Deconstructing Planning as Satisfiability

Henry Kautz
Department of Computer Science & Engineering
University of Washington
Seattle, WA 98195

Introduction

The idea of encoding planning as satisfiability was proposed in 1992 as a method for generating interesting SAT problems, but did not appear to be a practical approach to planning (Kautz & Selman 1992). This changed in 1996, when Satplan was shown to be competitive with current planning technology, leading to a mini-explosion of interest in the approach (Kautz & Selman 1996). Within a few years, however, heuristic search planning appeared to be vastly superior to planning as satisfiability, and many researchers wrote off the earlier success of the approach as a fluke. It was therefore rather surprising when Satplan won first place for optimal STRIPS planning in the 2004 ICAPS planning competition (Edelkamp *et al.* 2004). This talk will attempt to deconstruct the reasons for Satplan's successes and failures, and discuss ways the approach might be extended to handle "open" domains, metric constraints, and domain symmetries.

A Brief History of Satplan

The original Satplan "system" was actually a set of conventions for encoding STRIPS-style linear planning problems in propositional axiom schemas (Kautz & Selman 1992). The key technical advance in 1996 was introducing so-called parallel encodings, where several non-interfering actions could occur at the same time step (Kautz, McAllester, & Selman 1996). The first complete implementation of Satplan that took STRIPS notation as input was the MEDIC system of Ernst *et al.* (1997). The next year saw the release of Blackbox (Kautz & Selman 1998), which also performed "mutex propagation", a form of local-consistency reasoning introduced by Graphplan (Blum & Furst 1995), before generating each encoding. In the 1998 International Planning Competition, Blackbox's performance was comparable to the best entrants, which were by and large variations of Graphplan. However, in the 2000 IPC, Blackbox's performance was abysmal. Planning was dominated by Graphplan-style and state-space style heuristic search. No version of Satplan entered the 2002 competition.

A new implementation, Satplan04, was entered in the 2004 IPC (and in the 2006 competition, which is ongoing at the time this is written). In the track for optimal propositional (*i.e.*, non-metric) planning, Satplan was the winner by a wide margin (Edelkamp *et al.* 2004). It was judged to be the fastest and most robust planner in five domains, and the 2nd best in two domains. The best any other system did was placing 2nd in two domains.

Deconstructing Satplan's Success

Satplan04 did not include any new kinds of plan encodings or pre-processing. In fact, due to lack of time, the version that entered the competition did not even include the mutex propagation routine that was so crucial to blackbox's performance. The dramatic comeback for Satplan can be attributed to two factors: first, satisfiability solvers have steadily increased in power; and second, the optimal planning problems in the 2004, unlike nearly all of the problems in the previous competitions, were *intrinsically hard*.

Improvements in SAT Solvers

Table 1 illustrates the steady improvement in SAT solvers from 1997 to present on a series of optimal planning problems from IPC-4. Sato (Zhang 1997) and satz (Li & Anbulagan 1997) were state-of-the-art solvers in 1998. zChaff (Moskewicz *et al.* 2001) established the standard for true "industrial strength" SAT solving, and introduced novel data structures that could efficiently manage millions of learned (cached) clauses. Jerusat (Nadel 2002) improved on zChaff's branching and backtracking strategies, and won the the 2004 SAT competition industrial benchmark category.

It is remarkable that the two most recent systems, MiniSat (Eén & Sörensson 2003) and Siege (Ryan 2004), can solve problems with with 0.25 million variables in less than 30 seconds. Furthermore, while Siege is a proprietary, highly optimized system, the comparably fast system, MiniSat, is a straightforward, open-source implementation of the best recent published SAT techniques.

Intrinsically Hard Planning Problems

While the general STRIP planning problem is formally hard, particular planning *domains* may be solvable in low polynomial time. It is certainly the case that many kinds of day-to-day planning domains can be solved exactly and efficiently by both people and machines. Efficient domain-specific planning algorithms could be learned, evolved, or

wff	vars	clauses	sato 1997	satz 1997	zChaff 2001	jerusat 2002	MiniSat 2003	siege 2004
p05	3,656	31,089	13.23	0.61	0.01	0.01	0.02	0.01
p15	10,671	143,838	x	4.85	0.05	0.13	0.09	0.03
p18	34,325	750,269	x	x	13.92	6.59	2.55	4.85
p20	40,304	894,643	x	x	14.75	10.35	10.03	8.68
p28	249,738	13,849,105	x	x	846.72	79.59	27.80	12.74

Table 1: Comparison of SAT solvers on optimal planning problems from the Airport domain of IPC-4. Time in seconds for solving the final formula of optimal length on an Intel Xeon 2.8 GHz processor. Encodings were generated by Satplan04 using the "action-based" encoding. x = solver segfaulted or failed to solve the problem in 24 hours.

programmed. Furthermore, planning domains that might appear at first blush to be hard can be easy under a relaxed solution criteria. For example, although finding shortest plans in the blocks world is NP-hard, finding plans that are no more than twice as long as optimal is polynomial. Again, planning problems with relaxed solution criteria frequently occur in everyday life, and are solved with little conscious effort.

What is the point, then, of *domain independent* planning? One can argue that it is precisely to solve problems that do *not* admit a polynomial time solution. Planning is what we do when evolution, experience, and algorithmic analysis fails. On a more down to earth note, commercial applications of planning usually involve solving intrinsically hard problems.

The planning problems in the IPC competitions prior to 2004 were by and large easy to solve by heuristic forward-chaining search; in fact, it was possible to *prove* that many of the domains could be solved optimally with only a polynomial number of backtracks (Hoffmann 2002). By contrast, many of the 2005 IPC planning domains were deliberately designed to reflect real-world planning problems (Hoffmann *et al.* 2004) and were not (at least obviously) PTIME-solvable.

It is also possible to transform planning domains where feasible (non-optimal) planning is easy but optimal planning is hard to ones where feasible planning is also hard (Huang 2002). Such instances are useful for comparing planning algorithms without the complexity of taking solution quality into consideration.

Leveraging Plan Structure

What, then, distinguishes planning from pure satisfiability testing, or any other combinatorial search problem? The distinguishing feature is the regular *structure* of all planning problems; for example: the way that the evolution of each fluent over time is represented by sequence of propositions, and the way in which propositions representing actions are linked by clauses to those representing preconditions and effects. Further structure is generated by operator *schemas*: the set of propositions that result from instantiating an operator with different parameters connect to other propositions in a similar pattern. These patterns of structure can be viewed as various kinds of *symmetries* in the encoded problem.

Symmetry detection has been an important theme of much work in constraint satisfaction, satisfiability testing and planning (Joslin & Roy 1997; Shlyakhter 2001; Fox & Long 1999). While much of this work has concentrated on *automatically* detecting symmetries, it is often more straightforward and practical to provide the author of a planning domain with an easy way to state high-level symmetries (Beame, Agarwal, & Kautz 2003; Sabharwal 2005). In many domains either approach to exploiting symmetric structure (which in fact may both be employed) can lead to orders of magnitude speed-up.

Beyond SAT

The basic idea of encoding a classical planning problem as Boolean satisfiability can be generalized to encode more complex forms of planning to satisfiability problems in languages that are higher in the computational complexity hierarchy. For example, conditional planning in nondeterministic domains can be encoding as evaluating the truth of a quantified Boolean formula (QBF) (Rintanen 1999). Probabilistic planning can be encoded as *stochastic satisfiability*, an extension of QBF that includes a "random choice" quantifier (Majercik & Littman 1999).

Many planning problems involve reasoning about metric quantities, such as resources and metric time. One way to handle metric information is to compile a planning problem to both a Boolean formula and a set of linear inequalities, where the each inequality is associated with a particular proposition (Wolfman & Weld 1999; Shin & Davis 2005). An alternative approach would be to encode metric constraints as Boolean arithmetic in clausal form. Although such an approach may seem *prima facie* impractical, compiling metric constraints as SAT actually works well for some kinds of verification problems (Seshia 2005). Given the close connections between planning and verification, investigating the approach for metric planning appears worthwhile.

Currently the most fundamental limitation of the planning as satisfiability approach is the need to instantiate the entire propositional formula before attempting to solve it. If the domains contains many objects and predicates with multiple arguments, the instantiated formula may far exceed the memory limits of any current machine. Fully lifted (first-order resolution based) satisfiability testing is quite impractical. It may be fruitful, however, to explore the space of "slightly lifted" approaches. In particular, the fact that in

the solution to most planning problem nearly all predicates are false and nearly all clauses are trivially satisfied suggests that instantiation of positive predicates can be interleaved with search.

References

Beame, P.; Agarwal, A.; and Kautz, H. 2003. Using problem structure for efficient clause learning. In *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT 2003)*, 159–166.

Blum, A., and Furst, M. 1995. Fast planning through planning graph analysis. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 95)*, 1636–1642.

Edelkamp, S.; Hoffmann, J.; Littman, M.; and Younes, H. 2004. The 2004 international planning competition. http://ls5-www.cs.uni-dortmund.de/ edelkamp/ipc-4/main.html.

Eén, N., and Sörensson, N. 2003. An extensible sat-solver. In *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing*.

Ernst, M.; Millstein, T. D.; and Weld, D. S. 1997. Automatic SAT-compilation of planning problems. In *IJCAI*, 1169–1177.

Fox, M., and Long, D. 1999. The detection and exploitation of symmetry in planning problems. In *IJCAI*, 956–961.

Hoffmann, J.; Edelkamp, S.; Englert, R.; Liporace, F.; Thiébaux, S.; and Trü, S. 2004. Towards realistic benchmarks for planning: the domains used in the classical part of ipc-4.

Hoffmann, J. 2002. Local search topology in planning benchmarks: A theoretical analysis. In *Proceedings of the 6th International Conference on Artificial Intelligence Planning and Scheduling*.

Huang, Y.-C. 2002. Learning control knowledge for planning. PhD Thesis, Cornell University.

Joslin, D., and Roy, A. 1997. Exploiting symmetry in lifted CSPs. In *AAAI/IAAI*, 197–202.

Kautz, H., and Selman, B. 1992. Planning as satisfiability. In *Proceedings of the 10th European Conference on Artificial Intelligence*, 359–363. Wiley.

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 (AAAI-96)*, 1194–1201. AAAI Press. (Best Paper Award).

Kautz, H., and Selman, B. 1998. Blackbox: A new approach to the application of theorem proving to problem solving. In *AIPS-98 Workshop on Planning as Combinatorial Search*, 58–60.

Kautz, H.; McAllester, D.; and Selman, B. 1996. Encoding plans in propositional logic. In *Proceedings of the 4th International Conference on Knowledge Representation and Reasoning (KR-96)*, 374–385. Morgan Kaufmann.

Li, C. M., and Anbulagan. 1997. Heuristics based on unit propagation for satisfiability problems. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 366–371.

Majercik, S. M., and Littman, M. L. 1999. Contingent planning under uncertainty via stochastic satisfiability. In *AAAI/IAAI*, 549–556.

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 Design Automation Conference (DAC'01)*.

Nadel, A. 2002. Backtrack search algorithms for propositional logic satisfiability: Review and innovations. Master's Thesis, Hebrew Univesity.

Rintanen, J. 1999. Constructing conditional plans by a theorem-prover. *Journal of Artificial Intelligence Research* 10:323–352.

Ryan, L. O. 2004. Efficient algorithms for clause learning sat solvers. Masters Thesis, Simon Fraser University.

Sabharwal, A. 2005. Symchaff: A structure-aware satisfiability solver. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI)*, 467–474.

Seshia, S. A. 2005. Adaptive eager boolean encoding for arithmetic reasoning in verification.

Shin, J.-A., and Davis, E. 2005. Processes and continuous change in a sat-based planner. *Artificial Intelligence* 166:194–253.

Shlyakhter, I. 2001. Generating effective symmetry-breaking predicates for search problems. In *Proceedings of the Workshop on Theory and Applications of Satisfiability Testing*.

Wolfman, S. A., and Weld, D. S. 1999. The lpsat engine and its application to resource planning. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, 310–315.

Zhang, H. 1997. Sato: An efficient propositional prover. In *Proceedings of the International Conference on Automated Deduction*, 272–275.