Planning with Tests, Branches, and Non-Deterministic Actions as Satisfiability

Walter S. Lasecki and **Henry Kautz**
University of Rochester Computer Science
{wlasecki, kautz}@cs.rochester.edu
Technical Report #979

Abstract

We present an effective SAT encoding of planning with partial knowledge, tests, branches, and non-deterministic actions. As in recent work on compiling conformant and contingent planning into STRIPS, our encoding is based on representing knowledge states. Unlike previous approaches, however, fluents are conditioned on threads of execution, rather than on alternative choices of the initial state. Tests and other non-deterministic actions activate threads of execution, which are later deactivated by join operations.

The experiments we present in this paper compare our SAT approach with state of the art heuristic search planners on contingent and conformant planning problems. We then present results on solving conditional planning problems that cannot even be represented by competing approaches.

Introduction

Planning as satisfiability is an effective strategy for finding timespan-minimum solutions to deterministic planning problems (Kautz and Selman 1996; Kautz, McAllester, and Selman 1996; Kautz and Selman 1999). In this paper, we present a SAT encoding for *non-deterministic* domains where the system may have incomplete knowledge of the state of the world. The approach can be used to synthesize plans that include tests and branches, including nested tests, as well as arbitrary non-deterministic actions. The intuition behind the encoding is that a plan executes along a set of *threads*. Tests and non-deterministic actions split a thread into a set of threads. At any point in time, the system can be in different knowledge states on different threads. A merge operation combines threads by taking the intersection of what is known on all of the operation's threads.

The class of problems that can be represented by the system subsumes and is strictly more general than conformant planning and contingent planning problems. We describe the connections between our encoding and recent work on translating both conformant and contingent problems to STRIPS-style planning.

In addition to presenting the encoding and discussing its properties, we present experimental results from a simple, non-optimized implementation called T-Satplan, for "threaded Satplan". We show how the approach scales on a class of general non-deterministic planning problems, and provide some comparisons on contingent planning domains with a state of the art heuristic search planner. While the heuristic search approach is more efficient on domains that require little or no backtracking, as one might expect (Hoffmann 2005a), the SAT approach has an advantage on domains that require short makespans and many parallel actions, and allow reuse of threads.

Encoding

We describe the encoding for the case of atomic actions and fluents. Parameterized operators can be handled by instantiation followed by translation. Each fluent p is is represented by a pair of predicates, Kp and $K\neg p$, that stand for the system's knowledge state regarding the proposition. These knowledge predicates are indexed by a time step t and thread i. For example, $\neg Kp(t,i) \;\; \& \;\; \neg K\neg p(t,i)$ means the system does not know whether or not p holds at time t when executing thread i. A deterministic action a is also represented by a predicate that takes a time step and thread as arguments, with the convention that the effects of an action hold at the following time step on the same thread. Multiple non-conflicting deterministic actions may hold at the same time step on a thread. In many domains, it is necessary to be able to express *invariants*, formulas that must hold in every state, in addition to operators. For example, consider a robot domain where a key may be in one of three rooms, and the robot must open a locked door. If the robot does not find the key in the first two rooms, the system must be able to conclude that the key is in the third room, otherwise the problem would have no solution.

A non-deterministic action a is represented by a predicate whose arguments are a time step and sequence of N threads, where the action's preconditions hold on the first thread, and the N possible outcomes hold at the next time step on each corresponding thread. For example, the non-deterministic action of rolling a die at time t would be represented by a proposition of the form $a(t, i_1, \ldots, i_6)$. An important special case of non-deterministic actions are those that test (observe) some fluent p. The preconditions for a test should include that p is not known, and may include other conditions as needed. The action would have two possible outcomes, one asserting Kp and the other asserting $K\neg p$.

There are three special predicates:

- active(t, i) means thread i is active at time t.
- merge(t, i, j) means thread j merges into i at time t.
- inconsistent(t, i) means thread i at time t is inconsistent with respect to the domain invariance axioms.

We will describe the axioms in six groups: action precondition and effects; activating threads; merging threads; thread consistency; mutual exclusion; and frame axioms.

Precondition and effect axioms. Consider an action proposition $a(t, i_1, \ldots, i_k)$. Then for any precondition L, where L can be any of the forms Kp, $K\neg p$, $\neg Kp$, or $\neg K\neg p$, if the action occurs, the precondition holds on the first thread.

$$a(t, i_1, \ldots, i_n) \supset L(t, i) \tag{1}$$

If L_j is a knowledge literal in the *j-th* possible effect, then L_j holds on the *j-th* thread.

$$a(t, i_1, \ldots, i_n) \supset L_j(t, j) \tag{2}$$

Activating threads. Actions only occur on active threads. Each outcome of a non-deterministic action corresponds either to an activated thread, (exclusive) or to a thread that is inconsistent with the domain invariance axioms (as described below). Threads for outcomes after the first must have previously been inactive.

$$a(t, i_1, \ldots, i_n) \supset \text{active}(t, i_1)$$
$$a(t, i_1, \ldots, i_n) \supset \text{active}(t+1, i_j)$$
$$\vee\ \text{inconsistent}(t+1, i_j) \quad \text{for } 1 \leq j \leq n \tag{3}$$
$$\neg\text{active}(t+1, i_j) \vee \neg\text{inconsistent}(t+1, i_j)$$
$$a(t, i_1, \ldots, i_n) \supset \neg\text{active}(t, i_j) \quad \text{for } 2 \leq j \leq n \tag{4}$$

After a non-deterministic action, everything that is known and that is consistent with the effects of the action is still known on each resulting thread. Where L be any literal of the form Kp or $K\neg p$ such that $\neg L$ is not an effect of a:

$$a(t, i_1, \ldots, i_n)\ \ \&\ \ L(t, i_1) \supset L(t+1, i_j) \quad \text{for } 1 \leq j \leq n \tag{5}$$

Merging threads. Merging requires the threads to be active, and results in the second being deactivated.

$$\text{merge}(t, i, j) \supset \text{active}(t, i) \tag{6}$$
$$\text{merge}(t, i, j) \supset \text{active}(t, j) \tag{7}$$
$$\text{merge}(t, i, j) \supset \neg\text{active}(t+1, j) \tag{8}$$

The first thread retains what was known on *both* threads. Where L be any literal of the form Kp or $K\neg p$:

$$\text{merge}(t, i, j)\ \ \&\ \ L(t, i)\ \ \&\ \ L(t, j) \supset L(t+1, i) \tag{9}$$
$$\text{merge}(t, i, j)\ \ \&\ \ L(t+1, i) \supset L(t, i) \tag{10}$$
$$\text{merge}(t, i, j)\ \ \&\ \ L(t+1, i) \supset L(t, j) \tag{11}$$

If a thread is deactivated, then it must have been merged or become inconsistent.

$$\text{active}(t, j)\ \ \&\ \ \text{active}(t+1, j) \supset$$
$$\text{inconsistent}(t+1, j) \vee \exists i \neq j\ .\ \text{merge}(t, i, j) \tag{12}$$

Thread consistency. We define a predicate inc_i for each way in which a domain invariant can be violated. For example, given the invariant $p \vee q \vee r$, we could assert:

$$\text{inc}_1(t, i) \equiv (K\neg p(t, i)\ \ \&\ \ K\neg q(t, i)\ \ \&\ \ K\neg r(t, i)) \tag{13}$$

Suppose there are m such conditions. inconsistent is then defined as the disjunction of these conditions.

$$\text{inconsistent}(t, i) \equiv (\text{inc}_1(t, i) \vee \ldots \vee \text{inc}_m(t, i)) \tag{14}$$

Mutual exclusion axioms. As in a standard Satplan encoding, two deterministic actions are mutually exclusive at a time step and thread if one negates a precondition or effect of the other. Non-deterministic actions are mutually exclusive with all other actions on their first thread. Merge is mutually exclusive with all other actions on each of the threads being merged. Let a and a' be interfering deterministic actions, and b and b' be non-deterministic actions.

$$\neg a(t, i) \vee \neg a'(t, i) \tag{15}$$
$$\neg a(t, i) \vee \neg b(t, i_1, \ldots, i_n) \tag{16}$$
$$\neg b(t, i_1, \ldots, i_n) \vee \neg b'(t, i_1, \ldots, i_m) \tag{17}$$

Frame axioms. Explanatory frame axioms are used for deterministic actions because they allow parallel execution of non-interfering actions (Schubert 1990). Frame conditions for non-deterministic actions appear in the thread activation axioms above. Where $a_1, \ldots a_k$ is the set of actions with effect L, where L can be any of the forms Kp, $K\neg p$, $\neg Kp$, or $\neg K\neg p$:

$$\neg L(t, i)\ \ \&\ \ L(t+1, i) \supset a_1(t, i) \vee \ldots \vee a_k(t, i) \tag{18}$$

Related Work

Classic STRIPS planning is defined for fully-observed, deterministic domains, where the solution is a sequence of actions (Fikes and Nilsson 1971). Most research in non-probabilistic planning has focused on this case, or on the generalization of allowing solutions to contain parallel non-interacting actions, as in Graphplan (Blum and Furst 1997) and Satplan (Kautz, McAllester, and Selman 1996; Kautz and Selman 1996). The problem of finding plans that contain tests and branches was first investigated in the context of non-linear planning (Warren 1976; Peot and Smith 1992). Rintanen (Rintanen 1999) showed that the general problem of finding conditional plans is in a higher complexity class than finding STRIPS plans, and developed translations of conditional planning to solving quantified Boolean formulas (QBF). The complexity argument, however, does not rule out translations to STRIPS or SAT that do not guarantee the completeness of the solution, that handle only special cases, and/or that in the worst case are exponentially large.

Prior work most closely related to T-Satplan concerns translations from various kinds of conditional planning to STRIPS. Bonet and Geffner (Bonet and Geffner 2000) introduced the idea of representing incomplete information by a complete description of the knowledge state of the planner, *e.g.*, using fluents of the form Kp and $K\neg p$.

Conformant planning is the special case of conditional planning where the solution is restricted to be a linear sequence of actions. Although there are no explicit test operators, sensing can be implicitly performed by actions that have conditional effects. Palacios and Geffner (Palacios and Geffner 2007) present a translation of conformant planning into classic STRIPS. This involves introducing fluents that are conditioned on particular possibilities, called "tags", of the initial state. For example, the expression p/t can be thought of as meaning "if the initial state satisfied t, then p holds". If a fluent holds in a state for all possible choices in a set of tags for p, then a *merge* operator lets the planner conclude Kp. The translation is exponential in a quantity called the *conformant width* of the problem, which intuitively quantifies the amount of uncertainty in the initial state. They showed that many benchmark problems have low conformant width.

Contingent planning is more general than conformant planning in that plans may contain tests and branches, but uncertainty is still restricted to incomplete knowledge about the initial state. Contingent planning prohibits actions that cause loss of information or non-deterministic actions that are not tests. The Palacios and Geffner translation for conformant planning can be generalized to translate contingent planning to AND/OR tree search (Hoffmann 2005b) or to fully-observed STRIPS with non-deterministic actions (Albore, Palacios, and Geffner 2009) and solved by heuristic forward-chaining search. In this paper, our experiments include a contingent planning domain where we compare T-Satplan against the CLG planner (Hoffmann 2005b). Note that the notion of the conformant width of a problem can be generalized to the *contingent width*.

Tags are similar to threads in our model, and the merge operator to join. They differ in that each tag is associated with a particular proposition (or more generally, with a particular formula), while a thread can be associated with any outcome of any non-deterministic action. Threads can be reused after being deactivated, and can associated with different tests or outcomes at different points in a solution. For example, the experiments discussed below include a problem class where the required number of tags grows linearly with the size of the domain, but the required number of threads remains constant. We are currently working on devising a problem-instance complexity measure appropriate for full conditional planning with non-deterministic actions, and will compare it to contingent width for the special case of contingent planning.

Experiments

We tested T-Satplan using four problem classes, the first two were problems which CLG cannot represent, the third was a new problem that we use to compare T-Satplan to CLG in the presence of mutually exclusive actions and the final was part of the set of tests available with the CLG planner (Albore, Palacios, and Geffner 2009). The planner and problems are available at *www.cs.rochester.edu/u/wlasecki/TSatplan*.

Implementation. T-Satplan was implemented in Ruby and all tests were run using the Plingeling solver on a dual core Intel Core i5 machine running Fedora (Biere 2010). Our version of the encoder for T-Satplan was designed to be a proof of concept, and is not yet optimized. Even for classical problems for which the encoding remains unchanged from that of Satplan, our version runs significantly slower than Satplan. This is in large part due to the fact that we have not yet implemented a planning graph in order to prune unnecessary actions.

Code Check. First, we demonstrate results in a domain which contains uncertainty in the effects of actions. We model this by creating the `code-check` domain, in which a faulty program is corrected line by line until the program works. There are two versions of this problem, the first is simpler and contains only a single action that creates uncertainty, while the other contains two. The set of actions includes the following:

```
_CheckLine_
Preconditions: knows(-working)
Effects: knows(lineN),
-knows(compiling), -knows(-compiling),
-knows(working), -knows(-working)

_TestProgram_
Preconditions:
Effects: knows(working) | -knows(-working)

_Compile_
Preconditions: knows(-working)
Effects: knows(compiling) |
knows(-compiling), knows(-working)
```

Where the *check* action "fixes" a line of code, with the result being a state where either the code will compile or not (for version 2 only), and if it compiles, then the code will either work properly or not. This uncertainty introduced by the loss of information is not handled by CLG. Figure 1 shows the runtimes of T-Satplan in this domain.

	2 Lines	5 Lines	10 Lines	15 Lines	20 Lines
Steps	10	11	21	31	41
Threads	2	3	3	3	3
Gen Time	0.019	0.086	0.39	1.008	2.144
Parse Time	0.043	0.111	0.386	0.85	1.543
Solve Time	0.018	0.063	2.423	27.857	234.926
Total Time	0.08	0.26	3.199	29.715	238.613
Clauses	632	7691	30321	72401	138431
Literals	112	717	2322	4827	8232

Figure 1: Runtimes for the code-check domain.

N-way Nondeterminism. We show T-Satplan's performance in a domain which contains actions with more than 2 possible outcomes. Sensing actions are actions with 2 possible outcomes in CLG, but in some situations, actions could cause one of many outcomes. In this problem we have a single non-deterministic action that creates a number of branches, each with a different set of actions to reach the

goal. We demonstrate both action sets which can be run partially in parallel and sets which must be executed in order. CLG cannot directly handle this domain without preprocessing the problem to remove the n-way effect. Figure 2 shows the results from this domain.

	1 Act	2 Acts	5 Acts	20 Acts	10/2 Acts	20/5 Acts
3 Branches	0.072	0.08	0.104	0.341	0.423	7.437
4 Branches	0.186	0.213	0.311	0.95	1.182	15.108
5 Branches	2.519	2.065	2.449	5.252	6.84	54.436
6 Branches	51.586	53.494	58.583	77.672	108.713	402.837
7 Branches	5179.21	memout	memout	memout	memout	memout

Figure 2: Runtimes for the N-way Nondeterminism domain. x/y denotes x actions to be executed in total, with at most y performed in parallel at any given time.

Key Ring. In the `Key Ring` domain we have one door and N keys of which we are uncertain which, if any, key will unlock this door. If no key unlocks it, then a locksmith must be called as a default action. Each test action requires no prerequisite knowledge and the keys can be tried in any order, but the locksmith must be called only when it is certain that there is no key that will open the door. Once the door is opened, a series of parallelizable *moveBox* actions must be performed to fulfill the final goal of relocating the contents of the locked room. We focused most of our effort on this problem because it clearly demonstrates an environment in which larger numbers of branches each contain arbitrary numbers of parallel actions. It is representative of a broad set of problems which contain uncertainty in a threshold or selection event (such as whether or not your lottery numbers were winners), followed by sets of actions conditioned on the choice (the rest of your plans for the day).

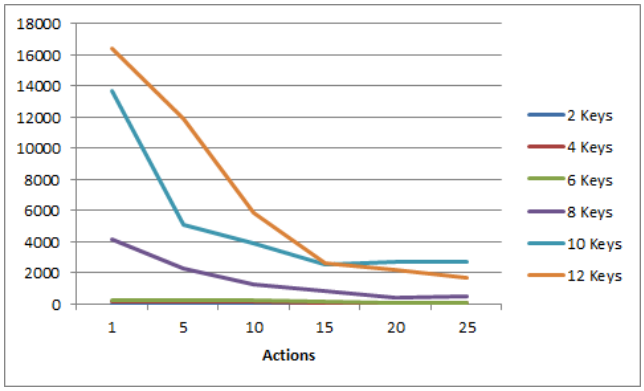

Figure 3: Graph of the ratio of runtime for T-Satplan compared to the runtime of CLG.

Much like Satplan, the SAT instances generated by T-Satplanare quite large (see Figure 5). However, modern SAT solvers are still able to handle these problems reasonably well. Despite a clear time advantage by CLG over T-Satplan, CLG's inability to identify mutually exclusive actions which can be performed in parallel means that the makespan of CLG's plans will be necessarily larger in the presence of parallelizable actions in a plan. We can see from Figure 3

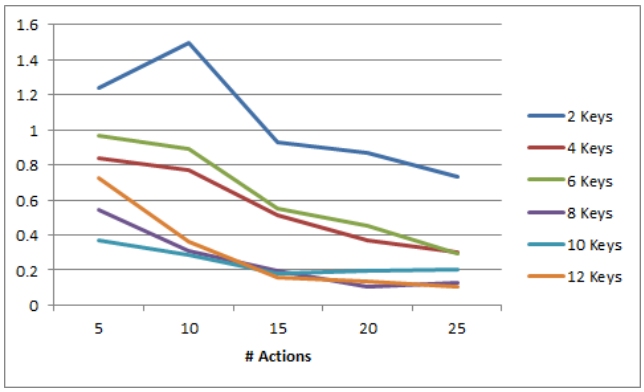

Figure 4: Graph of the ratio of improvement of 5-25 actions compared to the single action case.

that as the number of parallel actions increases for a given problem, the CLG's advantage shrinks not only in terms of difference in optimal makespan size, but we also see a reduction in the time improvement factor between CLG and T-Satplan. Figure 4 shows that the number of threads grows we see a similar trend. For larger numbers of threads, the cost of larger sequences of actions slows down CLG at a higher rate due to the fact that they must perform actions at $V * (A - 1)$ more time steps (total across all branches) than T-Satplan, where V is the total number of unknown variables and A is the number of actions on each branch. Figure 5 shows the sizes of the encodings for this problem.

		1 acts	5 acts	10 acts	15 acts	20 acts	25 acts
2 Vars	Clauses	12205	13653	15463	17273	19083	20764
	Literals	1068	1176	1311	1446	1584	1710
4 Vars	Clauses	111062	130878	155648	180418	205188	228903
	Literals	3065	3565	4190	4815	5445	6045
6 Vars	Clauses	229988	284604	352874	421144	489414	555579
	Literals	4475	5455	6680	7905	9135	10325
8 Vars	Clauses	404514	520810	666180	811550	956920	1098775
	Literals	6085	7705	9730	11755	13785	15765
10 Vars	Clauses	643280	855816	112140	1387156	1652826	1913211
	Literals	7895	10315	13340	16365	19395	22365
12 Vars	Clauses	760186	1011362	1325332	1639302	1953272	2260997
	Literals	9255	12115	15690	19265	22845	26355
14 Vars	Clauses	1348092	1887508	2561778	3236048	3910318	4574683
	Literals	12115	16615	22240	27865	33495	39045
16 Vars	Clauses	1831418	2616834	3598604	4580374	5562144	6531159
	Literals	14525	20305	27530	34755	41985	49125

Figure 5: Encoding size of T-Satplan's SAT instance.

Figures 6 and 7 show diagrams of the plans generated by T-Satplan and CLG respectively on the `Key-Ring` problem. T-Satplan's ability to maintain a compact representation lets the size of the resulting plan grow linearly in both number of unknown variables and makespan these cases.

Unix. Finally, we tested both planners using the `Unix` file system domain, a problem set used by the CLG planner (Albore, Palacios, and Geffner 2009). The task is to find a file in a directory structure and move it back to the root directory. Sensing actions allow the planner to determine whether or

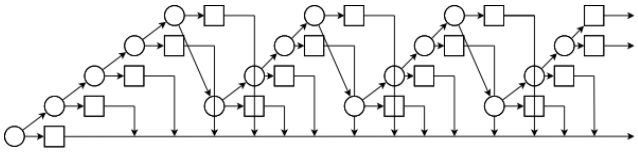

Figure 6: T-Satplan's plan for a domain with 16 unknown variables and 5 parallel actions. Circles represent sensing actions, while squares represent action steps. Here, each action step contains 5 parallel actions.

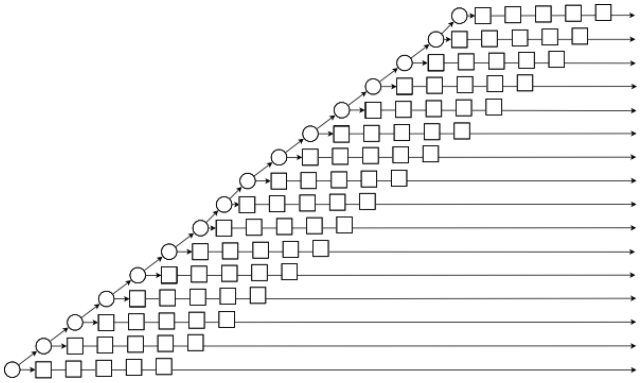

Figure 7: CLG's plan for a domain with 16 unknown variables and 5 (potentially parallel) actions.

not the file is in the current directory. If it is, the file can be moved to any directory chosen, otherwise the planner must continue to navigate the file system until the file is found. T-Satplan was able to find optimal solutions in terms of number actions taken where CLG was not. During the `Unix-1` test, CLG's longest branch contained 18 actions, whereas T-Satplan only used 15 actions in its plan.

In terms of computational time, Figure 8 shows T-Satplan performed as expected and was significantly slower, only able to finish one of the problems in this domain within two hours. The poor performance was most likely due to the fact that these problems were converted directly from the existing PDDL files without editing, so the current lack of any preprocessing greatly reduced the solving speed by including a large number of unnecessary actions in the domain.

Summary of Results. These tests show that T-Satplan has great potential for domains which contain both uncertainty and mutually exclusive actions. While not as fast as CLG on domains without mutually exclusive actons, T-Satplan is still able to find shorter paths in some cases since it is always able to find an optimal plan with respect to the longest possible path. As such, it shows promise for problems in which the cost of a longer makespan is greater than the cost of additional time spent finding the plan itself, as is the case in many industrial problems.

Conclusions and Future Work

SAT-based planning is an effective technique for timespan-minimum planning with interacting goals. Part of its appeal is its ability to immediately leverage the continual improve-

		T-S	CLG
Unix-1	Total Time	599.83	0.01
	Threads	3	3
	Makespan	15	18
Unix-2	Total Time	memout	0.21
	Threads	4	12
	Makespan	38	39
Unix-3	Total Time	memout	3.15
	Threads	NA	28
	Makespan	NA	86
Unix-4	Total Time	memout	52.69
	Threads	NA	60
	Makespan	NA	181

Figure 8: The runtimes for the Unix file system domain.

ments of general SAT solvers. Existing SAT-based planning systems handle classic fully-observed, deterministic STRIPS problems. This paper presents the first translation of planning with partial knowledge, tests, branches, and N-way non-deterministic actions to propositional satisfiability. Previous work has provided translations to QBF or to non-deterministic STRIPS. While there have been many advances in QBF solvers (Giunchiglia, Narizzano, and Tacchella 2004), they are much less scalable in practice than SAT solvers; thus, a translation to ordinary SAT remains of interest. The special case of translating conformant planning to SAT was, in principle, already solved, because the translation of conformant planning to classic STRIPS (Bonet and Geffner 2000) could be combined with the translation of classic STRIPS to SAT (Kautz, McAllester, and Selman 1996). However, we provide the first SAT translation for the contingent and general cases, and accordingly a solution method that does not rely upon heuristic search.

Our system T-Satplan provides a proof of concept that the approach is feasible, even with a far-from-optimal implementation. Performance on contingent planning benchmarks is reasonable, if not state of the art. More significantly, we can handle conditional planning problems that go beyond contingent planning. We also showed that the ability to reuse threads allows us to devise contingent planning problems whose solutions require fewer threads than tags, and thus require less space to be represented in a threaded SAT encoding than in a tagged STRIPS encoding.

Much work remains to be done. We are working on generalizing the notion of contingent width to a measure that will let us bound the size of our SAT encoding for any particular problem instance, and in defining problem classes that separate threaded from tagged encodings. Further experimental work will require us to reimplement T-Satplan so that it uses a planning-graph to prune the number and size of generated clauses, as is done in modern planning as satisfiability systems. This will allow for much larger domains to be solved, meaning empirical results can be collected from problems that our trend lines suggest T-Satplanwill outperform modern heuristic planners in. We are also working to guarantee timespan-minimal plans for non-worst case paths. Finally, we are investigating methods for encoding finding plans with loops as satisfiability, building off of recent suggestions for general planning by Srivastava *et. al.* (Srivastava, Immerman, and Zilberstein 2009).

References

Albore, A.; Palacios, H.; and Geffner, H. 2009. A translation-based approach to contingent planning. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI-09)*.

Biere, A. 2010. Lingeling, plingeling, picosat and precosat at sat race 2010. In *FMV Report Series*, 69. Institute for Formal Models and Verification, Johannes Kepler University.

Blum, A., and Furst, M. 1997. Fast planning through planning graph analysis. *Artificial Intelligence* 90:281–300.

Bonet, B., and Geffner, H. 2000. Planning with incomplete information as heuristic search in belief space. In *Proceedings of the 5th International Conference on AI Planning Systems (AIPS-2000)*, 52–61.

Fikes, R. E., and Nilsson, N. J. 1971. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189–208.

Giunchiglia, E.; Narizzano, M.; and Tacchella, O. 2004. Qbf reasoning on real-world instances. In *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT-2004)*, 105–121.

Hoffmann, J. 2005a. Where ignoring delete lists works: Local search topology in planning benchmarks. *Journal of Artificial Intelligence Research (JAIR)* 24:685–758.

Hoffmann, J. 2005b. Contingent planning via heuristic forward search with implicit belief states. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS-2005)*, 71–80.

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.

Kautz, H., and Selman, B. 1999. Unifying sat-based and graph-based planning. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, 318–325. Morgan Kaufmann.

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.

Palacios, and Geffner. 2007. From conformant into classical planning: Efficient translations that may be complete too. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling*.

Peot, M. A., and Smith, D. E. 1992. Conditional nonlinear planning. In *Proceedings of the 1st International Conference on AI Planning Systems (AIPS-92)*.

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

Schubert, L. 1990. Monotonic solution of the frame problem in the situation calculus: An efficient method for worlds with fully specified actions. In *Knowledge Representation and Defeasible Reasoning*. Kluwer Academic Press. 23–67.

Srivastava, S.; Immerman, N.; and Zilberstein, S. 2009. Finding plans with branches, loops and preconditions. In *ICAPS 2009 Workshop on Verification and Validation of Planning and Scheduling Systems*.

Warren, D. 1976. Generating conditional plans and programs. In *Proceedings of the Summer Conference on AI and Simulation of Behavior*.