

Planning with Specialized SAT Solvers

Jussi Rintanen

The Australian National University
Canberra, Australia

Abstract

Logic, and declarative representation of knowledge in general, have long been a preferred framework for problem solving in AI. However, specific subareas of AI have been eager to abandon general-purpose knowledge representation in favor of methods that seem to address their computational core problems better. In planning, for example, state-space search has in the last several years been preferred to logic-based methods such as SAT. In our recent work, we have demonstrated that the observed performance differences between SAT and specialized state-space search methods largely go back to the difference between a blind (or at least planning-agnostic) and a planning-specific search method. If SAT search methods are given even simple heuristics which make the search goal-directed, the efficiency differences disappear.

Introduction

A main benefit of logic-based representations is declarativity, the separation between the representation language and the algorithms for reasoning about the representation: a logic representation has a well-defined semantics that can be understood and reasoned about without knowing how it is going to be used computationally. Further, the automated reasoning methods available for example for the classical propositional logic are powerful, relatively simple, and due to their all-purpose nature, ideal for creating systems that combine multi-modal reasoning.

The promise of logic in planning was initially shown in the works by Kautz and Selman (1996) in the late 1990s, but has hence been overshadowed by more traditional algorithms, most notably state-space search with heuristic search algorithms (Bonet and Geffner 2001). The more open-ended state-space search framework has made it possible to easily combine several different search methods: most of the leading state-space search planners use multiple search modes and/or heuristics. However, progress after the initial successes of state-space search has been slow. As we will see in the experiments later, the scalability of the most recent planners is essentially at the same level of the planners from 2004 and even earlier. Also, the state-space search framework by itself does not support powerful modes of reasoning. For example, a large number of well-known state-space

search planners either limit to the simplest possible STRIPS language or have deficiencies in their handling more complex features, such as disjunction or conditionality, which in the SAT framework are trivial.

Arguably, the propositional logic provides a more flexible framework for representing the planning problem and additional features, such as control knowledge (Huang, Selman, and Kautz 1999) or symmetry-breaking constraints (Rintanen 2003), without needing any modifications in the search algorithm itself. But, the performance gap between the best SAT-based planners and the best planners overall has been perceived to be prohibitively wide, at least seemingly outweighing the benefits of the SAT framework.

However, we have recently shown that the performance gap disappears when introducing planning-specific heuristics to SAT solving: even a very simple one will lift the efficiency of SAT to the same level as with best state-space search planners (Rintanen a2010; b2010). These results, summarized in this paper, demonstrate that the advantages of declarative representation of planning problems don't conflict with efficiency. Interestingly, for finding *optimal* plans, SAT-based search (with reasonably weak assumptions) has recently been proved to be strictly more efficient than corresponding state-space search methods (Rintanen 2011b).

The structure of the paper is as follows. The next section explains the background of the work. Then we present the variable selection scheme (Rintanen a2010) and briefly discuss some heuristic extensions to it. In the experiments section we present a comparison between our planner and leading classical planners.

Preliminaries

The classical planning problem involves finding an action sequence from a given initial state to a goal state. The actions are deterministic, which means that an action and the current state determine the successor state uniquely. A state $s : A \rightarrow \{0, 1\}$ is a valuation of A , a finite set of *state variables*. In the simplest formalization of planning, actions are pairs (p, e) where p and e are consistent sets of propositional literals over A , respectively called *the precondition* and *the effects*. We define $\text{prec}((p, e)) = p$. Actions of this form are known as STRIPS actions for historical reasons. An action (p, e) is *executable* in a state s if $s \models p$. For a given state s and an action (p, e) executable in s , the unique

successor state $s' = \text{exec}_{(p,e)}(s)$ is determined by $s' \models e$ and $s'(a) = s(a)$ for all $a \in A$ such that a does not occur in e . This means that the effects are true in the successor state and all state variables not affected by the action retain their values. Given an initial state I , a plan to reach a goal G (a set of literals) is a sequence of actions o_1, \dots, o_n such that $\text{exec}_{o_n}(\text{exec}_{o_{n-1}}(\dots \text{exec}_{o_2}(\text{exec}_{o_1}(I)) \dots)) \models G$.

The basic idea in applying SAT to planning is, for a given set A of state variables, an initial state I , a set O of actions, goals G and a horizon length T , to construct a formula Φ_T such that $\Phi_T \in \text{SAT}$ if and only if there is a plan with horizon $0, \dots, T$. This formula is expressed in terms of propositional variables $a@0, \dots, a@T$ for all $a \in A$ and $o@0, \dots, o@T - 1$ for all $o \in O$. For a given $t \geq 0$, the valuation of $a_1@t, \dots, a_n@t$, where $A = \{a_1, \dots, a_n\}$, represents the state at time t . The valuation of all variables represents a state sequence so that the difference between two consecutive states corresponds to taking zero or more actions. This can be defined in several different ways (Rintanen, Heljanko, and Niemelä 2006). For our purposes it is sufficient that the step-to-step change from state s to s' by a set X of actions satisfies the following three properties: 1) $s \models p$ for all $(p, e) \in X$, 2) $s' \models e$ for all $(p, e) \in X$, and 3) $s' = \text{exec}_{o_n}(\text{exec}_{o_{n-1}}(\dots \text{exec}_{o_2}(\text{exec}_{o_1}(s)) \dots))$ for some ordering o_1, \dots, o_n of X .

Given a translation into propositional logic, planning reduces to finding a horizon length T such that $\Phi_T \in \text{SAT}$, and reading a plan from a satisfying assignment for Φ_T . To find such a T , early works sequentially tested Φ_1, Φ_2 , and so on, until a satisfiable formula was found. More efficient algorithms exist (Rintanen 2004).

The Variable Selection Scheme

After investigating several different ways of improving SAT-based planning by introducing better encoding schemes, we decided to look at the SAT solving process itself. The planners in the last 10 years had exclusively used the conflict-driven clause learning (CDCL) algorithm as their search method, as employed by the currently best SAT solvers (Moskewicz et al. 2001). The algorithm repeatedly chooses a decision variable, assigns a truth-value to it, and performs inferences with the unit resolution rule, until a contradiction is obtained (the empty clause is derived, or, equivalently, the current valuation falsifies one of the input clauses or derived clauses.) The sequence of variable assignments that led to the contradiction is analyzed, and a clause preventing the repeated consideration of the same assignment sequence is derived and added to the clause set.

Our first attempt to improve CDCL was to force it to choose as branching variables actions that contribute to the top-level goals. Generic SAT solvers and the VSIDS heuristic used by them choose branching variables blindly (from the point of view of the planning process), and our idea was to introduce a small bias to this process. However, it turned out that one does not have to use any of the strength of the VSIDS heuristic, as simply forcing the CDCL algorithm to do a plain form of depth-first backward chaining search is already a dramatic improvement.

The variable selection scheme (Rintanen a2010) is based on the following observation: each of the goal literals has to be made *true* by an action, and the precondition literals of each such action have to be made *true* by earlier actions (or, alternatively, these literals have to be *true* in the initial state.)

The first step in selecting a decision variable is finding the earliest time point at which a goal literal can become and remain *true*. This is by going backwards from the end of the horizon to a time point t' in which A) an action making the literal *true* is taken or B) the literal is *false* (and the literal is *true* or *unassigned* thereafter.) The third possibility is that the initial state at time point 0 is reached and the literal is *true* there, and hence nothing needs to be done. In case A we have an action already in the plan, and in case B we choose any action that can change the literal from *false* to *true* between t' and $t' + 1$ and use it as a decision variable.¹ In case A we push the literals in the precondition into the stack and find supporting actions for them.

The first version of our algorithm (Rintanen a2010), finds just one action in a depth-first manner, yielding an impressive performance, in comparison to earlier SAT-based planners. The second variant of the algorithm (Rintanen b2010) increased the performance still further, now surpassing the performance of best existing planners based on any search method. This second variant differs in two respects. First, its depth-first search is not terminated after one action is found, but proceeds further to identify several actions (10 in the algorithm description given below), one of which will be randomly chosen as the decision variable, to allow much earlier actions as decision variables than just those supporting the current subgoal. Second, we replaced the stack with a priority queue, which enables more flexible traversal orders.

The relaxed selection of branching variables that does not follow a strict backward chaining scheme was the stronger of the two improvements. However, a small modification to the initial selection scheme was vital: the set of multiple candidate actions computed should all contribute to the same top-level goal, and actions contributing to other top-level goals are ignored (Rintanen b2010).

The priority queue is controlled by a heuristic that orders the subgoals. When the preconditions of an action at time t become new subgoals and are pushed into the queue, we give a preference to the precondition which must have been true longer before t (i.e. its value is *true* for a higher number of time points preceding t). There are often implicit ordering constraints between the preconditions of an action, and this heuristic expresses a natural intuition about such orderings: because we always choose earliest possible actions that support a subgoal, the subgoals which must be satisfied earlier should be processed first, to avoid trying to satisfy the subgoals in the wrong order.

The algorithm for computing a set of actions that support currently unsupported top-level goals or preconditions of actions in the current partial plan is given in Fig. 1. For negative literals $l = \neg a$, $l@t$ means $\neg(a@t)$, and for positive literals $l = a$ it means $a@t$. Similarly, we define the valuation

¹Such an action must exist because otherwise the literal would have to be *false* also at $t' + 1$.

```

1: procedure support( $G, O, T, v$ )
2: empty the priority queue;
3: for all  $l \in G$  do push  $l@T$  into the queue;
4:  $X := \emptyset$ ;
5: while the priority queue is non-empty and  $|X| < 10$  do
6: pop  $l@t$  from the priority queue;
7:  $t' := t - 1$ ;
8: found := 0;
9: repeat
10: if  $v(o@t') = 1$  for some  $o \in O$  with  $l \in \text{eff}(o)$ 
11: then (* The subgoal is already supported. *)
12:   for all  $l' \in \text{prec}(o)$  do push  $l'@t'$  into the queue;
13:   found := 1;
14: else if  $v(l@t') = 0$  then
15:    $o :=$  any  $o \in O$  such that  $l \in \text{eff}(o)$  and  $v(o@t') \neq 0$ ;
16:    $X := X \cup \{o@t'\}$ ;
17:   for all  $l' \in \text{prec}(o)$  do push  $l'@t'$  into the queue;
18:   found := 1;
19:    $t' := t' - 1$ ;
20: until found = 1 or  $t' < 0$ ;
21: end while
22: return  $X$ ;

```

Figure 1: Computation of supports for (sub)goals

```

1:  $S := \text{support}(G, O, T, v)$ ;
2: if  $S \neq \emptyset$  then  $v(o@t) := 1$  for any  $o@t \in S$ ;
3: else
4: if there are unassigned  $a@t$  for  $a \in A$  and  $t \in \{1, \dots, T\}$ 
5: then  $v(a@t) := v(a@(t-1))$  for  $a@t$  with minimal  $t$ 
6: else  $v(o@t) := 0$  for any unassigned  $o@t$ ;

```

Figure 2: Variable selection for the CDCL algorithm

$v(l@t)$ for negative literals $l = \neg a$ by $v(l@t) = 1 - v(a@t)$ whenever $v(a@t)$ is defined. For positive literals $l = a$ of course $v(l@t) = v(a@t)$.

The procedure in Fig. 1 is the main component of the variable selection scheme for CDCL given in Fig. 2, in which an action is chosen as the next decision variable for the CDCL algorithm if one is available. If none is available, all subgoals are already supported. Some unassigned variables still typically remain, and the remaining fact variables are assigned the value they have in the predecessor state (line 5) and the action variables the value *false* (line 6). The code in Fig. 2 replaces VSIDS in the CDCL algorithm.

Comparison

We have compared our planner with the new heuristic (Mp), the same planner with the VSIDS heuristic (M), and the leading classical planners that use other search methods.

The test material was over one thousand instances from the planning competition problem sets from 1998 until 2008. Since the variable selection scheme is defined for the STRIPS language only, we chose all the STRIPS problems²,

²An extension of the heuristics for PDDL with conditional effects and disjunction yields similar results (Rintanen 2011a).

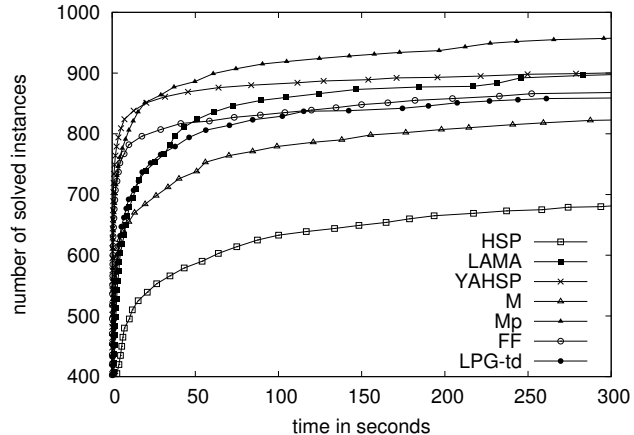


Figure 3: Number of instances that are solved in a given time

except that we did not choose benchmarks from an earlier competition if the same domain had been used in a later competition as well. We also excluded Schedule (2000) because to most planners it is difficult to ground efficiently. It is solved very efficiently by our planner and some other planners after it has been grounded.

All the experiments were run in an Intel Xeon CPU E5405 at 2.00 GHz with a minimum of 4 GB of main memory and using only one CPU core. We ran our planner for all of the problem instances, giving a maximum of 300 seconds for each instance. The runtime includes all standard phases of a planner, starting from parsing the PDDL description of the benchmark and ending in outputting a plan.

We compared our planners to LAMA (Richter, Helmert, and Westphal 2008), the winner of the last (2008) planning competition, and YAHSP (Vidal 2004) which placed second in the 2004 competition. These two planners appear to be the best performing classical planners until now. We ran the planners with their default settings, except for limiting LAMA’s invariant computation to a maximum of 60 seconds according to Helmert’s instructions, to adjust for the 300 second time limit we used.³

The configuration of our planner Mp is as in earlier papers (Rintanen a2010; b2010). The planner uses the \exists -step semantics encoding by Rintanen et al. (2006) and the algorithm B of Rintanen et al. (2006) with $B = 0.9$, testing horizon lengths 0, 5, 10, 15, ... and solving a maximum of 18 SAT problems simultaneously. M differs only in its use of VSIDS instead of the new heuristic.

The results are summarized in Figure 3, also including FF and LPG-td from the older planners. The curves show the number of problem instances solved (y axis) with a given time limit (x axis). The numbers of solved problems by benchmark domain are listed in Table 1. The first column is the number of (solvable) problem instances in each domain.

³The runtimes are with fixes to bugs in LAMA that affected Philosophers and Optical-Telegraph.

		Mp	MLAMA	YAHSP	
1998-GRID	5	4	2	5	5
1998-GRIPPER	20	20	20	20	20
1998-LOGISTICS	30	30	29	28	30
1998-MOVIE	30	30	30	30	30
1998-MPRIME	20	19	16	20	17
1998-MYSTERY	19	17	16	19	16
2000-BLOCKS	102	90	71	51	42
2000-LOGISTICS	76	76	76	76	76
2002-DEPOTS	22	22	21	16	18
2002-DRIVERLOG	20	20	15	20	20
2002-FREECELL	20	10	4	18	18
2002-ZENO	20	20	18	20	20
2004-AIRPORT	50	43	40	37	34
2004-OPTICAL-TELEGRAPH	14	14	14	2	13
2004-PHILOSOPHERS	29	29	29	11	29
2004-PIPESWORLD-NOTANK	50	32	15	44	50
2004-PSR-SMALL	50	50	50	50	50
2004-SATELLITE	36	32	29	30	36
2006-PATHWAYS	30	30	30	30	20
2006-PIPESWORLD	50	21	9	38	41
2006-ROVERS	40	40	40	40	27
2006-STORAGE	30	30	29	18	23
2006-TPP	30	30	26	30	30
2006-TRUCKS	30	30	19	8	11
2008-CYBER-SECURITY	30	30	30	18	12
2008-ELEVATORS	30	30	13	30	21
2008-OPENSTACKS	30	15	15	30	30
2008-PARCPRINTER	30	30	30	28	30
2008-PEGSOLITAIRE	30	30	25	29	29
2008-SCANALYZER	30	27	19	27	27
2008-SOKOBAN	30	6	2	18	18
2008-TRANSPORT	30	20	10	28	30
2008-WOODWORKING	30	30	30	28	28
total	1093	957	822	897	901
weighted score	33	29.10	24.86	27.53	28.23

Table 1: Number of instances solved in 300 seconds

The weighted score is the sum of the proportions of solved instances for every domain. Our earlier experiments (Rintanen a2010; b2010) showed that the quality of the plans produced by Mp is roughly the same as LAMA's.

Conclusions

We have presented simple heuristics for controlling the conflict-driven clause-learning algorithm when it is solving a planning problem which has been reduced to SAT. The performance of the resulting planner compares very favorably with best earlier planners.

A notable difference between our work and VSIDS (Moskewicz et al. 2001) is that we are not using weights of decision variables obtained from conflicts as a part of variable selection. Such weights would be able to order the top-level goals and subgoals in the computation of actions, based on their role in conflicts. This is a promising area for future improvement in the implementations of the heuristic.

References

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129(1-2):5–33.

Huang, Y.-C.; Selman, B.; and Kautz, H. 1999. Control knowledge in planning: benefits and tradeoffs. In *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI-99) and the 11th Conference on Innovative Applications of Artificial Intelligence (IAAI-99)*, 511–517. AAAI Press.

Kautz, H., and Selman, B. 1996. Pushing the envelope: planning, propositional logic, and stochastic search. In *Proceedings of the 13th National Conference on Artificial Intelligence and the 8th Innovative Applications of Artificial Intelligence Conference*, 1194–1201. AAAI Press.

Moskewicz, M. W.; Madigan, C. F.; Zhao, Y.; Zhang, L.; and Malik, S. 2001. Chaff: engineering an efficient SAT solver. In *Proceedings of the 38th ACM/IEEE Design Automation Conference (DAC'01)*, 530–535. ACM Press.

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks revisited. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI-08)*, 975–982. AAAI Press.

Rintanen, J.; Heljanko, K.; and Niemelä, I. 2006. Planning as satisfiability: parallel plans and algorithms for plan search. *Artificial Intelligence* 170(12-13):1031–1080.

Rintanen, J. 2003. Symmetry reduction for SAT representations of transition systems. In Giunchiglia, E.; Muscettola, N.; and Nau, D., eds., *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling*, 32–40. AAAI Press.

Rintanen, J. 2004. Evaluation strategies for planning as satisfiability. In López de Mántaras, R., and Saitta, L., eds., *ECAI 2004. Proceedings of the 16th European Conference on Artificial Intelligence*, 682–687. IOS Press.

Rintanen, J. 2011a. Heuristics for planning with SAT and expressive action definitions. In *ICAPS 2011. Proceedings of the Twenty-First International Conference on Automated Planning and Scheduling*. AAAI Press. to appear.

Rintanen, J. 2011b. Planning with SAT, admissible heuristics and A*. In Walsh, T., ed., *Proceedings of the 22th International Joint Conference on Artificial Intelligence*. AAAI Press. to appear.

Rintanen, J. 2010. Heuristics for planning with SAT. In Cohen, D., ed., *Principles and Practice of Constraint Programming - CP 2010, 16th International Conference, CP 2010, St. Andrews, Scotland, September 2010, Proceedings.*, number 6308 in Lecture Notes in Computer Science, 414–428. Springer-Verlag.

Rintanen, J. 2010. Heuristic planning with SAT: beyond uninformed depth-first search. In Li, J., ed., *AI 2010: Advances in Artificial Intelligence: 23rd Australasian Joint Conference on Artificial Intelligence, Adelaide, South Australia, December 7-10, 2010, Proceedings*, number 6464 in Lecture Notes in Computer Science, 415–424. Springer-Verlag.

Vidal, V. 2004. A lookahead strategy for heuristic search planning. In Zilberstein, S.; Koehler, J.; and Koenig, S., eds., *ICAPS 2004. Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling*, 150–160. AAAI Press.