Solving Difficult Problems by Viewing them as Structured Dense Graphs

Pavel Surynek, Lukáš Chrpa, and Jiří Vyskočil

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

Abstract. We are addressing the solving process over difficult AI problems such as planning and Boolean formula satisfaction in this paper. We propose a method for disentangling the structure of the problem hidden in its formulation. Namely we are viewing the problem as a graph with vertices end edges that we decompose into several complete sub graphs. The clique decomposition of the graph provides us with information about the structure of the problem which is then used for further reasoning. This reasoning is primarily targeted on early determining that a certain decision made during the search is unpromising. We implemented our new method and performed an experimental evaluation in planning with planning graphs and in Boolean formula satisfaction. The per-formed experiments showed order-of-magnitude improvements in comparison with other state-of-the-art approaches.

Keywords: problem solving, planning, SAT, planning graphs, consistency

1 Introduction

We are addressing the solving process over difficult problems known from artificial intelligence in this paper. Namely, we are interested in solving planning problems and Boolean formula satisfaction problems. These problems represent some of the most challenging issues in artificial intelligence as well as in theoretical computer science.

Our recent research focused on solving process over these problems showed that a structure of the problem has a large effect on the ability of a solver to solve it. Namely, the structure of a graph obtained by some kind of reformulation of the input problem is seemed to be a good measure of the difficulty the problem. We found that disentangling the structure of the problem hidden in the graph may significantly improve the process of search for solution.

The problem area of our choice has a wide variety of theoretical and practical implications. A planning problem is stated as a task of finding a sequence of actions resulting in achieving some goal [4]. The need of solving planning problems arises 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, 6] as well as the case of unmanned military devices [9].

Boolean formula satisfaction problems and SAT solving techniques play an extremely important role in theoretical computer science as well as in practice too. The question of whether there exists a complete polynomial time SAT solver is a key question for theoretical computer science and it is open for many years (the P vs. NP problem) [10]. On the other hand the practical use of SAT problems and SAT solvers in real life applications is also very intensive. Applications of SAT solving techniques range from microprocessor verification [32] to field-programmable gate array design [25].

Moreover, the described problem areas overlap. One of the most successful state of the art solver for planning problems uses translation of the input problem into a Boolean formula which is subsequently solved using SAT resolution techniques [21].

The idea behind our method itself is simple. We are viewing the input problem as a graph with vertices and edges. The vertices of the graph represent contributions to the solution and edges represent conflicts. The task is to select vertices which together form a solution by their contributions and that do not conflict with each other. That is, the selected vertices must form a stable set in the graph.

Conflicting vertices are those that cannot be both selected into a solution. For ex-ample in the domain of Boolean formulas, the conflicting vertices correspond to positive and negative literals of the same variable. At most one of such literals can be selected to contribute to the solution in this case.

Having the graph constructed from the input problem, a clique decomposition of this graph is found by a greedy algorithm (we do not need optimal clique decomposition; we need just some of the reasonable quality). The important property of the constructed clique decomposition is that at most one vertex from each clique can be selected to contribute to the solution. In this situation we can perform some kind of a contribution counting to rule out vertices of the graph that can never be selected for the solution. To do this, the maximum contribution of vertices of each clique is calculated. Then a vertex of a certain clique can be ruled out if the vertices from other cliques together with the selected vertex do not contribute enough to the solution.

For finding a clique decomposition of a reasonable quality by the greedy algorithm, the graph is required to be dense. It is expectable that the greedy algorithm would perform better on a graph with many edges than on a sparse one.

Although this problem reformulation is looking weak it provides a strong reasoning about the structure of the problem and about the effect of making a decision during solving process (selection of a vertex) on the existence of a solution. Moreover, on some difficult problems this kind of reasoning is strong enough to decide the problem itself without any search.

The organization of our paper is following. First we introduce some problem and a consistency technique for it. This may look like to be far apart from the reality of planning and SAT domain but it is not as we see later. The consistency technique is based on a clique decomposition of the graph of the defined problem. The clique decomposition serves as a method for disentangling the hidden structure of the problem. In next two sections, we put together the defined problem and the consistency technique with the problems of searching for a plan and searching for a satisfying valuation of a Boolean formula. The connection between the defined problem and application areas of planning and SAT solving reveals in this section. The proposed consistency technique is used for early determining that some decision made during solving process was wrong. Next, we put our work in relation to similar works in the field. Finally, we provide some discussion of our work and propose some future re-search directions of the studied topic.

2 Problem Definition

We formally define our problem and discuss its complexity in this section. The de-fined problem represents a common sub problem or reformulation for both planning and Boolean formula satisfaction.

Definition 1 (Contribution problem). Let S be a finite set of symbols. The set S represents an *objective* we want to satisfy. Next, let $G=(V,E)$ be an undirected graph where each vertex v from the set V is assigned a set of symbols $c(v)$ which is a (proper) subset of S. The task is to select a subset of vertices I such that $\bigcup_{v\in I} c(v)=S$ and no two vertices from I are adjacent through an edge from E (that is, $(\forall u\in I\ \forall v\in I)\{u,v\}\notin E$). The set of selected vertices thus forms a sable set in the graph G. Let us call the defined problem a *contribution problem* for further reference. The sets $c(v)$ associated with vertices of the set V are called *contributions to the objective* S. □

This problem may look rather artificial however it provides common basis for problems of our interest. Although the contribution problem seems to be not so hard, it is shown in [29] that it is *NP*-complete.

Since the contribution problem is *NP*-complete it is improbable that it can be solved without search. However, the efficiency of the search varies greatly depending on how informed the search is. Our method for improving the search is based on a special consistency technique which is able to discover wrong decisions in early stages.

3 Consistency Technique Based on Greedy Clique Decomposition

The formal description of a new consistency technique for improving the solving process of the contribution problem is provided in this section. We call our new technique a *projection consistency* according to the way how propagation is done through that. The projection consistency introduces some type of a global reasoning over the contribution problem. That is we consider the problem as a whole at once. It is designed to improve the search by discovering wrong decisions in early stages.

3.1 Basic Definitions from Constraint Programming

The consistency technique we develop in following paragraphs exploits some concepts of constraint programming methodology. Therefore some basic definitions from constraint programming are necessary. The key concept is a *constraint satisfaction problem.*

Definition 2 (Constraint satisfaction problem) [12]. 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 sequence of variables constrained by a constraint $c \in C$ is denoted as X_c. □

Definition 3 (Solution of CSP) [12]. A *solution* of a constraint satisfaction problem (X, D, C) is an assignment of values to the variables $\psi : X \rightarrow D$ such that all the constrains are satisfied for ψ, that is $(\forall c \in C)[x_1, x_2, \ldots, x_k] = X_c \Rightarrow [\psi(x_1), \psi(x_2), \ldots \ \ldots, \psi(x_k)] \in C$. □

CSPs are often solved using so called *constraint propagation*. If a domain of a variable is changed then this change is propagated into the domains of other variables through constraints. The quality of the solving algorithm is often tightly connected with the quality of propagation techniques 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 large parts of the problem (for example Régin's *allDifferent* constraint [26]). Our method is trying to follow the concept of such global propagation.

3.2 Clique Decomposition of Contribution Graph

The basic idea behind our consistency technique is to disentangle the structure of the graph of the contribution problem. This disentanglement is done by decomposing of the input graph into several complete sub-graphs (cliques). The knowledge of the clique decomposition of the contribution graph would allow us to identify a correspondence among vertices from a clique. Having a clique we know that at most one vertex from the clique can be selected to contribute into the objective S. Such property can be used for some kind of advanced reasoning afterwards. This is just the first part of the idea behind the projection consistency.

The second part of the idea of projection consistency is to take a subset of the given objective and to calculate how a certain clique of vertices contributes to satisfaction of the subset of the objective. This reasoning can be used to discover that some vertices within a certain clique do not contribute enough to the objective and therefore can be ruled out. Vertices 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.

Projection consistency assumes that a clique decomposition of the contribution graph is known. Thus we need to perform a preprocessing step in which a clique decomposition (clique cover) of the graph is constructed.

Let $G = (V, E)$ be a contribution graph. The task is to find a partitioning of the set of vertices $V = C_1 \cup C_2 \cup \ldots \cup C_n$ 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 E for $i = \{1, 2, \ldots, n\}$.

Cliques of the partitioning do not cover all the edges in general case. That is, $F = E - (C_1^2 \cup C_2^2 \cup \ldots \cup C_n^2) \neq \emptyset$ (where $C^2 = \{\{a,b\} \mid a,b \in C \,\&\, a \neq b\}$). Our requirement is to minimize n and $|F|$. 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].

As an exponential amount of time spent in preprocessing step is unacceptable it is necessary to abandon the requirement on optimality of the clique cover. It is sufficient to find some clique cover to be able to introduce projection consistency. However, the better the clique cover is (with respect to n and $|F|$) the better is the performance of projection consistency. Our experiments showed that a simple greedy algorithm provides satisfactory results.

The greedy algorithm we are using is listed as algorithm 1. Its complexity is polynomial in size of the input graph that is acceptable for preprocessing step.

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

Proof: Suppose that all the sets appearing in the algorithm are implemented as lists. A selection of a vertex of highest degree takes $O(|V|)$ steps and must be performed $O(|V|)$ times. Each edge from E 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 F. ■

The described greedy algorithm is particularly successful on dense graphs. Graphs obtained by reformulation of the difficult problems we studied mostly satisfy this property.

Algorithm 1: Greedy algorithm for finding clique cover

```
function cliqueCover(V,E) : pair
1:      $n \leftarrow 1$; $F \leftarrow \emptyset$
2:      while $V \neq \emptyset$ do
3:          $C_n \leftarrow \emptyset$; $V_n \leftarrow V$; $E_n \leftarrow E$
4:          while $V_n \neq \emptyset$ do
5:              $v \in V_n \mid (\forall u \in V_n) \deg_{(V_n,E_n)}(v) \geq \deg_{(V_n,E_n)}(u)$
6:              $C_n \leftarrow C_n \cup \{v\}$
7:              $V_n \leftarrow \{u \mid (\forall w \in C_n)\{u,w\} \in E_n\}$
8:              $E_n \leftarrow \{\{u,v\} \mid \{u,v\} \in E_n \;\&\; \{u,v\} \subseteq V_n\}$
9:              $F \leftarrow F \cup \{\{u,v\} \mid \{u,v\} \in E \;\&\; |\{u,v\} \cap C_n| = 1\}$
10:         $V \leftarrow V - C_n$
11:         $E \leftarrow E - (C_n^2 \cup F)$
12:         $n \leftarrow n+1$
13:     return $(\{C_1, C_2, \ldots, C_n\}, F)$
```

3.3 Clique Contribution Counting

For the following description of projection constraint consider the graph $G = (V, E)$ of a contribution problem for which a clique cover $V = C_1 \cup C_2 \cup \ldots \cup C_n$ of the set of vertices V with respect to the set of edges E was computed. Further suppose we have the set F consisting of edges outside the clique cover. Projection consistency is defined over the above decomposition for an objective p which is a subset of the objective S. The objective p is called a *projection objective* in this context. The fact that at most one vertex from a clique can be selected allows us to introduce the following definition.

Definition 4 (Clique contribution). A *contribution* of a clique $C \in \{C_1, C_2, \ldots, C_n\}$ to the projection objective p is defined as $\max(|c(v) \cap p| \mid v \in C)$. The contribution of a clique C to the projection objective 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 objective using the vertices from the clique cover. If for instance $\sum_{i=1}^{n} c(C_i, p) < |p|$ holds then the projection objective p cannot be satisfied. Nevertheless the projection consistency can handle a more general case as it is described in the following definitions.

Definition 5 (Supported vertex). A vertex $v \in C_i$ for $i \in \{1,2,\ldots,n\}$ is *supported* with respect to *projection consistency* with the projection objective p if $\sum_{j=1, j \neq i}^{n} c(C_j, p) \geq |p - c(v)|$ holds. □

Definition 6 (Consistency of a problem). The preprocessed instance of the contribution problem consisting of vertices $V = C_1 \cup C_2 \cup \ldots \cup C_n$, edges E, and the objective S is *projection consistent* with respect to a projection objective $p \subseteq S$, $p \neq \emptyset$ if every vertex $v \in C_i$ for $i = 1,2,\ldots,n$ is supported. □

If cliques of the clique cover are regarded as CSP variables and vertices 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 constraint bounds all the variables of the CSP problem. The constraint with respect to the projection objective $p \subseteq S$ is satisfied for an assignment $(C_1 = v_1, C_2 = v_2, \ldots, C_n = v_n)$ if $p \subseteq \bigcup_{i=1}^{n} c(v_i)$.

To enforce projection consistency over the contribution problem for some projection objective p we can easily remove values from the domains of variables. Specifically it is necessary to rule out values (vertices) which are not supported according to the definition 5 for the projection objective p. 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 objective only.

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

Proof: The proposition is easy to prove by observing that an unsupported vertex cannot participate in any solution satisfying the objective S. Let $v \in C_i$ be an unsupported vertex for $i \in \{1,2,\ldots,n\}$, that is $\sum_{j=1, j \neq i}^{n} c(C_j, p) < |p - c(v)|$. Thus after selecting the vertex v into the solution, there is no chance to satisfy the projection objective p. Hence there is even less chance to satisfy the objective S since it is superset of p. ■

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

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

Proof: Let $u \in C_j$ be an unsupported vertex after removal of v from C_i. That is after removal of v from C_i the inequality $\sum_{k=1,k\neq j}^{n} c(C_k,p) < |p - v(u)|$ holds.

First let us investigate the case when $i = j$. Observe that removal of v has no effect on the truth value of the expression $\sum_{k=1,k\neq j}^{n} c(C_k,p) < |p - c(u)|$. Hence the vertex u was unsupported even before v was removed.

For the case when $i \neq j$ the situation is similar. If $c(C_i,p) = c(C_i - \{v\}, p)$, then the removal of the vertex v has no effect on the truth value of the expression $\sum_{k=1,k\neq j}^{n} c(C_k,p) < |p - c(u)|$. If $c(C_i,p) > c(C_i - \{v\}, p)$, then $|p \cap c(v)| = c(C_i,p)$. From the assumption that $\sum_{j=1,j\neq i}^{n} c(C_j,p) < |p - c(v)|$ we have $\sum_{k=1}^{n} c(C_k,p) < |p|$. Hence also $\sum_{k=1,k\neq j}^{n} c(C_k,p) < |p - c(v)|$. ■

3.4 Propagation Algorithm

Algorithm 2: Projection consistency propagation algorithm

```
function propagateProjectionConsistency({C_1, C_2, ..., C_n}, p): set
1:      $\gamma \leftarrow 0$
2:      for $i = 1,2,\dots,n$ do
3:      |       $c_i \leftarrow calculateCliqueContribution(C_i, p)$
4:      |       $\gamma \leftarrow \gamma + c_i$
5:      for $i = 1,2,\dots,n$ do
6:      |       for each $v \in C_i$ do
7:      |       |       if $\gamma + |c(v) \cap p| < |p - c(v)| + c_i$ then $C_i \leftarrow C_i - \{v\}$
8:      return $\{C_1, C_2, \dots, C_n\}$

function calculateCliqueContribution(C, p): integer
9:      $c \leftarrow 0$
10:     for each $v \in C$ do
11:     |       $c \leftarrow \max(c, |c(v) \cap p|)$
12:     return $c$
```

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 2. The input of the algorithm is a projection objective p and the clique decomposition of the contribution problem.

Theorem 2 (Complexity of projection consistency). *Propagation algorithm for projection consistency with a projection objective* $p \subseteq S$ *over the contribution problem consisting of vertices* $V = C_1 \cup C_2 \cup \ldots \cup C_n$*, edges* E*, and an objective* S *runs in* $O(|p||S||V|)$ *steps.* □

Proof: Since the algorithm for enforcing projection consistency is quite straightforward it is easy compute its complexity. The auxiliary function *calculateCliqueContribution* performs $O(|p||C|)$ steps (the loop on lines 10-11 performs exactly $|C|$ iterations, each iteration of the loop takes $|p||S|$ steps). Hence lines 2-4 of the main function *propagateProjectionConsistency* takes $O(\sum_{i=1}^{n}|p||S||C|) = O(|p||S||V|)$. Finally lines 5-7 of the main function performs a conditional statement on line 7 $|V|$ times. Each check of the condition in the conditional statement on line 7 takes $|p||S|$ steps. Hence we have $O(|p||S||V|)$ steps in total. ■

3.6 How to Select Projection Objectives

Until now, we were not concerned with the question of how to select projection objectives for a problem with an objective S. The only condition on a projection objective p is that $p \subseteq S$. It is suitable to enforce projection consistency with respect to several projection objectives. Each of these objectives filters out different vertices from cliques of the decomposition. The selection of all the sub-objectives of the objective S is unaffordable since they are $2^{|S|}$ which is too many. Hence we can select only a limited number of projection objectives. At the same time the selection must be done carefully in order to achieve strongest possible filtration.

Our preliminary experimentations showed that propagation of a good quality can be obtained by using projection objectives which have the constant number of supporting vertices. More formally let $p_i = \{t \mid t \in S \; \& \; |\{v \in V \mid t \in c(v)\}| = i\}$ then projection consistency is enforced for every $i = \{1,2,\ldots\}$ for which $p_i \neq \emptyset$. Let us note that we do not know whether there exists a set of projection objectives which provides better results.

It takes $O(\sum_{i=1,2,\ldots \,\&\, p_i \neq \emptyset}|p_i||S||V|) = O(|S|^2|V|)$ steps to enforce projection consistency with respect to all projection objectives as defined above. If the projection consistency is enforced with respect to one projection objective it may happen that it becomes inconsistent with respect to another projection objective. Therefore the consistency should be enforced repeatedly in *AC-1* [12] style until cliques of vertices are changing. This takes $O(|S|^2|V|^2)$. 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 objectives p_i for $i = \{1,2,\ldots\}$ where $p_i \neq \emptyset$.

An example of projection consistent problem with respect to the multiple projection objectives p_i is shown in figure 1. In this example, all the vertices which are not part of the only one solution of the problem are ruled out by the projection consistency. So the problem was solved by projection consistency itself. On larger problems the situation is not so lucky in general.

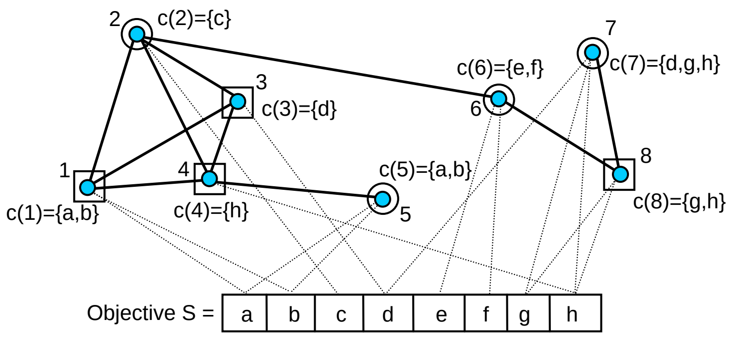

Fig. 1. An instance of the contribution problem consisting of a graph with eight vertices. Doted lines connecting a vertex and a symbol depict the contributions. Double-circled vertices depict the solution of the problem (that is, the set of vertices which together form the objective S by their contributions). Projection consistency with respect to multiple projection objectives is enforced in this contribution problem. Cliques detected by the greedy algorithm are: C_1={1,2,3,4}, C_2={6,8}, C_3={5}, and C_4={7}. Unsupported vertices for the projection objectives are: p_1={c,e,f}, p_2={a,b,d,g}, and p_3={h} are depicted by squared vertices. For example vertex 3 is unsupported for the projection objective p_1={c,e,f} since vertex 3 contributes by 0, C_2 contributes by 2, C_3 contributes by 0, and C_4 contributes by 0 which is together less than the size of p_1.

4 Application in Solving AI Planning Problems

In this section we describe application of contribution problems and projection consistency in the area of planning. Namely we exploit the technique developed in previous sections in solving planning problems over planning graphs by the *GraphPlan* algorithm [8].

For purposes of clarity we are using a simple language for expressing planning problems. To describe a 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 [18] Ghallab et al. show more approaches for describing planning problems). The following definitions assume a fixed language L.

Definition 7 (Atom). An *atomic formula* (atom) 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 $i = 1,2,\ldots,n$. □

Definition 8 (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. 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 9 (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 three 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 $\gamma(s,a) = (s - e^{-}(a)) \cup e^{+}(a)$. □

For purposes of planning graphs there are also assumed so called *no-op actions*. 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.

Definition 10 (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 11 (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 $i = 1, 2, \ldots, n$. □

4.1 Planning Graphs and GraphPlan Algorithm

The GraphPlan algorithm 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 extended planning graph is performed. The GraphPlan algorithm uses standard backtracking to extract plan from planning graphs. If the second phase is unsuccessful the process continues with the first phase. That is 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, …, 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 kth 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*). The next proposition layer P_{k+1} consists of all the positive effects of actions from A_{k+1} (this is the reason for having no-op actions).

Definition 12 (Independence of actions). A pair of actions $\{a,b\}$ is *independent* if $e^-(a) \cap (p(b) \cup e^+(b)) = \emptyset$ and $e^-(b) \cap (p(a) \cup e^+(a)) = \emptyset$. Otherwise $\{a,b\}$ is a pair of *dependent* actions. □

Definition 13 (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 14 (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 $\{p,q\} \subseteq e^+(c)$. □

4.2 Problem of Finding Supporting Actions for a 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 effectiveness of a method for solving problem of finding supports has a major impact on the performance of the planning algorithm as a whole. It is because this sub-problem must be solved many times during the process of finding a plan.

The problem of finding supports for a goal in this context is actually the contribution problem where the set of vertices is equal to A, set of edges is equal to μA, and each vertex corresponding to the action a is assigned the set of contributions $e^+(a)$. The objective is the goal g here. Having this we can apply the projection consistency technique. We use a backtracking based search from the standard GraphPlan algorithm augmented by maintaining projection consistency whenever a decision is made.

4.3 Experimental Results for Planning Problems

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 problems of finding supports within backtracking based plan extraction of the GraphPlan algorithm. We exactly follow the standard GraphPlan algorithm [8] except the part for solving the problem of finding supporting actions for a goal. The difference is that we are viewing the problem as contribution problem and maintain projection consistency with respect to the discussed projection objectives along the search for a solution of the problem. That is, whenever the backtracking algorithm makes a decision the projection consistency is enforced in order to prune the remaining search space.

We have made several experiments with our algorithm on simple planning domains. We were comparing the standard GraphPlan algorithm with our new version which maintains projection consistency.

Table 1. Time statistics of solving process over several planning problems. 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 *projection consistency* for contribution problems									
PlanGraph	***0.92***	***4.35***	***5.09***	***5.14***	***4.35***	***15.29***	***5.30***	***6.42***	***13.70***
Cliques	***0.09***	***9.48***	***1.09***	***1.09***	***0.87***	***46.29***	***0.77***	***6.54***	***4.67***
Extraction	***0.24***	***0.1***	***0.55***	***1.13***	***3.53***	***0.24***	***5.41***	***6.96***	***40.09***
Total	***1.25***	***13.93***	***6.73***	***7.36***	***8.75***	***61.82***	***11.48***	***19.92***	***58.46***

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 *projection consistency* for contribution problems									
PlanGraph	***40.0***	***57.5***	***44.1***	***56.1***	***99.08***	***40.9***	***204.9***	***86.3***	***369.8***
Cliques	***178.1***	***32.5***	***122.6***	***158.3***	***46.77***	***22.3***	***123.9***	***331.9***	***236.4***
Extraction	***12.2***	***12.3***	***29.1***	***37.5***	***49.95***	***376.7***	***121.5***	***53.9***	***992.4***
Total	***230.5***	***102.4***	***195.9***	***252.0***	***195.80***	***440.0***	***450.4***	***472.2***	***1598.7***

All the planning problems which were used for our experiments are available at the web site: http://ktiml.mff.cuni.cz/~surynek/research/iicai2007/. The planning domains are the same as that used for empirical tests in [31]. 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 [31]. 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 table 1.

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 by the gcc compiler version 3.4.3 with maximum optimization for the target machine (-O3 -mtune=opteron). No parallel execution was used.

4.4 Analysis of Experimental Results

The version of the GraphPlan algorithm based on contribution problems and maintaining projection consistency brings significant improvement on hard problems compared to the standard version. Notice that the version of the algorithm with projection consistency must perform clique decompositions before the search can proceed. Although the clique decompositions represent an overhead on easy problems the improvement in plan extraction with projection consistency overrides this disadvantage on hard problems.

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. 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 problems of finding supports does not represent any significant improvement (for example Hanoi tower problem 07 and 08).

5 Application in Solving Difficult SAT Instances

The proposed contribution problem and projection consistency can be also applied in solving Boolean formula satisfaction problems. We will formally describe details of the process of SAT problem reformulation in order to apply projection consistency in this section.

An excellent performance breakthrough was done in solving SAT problems over past years. Thanks to new algorithms and implementation techniques focused on real life SAT problems many of the today's benchmark problems [22, 27] are solved by state-of-the-art solvers [14, 16, 17] in time proportional to the size of the input. It seems that the difficulty of many SAT benchmark problems consists in their size only. Lot of smaller benchmark problems are solved in real-time by today's state-of-the-art SAT solvers. The observation that can be deduced upon these facts is that there is almost no chance to compete with best SAT solvers by own newly written SAT solver on these problems. That is why we are concentrating on difficult instances of SAT problems only, where the word difficult here means difficult for today's state-of-the-art SAT solvers.

A very valuable set of difficult problems was collected by Aloul [2]. Although these problems are small in length of the input formula they are difficult to be answered. The detailed discussion about hardness of these problems is provided in [3]. However one of the aspects is that these problems are mostly unsatisfiable (and this fact is well hidden in the problem). The solver cannot guess a solution using its advanced techniques and heuristics in such case and must really perform some search in order to prove that there is no solution. In the case of positive answer the satisfying valuation of variables serves a witness (of small size) certifying existence of at least one solution. If the solver obtains a witness, its task is complete. In contrast to this, there is no such small witness in the case of negative answer so the search must be performed.

Let $B = \wedge_{i=1}^{n} \vee_{j=1}^{m_i} x_j^i$ be the input Boolean formula in CNF form, where x_j^i is a literal (variable or its negation) for all possible i and j. A sub-formula $\vee_{j=1}^{m_i} x_j^i$ of the input formula B for every possible i is called a *clause*. The ith clause of the formula B will be denoted as CL_i in the following paragraphs. As it was mentioned in the introduction, the basic idea of the SAT problem reformulation consists in viewing the input formula as an undirected graph with vertices and edges in which the internal structure of the formula is captured in some way. To be concrete the graph will capture pairs of conflicting literals and will be constructed in several stages.

5.1 Inference of Conflicting Literals

We construct an instance of the contribution problem from the input formula. The instance is constructed in stages. We start by the construction of an undirected graph $G_B^1 = (V_B^1, E_B^1)$ which will represent trivially conflicting literals. The graph will be called a *trivial contribution graph*. The graph G_B^1 then undergoes further inference process by which additional conflicts will be inferred.

Construction of the trivial contribution graph is simple. A vertex is introduced into the graph G_B^1 for each literal occurring in the formula B, that is $V_B^1 = \bigcup_{i=1}^{n} \bigcup_{j=1}^{m_i} x_j^i$. The construction of the set of edges E_B^1 is also straightforward. An edge $\{x_j^i, x_l^k\}$ is introduced into the graph G_B^1 if the literals x_j^i are x_l^k are trivially conflicting, that is if $(x_j^i = b \wedge x_l^k = \neg b) \vee (x_j^i = \neg b \wedge x_l^k = b)$ for some Boolean variable b.

The resulting graph is evidently sparse, since there are edges only between literals of the same variable. As it is not a good starting point for our method, further inference mechanism for discovering more conflicting pairs of literals (more edges for the graph) must be applied. This further inference mechanism takes the already constructed graph G_B^1 and augments it by adding new edges, the result of this stage is a *final contribution graph* $G_B^2 = (V_B^2, E_B^2)$.

The process of construction of the graph G_B^2 exploits techniques known from standard SAT resolution approaches and from *constraint programming* [12] - *unit propagation* [13], *arc-consistency* (*AC*) [24] and *singleton arc-consistency* (*SAC*) [7]. Before proceeding with the construction of the graph G_B^2 let us recall these notions. While doing this we will adapt notations of these concepts slightly for the SAT domain to prepare them for our purposes. The following definitions assume the input formula B in the CNF form and a corresponding contribution graph G_B (for example the graph G_B^1 expressing trivial conflicts).

Definition 15 (Arc-consistency in SAT instance with respect to the contribution graph). Consider two clauses $CL_i = \vee_{j=1}^{m_i} x_j^i$ and $CL_k = \vee_{l=1}^{m_k} x_l^k$ for $i,k \in \{1,2,\dots,n\}$ of the formula B. A literal x_j^i ($j \in \{1,2,\dots,m_i\}$) from the clause CL_i is *supported* by the clause CL_k with respect to the given contribution G_B if there exists a literal x_l^k ($l \in \{1,2,\dots,m_k\}$) from the clause CL_k, such that the literals x_j^i and x_l^k are not in a conflict with respect to the graph G_B. An ordered pair of clauses (CL_i, CL_k)

of the formula B is called an *arc* in this context. An arc (CL_i, CL_k) for some $i,k \in \{1,2,\ldots,n\}$ is *consistent* (*or arc-consistent*) with respect to the contribution graph G_B if all the literals of the clause CL_i are supported by the clause CL_k with respect to the contribution graph G_B. The formula B is called *arc-consistent* with respect to the contribution graph G_B if all the arcs (CL_i, CL_k) for all $i,k = 1,2,\ldots,n$ are arc-consistent with respect to the contribution graph G_B. □

Definition 16 (Singleton arc-consistency in SAT instance with respect to the contribution graph). A literal x_l^k ($l \in \{1,2,\ldots,m_k\}$) from a clause CL_k for $k \in \{1,2,\ldots,n\}$ of the formula B is *singleton arc-consistent* with respect to the given contribution graph G_B if the formula obtained from B by replacing the clause CL_k by the literal x_l^k (the resulting formula is $(\wedge_{i=1}^{k-1} \vee_{j=1}^{m_i} x_j^i) \wedge x_l^k \wedge (\wedge_{i=k+1}^{n} \vee_{j=1}^{m_i} x_j^i)$) is arc-consistent with respect to the contribution graph G_B. □

Unsupported literals in the formula modified by replacing the clause CL_k by the literal x_l^k are in conflict with the literal x_l^k. This is quite intuitive, the selection of the literal x_l^k to be assigned the value *true* rules out some other literals. Hence these literals are in conflict with the selected literal x_l^k. Having singleton arc-consistency we are ready to infer new edges for the contribution graph.

The final contribution graph G_B^2 is constructed from the trivial contribution graph G_B^1 in the following way. Initially the graph G_B^2 is same as the graph G_B^1, that is we start with the initialization $V_B^2 \leftarrow V_B^1$ and $E_B^2 \leftarrow E_B^1$. Then for every literal corresponding to the vertex $v \in V_B^2$ singleton arc-consistency with respect to the trivial contribution graph G_B^1 is enforced. If the consistency discovers some unsupported literals, say literals $z_1, z_2, \ldots, z_m$, edges connecting vertices of y and z_i are added into the set of edges E_B^2 for all $i = 1,2,\ldots,m$.

Finally, if a literal corresponding to the vertex $v \in V_B^1$ occur in the clause $\vee_{j=1}^{m_i} x_j^i$, the symbol i is added to the set of contributions $c(v)$. The objective in the constructed contribution problem is represented by the set of integers $\{1,2,\ldots,n\}$.

Having the contribution problem for the input Boolean formula we are ready to apply projection consistency to reason about the problem.

5.2 Experimental Results for SAT Problems

We chose three state-of-the-art SAT solvers for comparison with our reformulation method. The SAT solvers of our choice were zChaff [16], HaifaSAT [17] and MiniSAT [14] (we used the latest available versions to the time of writing this paper). Our choice was guided by the results of several last SAT competitions [22, 27] in which these solvers numbered among winners. The secondary guidance was that complete source code (in C/C++) for all these solvers is available on web pages of their authors. As we implemented our method in C++ too, this fact allowed us to compile all source codes by the same compiler with the same optimization options which guarantees more equitable conditions for comparison (complete source code implementing our method in C++ can be found at the web page: http://ktiml.mff.cuni.cz/~surynek/ software/ssat/ssat.html).

Table 2. Experimental comparison of three SAT solvers over selected difficult benchmark SAT instances. We used the timeout of 10.0 minutes (600.00 seconds) for all the tests.

Instance	**Satisfiable**	**Number of variables / number of clauses**	**MiniSAT (seconds)**	**zChaff (seconds)**	**HaifaSAT (second)**
chnl10_11	unsat	220/1122	**34.30**	**7.54**	*> 600.00*
chnl10_12	unsat	240/1344	**101.81**	**9.11**	*> 600.00*
chnl10_13	unsat	260/1586	**200.30**	**11.47**	*> 600.00*
chnl11_12	unsat	264/1476	*> 600.00*	**33.49**	*> 600.00*
chnl11_13	unsat	286/1472	*> 600.00*	**187.08**	*> 600.00*
chnl11_20	unsat	440/4220	*> 600.00*	**329.57**	*> 600.00*
hole6	unsat	42/133	**0.01**	**0.01**	**0.01**
hole7	unsat	56/204	**0.09**	**0.04**	**0.02**
hole8	unsat	72/297	**0.49**	**0.23**	**0.94**
hole9	unsat	90/415	**3.64**	**1.46**	**478.16**
hole10	unsat	110/561	**39.24**	**7.53**	*> 600.00*
hole11	unsat	132/738	*> 600.00*	**32.36**	*> 600.00*
hole12	unsat	156/949	*> 600.00*	**372.18**	*> 600.00*
fpga10_11	unsat	220/1122	**44.77**	**12.58**	*> 600.00*
fpga10_12	unsat	240/1344	**119.26**	**33.82**	*> 600.00*
fpga10_13	unsat	260/1586	**362.24**	**76.15**	*> 600.00*
fpga10_15	unsat	300/2130	*> 600.00*	**274.84**	*> 600.00*
fpga10_20	unsat	400/3840	*> 600.00*	**546.00**	*> 600.00*
fpga11_12	unsat	264/1476	*> 600.00*	**55.70**	*> 600.00*
fpga11_13	unsat	286/1742	*> 600.00*	**237.54**	*> 600.00*
fpga11_14	unsat	308/2030	*> 600.00*	*> 600.00*	*> 600.00*
fpga11_15	unsat	330/2340	*> 600.00*	*> 600.00*	*> 600.00*
fpga11_20	unsat	440/4220	*> 600.00*	*> 600.00*	*> 600.00*

Table 3. Experimental comparison of three SAT solvers with the method using projection-consistency over selected difficult benchmark SAT instances. Again timeout of 10.0 minutes (600.00 seconds) for all the tests was used.

Instance	**Decided by projection consistency**	**Cliques (count x size)**	**Decision (seconds)**	**Speedup ratio w.r.t. MiniSAT**	**Speedup ratio w.r.t zChaff**	**Speedup ratio w.r.t HaifaSAT**
chnl10_11	yes	20 x 11	**0.43**	**79.76**	**17.53**	*> 1395.34*
chnl10_12	yes	20 x 12	**0.60**	**169.68**	**8.51**	*> 1000.00*
chnl10_13	yes	20 x 13	**0.78**	**256.79**	**14.70**	*> 769.23*
chnl11_12	yes	22 x 12	**0.70**	*> 857.14*	**47.84**	*> 857.14*
chnl11_13	yes	22 x 13	**0.92**	*> 652.17*	**203.34**	*> 652.17*
chnl11_20	yes	22 x 20	**5.74**	*> 104.42*	**57.41**	*> 104.42*
hole6	yes	6 x 7	**0.01**	**1.0**	**1.0**	**1.0**
hole7	yes	7 x 8	**0.02**	**4.5**	**2.0**	**1.0**
hole8	yes	8 x 9	**0.04**	**12.25**	**5.75**	**23.5**
hole9	yes	9 x 10	**0.08**	**45.5**	**18.25**	**5977.00**
hole10	yes	10 x 11	**0.13**	**301.84**	**57.92**	*> 4615.38*
hole11	yes	11 x 12	**0.20**	*> 3000.00*	**161.8**	*> 3000.00*
hole12	yes	12 x 13	**0.30**	*> 2000.00*	**1240.6**	*> 2000.00*
fpga10_11	yes	20 x 11	**0.46**	**97.32**	**27.34**	*> 1304.34*
fpga10_12	yes	20 x 12	**0.64**	**186.34**	**52.84**	*> 937.50*
fpga10_13	yes	20 x 13	**0.84**	**431.23**	**90.65**	*> 714.28*
fpga10_15	yes	20 x 15	**1.39**	*> 431.65*	**197.72**	*> 431.65*
fpga10_20	yes	20 x 20	**4.72**	*> 127.11*	**115.67**	*> 127.11*
fpga11_12	yes	22 x 12	**0.76**	*> 789.47*	**73.28**	*> 789.47*
fpga11_13	yes	22 x 13	**1.01**	*> 594.05*	**235.18**	*> 594.05*
fpga11_14	yes	22 x 14	**1.30**	*> 461.53*	*> 461.53*	*> 461.53*
fpga11_15	yes	22 x 15	**1.67**	*> 359.28*	*> 359.28*	*> 359.28*
fpga11_20	yes	22 x 20	**5.96**	*> 100.67*	*> 100.67*	*> 100.67*

Again, all the tests were run on a machine with two AMD Opteron 242 processors (1600 MHz) with 1GB of memory under Mandriva Linux 10.2. Our method as well as the listed SAT solvers were compiled by the gcc compiler version 3.4.3 with options provided maximum optimization for the target testing machine (-O3 -mtune=opteron). As well as in the planning tests, no parallel processing was used.

The testing set consisted of several difficult unsatisfiable SAT instances. This set of benchmark problems was collected by Aloul [2] and is provided at his research web page. Let us briefly say that hard SAT instances often encodes so called pigeon hole principle. That is, the problem asks whether it is possible to place $n+1$ pigeons in n holes without two pigeons being in the same hole.

For each benchmark SAT instance we measured the overall time necessary to decide its satisfiability. The results are shown in table 2 and table 3. The speedup obtained by using our method compared to a selected SAT solver is also shown.

5.3 Analysis of Experimental Results

As it is evident from our experimentation the proposed method brings significant improvement in term of time necessary for decision of the selected difficult benchmark problems. The improvements are in orders of magnitudes with respect to all tested state-of-the-art SAT solvers. It seems that the improvement on selected benchmarks is exponential with respect to the best tested SAT solver. The conclusion is that there is still room to improve SAT solvers. However the domain of the improvement is more likely over difficult instances of SAT problems which are typically unsatisfiable.

We also tested our approach on SAT instances which are easy for the tested SAT solvers. These instances are mostly satisfiable and our method based on projection consistency does not decide the instance itself without search. In such situations our method does not provide competitive results. So the question may be now what to do when we have a new problem of unknown difficulty. That is shall we use our method or the SAT solver of our choice directly? It is an open question now. One possible solution is to run both the method based on projection consistency and the SAT solver in parallel (on a machine with more than one processor we may obtain an exponential speedup; on a machine with only one processor we may obtain an exponential speedup at the expense of constant slowdown). The second and more promising solution is to integrate projection consistency into the SAT solver directly.

6 Related Works

Our work is a generalization of approaches proposed by Surynek in [29, 30]. Here we provide a unified formalization of proposed method and show the method in a more general perspective.

The idea of exploiting structural information for solving problems is not new. There is lot of works concerning this topic. Many of these works are dealing with methods for breaking symmetries [3, 5, 11]. We share the goal with these methods, which is to reduce the search space. However we differ in the way how we are doing this. We are rather trying to infer what would happen if the search over the problem proceeds in some way. And if that direction seems to be unpromising the corresponding part of the search space is skipped. Symmetry breaking methods are rather trying no to do the same work twice (or more times) by clever a transformation of the original problem.

Next, let us note that even the detection of cliques in the structure of the problem is not new. A work dealing with a consistency based on cliques of inequalities was published by Sqalli and Freuder [28]. They use information about cliques to reach more global reasoning about the problem. Another work dealing with the similar ideas is [15] in which the authors use graph structure of the problem to transform it into another formulation based on global constraints which provide stronger propagation than the original formulation.

GraphPlan algorithm and planning graphs were intensively studied after their first introduction in [8]. Although GraphPlan algorithm is no longer considered to be state-of-the-art, the planning graphs still provides a good structure for reasoning about step-optimal planning.

Kambhampati's successful idea to formulate plan extraction from planning graph as CSP is presented in [19]. He evaluates the use of various constraint programming techniques and its impact on the effectiveness of plan extraction. Several extensions of expressivity of planning graphs are described in [20]. 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 [23] 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.

7 Conclusions and Future Work

We proposed a novel consistency technique which we called projection consistency. The technique is designed to prune the search space by early determining unpromising decisions. The idea behind projection consistency is to disentangle the structural information hidden in the problem formulation. This is done by viewing the problem as a graph in which complete sub-graphs are found by the greedy algorithm.

We show two areas of application of our method in artificial intelligence - solving planning problems and Boolean formula satisfaction. Namely we use projection consistency as technique to prune the search space during extraction of plans from planning graphs by the GraphPlan-style algorithm and as a technique which helps to decide difficult SAT instances. In both areas of applications we empirically showed the usefulness of the method. To be concrete, the improvement in terms of overall solving time is in order of magnitudes when projection consistency is used in some way. Although the demonstrated applications cover only a limited part of problem solving field in artificial intelligence, we hope that it may be useful in general.

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 projection goals. But it is also possible to do it by slight modification of the definition of the supported vertex. Instead of the expression $\sum_{j=1, j\neq i}^{n} c(C_j, p) \geq |p - c(v)|$ in the definition 5 one can use $\sum_{j=1, j\neq i}^{n} c(C_j, p - c(v)) < |p - c(v)|$. Unfortunately this change causes that monotonicity (proposition 2) no longer holds. Hence the complexity of propagation algorithm increases. The solution may be a better propagation algorithm.

For future we plan to further tune the method to be able to cope better with the problems having few edges in their contribution graphs. This is the case of certain difficult SAT instances. This may be done by some alternative consistency technique instead of singleton arc-consistency.

Finally, the interesting research direction is some kind of a combination of existing symmetry breaking methods and the proposed projection consistency. Since our method is a preprocessor for SAT instance in fact, integration with existing SAT preprocessors (such as that used by MiniSAT [14]) would be also interesting.

References

1. Ai-Chang, M., et al.: MAPGEN: Mixed-Initiative Planning and Scheduling for the Mars Exploration Rover Mission. IEEE Intelligent Systems 19(1), 8-12, IEEE Press, 2004.

2. Aloul, F. A.: Fadi Aloul's Home Page - SAT Benchmarks. Personal Web Page. http://www.eecs.umich.edu/~faloul/benchmarks.html, University of Michigan, USA, (March 2007).

3. Aloul, F. A., Ramani, A., Markov, I. L., Sakallah, K. A.: Solving Difficult SAT Instances in the Presence of Symmetry. Proceedings of the 39th Design Automation Conference (DAC-2002), 731-736, USA, ACM Press, 2002.

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

5. Benhamou, B., Sais, L.: Tractability through Symmetries in Propositional Calculus. Journal of Automated Reasoning, volume 12-1, 89-102, Springer-Verlag, 1994.

6. Bernard, D. et al.: Remote Agent Experiment. Deep Space 1 Technology Validation Report. NASA Ames and JPL report, 1998.

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

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

9. 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.

10. Cook, S. A.: The Complexity of Theorem Proving Procedures. Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, 151-158, USA, ACM Press, 1971.

11. Crawford, J. M., Ginsberg, M. L., Luks, E. M., Roy, A.: Symmetry-Breaking Predicates for Search Problems. Proceedings of the 5th International Conference on Principles of Knowledge Representation and Reasoning (KR-96), 148-159, Morgan Kaufmann, 1996.

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

13. Dowling, W., Gallier, J.: Linear-time algorithms for testing the satisfiability of propositional Horn formulae. Journal of Logic Programming, 1(3), 267-284, Elsevier, 1984.

14. Eén, N., Sörensson, N.: The MiniSat Page. Research Web Page. http://www.cs.chalmers.se/ Cs/Research/FormalMethods/MiniSat/Main.html, Chalmers University, Sweden, (March 2007).

15. Frisch, A. M., Miguel, I., Walsh, T.: CGRASS: A System for Transforming Constraint Satisfaction Problems. Barry O'Sullivan (Editor): Recent Advances in Constraints, 15-30, LNCS 2627, Springer-Verlag, 2003.

16. Fu, Z., Marhajan, Y., Malik, S.: zChaff. Research Web Page. http://www.princeton.edu/ ~chaff/ zchaff.html, Princeton University, USA, (March 2007).

17. Gershman, R., Strichman, O.: HaifaSat – a new robust SAT solver. Research Web Page. http://www.cs.technion.ac.il/~gershman/HaifaSat.htm, Technion Haifa, Israel, (March 2007).

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

19. 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.

20. 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.

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

22. Le Berre, D., Simon, L.: SAT Competition 2005. Competition Web Page, http://www.satcompetition.org/2005/, Scotland, (March 2007).

23. 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.

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

25. Nam, G. J., Sakallah, K. A., Rutenbar, R.: A New FPGA Detailed Routing Approach via Search-Based Boolean Satisfiability. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, volume 21-6, 674-684, IEEE Press, 2002.

26. 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.

27. Sinz, C.: SAT-Race 2006. Competition Web Page, http://fmv.jku.at/sat-race-2006/, USA, (March 2007).

28. Sqalli, M. H., Freuder, E. C.: Inference-Based Constraint Satisfaction Supports Explanation. Proceedings of the 13th National Conference on Artificial Intelligence and 8th Innovative Applications of Artificial Intelligence Conference (AAAI-96 / IAAI-96), 318-325, AAAI Press / The MIT Press, 1996.

29. Surynek, P.: Projection Global Consistency: An Application in AI Planning. Technical report, ITI Series, 2007-333, http://iti.mff.cuni.cz/series, Charles University, Prague, Czech Republic, 2007.

30. Surynek, P.: Solving Difficult SAT Instances Using Greedy Clique Decomposition. Accepted to 7th Symposium on Abstraction, Reformulation, and Approximation (SARA-2007), Canada, 2007.

31. 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.

32. Velev, M. N., Bryant, R. E.: Effective Use of Boolean Satisfiability Procedures in the For-mal Verification of Superscalar and VLIW Microprocessors. Journal of Symbolic Computation (JSC), volume 35-2, 73-106, Elsevier, 2003.