Disertační práce byla vypracována v rámci doktorského studia, které uchazeč absolvoval na Katedře teoretické informatiky a matematické logiky na Matematicko fyzikální fakultě Univerzity Karlovy.

Uchazeč: RNDr. Pavel Surynek

Školitel: Doc. RNDr. Roman Barták, Ph.D.
Katedra teoretické informatiky a matematické logiky
Matematicko-fyzikální fakulta, Univerzita Karlova v Praze
Malostranské náměstí 2/25, 118 00 Praha 1, Česká republika

Školící pracoviště: Katedra teoretické informatiky a matematické logiky
Matematicko-fyzikální fakulta, Univerzita Karlova v Praze
Malostranské náměstí 2/25, 118 00 Praha 1, Česká republika

Oponenti: Prof. RNDr. Olga Štěpánková, CSc.
Katedra kybernetiky
Fakulta elektrotechnická, České vysoké učení technické v Praze
Technická 2, 166 27 Praha 6, Česká republika

Prof. RNDr. Peter Vojtáš, DrSc.
Katedra softwarového inženýrství
Matematicko-fyzikální fakulta, Univerzita Karlova v Praze
Malostranské náměstí 2/25, 118 00 Praha 1, Česká republika

Autoreferát byl rozeslán dne: .....

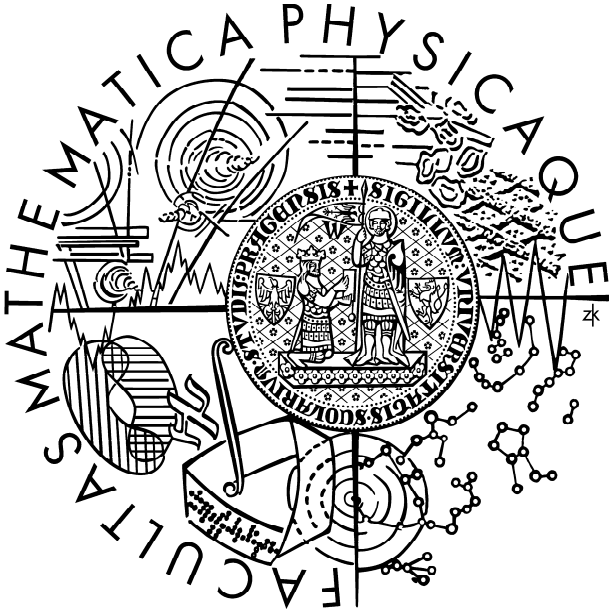
Obhajoba se koná dne ..... v ..... hod. před komisí pro obhajoby disertačních prací oboru I-1 na MFF UK, Malostranské náměstí 2/25, 118 00 Praha 1, v posluchárně .....

S doktorskou disertační prací je možno se seznámit na studijním oddělení pro doktorské studium MFF UK, Ke Karlovu 3, 120 00 Praha 2.

Prof. RNDr. Petr Štěpánek, DrSc.
předseda rady doktorského studijního oboru I-1 teoretická informatika
Katedra teoretické informatiky a matematické logiky
Matematicko-fyzikální fakulta, Univerzita Karlova v Praze
Malostranské náměstí 2/25, 118 00 Praha 1, Česká republika

CHARLES UNIVERSITY IN PRAGUE
FACULTY OF MATHEMATICS AND PHYSICS

ABSTRACT OF THE DOCTORAL DISSERTATION



RNDr. PAVEL SURYNEK

CONSTRAINT PROGRAMMING IN PLANNING

ADVISER: DOC. RNDr. ROMAN BARTÁK, Ph.D.

DEPARTMENT OF THEORETICAL COMPUTER SCIENCE AND
MATHEMATICAL LOGIC

PRAGUE 2008

1 Introduction

The primary topic of the thesis is an application of *constraint programming* (Dechter, 2003) in *artificial intelligence planning* (Ghallab *et al.*, 2004).

Planning problems represent an intensively studied research area of artificial intelligence. The planning problem is stated as a task of determining a sequence of actions for a certain agent (or a group of agents) whose execution achieves a given goal (set of propositions) from a given initial state (set of propositions). This sequence of actions is called a *plan* in this context. Due to their complexity, which is far beyond tractability, solving planning problems is an area still requiring innovations. Another important motivation for developing more powerful solving techniques for planning problems is their practical importance. The areas of application of planning technology range from industrial planning of manufacturing operations to planning of actions for autonomous planetary exploration agents.

From a wider point view, the research effort was not concentrated on planning problems only but also on improving search for a solution of a certain problem generally. Therefore, results achieved in the area of solving *satisfiability problems* (Davis and Putnam, 1960; Cook, 1971) are also presented in the thesis (a satisfiability problem is a task of finding a valuation of variables that satisfies a given logical formula). The satisfiability problems represent a challenge for research in artificial intelligence. They are also used as the main benchmark suit for measuring qualities of improvements in search techniques since large collections of benchmark problems are available. Again, the motivation for improving techniques for solving satisfiability problems is not only theoretical. Real life applications where satisfiability problems must be solved account for example automated software testing or microprocessor verification.

Let us adopt a simplified intuitive statement that one of the engineering goals of *artificial intelligence* is to build an intelligent agent that reasons and behaves rationally when it is solving problems (Russell and Norvig, 2003). The rational reasoning and behavior is defined as the reasoning and behavior typical for humans when they are solving problems. Solving of planning and satisfiability problems produces something that can be interpreted as rational reasoning and behavior. For example, acting according to plan which solves a certain planning problem looks like a rational behavior. From this point of view, the topics studied in this thesis contribute to building of such an intelligent agent. And hence contribute to one of the goals of artificial intelligence as a science.

The main source of inspirations for improvements in solving planning problems and satisfiability problems was another area of artificial intelligence - *constraint programming*. In particular, constraint programming methodology provides a concept of so called *consistency techniques* (or propagation/filtering techniques). Intuitively said, a consistency technique can be used to predict the impact of decisions made during the solving process on

future evolution of the search process. Thus the consistency technique reduces the number of candidates for solution that need to be tested (a series of decisions that does not lead to the solution is identified early).

2 Contributions

The contribution of the thesis to above topics of artificial intelligence consists in developing of new concepts inspired by constraint programming.

We focused on improving the solving technique for planning problems over so called *planning graphs* (Blum and Furst, 1997). Planning graphs and the related algorithm *GraphPlan* represent one of the most successful concepts for working with so called *concurrent plans* - plans that allow more than one action to be performed in a single time-step. We identified some weak points of the GraphPlan algorithm. Namely, the process how supporting actions for a certain goal are searched is very inefficient within the original algorithm (the problem itself is *NP*-complete). We improved this process by formulating the problem as a *constraint satisfaction problem* and solved it using maintaining a certain type of local consistency (*arc-consistency* - Mackworth, 1977).

The relatively successful application of local consistency was an inspiration to develop a more global and more specialized type of consistency for the problem. The new type of consistency is based on disentangling the structure of the problem hidden in its formulation. We visualized the problem as a graph. This uncovered that the graphical structure of the problem of finding supporting actions is far from random. The graph is typically formed by a small number of large complete sub-graphs (plus some additional edges). The consistency that utilizes this structure for solving the problem of supports is described in the thesis. This new technique is called *projection consistency*.

The concept of projection consistency was further improved. We found that it can be used to define a special class of the problem of finding supporting actions that can be solved in polynomial time. A special variant of the solving process for the problem of finding supports within the GraphPlan algorithm was proposed. The solving process utilizes the class of problems solvable in polynomial time. The experimental evaluation unexpectedly showed that this enhancement of GraphPlan algorithm becomes competitive with today's state-of-the-art planners on certain problems (this was unexpected because of a not well optimized testing implementation compared to fine tuned implementations of state-of-the-art planners).

Finally, we applied the experiences gained during experimentation with planning problems in solving Boolean satisfaction problems. We proposed a special preprocessing of satisfiability problems which helps the general solver to decide the problem. It is again based on disentangling the structure of the formulation of the problem - again the problem is interpreted as a graph. Contrary to problems of finding supporting actions the graphical

representation of satisfiability problems is unstructured and must be processed further to make the structures visible. The method for preprocessing satisfiability problems was called *clique consistency* since it is again based on complete sub-graph structures.

3 Arc-consistency in Planning Graphs

The planning graph (Blum and Furst, 1997) is a polynomial size data structure which encapsulates a necessary but insufficient condition for the reachability of a given goal. It can be used to answer the question whether it is possible to reach a certain goal in a certain time-step. An example of the plannig graph is shown in figure 1. The first algorithm based on *planning graphs* is *GraphPlan* (Blum and Furst, 1997).

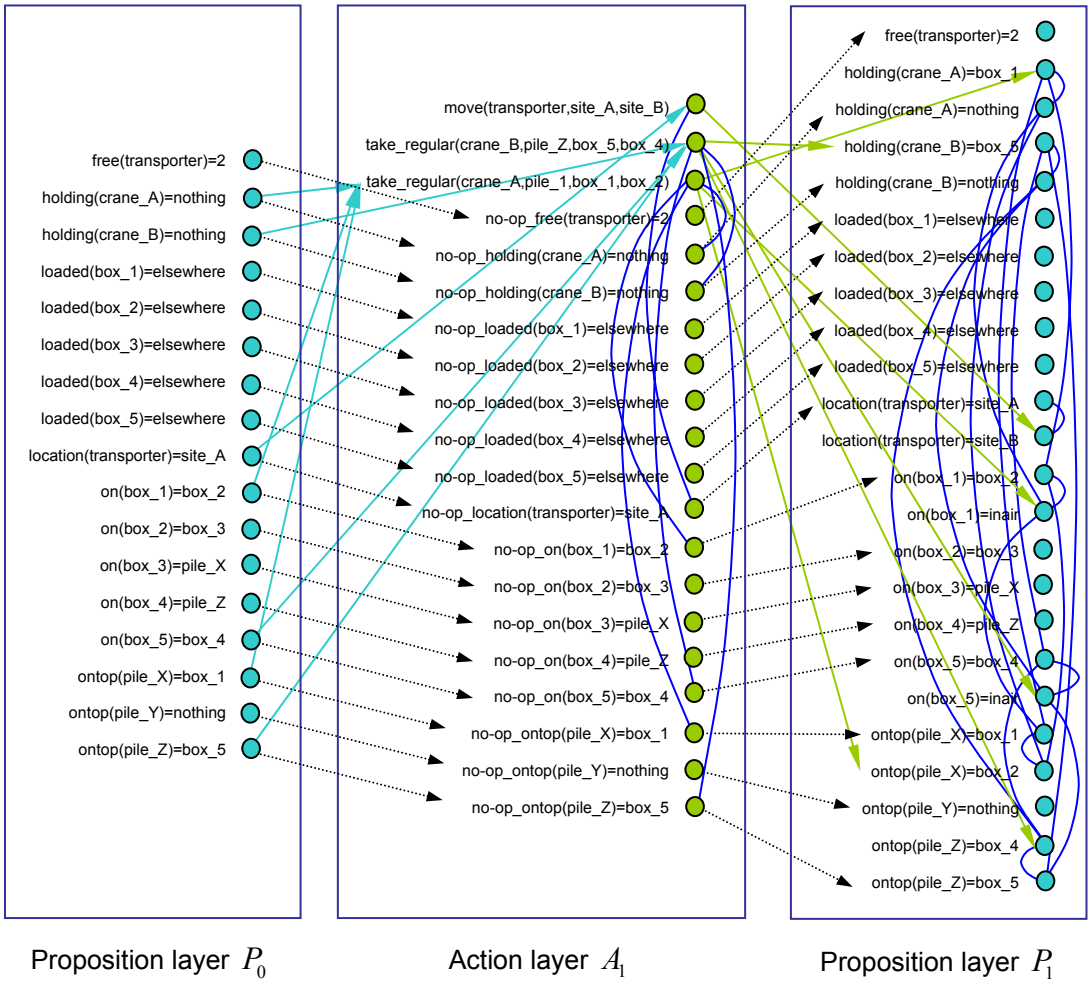


Figure 1. *STRUCTURE OF PLANNING GRAPH.* First three layers of the planning graph for the dock worker robots planning problem. Proposition and action mutexes are represented by blue arcs.

The planning graph is a structure consisting of alternating layers of two types - *action layers* and *proposition layers*. The *k*th action layer of the planning graph contains actions that can be reached in the *k*th time-step. The *k*th proposition layers contains proposition that can be reached in the *k*th time-step. Each layer is accompanied by the set of *mutexes*. A mutex is a pair of actions or a pair propositions. The mutex represents the property that its pair of actions (pair of propositions) cannot be performed together (cannot hold simultaneously).

The GraphPlan algorithm itself is based on state reachability analysis using the data structure of planning graphs. The algorithm consists of two interleaved phases. The first phase is represented by *incremental expansion* of the planning graph. The incremental expansion of the planning graph is just adding one action and one proposition layer respectively as new last two layers of the planning graph.

The second phase of the GraphPlan algorithm consists in an attempt to *extract a plan* from the planning graph. This phase is performed only if the given goal is reachable in the constructed planning graph. If the plan extraction is unsuccessful then the algorithm continues with the planning graph expansion phase. These two phases are repeated till the planning problem is solved or a termination condition is reached.

3.1 Constraint Programming

Constraint programming (*CP* - Dechter, 2003) provides a framework for modeling and solving a large variety of problems arising in combinatorial optimization, artificial intelligence, computer graphics, planning, and scheduling etc.

The central concept of constraint programming is a *constraint satisfaction problem (CSP)* - it consists of *variables* and of *constraints*. Each variable has assigned a finite *domain* of values for it. A constraint is an arbitrary relation over the domains of the variables in the scope of the constraint. Constraints represent relations between variables and restrict the values of the variables to allowed combinations only. To solve a given constraint satisfaction problem it is necessary to assign values to variables such that all the constraints are satisfied.

There is a wide variety of techniques for solving constraint satisfaction problems (Dechter, 2003). Techniques for solving CSPs range from *backtracking-based search* to *local search* (Baker, 1995; Harvey, 1995). In the thesis we are focusing on backtracking-based methods only. The reason is that these methods are able to prove non-existence of a solution (the search space is explored systematically; nothing is skipped) and this is a property we need in our applications of CP.

One of the most powerful techniques developed in constraint programming are so called *consistency techniques.* They are used for ruling out *inconsistent* values or the tuples of values from further consideration. If the value or the tuple of values is inconsistent then

it cannot be part of any solution of the problem. The main area of application of consistency techniques are general search algorithms for solving CSPs. The early removal of inconsistencies from the problem saves the time which the search algorithm would consume by unsuccessful attempts to assign inconsistent values to the variables otherwise.

The most widely used type of consistency is *arc-consistency* (Mackworth, 1977). It combines simplicity of implementation with the relatively strong pruning power. Arc-consistency is a technique that removes values (not tuples of values) from the domains of variables. Figure 2 shows enforcing arc-consistency over a single binary constraint. There is a variety of algorithms for enforcing arc-consistency - *AC-1*, *AC-2* (Waltz, 1975), *AC-3* (Mackworth, 1977), *AC-3.1* (Zhang and Yap, 2001), *AC-4* (Mohr and Henderson, 1986), *AC-6* (Bessière and Cordier, 1993), and *AC-2000/AC-2001* (Bessière and Régin, 2001).

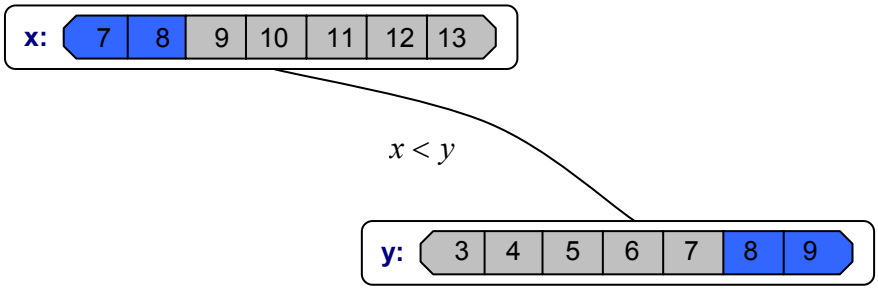


Figure 2. *ARC-CONSISTENCY OF A BINARY CONSTRAINT.* Values removed from the domains are depicted by the grey color.

Another successful technique from constraint programming is represented by so called *global constraints*. A global constraint is a specialized constraint modeling a specific sub-problem. The global constraint is a relation over a higher number of variables. It is accompanied with a specialized and efficient algorithm for enforcing certain type of consistency. An example of global constraint is *allDifferent* constraint (Régin, 1994). Its filtering procedure for enforcing consistency is based on algorithms for computing maximum network flows (Ahuja *et al.*, 1993).

3.2 Maintaining Arc-consistency within GraphPlan

We are focusing on improving of the plan extraction phase of the GraphPlan algorithm. By using concepts and techniques from *constraint programming* (Dechter, 2003) we improve the search for a plan. We are using maintaining *arc-consistency* (Mackworth, 1977) during extraction of a plan from the planning graph to reduce the search space.

We deal with the problem of finding a mutex-free set of actions in an action layer of the planning graph that together satisfy a certain goal (they support the goal). An example

of the *problem of finding supports* for a sub-goal is shown in figure 3. If we examine the course of execution of the GraphPlan algorithm, we can observe that typically huge numbers of sub-goals must be satisfied or proved to be unsatisfiable along the search for the global goal in the standard GraphPlan algorithm. The effectiveness of a method for solving the problem of finding supports has therefore a major impact on the performance of the planning algorithm as a whole. Unfortunately the problem of finding supports is NP-complete.

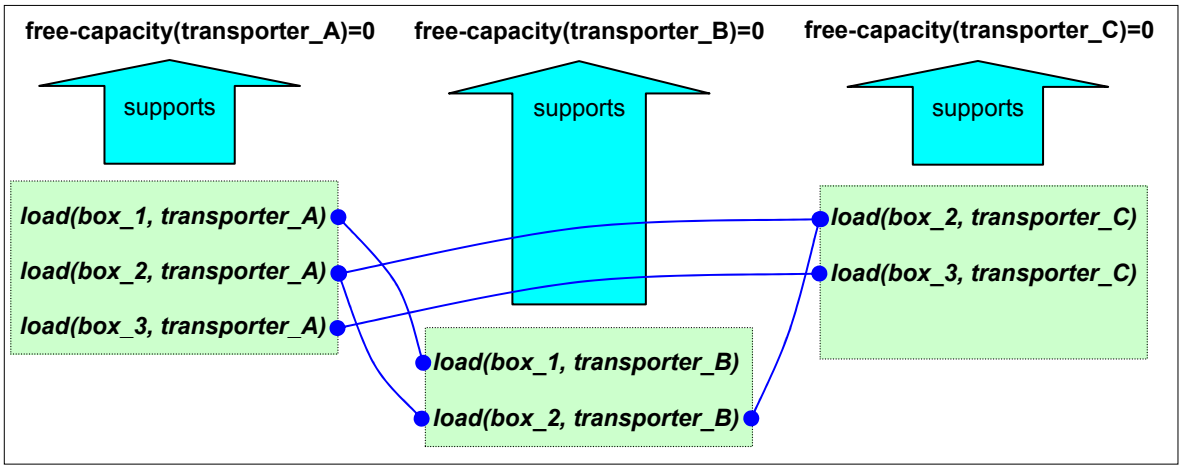


Figure 3. *PROBLEM OF FINDING SUPPORTS FOR A SUB-GOAL.* Example of a problem of finding supports for a sub-goal. Three transporters must be loaded. Each transporter has a capacity of one box. Mutexes are depicted as arcs between actions (mutexes are between actions loading the same box to different transporters).

Compared to the simple uninformed backtracking used to solve the problem of finding supporting actions for a sub-goal constraint programming provides more effective tools to solve similar combinatorial problems. We model the problem of finding supporting actions for a sub-goal as a CSP. Having this formulation, we use *arc-consistency* for pruning the search space during the search for supporting actions. The constraint model is built and solved whenever a sub-goal arises during the plan extraction phase.

The constraint model consists of two sub-models called *activity cluster* and *support cluster*. There is one special constraint connecting these two clusters. This special constraint controls propagation of changes between the clusters. The activity cluster is used for fast propagation after selection of an action. It provides a fast detection of actions mutually excluded with the currently selected action. The support cluster represents the main sub-model. It is richer than the activity cluster and arc-consistency can infer more in this cluster. The drawback of the support sub-model is that it is large and in practice enforcing arc-consistency in this sub-model takes non-trivial amount of time.

The uninformed backtracking within the plan extraction phase which solves the problem of finding supporting actions can be replaced by solving of the proposed constraint

model. To solve the proposed constraint model we use the standard chronological based backtracking (Dechter, 2003) augmented by various degrees of constraint propagation and certain type of dynamicity (the model is changed during search).

3.3 Variants of Constraint Propagation

We describe three variants of propagation scheme for solving the proposed constraint model in the thesis. The individual variants represent different levels of consistencies enforced in the constraint model. The variants are named: *variant A*, *variant B*, and *variant C*. The *variant A* represents the weakest level of consistency; on the other hand this variant propagates quickly. The *variant C* represents the strongest level of consistency but it is computationally most expensive. The *variant B* represents a compromise between *variant A* and *variant C*.

Propagation scheme of *variant A*: Arc-consistency in the activity cluster is enforced every time a variable in the activity cluster is assigned a value. The next step consists of propagation of the changes made in the activity cluster into the support cluster. This variant is functionally equivalent to *forward checking* (Dechter, 2003) in the support cluster (however this version is faster thanks to simplicity of the activity cluster). ●

Propagation scheme of *variant B*: In this propagation scheme we proceed similarly as in the *variant A*. Arc-consistency is enforced in the activity cluster and changes are propagated into the support cluster. This propagation is done in the same way as in the *variant A*. In addition to the *variant A*, changes in the support cluster are propagated back to the activity cluster. ●

Propagation scheme of *variant C*: This variant further evolves the previous variant. Now consistency is enforced in both clusters. Propagation in both directions between the clusters is done in the same way as in previous variants. ●

For uncovering the behavior of the process of solving constraint models connected with various constraint propagation variants we used an experimental evaluation. We performed a set of experiments to evaluate the contribution of the proposed constraint model and arc-consistency maintaining framework for solving the problems of finding supporting actions. Our experiments are targeted on comparison of the basic version of the GraphPlan algorithm with the enhanced versions of the GraphPlan algorithm which use the constraint model and a variant of solving technique (*variant A*, *variant B*, and *variant C*).

We implemented all the tested algorithms. All the tested algorithms were implemented in the C++ language (Stroustrup, 1986) and were compiled under identical conditions using gcc compiler version 3.4.3 (GNU Project, 2008) with options providing maximum optimization for the target testing machine (-O3 -mtune=opteron). The tests were run on a ma-

chine with two AMD Opteron 242 processors (1600 MHz) with 1GB of memory under Mandriva Linux 10.2 (Mandriva, 2008).

We used three types of planning environments - dock worker robots environment (Ghallab *et al.*, 2004), refueling planes environment (which is original) and towers of Hanoi planning environment. Several instances of planning problems of various difficulties from each proposed planning environment were used. All the problems used in our experimentation were solvable (that is, the algorithm terminated with the answer that solution exists and retuned one solution). Along the execution of experimental tests, we collected variety of statistical data characterizing performance of the individual algorithmic techniques.

The comparison of the overall solving time of the standard GraphPlan algorithm and the enhanced versions based on maintaining arc-consistency is shown in figure 4.

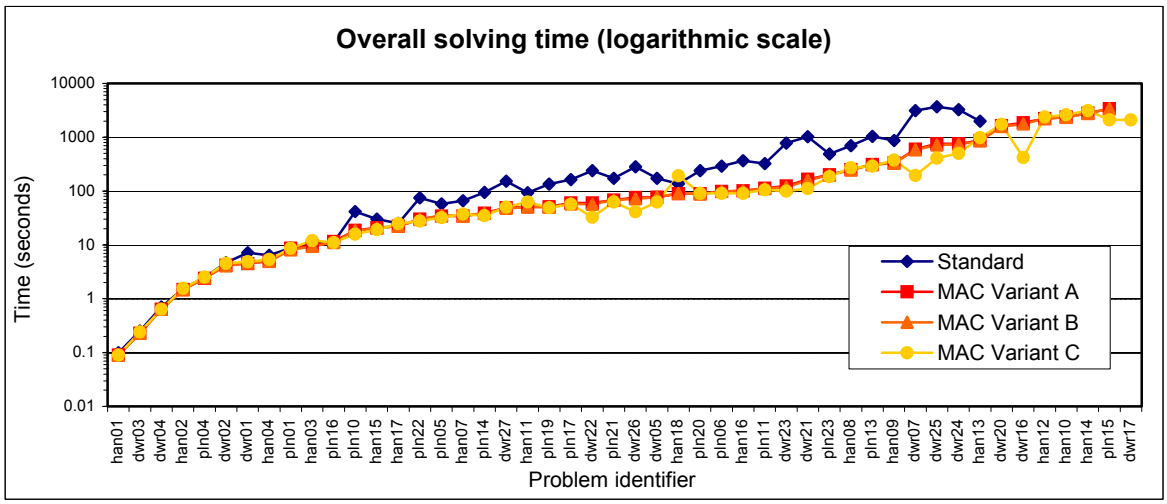


Figure 4. *COMPARISON OF OVERALL SOLVING TIMES (LOGARITHMIC SCALE) - (STD, VARA, VARB, VARC).* Comparison of the overall solving time of the standard GraphPlan algorithm and enhanced versions which use maintaining arc-consistency for solving the problem of finding supports (standard version and variants *A*,*B*, and *C* of the propagation scheme are compared). Problems on the horizontal axis are listed in the ascending order according to the time consumed by *variant A*. Time limit of 1 hour for each problem is used.

The standard GraphPlan is compared with arc-consistency propagation variants *A*, *B*, and *C*. Problems are ordered along the horizontal line. Each problem is identified by a prefix ("dwr" for Dock Worker Robots planning environment, "han" for Towers of Hanoi planning environment, and "pln" for Refueling Planes planning environment) followed by the number of the problem. Problems are listed along the horizontal axis in the ascending order according to the solving time using *variant A* propagation scheme (this ordering allows to depict deviation of the standard GraphPlan and the *variant C* propagation scheme). The time limit of 1 hour is used.

The graph show that GraphPlan enhanced by any of the variants of propagation for maintaining arc-consistency in the constraint model of the problem of finding supporting

actions outperforms the standard version in terms of overall problem solving time (for example the improvement is up to 1600% when we compare standard GraphPlan and the *variant C* on the problem dwr07). Notice that we improved only the plan extraction phase. Phase of building planning graph remains the same, so the more time is spent in plan extraction phase the better is the improvement.

The comparison of the time spent in plan extraction phases (that is we do not account time spent by building planning graph) is shown in figure 5. The ordering of problems along the horizontal axis in figure 5 is the same as in figures 4. So, the portion of time spent by building planning graphs and in plan extraction phase can be observed. The differences among individual methods are more expressed since the graph building time (which is the same for all the methods) is ruled out.

Experiments showed that there is a significant improvement of the time of extraction phase as well as in overall solving time in comparison with standard GraphPlan when the constraint model with maintaining arc-consistency is used.

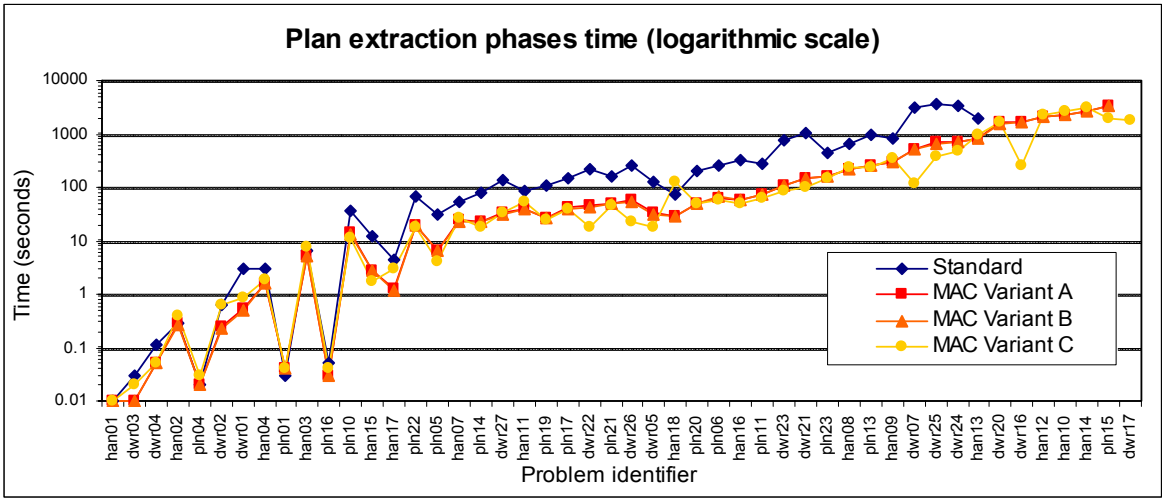


Figure 5. *COMPARISON OF PLAN EXTRACTION PHASES TIMES (LOGARITHMIC SCALE) - (STD, VARA, VARB, VARC)*. Comparison of the plan extraction time of the standard GraphPlan algorithm and enhanced versions which use maintaining arc-consistency for solving the problem of finding supports (standard version and variants *A*,*B*, and *C* of the propagation scheme are compared). Problems on the horizontal axis are listed in the ascending order according to the overall solving time consumed by *variant A* (same ordering as in figure 4). Time limit of 1 hour for each problem is used again.

4 Projection Consistency

Our new global consistency is based on discovering the hidden structural information in the constraint model. We are viewing the given problem as a graph in which we search for structures. We found that complete sub-graphs represent the valuable structures with re-

spect to the task of solving the problem of finding supports. We call the global constraint and the associated consistency based on structural decomposition of the problem *projection global constraint* and *projection consistency* respectively. The name of the concept was chosen according to consistency checking with respect to a sub-problem - we *project* the consistency into the smaller space.

Projection consistency was motivated by observation of layers of the planning graph when they are visualized as graphs (actions are vertices and mutexes are edges) - let us call such graphs *mutex graphs*. These mutex graphs embody high density of edges on majority of testing planning problems (however our method works with sparse mutex graphs as well). We consider that the most important factor is that certain sets of actions are intrinsically pair-wise mutually excluded. Such set of actions induces a complete sub-graph - a clique - within the mutex graph.

The knowledge of clique decomposition of the mutex graph allows us to do quite strong reasoning since at most one action from a clique can be selected. The second part of the idea of projection constraint is to take a subset of propositions of a given sub-goal and to calculate how a certain clique of actions contributes to satisfaction of the subset of propositions. This reasoning can be used to discover that some actions within a certain clique do not contribute enough to the selected subset 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 a smaller number of alternatives must be considered.

4.1 Preprocessing Step: Clique Decomposition

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.

Unfortunately, the problem of finding the optimal clique cover with respect to some criterion is typically *NP*-complete (Golumbic, 1980). 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 introduce projection constraint. Our experiments showed that a simple greedy algorithm provides satisfactory results. Its complexity is polynomial in the size of the input graph which is acceptable for preprocessing step.

The example of a mutex graph of the action layer of the planning graph and its clique decomposition by the greedy algorithm is shown in figure 6. It is the real-life mutex graph which was extracted from the action layer from the planning graph for a Dock Worker Robots problem.

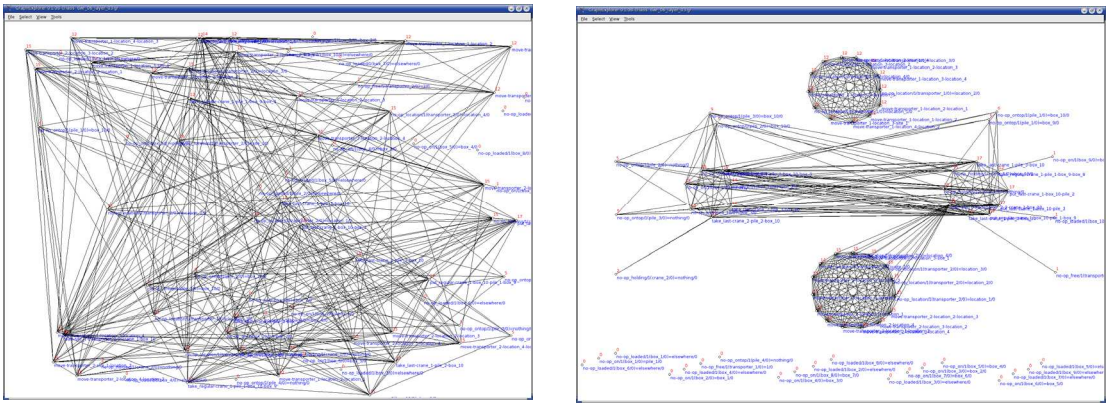


Figure 6. *ILLUSTRATION OF MUTEX GRAPH AND CLIQUE DECOMPOSITION.* The left part of the figure is an illustration of a mutex graph obtained from the action layer of the planning graph for a slightly more complex Dock Worker Robots problem. Vertices represent actions and edges represent action mutexes. The vertices representing actions are placed randomly in the window. The right part of the figure shows an illustration of clique decomposition of the graph on the left. The individual cliques of actions are depicted by grouping of vertices into clusters.

4.2 Counting Derived from Clique Decomposition

Projection consistency is defined over the decomposition of a mutex graph for a sub-goal p of the main goal. The sub-goal p is called a *projection goal* in this context. The projection goal represents some kind of a parameter for the new consistency. The new consistency has different results for different projection goals.

The fact that at most one action from a clique can be selected allows us to use special counting derived from the clique decomposition. For instance we can calculate the maximum number of propositions within the projection goal that can be satisfied by actions from the clique. We call this value the *contribution of the clique* to the given projection goal p. An action cannot participate in the solution if the number of propositions that are satisfied by the action plus the sum of contributions of other cliques is smaller than the size of the projection goal. This counting forms the core of the projection consistency.

4.3 Experimental Evaluation

We implemented the projection consistency enforcing algorithm in C++ and we have integrated it into our implementation of the GraphPlan algorithm in order to improve the solving process of the problems of finding supporting actions for a goal. For the experimental evaluation itself we used the same set of problems as in the case of evaluation of constraint models for maintaining arc-consistency. That is, we used several instances of planning problems from Dock Worker Robots environment, Towers of Hanoi environment, and from

the Refueling Planes environment. Again the same statistical characteristics were collected. The tests were performed on the same hardware and in the same software environment as the previous experimental evaluation.

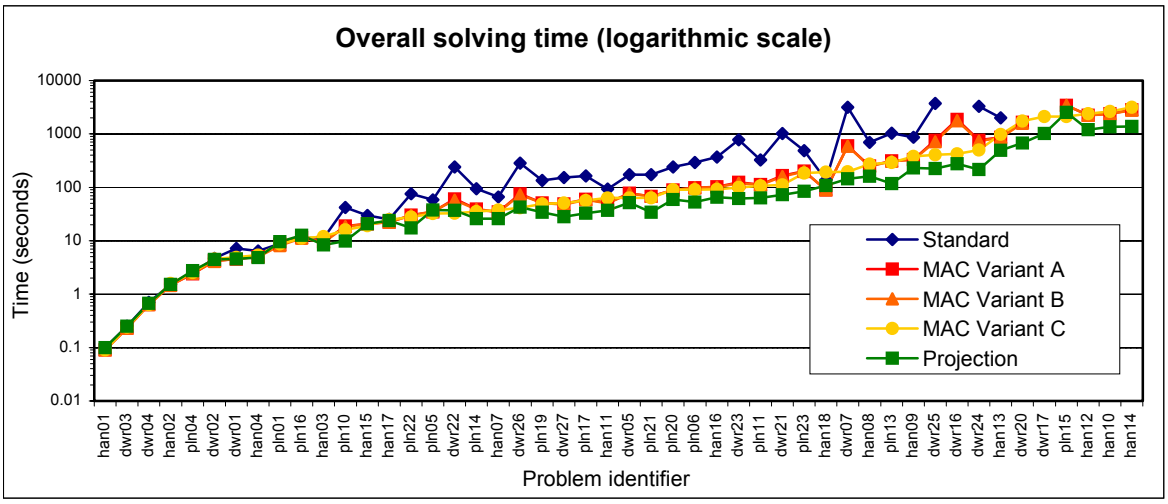


Figure 7. *COMPARISON OF OVERALL SOLVING TIMES (LOGARITHMIC SCALE) - (STD, VARA, VARB, VARC, PRJ).* Comparison of the overall solving time of the standard GraphPlan algorithm and enhanced versions which use maintaining arc-consistency and maintaining projection consistency for solving the problem of finding supports (standard version and variants *A*,*B*, and *C* and projection consistency propagation schemes are compared). Problems on the horizontal axis are listed in the ascending order according to the time consumed by the *variant C*. Time limit of 1 hour for each problem is used.

Since the projection consistency was further improved we postpone the presentation of more detailed results into the next section. The results in figure 7 shows the comparison of the standard version of the GraphPlan algorithm and the enhanced versions which use maintaining arc-consistency of variants *A*, *B*, and *C* and maintaining projection consistency. The ordering of problems along the horizontal axis is according to the ascending time consumed by the *variant C*. The results show that the version with maintaining projection consistency is the best on almost all the tested problems. Notice that projection consistency requires an extra time for building the clique cover and still it is faster.

5 Tractability using Projection Consistency

Consider that we have a clique decomposition of the mutex graph of a certain action layer of the planning graph. Next consider that a set of propositions supported by actions in the clique is constructed for each clique. We noticed that the intersection graph (Golumbic,

1980), where vertices are these sets and edges are their non-empty intersections, has typically a simple structure resembling interval graphs.

5.1 Tractability

We developed a method that is trying to utilize the above observation for solving the problem of supporting actions in connection with projection consistency. More precisely, we developed a polynomial time solving procedure for solving the problem of supports when the above intuitively defined graph is acyclic. The approach is very similar to that used for enforcing arc-consistency in an acyclic constraint network (Dechter, 2003). Next, we proposed a heuristic that guides the solving process of the general problem of supports that is trying to simplify the problem to one belonging into the tractable class.

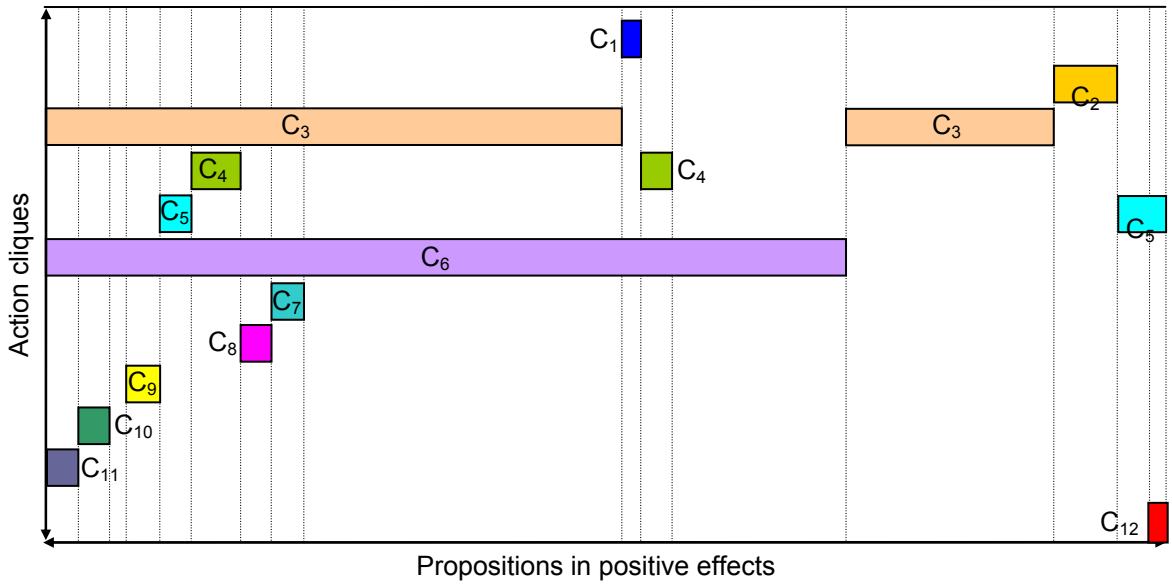


Figure 8. *A DIAGRAM OF MERGED POSITIVE EFFECTS OF THE CLIQUE DECOMPOSITION.* A diagram of merged positive effects of cliques of action layer clique decomposition. Each line of the diagram represents a clique. The scope of the merged positive effect of the clique is depicted as one or more horizontal bars. The width of bars is proportional to the number of actions in the individual action cliques. The diagram was constructed according to an action layer of the planning graph for an instance of the Dock Worker Robots domain.

The obstacle is that not every instance of the problem of supports belongs to the described class. An example of a interval diagram constructed from the problem of supports is shown in figure 8. The corresponding clique intersection graph is shown in figure 9. According to these figures the problem does not belong the tractable class (the graph of the figure 9 is not acyclic). However, the problem is very close to our tractable class. The clique intersection graph can be made acyclic by removing a single vertex. The vertex re-

moval corresponds to the selection of an action from the clique corresponding to the vertex, namely C_6. After having the clique intersection graph acyclic we can use the specialized solving algorithm based on projection consistency. For selecting actions we suggest to use highest degree heuristics. That is an action from the clique of the highest degree of the clique intersection graph (merged with corresponding mutex network) is selected preferably.

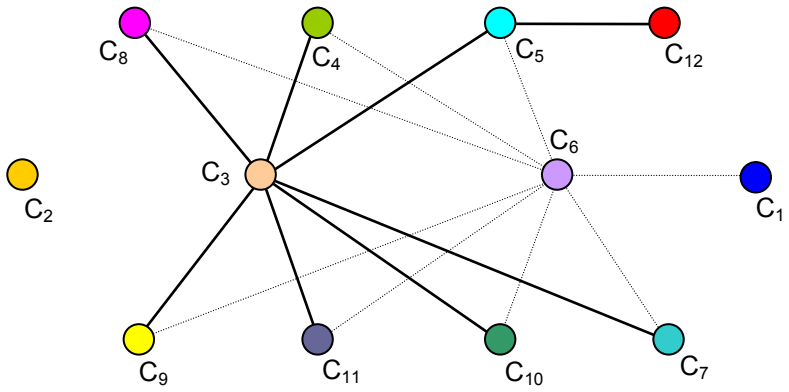


Figure 9. *CLIQUE INTERSECTION GRAPH.* An intersection graph of merged positive effects of cliques of the action layer clique decomposition form figure 3.11. The effect of removing of the cycle-cut-set consisting of the vertex C_6 is denoted by doted edges.

5.2 Experimental Evaluation

We evaluated the proposed approach using our experimental implementation written in C++. The integration of the proposed consistency enforcing algorithm into the solving algorithm is similar as that in the case of maintaining arc-consistency and maintaining projection consistency. The algorithm follows the standard GraphPlan algorithm except the part for solving the problem of finding supporting actions for a goal. For this we use maintaining projection consistency with the heuristic for preferring the tractable case (action from a clique of the highest degree is preferably selected) and when the tractable case is reached we switch to the specialized polynomial time solving process.

We used the same set of planning problems as in the previous sections for the experiments. This allows a direct comparison of performance of all the proposed method. The set of planning problems consists of several instances of various difficulties of Dock Worker Robots, Towers of Hanoi and Refueling Planes planning domain.

We compared the proposed method for solving the tractable case of the problem with the standard GraphPlan, with the *variant C* which maintains arc-consistency and with the version which maintains pure projection consistency. The experiments were again run on the same machine and in the same software environment.

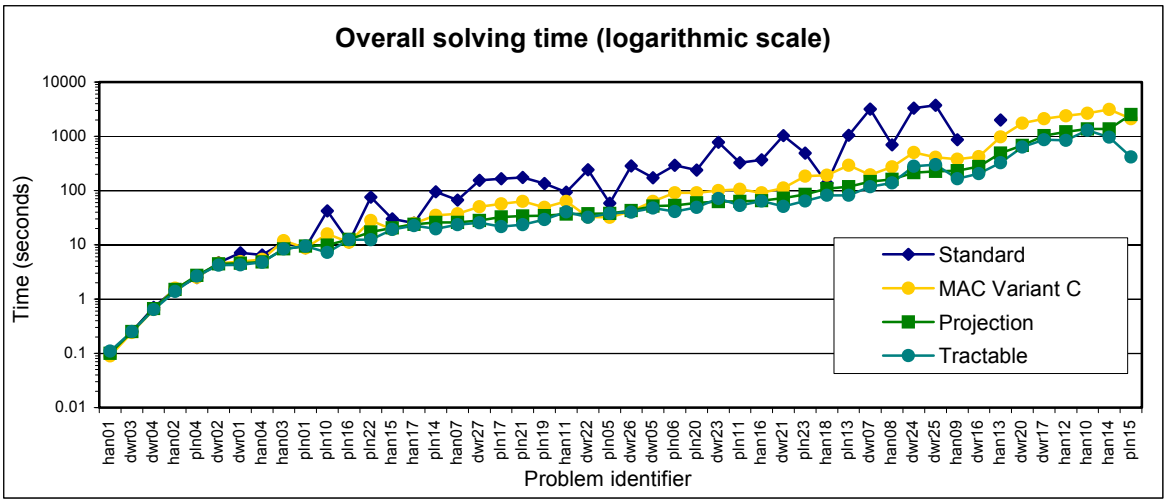


Figure 10. *COMPARISON OF OVERALL SOLVING TIMES (LOGARITHMIC SCALE) - (STD, VARC, PRJ, TRACT).* Comparison of the overall solving time of the standard GraphPlan algorithm and enhanced versions which use maintaining arc-consistency of variant C, maintaining projection consistency, and maintaining projection consistency with preference of tractable case for solving the problem of finding supports. Problems on the horizontal axis are listed in the ascending order according to the solving time consumed by the maintaining projection consistency algorithm. Time limit of 1 hour for each problem is used.

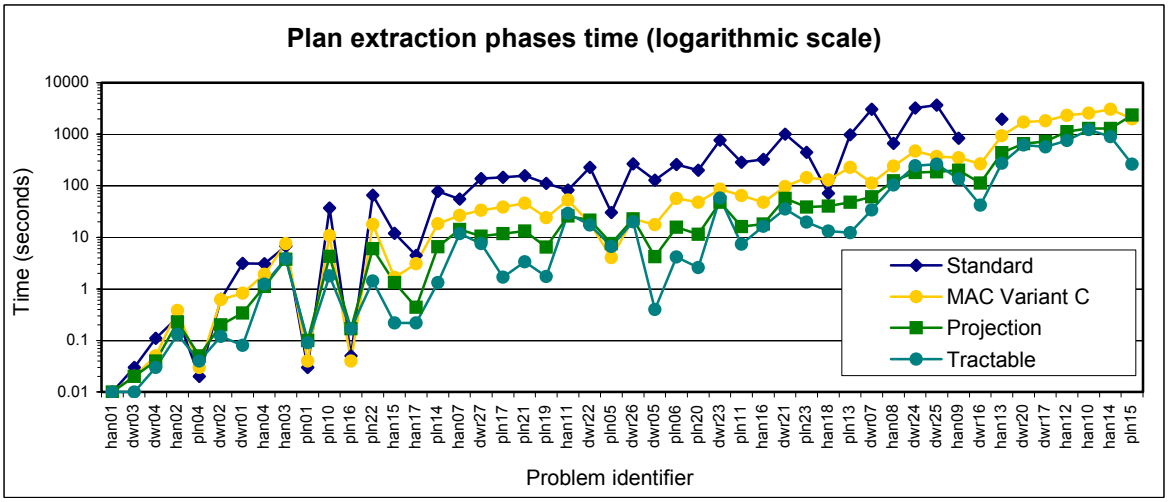


Figure 11. *COMPARISON OF PLAN EXTRACTION PHASE TIMES (LOGARITHMIC SCALE) - (STD, VARC, PRJ, TRACT).* Comparison of time spent in plan extraction phases of the standard GraphPlan algorithm and enhanced versions which use maintaining arc-consistency of variant C, maintaining projection consistency, and maintaining projection consistency with preference of tractable case for solving the problem of finding supports. Problems on the horizontal axis are listed in the ascending order according to the solving time consumed by the pure maintaining projection consistency algorithm.

The comparison of the overall solving time (planning graph building time + time of plan extraction phases) of the proposed method based on tractable case with the standard GraphPlan, maintaining arc-consistency method - *variant C*, and the method using projection consistency is shown in figure 10. The comparison of times spent in plan extraction phases is shown in figure 11.

The improvement in overall problem solving time is up to 200% compared to the version which uses projection consistency. The improvement in plan extraction time is up to 1000%.

6 Difficult Planning Problems

We found that our approach based on the tractable class of problems of supports is especially successful on difficult problems which force the planner to really perform search to find the solution or to prove that there is no solution (Urquhart, 1987). Such problems encapsulates for example an instance of the Dirichlet's box principle (place $n+1$ pigeons into holes n so that no two pigeons are placed in the same hole). Although these problems are short in length of the input, they are hard to be answered. This property of the box principle makes the solver to perform exhaustive search to prove that the problem has no solution. The algorithm for the problem of finding supports of the tractable class has an advantage in such situation. It can detect insolvability of some sub-problems quickly in polynomial time and can prune large parts of the search space in this way.

The performed competitive comparison of our experimental planning system implementing the algorithm for the tractable class with several state-of-the-art planners on the mentioned difficult problems showed surprising results. We chose several planners participating in the International Planning Competition (*IPC*) (Gerevini *et al.*, 2006) for our experimental evaluation. Although it was not our goal to compete with planners from the *IPC* by our experimental planner, the result was that our experimental planner performs better than some of the winners of the *IPC* on selected problems.

For the competitive evaluation we used several problems encoding (insolvable) Dirichlet's box principle. We compared our version of the GraphPlan algorithm that prefers tractable class with several state-of-the-art planners. We selected optimal planners *MaxPlan* (Zhao *et al.*, 2007), *SATPlan* (Kautz *et al.*, 2007), *CPT 1.0* (Vidal and Geffner, 2007), and *IPP 4.1* (Koehler, 2007) and satisfying planners *SGPlan 5.1* (Hsu *et al.*, 2007) and *LPG-td 1.0* (Gerevini and Serina, 2007). The results are shown in table 1. All the tests were performed on the same testing machine and in the same software environment as previous tests.

The improvement obtained by using tractable class preference algorithm is in order of magnitude compared to the selected state-of-the-art planners on the selected problems. Experiments showed that current planners do not cope well on the selected insolvable prob-

lems. Even some planners seems to be falling into an infinite loop (*SATPlan* and *CPT*), another planner incorrectly solved insolvable problems (*SGPlan*). We consider this property as a major obstacle for practical usage of these planners (a user does not get the answer whether the problem can be solved). Although the selected class of testing problems is rather limited, it represents an important class of difficult problems (Urquhart, 1987) which intrinsically require search to be answered (the solution cannot be guessed).

Instance	**Our planner** (seconds)	Speedup ratio w.r.t **SGPlan 5.1**	Speedup ratio w.r.t **IPP 4.1**	Speedup ratio w.r.t. **MaxPlan**	Speedup ratio w.r.t. **SATPlan**	Speedup ratio w.r.t. **CPT**	Speedup ratio w.r.t. **LPG-td**
ujam-02_01	**0.06**	N/A	N/A	N/A	N/A	**1.00**	N/A
ujam-03_02	**0.54**	N/A	**0.02**	**+ 0.04**	N/A	N/A	***+ 9.25***
ujam-04_03	**3.39**	N/A	**0.19**	**+ 0.30**	N/A	N/A	***+ 1.76***
ujam-05_04	**23.88**	N/A	***3.50***	**+ 0.35**	N/A	N/A	**+ 0.37**
ujam-06_05	**177.9**	N/A	***> 3.37***	***> 3.37***	N/A	N/A	***> 3.37***
ujam-07_06	> 600	N/A	N/A	N/A	N/A	N/A	N/A
ujam-08_07	> 600	N/A	N/A	N/A	N/A	N/A	N/A
ujam-09_08	> 600	N/A	N/A	N/A	N/A	N/A	N/A
ujam-10_09	> 600	N/A	N/A	N/A	N/A	N/A	N/A
jam-02_01	**0.03**	N/A	N/A	***8.66***	***5.66***	**1.00**	N/A
jam-03_02	**0.09**	N/A	N/A	***2.77***	***1.77***	**0.44**	N/A
jam-04_03	**0.25**	N/A	**0.08**	***1.76***	**0.68**	**0.68**	N/A
jam-05_04	**0.60**	N/A	***1.08***	***1.80***	**0.38**	***8.38***	N/A
jam-06_05	**1.29**	N/A	***20.00***	***2.14***	**0.71**	***177.32***	N/A
jam-07_06	**2.48**	N/A	***> 241.93***	***12.46***	***1.21***	***> 241.93***	N/A
jam-08_07	**4.60**	N/A	***> 130.43***	***49.56***	***3.19***	***> 130.43***	N/A
jam-09_08	**8.77**	N/A	***> 68.42***	***> 68.42***	***17.33***	***> 68.42***	N/A
jam-10_09	**17.05**	N/A	***> 35.19***	***> 35.19***	***> 35.19***	***> 35.19***	N/A
holes-02_01	**0.00**	N/A	N/A	N/A	N/A	N/A	N/A
holes-03_02	**0.02**	N/A	N/A	**+ 0.50**	N/A	N/A	***200.00***
holes-04_03	**0.04**	N/A	N/A	***+ 26.00***	N/A	N/A	***125.00***
holes-05_04	**0.12**	N/A	**0.08**	***+ 125.00***	N/A	N/A	***41.66***
holes-06_05	**0.27**	N/A	**0.44**	***+ 1000.00***	N/A	N/A	***18.51***
holes-07_06	**0.55**	N/A	***3.44***	***> 1090.00***	N/A	N/A	***10.90***
holes-08_07	**1.08**	N/A	***28.09***	***> 555.55***	N/A	N/A	***12.96***
holes-09_08	**2.08**	N/A	***276.25***	***> 288.46***	N/A	N/A	***> 288.46***
holes-10_09	**4.06**	N/A	***> 147.78***	***> 147.78***	N/A	N/A	***> 147.78***

Table 1. PERFORMANCE COMPARISON OF PLANNERS ON DIFFICULT PROBLEMS - PART II. Comparison of selected state-of-the-art planner with our experimental planning system on several hard planning problems encoding Dirichlet's box principle. The symbol N/A indicates that a comparison cannot be made.

7 Consistency in Boolean Satisfiability

Our contribution to solving SAT problems consists of preprocessing and reformulating the input Boolean formula in the *CNF* (*conjunctive normal form* - conjunction of disjunctions). The result of this processing is either the answer whether the input formula is unsatisfiable or a new formula (hopefully simpler) with the same set of satisfying valuations as that of the input one. If the input formula is not decided by the preprocessing phase then the pre-

processed formula is sent to the SAT solver of the user's choice. The idea behind this process is to make the task for the SAT solver easier by deciding the input formula within the fast preprocessing phase or by providing an equivalent but simpler formula to the SAT solver.

We are viewing the input Boolean formula in a CNF as a graph. For each *literal* (variable or its negation) of the input formula we consider a vertex and for each conflict between literals we consider an edge. Conflicting literals are those that cannot be both satisfied in a single valuation of variables, for example positive and negative literals of the same variable are conflicting. Generally, a set of literals of a formula is conflicting if the formula entails that at most one of the literals can be *true*. To be able to use our reasoning based on the clique decomposition we need a graph with appropriately large *complete sub-graphs* (cliques). That is, we need some kind of a good approximation of the sets of conflicting literals. Unfortunately the graph arising from the above interpretation of the Boolean formula in *CNF* is rather sparse (the largest clique is of size 2). That is why we apply further inference by which we deduce more conflicts between the literals and which allow us to introduce more edges into the graph. We are using singleton arc-consistency (Bessière and Debruyne, 2005) as the inference technique for deducing new edges.

Having the graph constructed from the input CNF formula, a clique decomposition of this graph is found by a greedy algorithm (we do not need an optimal clique decomposition; we need just some of the reasonable quality). The important property of the constructed clique decomposition is that at most one literal from each clique can be assigned the value *true*. In this situation we perform similar contribution counting as in the case of projection consistency to rule out the literals that can never be *true*. To do this, the maximum number of satisfied clauses by literals of each clique is calculated. Then a literal of a certain clique can be ruled out if the literals from the other cliques together with the selected literal do not satisfy enough clauses to satisfy the input formula.

7.1 Experimental Results

We chose three state-of-the-art SAT solvers for comparison with our reformulation method. The SAT solvers of our choice were *zChaff* (Fu *et al.*, 2007), *HaifaSAT* (Gershman and Strichman, 2007) and *MiniSAT* (a version with *SATElite* preprocessing integrated) - (Eén and Sörensson, 2007) - (we used the latest available versions to the time of writing this thesis). Our choice was guided by the results of several last SAT competitions - *SAT Competition 2005* and *SAT Race 2006* (Le Berre and Simon, 2005; Sinz, 2006) in which these solvers belonged to the winners. We again used the same testing environment as in the previous cases.

The testing set consisted of several difficult unsatisfiable SAT instances. This set of benchmark problems was collected by Aloul (2007) and it is provided at his research web

page. The details about hardness and construction of these instances are discussed in (Aloul *et al.*, 2002). For each benchmark SAT instance we measured the overall time necessary to decide its satisfiability. The results are shown in table 2. The speedup obtained by using our method compared to tested SAT solvers is also shown.

As it is evident from our experiments the proposed method brings significant improvement in terms of time necessary for the decision of the selected difficult benchmark problems (Pigeon holes, FPGA routing instances).

Instance	**Decided by preprocessing**	**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*
urq3_5	no	47 x 2	**130.15**	**0.73**	*N/A*	*N/A*
urq4_5	no	73 x 2	*> 600.00*	*N/A*	*N/A*	*N/A*
urq5_5	no	120 x 2	*> 600.00*	*N/A*	*N/A*	*N/A*
urq6_5	no	179 x 2	*> 600.00*	*N/A*	*N/A*	*N/A*
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*

Table 2. EXPERIMENTAL COMPARISON OF SAT SOLVERS. Experimental comparison of three SAT solvers with the method using clique-consistency over the selected difficult benchmark SAT instances. Again timeout of 10.0 minutes (600.00 seconds) for all the tests was used.

8 Conclusions

The thesis represents contribution to the areas of solving planning problems and solving Boolean satisfiability problems. The main results of this thesis are the improvements of the process of solving problem of supporting actions for a goal within the context of GraphPlan algorithm and special preprocessing method for solving Boolean satisfiability problems.

We gradually describe several variants of solving the problem of finding supporting actions for a goal within the GraphPlan algorithm. Solving of this problem is one of the

weakest points of the whole algorithm. First, we proposed to model the problem as a constraint satisfaction problem and to maintain arc-consistency throughout the process of search for solution of the problem of finding supports. Several variants of propagation of arc-consistency were proposed. The GraphPlan algorithm enhanced by every variant of propagation of arc-consistency proved to be better than the original version in all the major measurable characteristics.

The promising results obtained by maintaining arc-consistency served as an inspiration for developing a specialized global consistency. This consistency exploits some structural properties of the problem of supports. Arc-consistency performs propagations only in a local neighborhood of currently explored part of the problem which seems to be weak. The idea was therefore to develop global consistency that would take into account the whole problem at once. These ideas put into reality are represented by the definition of projection consistency. This is a special type of consistency which exploits the structure of graphical representation of the problem of finding supporting actions. Namely, the consistency exploits the decomposition of the graph of the problem into several complete graphs - cliques. The knowledge of cliques decomposition allows us to use special counting arguments to rule out the actions from further considerations. It was theoretically shown that this approach can be stronger than local consistency such as arc-consistency. The performed experimental evaluation showed significant improvements compared to the best version that uses arc-consistency. The improvements were in the overall solving time as well as in other characteristics.

The final enhancement of the consistencies for solving the problem of finding supporting actions was the definition of a so called tractable class of the problem. It is in fact the class of problems defined using the proposed projection consistency that can be solved in polynomial time. This tractable class together with the heuristics that transform the general problem (not belonging to the class) to the problem belonging to the class forms the best performing algorithmic technique for solving the problem. The experimental evaluation showed that some planning problems can be solved even without backtracking using this enhancement within the GraphPlan algorithm. Compared to the previously developed techniques for solving the problem, the tractable class brings further improvements in comparison to the version which uses pure projection consistency.

The results achieved in solving Boolean satisfiability problems are represented by the special preprocessing technique which can be used to simplify the problems before they are passed to the general solving system. The success of exploiting the structural properties of the problem by projection consistency was an inspiration for finding similar structures in the Boolean satisfaction problems. The situation here was complicated by the fact that raw Boolean satisfaction problem lacks structures when it is interpreted graphically. Therefore several inference stages were proposed to make the structures visible and allow their detection and usage. The resulting consistency technique was called clique consistency since it again exploits cliques of the graphical interpretation of the problem. The clique consistency

is in fact adaptation of projection consistency for Boolean satisfaction problems. The experimental evaluation showed that usage of clique consistency can outperform today's best Boolean satisfaction solvers (SAT solvers) on the selected set of difficult problems.

9 Bibliography

Ravindra K. **Ahuja**, Thomas L. **Magnanti**, and James B. **Orlin (1993)**. *Network Flows: Theory, Algorithms and Applications.* Prentice Hall.

Fadi A. **Aloul**, Arathi **Ramani**, Igor L. **Markov**, and Karem A. **Sakallah (2002)**. *Solving difficult SAT instances in the presence of symmetry*. Proceedings of the 39th Design Automation Conference (DAC 2002), New Orleans, LA, USA, 731-736, ACM Press.

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

Andrew B. **Baker (1995)**. *Intelligent Backtracking on Constraint Satisfaction Problems: Experimental and Theoretical Results*. Doctoral Thesis, University of Oregon, Eugene, OR, USA.

Christian **Bessière** and Marie-Odile **Cordier (1993)**. *Arc-Consistency and Arc-Consistency Again.* Proceedings of the 11th National Conference on Artificial Intelligence (AAAI 1993), Washington, DC, USA, 108-113, AAAI Press/MIT Press.

Christian **Bessière** and Romuald **Debruyne (2005)**. *Optimal and Suboptimal Singleton Arc Consistency Algorithms*. Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005), Edinburgh, Scotland, United Kingdom, 54-59, Professional Book Center.

Christian **Bessière** and Jean-Charles **Régin (2001)**. *Refining the Basic Constraint Propagation Algorithm.* Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2001), Seattle, WA, USA, 309-315, Morgan Kaufmann Publishers.

Avrim L. **Blum** and Merrick L. **Furst (1997)**. *Fast planning through planning graph analysis.* Artificial Intelligence, Volume 90 (1-2), 281-300, AAAI Press.

Stephen A. **Cook (1971)**. *The Complexity of Theorem-Proving Procedures.* Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC 1971), Shaker Heights, OH, USA, 151-158, ACM Press.

Martin **Davis** and Hilary **Putnam (1960)**. *A Computing Procedure for Quantification Theory.* Journal of the ACM, Volume 7 (3), 201-215, ACM Press.

Rina **Dechter (2003)**. *Constraint Processing*. Morgan Kaufmann Publishers.

Niklas **Eén** and Niklas **Sörensson (2007)**. *The MiniSat Page*. Research web page, http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/Main.html, Chalmers University, Sweden, (March 2007).

Zhaohui **Fu**, Yogesh **Marhajan**, and Sharad **Malik (2007)**. *zChaff*. Research Web Page. http://www.princeton.edu/ ~chaff/zchaff.html, Princeton University, USA, (March 2007).

Alfonso **Gerevini**, Yannis **Dimopoulos**, Patrik **Haslum**, and Alessandro **Saetti (2006)**. *5th International Planning Competition*. Event in the context of the 16th International Conference on Automated Planning and Scheduling (ICAPS 2006), Cumbria, UK, http://ipc5.ing.unibs.it, University of Brescia, Italy, (May 2007).

Alfonso **Gerevini** and Ivan **Serina (2007)**. *Homepage of LPG*. Research web page, http://zeus.ing.unibs.it/lpg/, University of Brescia, Italy, (April 2007).

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

Malik **Ghallab**, Dana S. **Nau**, and Paolo **Traverso (2004)**. *Automated Planning: theory and practice.* Morgan Kaufmann Publishers.

GNU Project (2008). *GCC, the GNU Compiler Collection.* Project Web Page. http://gcc.gnu.org/, (March 2008).

Martin C. **Golumbic (1980)**. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press.

William D. **Harvey (1995)**. *Nonsystematic Backtracking Search*. Doctoral Thesis, University of Oregon, Eugene, OR, USA.

Chih-Wei **Hsu**, Benjamin W. **Wah**, Ruoyun **Huang**, and Yixin **Chen (2007)**. *SGPlan 5: Subgoal Partitioning and Resolution in Planning*. Research web page. http://manip.crhc.uiuc.edu/programs/SGPlan/index.html, University of Illinois, USA, (April 2007).

Henry A. **Kautz**, Bart **Selman**, and Jörg **Hoffmann (2007)**. *SATPLAN*. Research web page. http://www.cs.rochester.edu/u/kautz/satplan/index.htm, University of Rochester, NY, USA, (April 2007).

Jana **Koehler (2007)**. *Homepage of IPP*. Research web page, http://www.informatik.uni-freiburg.de/~koehler/ipp.html, University of Freiburg, Germany, (April 2007).

Jana **Koehler**, Bernhard **Nebel**, Jörg **Hoffmann**, and Yannis **Dimopoulos (1997)**. *Extending Planning Graphs to an ADL Subset*. In Proceedings of the 4th European Conference on Planning (ECP-97), Toulouse, France, 273-285, LNAI 1348, Springer-Verlag.

Daniel **Le Berre** and Laurent **Simon (2005)**. *SAT Competition 2005*. Competition Web Page, http://www.satcompetition.org/2005/, Scotland, (March 2007).

Mandriva (2008). *Mandriva 10th Year of Innovation.* Commercial Web page. http://www.mandriva.com/, (March, 2008).

Roger **Mohr** and Thomas C. **Henderson (1986)**. *Arc and Path Consistency Revisited.* Artificial Intelligence, Volume 28 (2), 225-233, Elsevier Science Publishers.

Alan K. **Mackworth (1977)**. *Consistency in Networks of Relations*. Artificial Intelligence, , Volume 8 (1), 99-118, Elsevier Science Publishers.

Stuart **Russell** and Peter **Norvig (2003)**. *Artificial Intelligence: A Modern Approach (second edition)*. Prentice Hall.

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

Bjarne **Stroustrup (1986)**. *The C++ Programming Language.* Addison-Wesley.

Alasdair **Urquhart (1987)**. *Hard examples for resolution*. Journal of the ACM, Volume 34, 209-219, ACM Press.

Vincent **Vidal** and Hector **Geffner (2004)**. *Branching and Pruning: An Optimal Temporal POCL Planner-based on Constraint Programming.* In Proceedings of the AAAI Workshop on Integrating Planning Into Scheduling, event in the context of the AAAI 2004 conference, USA.

Vincent **Vidal** and Hector **Geffner (2007)**. *CPT Description.* Research web page, http://www.cril.univ-artois.fr/~vidal/cpt.html, Université D'Artois, France, (April 2007).

David L. **Waltz (1975)**. *Understanding line drawings of scenes with shadows.* The Psychology of Computer Vision, P. Winston (editor), 19–91, McGraw-Hill, New York.

Yuanlin **Zhang** and Roland H. C. **Yap (2001)**. *Making AC-3 an Optimal Algorithm.* Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2001), Seattle, WA, USA, 316-321, Morgan Kaufmann Publishers.

Xing **Zhao**, Yixin **Chen**, Weixiong **Zhang**, and Ruoyun **Huang (2007)**. *MaxPlan.* Research web page, http://www.cse.wustl.edu/~chen/maxplan/, Washington University in St. Louis, MO, USA, (April 2007).