Investigating the Effect of Relevance and Reachability Constraints on SAT Encodings of Planning

Minh Binh Do, Biplav Srivastava & Subbarao Kambhampati
Email: {binhminh, biplav, rao}@asu.edu
Department of Computer Science and Engineering
Arizona State University, Tempe, AZ 85287-5406.

Abstract

Currently, Graphplan and Blackbox, which converts Graphplan's plan graph into the satisfaction (SAT) problem, are two of the most successful planners. Since Graphplan gains its efficiency from the forward propagation of reachability based mutual exclusion constraints (mutex) and their backward use, it has been believed that SAT encoding will also benefit from mutexes. In this paper, we will try to answer two important questions: (1) Are mutual exclusions actually useful for solution extraction in SAT encoding? (2) Are there other useful constraints that can be propagated on the planning graph which may help SAT solvers ? Our experiments with systematic solvers Relsat and Satz shows that though forward mutexes are useful in general, there are domains in which mutex constraints can slow down search. Moreover, we introduce the notion of backward mutex and their propagation which is based on relevance analysis and implement it in Blackbox. We find that the addition of relevance based backward mutual exclusions helps speedup the Relsat solver in solving the SAT encoding of many standard planning problems.

Introduction

Graphplan(Blum & Furst 1995) is currently one of the more efficient algorithms for solving general classical planning problems. It uses forward propagation of mutual exclusion (*mutex*) relationships in the graph expansion phase to derive reachability-based constraints (referred to as *fmutex* henceforth), and later uses them in the backward search to prune out non-solution branches.

Recently, in (Kautz & Selman 1996),(Kautz & Selman 1998), impressive results have been obtained by encoding the plan graph as a SAT problem and using fast SAT solvers to obtain the solutions. While it has been clear from the beginning that forward mutex propagation is useful in Graphplan system, the question of whether mutexes are beneficial for SAT encodings has not been addressed directly. After all, constraints correspond to more clauses in a SAT encoding and this can potentially degrade performance if model checking cost is not compensated by increased unit propagation. We empirically show that for most of the domains, fmutex constraints help in speeding up search. However, there are domains in which putting them in the encoding actually slows down the search.

While fmutexes are based on the reachability analysis of the planning problem, the equally important relevance-based constraint probagation is not currently employed by Graphplan and Blackbox. As search in Graphplan is in backward direction, relevance based relationships mimic search and will not be useful. But once a SAT encoding is available, model checking in SAT is directionless and can benefit from relevance based constraints.

A first step in this direction is a routine in Blackbox to eliminate irrelevant actions and propositions from the graph to make the search more focussed. We focus on relevance analysis and introduce the backward mutex (for short, *bmutex*) relationship that can find relevance-based constraints more directly. We show that bmutex helps in finding better quality solutions and speeds up search. Like fmutexes, bmutexes restrict the use of certain pair of actions or propositions in the same level together. However, they have different semantics: when conditions (X, Y) are fmutex, it means "X, Y can not be true together", while when conditions (X,Y) are bmutex, it means "X, Y need not to be true together in any state visited by a minimal plan".

We have implemented bmutex in Blackbox and empirical evidence shows that adding bmutex to the SAT encoding (along with fmutex) expedites search in most of the domains tested using the systematic Relsat solver, with speed up of up to 10x in some problems. By restricting actions that give redundant effects to not occur in the same plan, bmutex also enables better quality solutions in most tested problems. We note that acting alone, fmutex constraints are generally more powerful than bmutex constraints in pruning search in SAT. However, bmutex in conjunction with fmutex can be more effective than fmutex itself.

The paper is organized as follows. We start with a discussion on the Graphplan algorithm and the role of forward mutex relations. Next, we show the connection between Graphplan, constraint satisfaction problem, and SAT. We proceed to present bmutex constraints and give a detailed discussion on how to find, maintain, and use bmutexes.

Next, we provide empirical results to validate the utility of fmutexes and bmutexes in SAT. Finally, we conclude with related work and future directions.

Graphplan and fmutex

The Graphplan algorithm consists of two interleaved phases, forward graph construction, and backward solution extraction. In the graph construction phase (see Figure 1), level 0 contains the initial state. The algorithm incrementally extends the graph by appending one action level, which contains all actions that are applicable, and a proposition level which is the union of the effects of newly added action level and the last proposition level. During the graph expansion phase, it also calculates the mutex constraints for each newly added action and fact levels. Mutexes are defined as follows:

- Two actions A, and B are mutex if one action deletes any precondition or effect of the other (*static mutex*) or if there exists a pair of preconditions, one of A and one of B that are mutex (*dynamic mutex*).
- Two facts P, and Q are mutex if all actions supporting P (i.e. have P as one of their effect) are pair-wise mutex with all actions supporting Q.

The constructed graph contains links between actions and their preconditions in the earlier proposition level, and their effects in the next level. Notice that Graphplan also introduces "noop" actions, which provide the links between the same facts appearing in consecutive levels. The plan graph in Graphplan, with its forward mutexes, shows an upper bound on what literals can be achieved in parallel at each level. We call fmutexes "reachability-based mutexes" because the plan graph and fmutexes tell us about the states that can be reached given the initial state and a set of actions. An interesting point to note is that fmutex propagation follows the AND(fact level)/OR(action level) rule. AND means for two facts to be fmutex, all the actions that support them must be pair-wise mutex. OR means if any two preconditions of two actions are mutex, then that pair of action is mutex.

The solution extraction phase only starts when all goals appear pair-wise non-mutex at some level k. The search phase on the k-level plan graph involves recursively checking backward from the goal level whether there exists a k-level subgraph S in which:

- Every goal is present in S.
- Every fact node in S must have some node in lower action level linked to it.
- Every action node in S must have the entire links with its preconditions in S.
- There are no two nodes in the same level in S that are mutex with each other.

Graphplan's backward search constructs a solution graph incrementally from the last level. The fmutex propagation helps to ensure that subgraphs which do not satisfy the last property above will be detected and pruned early.

Graphplan and CSP

Before we talk about the SAT encoding of the plan graph in the next section, due to the fact that SAT can be considered as boolean constraint satisfaction problem (*CSP*), it is useful to discuss the connections between Graphplan and CSP. The relation between solution extraction in Graphplan and search in CSP has been discussed in detail in (Kamphampati 1999) and (Weld 1999), which shows how one can convert a plan graph to dynamic and standard CSP.

To solve CSPs efficiently, we have to add enough constraints to increase the local consistency of the problem. The way Graphplan selectively puts actions in the graph, and the way fmutex propagation process is used by Graphplan, can be seen as constraint propagation in CSP to enforce a partial directed 1-consistency and 2-consistency of the problem(Kamphampati 1999). The fact that fmutexes are all constraints between a pair of conditions ensures that they will help strengthen the partial 2-consistency of the problem[1]. In a way similar to fmutex propagation, under certain rules, we can also derive constraints that help enforce partial 1-consistency and 2-consistency from the goal state. When building the plan graph, we already know that to make it compact, Graphplan will never put in an action if not all of its preconditions appear in earlier fact level, or some of them are fmutex. We can do the same thing but going backward from the goal state to enforce partial 1-consistency, and make the graph even smaller by only taking into consideration *relevant* conditions, which are defined backward from the goal state as follow:

- All goals are relevant.
- All actions that give some relevant effect will be relevant.
- All facts that suport some relevant action will be relevant.

Currently, this cheap process is used by Blackbox to eliminate irrelevant conditions from the plan graph before converting it to a SAT problem[2]. From now on, whenever we mention about the SAT encoding of Blackbox, we mean the encoding of the relevant part of the graph.

[1]We can propagate constraints of size bigger than 2 in the way similar to mutex propagation in Graphplan. (Brafman 1999) discusses about k-size reachability and relevance-based constraints, but no result other than k=1 were shown.

[2]There is no need in Graphplan for such a procedure. For each subgoal set at each level, it takes into consideration only actions in the last action level that connect to facts in that set.

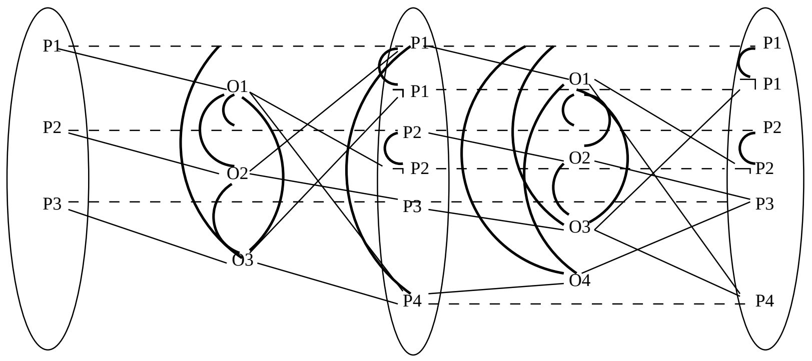

Figure 1: *Plan graph with fmutex propagation. Arcs refer to mutual exclusion relationships between actions or facts.*

To help enforce 2-consistency more completely, we introduce backward mutexes which are propagated from the goal state in a similar manner as forward mutex propagation, in a later section.

Blackbox and its SAT encoding of the plan graph

Blackbox planner(Kautz & Selman 1998) converts the plan graph of Graphplan to SAT clauses and uses fast systematic (Relsat(Bayardo & Schrag 1997), Satz(Li & Anbulagan 1997)) or randomized (Walksat(Kautz & Selman 1996), Relsat-rand, Satz-rand) SAT solvers to find a model in the encoding. First, every node (action or fact) in the relevant part of the plan graph is assigned an unique number. Then, the following types of SAT clause are generated from the plan graph by Blackbox:

1. All goals are set to value true.
2. All facts that hold in the initial state are set to be true.
3. For each fact F in the level other than level 0, add an axiom: F $\Rightarrow$ $A_1 \vee A_2 .... \vee A_n$, in which $A_1 ... A_n$ are actions in the lower level that give F. This axiom ensures that for each fact in the graph to be true, there should be at least one action in the lower level that supports it.
4. For each action A, where $P_1..P_n$ are A's preconditions, and $E_1..E_n$ are A's effects, add axioms: $A \Rightarrow P_1$; $A \Rightarrow P_2$.... $A \Rightarrow P_n$ (4.1), and $A \Rightarrow E_1$; $A \Rightarrow E_2$.... ; $A \Rightarrow E_n$ or $\neg A \vee E_1$ (4.2). Those axioms state that if one action is true, then its preconditions and effects must also be true.
5. For each mutex constraint (A,B) in the graph, add one axiom: $A \Rightarrow \neg B$, or $\neg A \vee \neg B$ to state that A and B can not be true together.

By default, Blackbox uses clauses of type 1, 2, 3, 4.1, and 5 (only for mutexes at action level) to encode the plan graph.

Bmutex and relevance analysis

Unlike reachability-based fmutexes, bmutexes are relevance-based relations between conditions that support the same (relevant) effects at the next level. Bmutexes are counted starting from the last level (goal level), and propagates backward to the initial state level in the relevant part of the plan graph as follows:

Facts:

- Facts in the goal level are not bmutex with each other.
- Two relevant facts P and Q are bmutex if all relevant actions that have P as one of their preconditions are pairwise bmutex *or* fmutex with all actions that have Q as one of their precondition.

Actions: Let us define E(A) as the set of action A's relevant effects. Action A_1 is bmutex with action A_2 if:

- $E(A_1)$ subsumes $E(A_2)$.
- All facts in $E(A_1) \backslash E(A_2)$ are pair-wise bmutex with all facts in $E(A_2) \backslash E(A_1)$ (where '$\backslash$' is set difference operator).

Following is a simple example illustrated in Figure 2. Here, all the *relevant* nodes are in bold typeface. Starting from the goal state ($\neg P_1$,P_3,P_4), we can see that (O_2,O_4) are bmutex because they both give the same effect P_3, and (O_1,O_3) are bmutex because O_3's effects ($\neg P_1$,P_4) subsumes O_1's effect (P_4). Continuing to the next level, we see that (O_2, noop-P_3) are bmutex, and (O_2,O_3) are fmutex. Since all the consumers of (P_2 and P_3) are pair-wise mutually exclusive, they are also bmutex.

Basically, the expectation is that by using bmutex, one can prune unnecessary actions in any feasible solution that can be achieved from current search branch; and this may speedup the search by avoiding inoptimal branches. We believe that in hard planning problems, in which not many conditions can be true together in any solution, bmutex

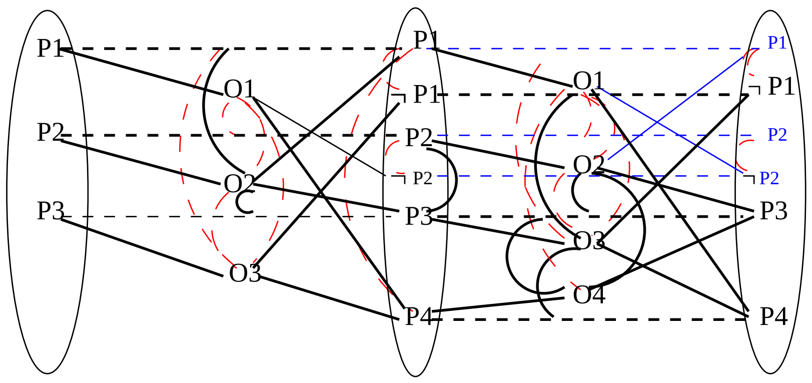

Figure 2: *Backward mutexes with goal ($\neg P_1, P_3, P_4$). Solid arcs refer to bmutexes while dotted arcs refer to fmutexes between actions or facts.*

will help in speeding up the search by explicitly restricting which pairs of conditions in each level will not be true together.

In contrast to fmutex, which states that two conditions in the same level *can not* be true together, bmutex indicates that two conditions *need not* be true together. In essence, fmutex encodes *hard constraints* while bmutex encodes *soft constraints* that can be violated without compromising the correctness of the solution. Moreover, recall that fmutex propagation follows the AND/OR rule. Bmutex propagation follows the AND(fact level)/AND(action level) rule. AND means for two facts to be bmutex, all actions that consume those facts must be pair-wise mutex. Also, any two actions are bmutex, if all of their effects are pair-wise bmutex.

However, in our experiment with SAT encoding, we force all bmutex constraints to be satisfied when solving the SAT encoding and encode them in the same way as normal fmutexes. Thus, if two conditions A, B are found to be bmutex, then we will add the clause $\neg A \vee \neg B$ to the encoding.

Empirical Evaluation

The objective of our experiments is to test if mutual exclusions are useful for solution extraction in SAT encoding and also to check if bmutex constraints can benefit the directionless search in SAT. We used Blackbox(Kautz & Selman 1999) as out testbed which allows plan graph of Graphplan to be encoded as a SAT problem and solved with multiple SAT solvers.

In our experiments, we ran Blackbox with the default SAT encoding and used the systematic Relsat and Satz solvers. We considered the following variations[3]: (1) with only fmutexes (dynamic+static), (2) with only bmutexes (bmutex+static), (3) with both fmutexes and bmutexes and, (4) with only static mutexes ("-noexcl" flag). For each problem, we use the cutoff backtrack limit of 1 million and 10 different random seeds to start the search. The same 10 random seeds are used for all the problems to make comparisons meaningful. The average times, length of the solution, and speedup are taken by the average output from the running on 10 different chosen seeds. The parallel blockworld domain is taken from the HSP(Bonet & Geffner 1999) planner distribution, the logistic domain is from the Blackbox distribution, and all other domains are from AIPS98 planning competition(McDermott 1998). We concentrate more on problems which are quite hard to solve. All the experiments are done on Sun Ultra 5 Unix machine with 256 MB RAM.

Table 1 shows the utility of fmutex relationships. The table shows that while fmutex helps to make problems that are hard to solve or unsolvable in alloted time become solvable in shorter time, there are domains in which removing them from the encoding helps to speed up the search. For example, for problem Bw03 in blockworld domain, Relsat solver can get a solution for the encoding with both fmutexes and bmutexes in fastest time, followed by encoding with only bmutex, and noexcl. The encoding with only fmutex is actually the slowest one. Thing gets more serious for the Satz solver, for which only the noexcl encoding can be solved (in quite short time), and we can not get the solution with any other encoding in 3 hours. The result shows that while for most of the domains, the utility of fmutex in cutting down bad search branches early compensates for the overhead of more clauses in the SAT encoding, there do exist domains in which the role is reversed.

Result in Table 2 shows the effect of adding bmutex constraints to the SAT encodings and solving them using the Relsat solver. We can see that for most of the problems in different domains, bmutex helps in speeding up search

[3]By default, Blackbox puts static mutexes into the SAT encoding even when the no exclusion ("-noexcl") flag is set. We do not modify this because static mutexes can be seen as fmutexes or bmutexes or just as part of the base SAT encoding.

Name	Relsat				Satz			
	f	b	f+b	noexcl	f	b	f+b	noexcl
Log07	114.0	x	196.9	x	33.72	x	75.17	x
Log15	2153.7	x	757.8	x	3430	x	8596	x
Log16	775.9	x	369.4	x	106	x	127	x
Bw02	2.28	1.83	2.11	2.85	6.97	5.39	7.61	4.75
Bw03	194.6	126.3	62.3	139.6	x	x	x	47.27
Grid01	71.9	x	53.2	x	x	x	x	x
Ferry01	332.8	x	149.4	x	51.64	3416	63.02	4564

Table 1: Utility of fmutexes and bmutex. "x" sign means that problem could not be solved in 3 hours.

on an average by 2-5 times. For some specific seeds on some problems, we got speedup of up to 20x. The bigger speed up comes from parallel domains like Blockworld and Logistic, and the smaller speedups are from serial domains like Grid, Ferry, and Hanoi. Especially for problem Bw04, we were able to solve 8 out of 10 seeds if we include the bmutexes constraints, but only 4 of 10 for encoding without bmutexes in alloted time of 3 hours. Because we do not have a uniform results for both encodings, the speedup for this problems is calculated by dividing the average times we got for each type of encoding. The columns about average length of the solution in Table 2 indicate that we also normally get a smaller length when we include the bmutex constraints in the SAT encodings. These results on Relsat solver confirm our argument about the effects of bmutexes on searching times, and lengths of the solutions.

Even though we got good improvements of using Relsat solver for encoding with bmutexes, it is not the case for Satz solver. The inclusion of bmutexes actually slowed down the searching time for all tested problems. It is difficult to explain the different effects of bmutexes on Relsat and Satz, because even thought Satz and Relsat are both based on DPLL procedure (Bayardo & Schrag 1997; Li & Anbulagan 1997), they use different heuristic formulations to choose candidate set, and different scoring function to score the effect of doing full unit propagation on variables in the candidate set[4]. It is not clear that bmutexes have positive effect on Relsat because there is some relation between bmutex and conflict based backtracking and learning used by Relsat, or bmutex just suits the variable and value ordering heuristic used by Relsat.

Related work

Recently, several articles have been published which investigate the relevant effects in both Graphplan and SAT encodings of planning problem. In the original paper about Graphplan (Blum & Furst 1995), the authors discussed about "minimal action set", which basically has a similar effect like bmutex in restricting actions with the same relevant effects to appear in the solution. An idea similar to bmutex was proposed in (Kambhampati, Parker & Lambrecht 1997) but it was never implemented. Nebel et al, in (Nebel, Dimopoulos, & Koehler 1997), showed that even after eliminating a lot of irrelevant effects by using a fast, simple routine, the relevant part of Graphplan still suffers from left-over irrelevant facts and actions in the plan graph. They suggest the heuristic of choosing the minimum set of initial facts to achieve the goals. The heuristic works fine; and bmutex, by restricting the number of actions or conditions which can be chosen at each level, basically accomplishes similar effect.

There is also some work on investigating reachability-based and relevance-based constraints on SAT encoding. In (Brafman 1999), Brafman analyzed and tested the reachable-k, and relevant-k algorithms on both linear and Graphplan-based SAT encodings. The reachable-k algorithm is similar to the extended version of mutex propagation in graphplan, while the relevant-k algorithm is an backward-chaining version of reachable-k. Relevance-k algorithm differs from relevance-based bmutex discussed in this paper, even though they both work only on the relevant part of the plan graph. While bmutexes are constraints that rule out unnecessary actions, relevance-k tries to rule out every set of actions that can not exist together in any plan. Unfortunately, the paper only shows the result for reachable-1 and relevant-1.

While k-size reachability and relevance constraints have been discussed in literature, there has been no published account to our knowledge of anyone implementing them for ($K \succ 2$). As quoted in (Kautz & Selman 1999), Blum observes that computing higher order mutexes may not be useful. One reason for this has been the belief that it would be too heavy an overhead for too few mutexes. It is worth noting that while the overhead increase exponentially, k-size mutex constraints are getting weaker in ruling out branches as k increases; one reason why 2-size mutexes are the most popular constraints in all encoding.

Some researchers(Huang, Selman, & Kautz 1997) have

[4]Similar issues in explaining the performance of these SAT solvers was also faced by (Kautz & Selman 1999) and they settled on different scoring functions as the plausible explanation.

Name	fmutex		fmutex+bmutex		Speedup	
	time (s)	length	time (s)	length	Avg	Std.Dev
Log07	114.0	100.5/14	196.9	98/14	0.61x	0.37
Log15	2153.7	99/12	757.8	92.5/12	3.95x	4.85
Log16	775.9	84.5/16	369.4	84.8/16	2.17x	0.58
Log17	1744.2	78.1/17	743.3	80.8/17	2.53x	0.98
Bw03	194.6	16.7/5	62.3	16.3/5	5.58x	4.86
Bw04*	4257.3	20.0/6	1830.1	17.9/6	2.33	-
Grid01	71.9	14/14	53.2	14/14	1.45x	0.67
Grid02	220.5	22.5/21	405.5	21.8/21	0.87x	0.88
Ferry01	332.8	16/16	149.4	16/16	2.32x	0.77
Ferry02	21579.8	20/20	7310.8	20/20	3.06x	0.62
Hanoi04	56.0	15/15	47.5	15/15	1.19x	0.25

Table 2: Utilities of fmutex and bmutex for systematic Relsat solver. For problem Bw04, with bmutex, we can solve it with 8 of 10 seeds. Without bmutex, we can only solve 4 of 10 before running out of memory after about 3 hours of running time.

looked at including domain knowledge based constraints in SAT to speedup search. Even the newest Blackbox version (3.6) has the capability of adding domain specific information into the encoding. We see bmutex type relevance based constraints as an orthogonal approach to expedite search in a domain independent manner.

Future directions

In future, we plan to look at other forms of relevance based constraints. As an example, inseparability relations (insep) represent constraints that a set of conditions must be true together, or none of them can be true. Inseparability constraints are also propagated backward from the goals state following the rule:

- All goals are inseparable.
- Two set of action A and B are inseparable iff the set of facts supported by A is pair-wise inseparable with the set of facts supported by B.
- Two set of facts P and Q are inseparable iff the set of action supported by P is pair-wise inseparable with the set of action supported by Q.
- If two conditions are mutex, then they can not be inseparable.

Set A = (A_1,A_2...A_n) is inseparable with set B = (B_1,B_2....B_m) can be described in SAT as: ($A_1 \vee A_2$ ... $\vee A_n$) $\Rightarrow$ ($B_1 \vee B_2$.... $\vee B_m$) and ($B_1 \vee B_2 \vee$..... B_m) $\Rightarrow$($A_1 \vee A_2$ .... $\vee A_n$).

Unlike bmutex, in which we have strict control over the size of mutex that we want to find, inseps's size normally grows larger when we go further backward. The number of variable involved in insep clauses also gets bigger and this can weaken the constraints. Note that any insep constraint between two set only states that any one action in set A must be true with any one action in set B.

Incidentally, in PGraphplan(Blum & Langford 1999), an inseperability type value propagation heuristic (in the context of probabilistic planning) is implemented and the authors comment that bmutex form of constraints may be difficult to propagate. The relative promise of bmutex encourages us to investigate inseparability and other forms of domain-independent constraints that can guide search in planning by SAT, in future.

Conclusion

We investigated the effect of reachability-based fmutex and relevance-based bmutex on the solving of planning problems in SAT. The difficulty of a SAT instance depends on the number of variables and the number of clauses, so adding a large number of mutexes to the SAT encoding may indeed make the problem harder to solve. Our experiments shows that though forward mutexes are useful in general, there are domains in which fmutex constraints can slow down search. We introduced relevance-based backward mutex constraints and showed that even with small number of bmutex, the SAT encoding is significantly easier to solve in some domains along with fmutex. We provided some analysis on the utilities of bmutex for different SAT solvers and plan to investigate other forms of relevance based constraints and different SAT solvers in future.

References

Bayardo, R., and Schrag, R. 1997. Using CSP Look-Back Techniques to Solve Real-World SAT Instances. In the Proceedings of AAAI97.

Blum, A., and Furst, M. 1995. Fast planning through planning graph analysis. In the Proceedings of IJCAI-95, 1636-1642.

Blum, A., and Langford, J. 1999. Probabilistic Planning in the Graphplan Framework. In Proc. 5th European Conference on Planning.

Bonet, B., and Geffner, H. 1999. Planning as heuristic search: New results. In Proc. 5th European Conference on Planning.

Brafman, R. 1999. Reachability, Relevance, Resolution and the Planning as Satisfiability Approach. *IJCAI-99*.

Eugene, F., and Quiang, Y. 1992. Formalizing Plan Justifications. *In Proceedings of CSCSI-92, 9-14*.

Huang, Y.; Selman, B.; and Kautz, H. 1999. Control Knowledge in Planning: Benefits and Tradeoffs. *Proc. AAAI-99, Orlando.*.

Jeroslow, R.E., and J.Wang. 1990. Solving propositional satisfiability problems. Annals of Mathematics and AI1.

Kamphampati, S. 1999. Planning Graph as a (Dynamic) CSP: Exploiting EBL, DDB and other CSP Search Techniques in Graphplan. *To be published: Journal AI Research*.

Kambhampati, S.; Parker, E.; and Lambrecht, E. 1997. Understanding and extending graphplan. *Proc. ECP*.

Kautz, H. and Selman, B. 1996. Pushing the envelope: Planning, Propositional Logic and Stochastic Search. *Proc. AAAI 96*.

Kautz, H., and Selman, B. 1998. BLACKBOX: A New Approach to the Application of Theorem Proving to Problem Solving. *Workshop Planning as Combinatorial Search, AIPS-98, Pittsburgh, PA, 1998.*

Kautz, H., and Selman, B. 1999. Unifying SAT-based and Graph-based Planning. *Proc. IJCAI99.*

Li, C.M., and Anbulagan. 1997. Heuristics based on unit propagation for satisfiability problems. *Proc IJCAI-97*.

McDermott, D. 1998. AIPS-98 Planning Competition Results. At ftp.cs.yale.edu/pub/mcdermott/aipscomp-results.html.

Nebel, B.; Dimopoulos, Y.; and Koehler, J. 1997. Ignoring irrelevant facts and operators in plan generation. *Proc. ECP-97*.

Weld, D. 1999. Recent Advances in AI Planning. AI Magazine, 1999.