

Action Constraints for Planning

Ulrich Scholz

Darmstadt University of Technology
Alexanderstraße 10, 64283 Darmstadt, Germany
scholz@informatik.tu-darmstadt.de
<http://aida.intellektik.informatik.tu-darmstadt.de/~scholz/>

Abstract Recent progress in the applications of propositional planning systems has led to an impressive speed-up of solution time and an increase in tractable problem size. In part, this improvement stems from the use of domain-dependent knowledge in form of state constraints. In this paper we introduce a different class of constraints: *action constraints*. They express domain-dependent knowledge about the use of actions in solution plans and can express strategies which are used by human planners. The use of action constraints results in a tendency to better plans. We explain how to calculate and apply action constraints in the framework of parallel total-order planning, which is the design of the most powerful planners at the moment. We present two classes of action constraints and demonstrate their capabilities in the planner PROBAPLA.

1 Introduction

Recent progress in the applications of propositional planning systems has led to an impressive speed-up of solution time and an increase in tractable problem size. In part, this improvement stems from the use of domain-dependent knowledge [BK96, KS96]. This knowledge can be formalized as axioms about properties of plans and many planning formalisms can incorporate them easily. A domain-independent planning system has to generate these domain-dependent axioms automatically via preprocessing of the planning problem. By now, these preprocessing methods have concentrated on the generation of state constraints [GSb98, FL98] which are very successful. Their expressiveness does not cover the whole space of domain-dependent knowledge, which gives rise to the hope that other types of constraints can advance planning in a similar way.

Human planners take a different approach for using domain-dependent knowledge: They try to avoid actions and action sequences without useful effect or to try the best combination of actions for solving a subproblem. Similar to state constraints, this knowledge can be formulated as *action constraints*. These eliminate some preliminary plans and solutions from the search space which allows to solve larger problems in shorter time. An additional effect is a tendency to better plans.

Current planners use action constraints as heuristic [KS98] or as post-processing technique for specific domains [AK97] in a domain-dependent way. Our approach is different: We present action constraints for domain-independent planning. This requires to consider several issues which depend on each other: (1) What class of action constraints do we want to use? (2) How does the corresponding constraint look like for the used planning formalism? (3) How can we calculate the domain-dependent knowledge for a given planning problem? (4) How do we use the action constraint once it is calculated, and finally (5) what are the expected improvements of its use? The designer of a planning system has to weigh up these issues. This paper presents some points in her decision space and explains their interconnections.

The paper is organized as follows: In the next section we introduce action constraints and action patterns (1,2). The following section explains how to find and use action constraints based on patterns for a parallel total-order planner and for a given planning problem (2,3,4). Then we present two constraints (1): The *rsa* constraint deals with the subsumption of action sequences of the length two. The *top* constraint allows total-order planners to have a similar search space as partial-order ones. The final part of the paper demonstrates these constraints (5) and gives concluding remarks.

2 Action Constraints

The quality of a planning method is measured by three criteria: planning speed, the class of planning problems it can handle, and the quality of its solutions. State constraints are a successful method to improve the first two criteria, but unfortunately they are not helpful in regard of the third. Rating the quality of plans and searching for good solutions requires to discern between different solutions and to discard unwanted ones but state constraints hold for *all* plans. Furthermore, the common definition of a good plan is to be short and to contain few actions. These criteria do not involve properties of states. For this reason we introduce constraints about actions which are axioms that hold for all actions in a wanted solution.

Practical planning requires the use of fast and secure techniques. For this reason we introduce a restricted class of action constraints which are based on patterns of actions. Their use is similar to a keyhole technique used by human planners, which examine actions in adjacent time steps. In case that they find a suboptimal pattern of actions they remove it or replace it with a more optimal one. An example is the representation of a door. A human planner does not open and close a door without an action inbetween. Such a sequence of actions is suboptimal in any case and she detects them by examining a small part of a plan.

An *action pattern* is a set of actions together with their mapping to time steps, which is part of a plan. An action constraint is *based* on a set of patterns \mathcal{P} iff it eliminates all plans which contain a pattern of \mathcal{P} and holds for all other plans. *Replacing* an action pattern A with A' in a plan means to remove exactly the

actions of A and to insert exactly the actions of A' . An action pattern does not depend on other properties of plan, like the activity of facts or the time step it is placed at. This simplifies the calculation and application of the corresponding action constraint.

In this paper we use a compact way to write action patterns: Actions on the same time step are given as a set and adjacent time steps are connected by \circ . In case that we assign a single action to a time step, we drop the brackets. For the above example of representing a door, the action constraint can be formulated as $\forall a. name(a, 'open door') \rightarrow \nexists a'. a \circ a' \wedge name(a', 'close door')$. The replacement operation removes this action pattern by replacing it with the empty pattern.

The elimination of solutions from the plan space can make a planning problem unsolvable for a planning system, as it is the case with bounding the length of the considered plans. On the other hand we expect planning systems to compute a single solution and loosing some solutions can be an advantage. If a technique guarantees to preserve at least one solution of a solvable planning problem, it is safe to apply this technique. We call planning methods with that property *solution-safe*.

Some planning systems, like CNF-based ones, search for arbitrary solutions and stop after they find the first one. Action constraints which eliminate plans other than the optimal ones improve the solution quality of these planners. An action pattern A is called *suboptimal* according to an objective function o iff (1) every solution plan P which contains A can be transformed into a solution P' not containing A by replacing A with another action pattern, and (2) P' is better or at least of equal quality as P according to o . Action constraints based on suboptimal patterns are always solution-safe: The better plan does not contain the pattern, so that it complies with the constraint.

The elimination of suboptimal solutions and the general reduction of search space size has an effect on the planning speed and the size of tractable problems. Often it is harder to find an optimal solution than finding an arbitrary one, e. g. it is possible to find a solution to a `blocksworld` problem in polynomial time but finding an optimal solution is NP-complete [By194]. On the other hand, reducing the search space can result in a faster search, even if the solution set is not changed.

3 Generation and Use

The idea of action constraints is useful for planning in general. For example it seems to be helpful for any planning system and any planning problem to avoid action patterns without effect. Nevertheless, an action constraint for a specific planning problem is domain-dependent knowledge, so domain-independent planners have to calculate them from the problem description. Planning systems use various formalisms, representations, and search methods. Action constraints form one small wheel in this machinery and have to adopt its characteristics.

For example consider the definition of conflict: Parallel planners cannot assign actions to the same time step which add and delete the same fact. With linear

a_1 : pre \emptyset	a_2 : pre f_1	a_3 : pre f_1	a_4 : pre \emptyset
add f_1	add f_2	add \emptyset	add f_2
del \emptyset	del f_1	del f_1	del f_1

Figure 1. Example of simple action patterns. An action a *annuls* an action a' iff $\text{del}(a)$ is a superset of $\text{add}(a')$ and patterns of annulling actions can be suboptimal. The following three annulling action patterns are different in respect to their optimality and replacement operation: $a_1 \circ a_2$ is not suboptimal as f_2 can be necessary for a solution and a_2 depends on a_1 . The patterns $a_1 \circ a_3$ and $a_1 \circ a_4$ are suboptimal but require different replacement operations. The first one has no combined effect and can be removed. The second pattern has the combined effect of adding f_2 . Action a_4 does not depend on an effect of a_1 , so that a_1 can be removed.

planning methods these *add/del conflicts* cannot occur. Action patterns which are suboptimal for a linear planner can be optimal for a parallel one, if all its replacements result in an add/del conflict. Some action constraints are based completely on a characteristic of a planning formalism, so they are not usable for planners with different design. For example the patterns for the **top** constraint, explained in Sect. 5, are usable only for total-order planners.

A generic way to find a pattern-based action axiom for a planning problem is to enumerate and examine all patterns of the considered size. For many classes of constraints there are more efficient algorithms: The actions of a pattern are related, so that enumerating the facts required or changed by an action and the actions which require or change a fact cuts the search space. The same holds for finding a replacement pattern. After calculating all patterns it is easy to find the corresponding axiom: It has to be violated for every plan which contains a suboptimal pattern and not violated otherwise.

Even for simple classes of patterns it can be hard to specify a set of replaceable action patterns and their replacements. Figure 1 presents annulling pairs of actions. The patterns of this class are divided into three subclasses which have to be handled differently.

Planning systems use action constraints in various ways. The action axiom can be encoded into a plan space representation so that it does no longer include the corresponding suboptimal plans or it can be used to prune the search tree directly. In case that a planner calculates the corresponding replacement operation, it can perform the replacements during its search phase. If the replacement operation just eliminates an action, the planner can remove this action from the planning domain in advance. Ambite and Knoblock [AK97] perform the replacements as post-processing. They introduce a planning method which attempts to find an initial solution quickly. Then, they rewrite this possibly suboptimal plan by applying the action constraints as rewriting rules.

In this paper we assume a parallel total-order planning formalism. An action a consists of preconditions, added effects, and deleted effects, abbreviated by $\text{pre}(a)$, $\text{add}(a)$, and $\text{del}(a)$, respectively. Preconditions and effects are sets of positive facts without quantification, and actions are not allowed to add one of their

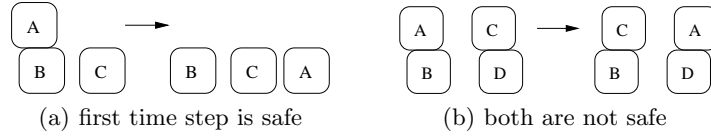


Figure 2. Examples for replaceable action sequences in the `blocksworld` domain. In Fig. 2(a) the sequence $\text{move}(A, B, C) \circ \text{move}(A, C, \text{Table})$ can be replaced in a solution-safe way by $\text{move}(A, B, \text{Table})$ on the time step of the first action. The table is always clear, so that this fact cannot be the cause of a conflict. In Fig. 2(b) we cannot replace any sequence if we want to find a plan with optimal length because it is not possible to insert the replacing action into the first or the second time step: Blocks A and C mutually occupy the target location of each other.

preconditions. An action in a plan has a conflict in case that one of its preconditions is inactive. In addition, two actions on the same time step have a conflict if one deletes a precondition or an added effect of the other. Several successful planning systems use this design for their internal representation: `GRAPHPLAN` as well as `BLACKBOX` and other planners of the AIPS’98 planning competition. In the following we present two action constraints, `rsa` and `top`.

4 Replaceable Sequences of Actions

In many planning domains there are sequences of actions which have the same effects than a single action.¹ Examples are planning domains with locations: Instead of moving the same object in subsequent time steps it is often possible to place it in its final location right away. Eliminating these `replaceable` sequences of `actions` `rsa` leads to more optimal plans. Due to the computational complexity we will examine only sequences of the length two which are not suboptimal otherwise.

A sequence $a_1 \circ a_2$ is replaceable by an action a but a cannot replace a_1 or a_2 directly if both a_1 and a_2 have weaker preconditions, more added effects, or less deleted effects than a . Their sequence is replaceable by a , so that each of a_1 and a_2 has to have the missing preconditions or deleted effects, and delete or require the extra added effects of the other. According to these cases, replaceable action sequences are defined as follows:

Definition 1. *The action sequence $a_1 \circ a_2$ is called replaceable by an action a in case that a fulfills*

$$\begin{aligned} \text{pre}(a) &\subseteq \text{pre}(a_1) \cup \text{pre}(a_2) \setminus \text{add}(a_1) \wedge \\ \text{add}(a) &\supseteq (\text{add}(a_1) \setminus \text{del}(a_2)) \cup (\text{add}(a_2) \setminus \text{pre}(a_1)) \wedge \\ \text{del}(a) &\subseteq (\text{del}(a_1) \setminus \text{add}(a_2)) \cup \text{del}(a_2) \end{aligned}$$

¹ The same is true for single actions. Due to the limited space we will not exemplify this case.

If no action is in parallel to a replaceable action sequence $a_1 \circ a_2$, the sequence is suboptimal and its replacement reduces the plan length by one time step. This is always the case for linear plans. Actions in parallel to the sequence can conflict with the replacing action. Placing a replacement in a time step is solution-safe if such a conflict cannot occur. Figure 2(a) exemplifies this case. If the replacing action can cause a conflict in both time steps, the replacement operation has to insert it into an additional time step. Figure 2(b) shows that this can result in plans with suboptimal length. Besides the mentioned add/del problem, these conflicts occur if a_1 adds a fact which is required at the next time step or a_2 deletes a fact which is required at the preceding time step. The replacing action can add these facts too late or delete them too early, respectively.

5 Searching a Total-Order Representation in Partial-Order Style

Beginning with the view of planning as satisfiability problem [KS92] and the introduction of planning graphs [BF95], the most powerful planners are based on a total-order plan representation. Despite this fact, it is well known that total-order planners can have an exponentially larger search space than partial-order ones [MBD94]. Moreover, the search space of a parallel total-order representation can be even larger.

The reason for the large number of totally ordered plans compared to partially ordered ones is their commitment to a specific order of unrelated actions. Parallel total-order representations have even more possible orderings than linear ones: They have ‘parallel’ as a third ordering relation. The search space of a partial-order and a total-order planner are related. A *totalization* of a partially ordered plan is a total order over the plan’s actions that is consistent with the existing partial order.²

In order to construct a totally ordered search space which is similar in size than the corresponding partial one, we partition the set of totally ordered plans such that its representative set is similar to the set of partially ordered plans. The **top** action constraint allows a **total-order planner** to eliminate all plans besides the representative set. It is related to the orderings introduced by a partial-order planning algorithm.

According to [BW94], a simple partial-order planner inserts an action a_1 together with the ordering constraint $a_1 \prec a_2$ into a plan P if a_1 adds an unsupported precondition f of a_2 . In addition, one of $a_3 \prec a_1$ or $a_2 \prec a_3$ is added if an action a_3 of P deletes f and could be ordered between a_1 and a_2 . All other action remain unordered. The planner starts with a plan containing a_0, a_∞ and $a_0 \prec a_\infty$, where a_0 adds all initial facts and a_∞ has the goal facts in its preconditions. A plan is a solution if it supports all preconditions of all of its actions.

We expect actions which are unordered in the partially ordered plan to be in parallel in its totally ordered counterpart. Unfortunately, actions which have

² This definition is similar to *linearization* in [MBD94], but the word linear in combination with parallel actions seems to be awkward.

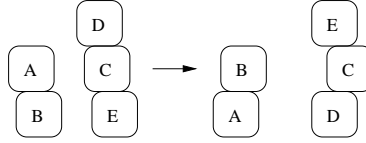


Figure 3. Example for the relation between a total-order and a partial-order search space. Flipping the right tower requires at least three time steps. The two actions for the left tower are $a_1 = \text{move}(A, B, \text{Table})$ and $a_2 = \text{move}(B, \text{Table}, A)$, and there are three different ways to assign them to the three time steps. Action a_1 adds the fact $\text{on}(A, \text{Table})$ which is required by a_2 , so that a partial-order planner orders them with $a_1 \prec a_2$. The corresponding **top** constraints require a_1 in the first time step and a_2 in the second, so that they eliminate the other two solutions.

an add/del conflict caused by unnecessary facts can remain unordered. For this reason we have to extend the above algorithm by the following operation: As part of the introduction of actions, the planner adds one of the ordering constraint $a \prec a'$ or $a' \prec a$ if a and a' are not ordered otherwise and they have an add/del conflict. A total-order planner using **top** action constraints has a solution space similar to the one of this extended planner. Figure 3 gives an example.

Theorem 1. *For every partially ordered plan P_{po} found by the extended algorithm there is exactly one totally ordered plan P_{to} which fulfills the following conditions:*

1. *In case that two actions a, a' in P_{po} are ordered with a constraint $a \prec a'$, a is placed in an earlier time step of P_{to} than a' .*
2. *For all actions a' of P_{to} (except the ones in the first time step) we have: If a' is in time step t , there is an action a in the time step directly preceding t such that the constraint $a \prec a'$ is in P_{po} .*

It is obvious that every partially ordered plan has a totalization which fulfills these conditions and vice versa. Also there can be no partially ordered plan which has two different totalizations of this kind. This would require an action a to have two possible time steps but a would have to observe the same ordering relations for both totalizations. This would violate condition two. The corresponding action constraint is $\forall a' \in P. (\nexists a. a \text{ is before } a' \text{ in } P) \vee (\exists a. a \circ a' \in P \wedge a \xrightarrow{\text{top}} a')$, where $a \xrightarrow{\text{top}} a'$ is defined as $\text{pre}(a) \cap \text{del}(a') \neq \emptyset \vee \text{del}(a) \cap \text{add}(a') \neq \emptyset \vee \text{add}(a) \cap \text{del}(a') \neq \emptyset \vee \text{add}(a) \cap \text{pre}(a') \neq \emptyset$. The relation $\xrightarrow{\text{top}}$ holds for all pairs of actions which are ordered by the partial-order planner.

Planners can use a subset of **top**-relations for an action constraint. An example is the domain-dependent heuristic ‘simplifying’ presented by Kauts and Selman [KS98]. The set of constraints resulting from this heuristic is subsumed by the set of **top** constraints.

Similar to the parallel case, **top**-style constraints can be used to eliminate unnecessary orderings in linear plans. Here, actions have at most one direct

domain	instance	nb actions	nb facts	optimal length	optimal size
blocksworld	bw_large.a	648	90	4	10
	bw_large.c	3150	240	8	18
logistics	logistics.d	7180	378	14	73*
	logistics.e	14244	538	14	87*
D^mS^{2*}	sfacts40	81	122	81	81

Table 1. Problem domains and instances used in this paper. The **blocksworld** domain does not have a robot arm. The instances `bw_large.a` and `bw_large.c` are well known from the literature. `logistics.d` is an instance of the logistics-typed-length domain, taken from the BLACKBOX distribution and `logistics.e` is an extension of it. The **D^mS^{2*}** domain is designed by Barret and Weld. The table lists the number of actions and facts of these domains, the minimal length of a parallel solution, and the minimal number of actions of that solution. Note that numbers marked with a ‘*’ are the best of our knowledge.

predecessor, so that the second condition of Theorem 1 does not apply. Instead, we assign an index to actions and disallow pairs of actions a, a' for which the following holds: (1) a is on an earlier time step than a' , (2) there are no actions a_1, \dots, a_n with $a \prec a_1 \prec \dots \prec a_n \prec a'$ and $a \xrightarrow{\text{top}} a_1 \xrightarrow{\text{top}} \dots \xrightarrow{\text{top}} a_n \xrightarrow{\text{top}} a'$, and (3) $\text{index}(a) > \text{index}(a')$. A linear planner using forward search, like TLPLAN [BK96], can use linear **top** constraints easily by maintaining a list of actions which can still meet condition (2) according to the definition of the **top** relation.

6 Results and Conclusions

We demonstrate the presented action constraints as part of the planner PROBAPLA on several large instances of the **blocksworld**, **logistics**, and **D^mS^{2*}** domain, as explained in Tab. 1. PROBAPLA is a parallel total-order planner which analyzes a planning domain to generate state and action constraints, [Sch97] describes an early version. PROBAPLA builds a bounded plan space representation which is searched via an encoding into CNF. The current version of PROBAPLA requires the specification of the desired plan length together with the planning problem. For this reason we cannot exemplify the effect of the action constraints on the solution length. In addition to **top** and **rsa**, PROBAPLA uses the simple action constraint **const** to eliminate inapplicable actions and generates **SMF** state constraints, as described in [Sch98].

Table 2 gives the time required for the calculation of the constraints and the number of patterns they find in the planning problems. The **blocksworld** domain has a high interconnection: Every block can be on top of each other and each action affects three blocks. This results in a high number of **top** relations and replaceable sequences together with a high calculation time of the patterns. As explained in Fig. 2(b), the replacement by moves from blocks to blocks is not safe, so that the fraction of solution-safe **rsa** is small.

The actions of the **logistics** domain are much less coupled, eg. airplanes can land on airports only but not on trucks. This results in a small number of

instance	t	#top	t	#rsa	#safe
bw_large.a	0.3	128664	4.3	8568	1008
bw_large.c	9.0	$1.2 \cdot 10^6$	177.9	79170	5460
logistics.d	0	4620	0.3	150	150
logistics.e	0	7680	0.7	276	276
sfacts40	0	4840	0	0	0

Table 2. Table presenting the time to generate the action constraints for the problem instances and the number of the corresponding patterns. Time is given in seconds, measured on a SUN Ultra 10 with 300 MHz and 196 MB. Columns labeled #top, #rsa, and #safe give the number of top-relations, replaceable sequences, and replaceable sequences which are solution-safe, respectively.

instance	PROBAPLA			with top		and with rsa		BLACKBOX		
	t	size	succ	t	size	t	size	t	size	succ
bw_large.a	1.0	10.9		1.3	10.6	5.1	10	2.2	10	
bw_large.c	6.4	30.5		16.2	30.1	185.4	29.7	—		0%
logistics.d	9.6	105.5		7.3	102.4	7.2	77.3	44.6	98.0	95%
logistics.e	27.7	125.0		70.1	123.2	54.0	97.6	104.7	116	56%
sfacts40	218.4	81	99%	142.8	81	142.8	81	—		0%

Table 3. Performance of PROBAPLA and BLACKBOX version 3.4. Times for PROBAPLA combine the time to generate the CNF formula and the time to solve this formula with satz-rand version 2.0. Both planners have the optimal plan length as input and the numbers are averaged over 100 runs. Columns labeled size give the total number of actions. In case a planner did not find a solution in every attempt, the column succ gives the percentage of successful runs. PROBAPLA is tested three times: without top and rsa, with top, and with both. The largest D^mS^{2*} instance solvable by BLACKBOX is **sfacts13** taking 38.1 seconds. PROBAPLA solves this instance with 27 actions and 41 facts in 1.7 seconds.

top relations and replaceable sequences and their calculation takes much less time. Note that all *rsa* of the *logistics* domain are solution-safe. For the D^mS^{2*} domain the number of top relations is high compared to the number of actions. It is a hard domain and PROBAPLA can solve only small problem instances, for which the calculation of the top relations is fast. This domain does not have replaceable action sequences and the search for *rsa* fails quickly.

Table 3 shows the trade-off between the time required to find a plan and the use of the action constraints *rsa* and *top*. In addition, it compares PROBAPLA to the planner BLACKBOX. The high interconnection in the *blocksworld* domain and the small number of solution-safe replaceable action sequences makes these actions constraints a bad choice for its instances: The increase in quality is minimal compared to the increase in planning time. This is different for the *logistics* domain. For the longer instance both *top* and *rsa* reduced the size of the solutions by 22% while the total run time was less than doubled. For problems of the D^mS^{2*} domain, a solution with optimal length has optimal size, too. The *top* constraint reduces the run time by one third.

These results show that the application of action constraints is not favorable for all planning domains: The designer of a planning system has to trade their potential effects against their time requirements. The `rsa` constraint shows the difficulties of finding the right choice: It results in little improvement for the `blocksworld` domain but performs much better for `logistics`. Nevertheless the replacement of action sequences is favorable if optimal plans are of high value or the computational effort of computing the constraints is small. The latter can be the case after the application of preprocessing techniques which reduce the number of actions, like the RIFO method [NDK97]. Furthermore Ambite and Knoblock [AK97] showed that `rsa` performs well as post-processing method for `blocksworld`.

In this paper we presented pattern-based action constraints as a mean to improve the quality to reduce the run time of planning systems. We explained the connections between the use of action constraints and the optimality of plans, showed that action constraints can be based solely on the specifics of a planning formalism, and gave examples how to calculate and use them. We presented two action constraints in detail and demonstrated their potential for improvements in planning speed and plan quality with the parallel total-order planner PROBA-PLA. For both constraints we explained their use with linear total-order planners. Finally we showed that the effects of an action constraint varies for different planning domains.

References

- [AK97] José Luis Ambite and Craig A. Knoblock. Planning by rewriting: Efficiently generating high-quality plans. In B. Kuipers and B. Webber, editors, *Proc. 14th National Conference on Artificial Intelligence*, pages 706–713, Providence, USA, July 1997. AAAI Press/MIT Press.
- [BF95] Avrim Blum and Merrick Furst. Fast planning through planning graph analysis. In C. Mellish, editor, *Proc. 13th International Joint Conference on Artificial Intelligence*, pages 1636–1642, Montréal, Canada, August 1995. Morgan Kaufmann.
- [BK96] Fahiem Bacchus and Froduald Kabanza. Using temporal logic to control search in a forward chaining planner. In M. Ghallab and A. Milani, editors, *New Directions in Planning*, pages 141–153. IOS Press, 1996.
- [BW94] Anthony Barrett and Daniel S. Weld. Partial-order planning: Evaluating possible efficiency gains. *Artificial Intelligence*, 67:71–112, 1994.
- [Byl94] Tom Bylander. The computational complexity of STRIPS planning. *Artificial Intelligence*, 69:165–204, 1994.
- [FL98] Maria Fox and Derek Long. The automatic inference of state invariants in TIM. *Journal of Artificial Intelligence Research*, 9:367–421, December 1998.
- [GSb98] Alfonso Gerevini and Lenhard Schubert. Inferring state constraints for domain-independent planning. In *Proc. 15th National Conference on Artificial Intelligence*, pages 905–912, Madison, USA, July 1998. AAAI Press/MIT Press.
- [KS92] Henry Kautz and Bart Selman. Planning as satisfiability. In B. Neumann, editor, *Proc. 10th European Conference on Artificial Intelligence*, pages 359–363, Vienna, Austria, August 1992. John Wiley & Sons, Ltd.

- [KS96] Henry Kautz and Bart Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proc. 13th National Conference on Artificial Intelligence*, pages 1194–1201, Portland, USA, August 1996. AAAI Press/MIT Press.
- [KS98] Henry Kautz and Bart Selman. The role of domain-specific knowledge in the planning as satisfiability framework. In *Proc. 4th Conference on Artificial Intelligence Planning Systems*, pages 181–189, Pittsburgh, USA, June 1998. AAAI PRESS/MIT Press.
- [MBD94] Steven Minton, John Bresina, and Mark Drummond. Total-order and partial-order planning: A comparative analysis. *Journal of Artificial Intelligence Research*, pages 227–262, December 1994.
- [NDK97] Bernhard Nebel, Yannis Dimopoulos, and Jana Koehler. Ignoring irrelevant facts and operators in plan generation. In S. Steel and R. Alami, editors, *Proc. 4th European Conference on Planning*, volume 1348 of *Lecture Notes in Computer Science*, pages 338–350, Toulouse, France, September 1997. Springer.
- [Sch97] Ulrich Scholz. Planning by local search. Diplomarbeit, Technische Universität Darmstadt, Fachbereich Informatik, Alexanderstraße 10, 64283 Darmstadt, Germany, December 1997.
- [Sch98] Ulrich Scholz. Strategien zur Domänenanalyse. In *12. Workshop "Planen und Konfigurieren" (PuK '98)*, Bericht tr-ri-98-193, Reihe Informatik, pages 17–22, FB Mathematik/Informatik, Warburgerstraße 10, 33098 Paderborn, Germany, April 1998.