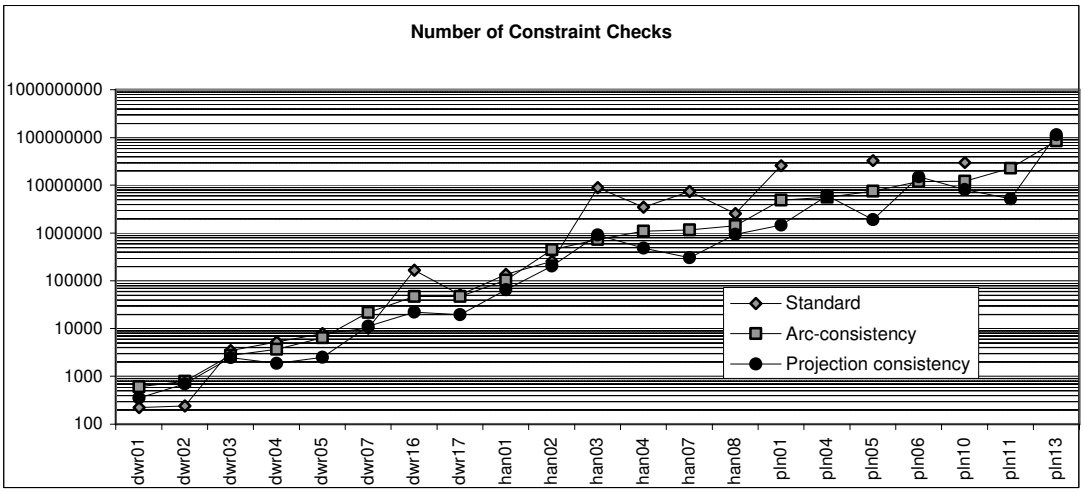

Fig. 4. Number of constraint checks of several planning problems. Time range uses logarithmic scale. Again, several results of standard GraphPlan for hard problems are missing due to timeout (2 hours).

6.2 Implementation Notes

Our experimental planning system which we used to produce the empirical tests is not a state-of-the-art planner. At the current stage it cannot compete with planners from International Planning Competition (IPC) [7]. It is caused partially by a not well optimized implementation and partially by the fact that we do not use any domain specific heuristics. Nevertheless it is not our goal to compete with planners participating in IPC at the current stage. We are rather focusing on understanding the structure of planning problems and on utilizing this knowledge to improve the solving process.

For our empirical tests we used standard variable and value selection heuristics. Specifically an atom with the smallest number of supporting actions is always selected as first to be satisfied. Then supporting action are tried starting with the action that is least constrained.

There is also another important implementation issue concerning nogood recording. We use unrestricted nogood recording within our experimental planning system. A special multiple-valued decision tree [19] is used to store nogoods. The tree is optimized for space by preferring low branching towards root and high branching towards leaves. Our minor experiments showed that the decision tree requires space of about $30\% - 10\%$ of sum of sizes of all stored nogoods on the testing problems.

The last implementation issue we would like to mention is that we use state variable representation for planning problems [8]. Compared to classical representation the state variable representation provides easier expressing of actions and also some performance advantages directly connected with this fact.

7 Related Works

The main difference of our approach from other approaches exploiting another formalism (CSP, SAT) for solving planning problems [10, 11, 15, 17, 12, 13, 14] is that we do not formulate the planning problem in another formalism as a whole. We use constraint programming approach only to solve a sub-problem arising during search. Moreover we not only model the supports problems in constraint programming formalism, we extend the formalism by introducing new type of consistency to model the sub-problem in a better way.

Kambhampati's successful idea to formulate plan extraction from planning graph as CSP is presented in [10]. He evaluates the use of various constraint programming techniques and its impact on the effectivity of plan extraction. Several extensions of expressivity of planning graphs are described in [11]. From our point of view the most interesting idea is to generalize mutex relations and its propagation in planning graph. Another approach is presented in [17] by Lopez and Bacchus. Again they model the planning problem in planning graph representation as CSP. The originality of their technique consists in making transformations of the obtained CSP which uncovers additional structural information about the problem.

One of the most successful in term of speed of solving the planning problems are planning algorithms based on Boolean formula satisfiability encoding [15, 12, 13, 14]. Here the speed of solving algorithm also depends on the quality of encoding. The encoding of planning problems based on planning graph representation was by Kautz and Selman studied in [15].

The also successful algorithm CPlan of Van Beek and Chen [24] uses hand tailored CSP encoding of a planning problem. The success of their approach is accounted to well designed numeric constraints that bind spatiotemporally distant object of the planning world.

Finally let us mention that a greedy search for mutex cliques was also used by Blum and Furst in their original GraphPlan [4]. But they were trying to detect mutex cliques from different reasons. They used the discovered mutex cliques to identify state variables (which we have intrinsically in problem formulation from the beginning) and to reduce memory requirements.

8 Conclusion and Future Work

We proposed a novel consistency technique which we called projection consistency. The technique is designed to prune the search space during extraction of plans by the GraphPlan-style algorithm. We theoretically showed that the projection consistency has faster propagation algorithm than arc-consistency propagation algorithm AC-3 which application on the same problem was recently studied by Surynek in [22]. Empirical test showed improvements in order of magnitudes compared to the standard GraphPlan and also compared to the version using arc-consistency. The improvements are both in overall time as well as in number of constraint checks.

There is a lot of future work. The first interesting issue is how to make projection consistency stronger. This may be done by other types of projections. But it is also possible to do it by slight modification of the definition of the supported action. Instead of the expression $\sum_{j=1,j\neq i}^{n} c(C_j, p) \geq \left| p - e^{+}(a) \right|$ in the definition 10 one can use $\sum_{j=1,j\neq i}^{n} c(C_j, p - e^{+}(a)) < \left| p - e^{+}(a) \right|$. Unfortunately this change causes that monotonicity (proposition 2) no longer holds. And hence the complexity of propagation algorithm increases. The solution may be better propagation algorithm.

The next important issue about the projection consistency which we are currently intensively studying is a tractable case. That is the case when it is possible to solve the supports problem in polynomial time. We identified that such case arise when intersection graph [9] of scopes (sets of atoms) of cliques of the decomposition is acyclic. Currently we are working on heuristics for preferring such cases.

The similarity between Boolean formula satisfaction problem and supports problem as it is shown in theorem 1 leads us to the question whether it is possible to exploit projection consistency for solving SAT problems. The expectable question is also how to extend the presented ideas for planning graphs with time and resources [16, 21]. Since the planning graphs for complex problems are really large the related question is also how to make planning graphs unground and how to get rid of huge numbers of no-operation actions.

References

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

2. Allen, J., Hendler, J., Tate, A. (editors): Readings in Planning. Morgan Kaufmann Publishers, 1990.

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

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

5. Boeing Co.: Integrated Defense Systems - X-45 J-UCAS. http://www.boeing.com/defense-space/military/x-45/index.html, Boeing Co., USA, October 2006.

6. Dechter, R.: Constraint Processing. Morgan Kaufmann Publishers, 2003.

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

8. Ghallab, M., Nau, D. S., Traverso, P.: Automated Planning: theory and practice. Morgan Kaufmann Publishers, 2004.

9. Golumbic, M. C.: Algorithmic Graph Theory and Perfect Graphs. Academic Press, 1980.

10. Kambhampati, S.: 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.

11. Kambhampati, S., Parker, E., Lambrecht, E.: Understanding and Extending GraphPlan. In Proceedings of 4th European Conference on Planning (ECP-97), 260-272, LNCS 1348, Springer-Verlag, 1997.

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

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

14. Kautz, H. A., Selman, B.: 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.

15. Kautz, H. A., Selman, B.: 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.

16. Long, D., Fox, M.: Exploiting a GraphPlan Framework in Temporal Planning. In Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS-2003), 52-61, AAAI Press, 2003.

17. Lopez, A., Bacchus, F.: 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. Mackworth, A. K.: Consistency in Networks of Relations. Artificial Intelligence 8, 99-118, AAAI Press, 1977.

19. Miller, D. M., Drechsler, R.: On the Construction of Multiple-Valued Decision Diagrams. In Proceedings of the 32nd IEEE International Symposium on Multiple-Valued Logic (ISMVL-2002), 245-253, Boston, Massachusetts, USA. IEEE Computer Society, 2002.

20. Régin, J. C.: A Filtering Algorithm for Constraints of Difference. In Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94), 362-367, AAAI Press, 1994.

21. Smith, D. E., Weld, D. S.: Temporal Planning with Mutual Exclusion Reasoning. In Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99), 326-337, Morgan Kaufmann, 1999.

22. Surynek, P.: Maintaining Arc-consistency over Mutex Relations in Planning Graphs during Search. Accepted to the 20th FLAIRS conference, Key West, Florida, USA, 2007. Also available as technical report in ITI Series, http://iti.mff.cuni.cz/series/index.html, Charles University , Prague, Czech Republic, 2007. And also available as technical report at personal web http://ktiml.mff.cuni.cz/~surynek, 2007.

23. Surynek, P.: Constraint Based Reasoning over Mutex Relations in Planning Graphs during Search. Technical report, ITI Series, http://iti.mff.cuni.cz/series/index.html, Charles University, Prague, Czech Republic, 2007. Also available as technical report at personal web http://ktiml.mff.cuni.cz/~surynek.

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