A Compact and Efficient SAT Encoding for Planning

Author

Robinson, N, Gretton, C, Pham, DN, Sattar, A

Published

2008

Conference Title

ICAPS 2008 - Proceedings of the 18th International Conference on Automated Planning and Scheduling

A Compact and Efficient SAT Encoding for Planning

Nathan Robinson, Charles Gretton, Duc-Nghia Pham, and Abdul Sattar
SAFE Program, Queensland Research Lab, NICTA and
Institute for Integrated and Intelligent Systems, Griffith University, QLD, Australia
{nathan.robinson,charles.gretton,duc-nghia.pham,abdul.sattar}@nicta.com.au

Abstract

In the planning-as-SAT paradigm there have been numerous recent developments towards improving the speed and scalability of planning at the cost of finding a step-optimal parallel plan. These developments have been towards: (1) Query strategies that efficiently yield approximately optimal plans, and (2) Having a SAT procedure compute plans from relaxed encodings of the corresponding decision problems in such a way that conflicts in a plan arising from the relaxation are resolved cheaply during a post-processing phase. In this paper we examine a third direction of tightening constraints in order to achieve a more compact, efficient, and scalable SAT-based encoding of the planning problem. For the first time, we use *lifting* (i.e., *operator splitting*) and *factoring* to encode the corresponding n-step decision problems with a parallel action semantics. To ensure compactness we exploit *reachability* and *neededness* analysis of the *plangraph*. Our encoding also captures state-dependent mutex constraints computed during that analysis. Because we adopt a lifted action representation, our encoding cannot generally support full action parallelism. Thus, our approach could be termed approximate, planning for a number of steps between that required in the optimal parallel case and the optimal linear case. We perform a detailed experimental analysis of our approach with 3 state-of-the-art SAT-based planners using benchmarks from recent international planning competitions. We find that our approach dominates optimal SAT-based planners, and is more efficient than the relaxed planners for domains where the plan existence problem is hard.

Introduction

Currently the best approaches for domain independent step-optimal planning are SAT-based. The winner of the optimal track in the 2004 International Planning Competition (IPC-4) was SATPLAN-04, and in the 2006 competition (IPC-5) SATPLAN-06 (Kautz, Selman, & Hoffmann 2006) and MAXPLAN (Chen, Xing, & Zhang 2007) tied for first place. These solvers, all descended from BLACKBOX (Kautz & Selman 1999), compile the problem posed by an n-step *plangraph* (Blum & Furst 1997) into a *conjunctive normal form* (CNF) formula. A plan is then computed by a dedicated SAT solver; such as Lawrence Ryan's SIEGE (Ryan 2003). Compilation is fairly direct. Each variable in the CNF corresponds to whether an action is executed at a time step, or whether a proposition is true at a time step. The planning constraints, such as *frame axioms*, *conflict exclusion*, and that *an action implies its preconditions and effects*, are encoded naturally in terms of those variables. Consequently, the biggest drawback to BLACKBOX successors is the enormous sized CNFs they generate. For example, the 19-step decision problem generated by SATPLAN-06 for problem P-22 from the IPC-5 PIPESWORLD domain has over 20 million clauses and 47 thousand variables. Theoretically, in the worst case the number of clauses needed for each time step is polynomially dominated by the number of action variables for that time step. Although this size blowup is mitigated in the above solvers by performing *reachability* and *neededness* analysis in constructing the plangraph – and consequently reducing the number of variables and clauses in the CNF – those solvers are unable to tackle problems of the size satisficing planners routinely solve (Gerevini *et al.* 2006; Hoffmann & Edelkamp 2005).

To mitigate the problem of size blowup, the MEDIC system (Ernst, Millstein, & Weld 1997) uses a *lifted* representation of actions (Kautz & Selman 1992). Compared with SATPLAN-style flat representations, for direct-encodings of the linear planning problem – where all action parallelism is forbidden – fewer variables are required to describe operators. For an n-ary operator where each argument is drawn from a set of objects Σ, lifting needs $\mathrm{O}(n|\Sigma|)$ variables, whereas a flat representation needs $\mathrm{O}(|\Sigma|^n)$. In the MEDIC system axioms that encode action constraints are *factored*, leading to relatively few clauses in the CNF. Unfortunately MEDIC only exploits *lifting* if execution of multiple actions in parallel is forbidden. Thus, planning usually requires more plan steps compared to competing parallel systems.[1] For example, for problem P-07 from the IPC-5 ROVERS domain, MEDIC requires 18 steps, whereas SATPLAN-06 requires only 5 steps.

The above optimal SAT-based planning systems are not fast and do not scale well compared with approximately-optimal (Streeter & Smith 2007; Rintanen 2004) and sat-

[1] Indeed, action parallelism was introduced to reduce the planning horizon; not because domains model parallelism inherent in the real-world.

isficing planning systems for two key reasons. First, the proposed encodings are too big. Secondly, *query strategies* they use to efficiently find a step-optimal plan become inefficient when step-optimality is only desirable. Such strategies include the *binary-search* used by an early version of SATPLAN (Kautz & Selman 1996), and the *ramp-up* and *ramp-down* strategies employed by SATPLAN-04/06 (Kautz 2006) and MAXPLAN (Chen, Xing, & Zhang 2007) respectively. *Ramp-up* starts with $n = 1$, and then incrementally increases this by one step until a solution is found. *Ramp-down* starts with $n = k$ for some upper bound k (obtained in practice using a fast satisficing planner) and proceeds towards the optimal.

Rintanen, Heljanko, & Niemelä (2006) propose reducing the number of queries required to find a plan by relaxing constraints on action parallelism. Their work also addresses the problem of large CNF, presenting the first direct SAT encoding with size linearly bounded by the size of the problem at hand. Later Wehrle & Rintanen (2007) adapted this work from its original setting, ADL planning, to the STRIPS case, also further relaxing constraints on action parallelism and availability. These approaches both exploit the concept of *post-serialisability* from Dimopoulos, Nebel, & Koehler (1997), thereby allowing a set of conflicting actions to be prescribed at a single time step provided a scheme, computed *a priori*, is available for generating a valid serial execution. In their encoding, *conflict exclusion* axioms – i.e., constraints that prevent conflicting actions being executed in parallel – are exchanged for axioms ensuring that conflicting parallel executions respect the serialisation scheme. Consequently, fewer decision problems are usually required to be solved using ramp-up compared with other existing encodings. On the downside, the length of solution plans are relatively long.

In this paper we develop a compact SAT encoding for parallel planning. Along the lines of BLACKBOX, we propose a CNF encoding capturing the problem posed by the n-step plangraph. Along the lines of MEDIC, we use a lifted action representation, however unlike MEDIC, our encoding permits some action parallelism. In this last respect the direction we take is somewhat opposite to that of Rintanen, Heljanko, & Niemelä (2006). Whereas they allow conflicting actions to be prescribed to the same time step, we explicitly prohibit parallel execution of conflicting actions and of actions that "interfere" given their lifted representation. The result is a compact SAT encoding based on the problem posed by an n-step plangraph with a restricted parallel action semantics.

In the following section we review planning and plangraph analysis. We then discuss our compilation of the n-step problem posed by a plangraph into CNF with a limited form of action parallelism. Finally, we present detailed empirical results, followed by concluding remarks, and a discussion of future research directions.

Problem and Notations

Planning

A propositional planning problem is given in terms of a finite set of objects Σ, first-order STRIPS-like planning operators of the form $\langle o, \mathit{pre}(o), \mathit{add}(o), \mathit{del}(o)\rangle$, and predicates Π. Here, o is an expression of the form $\mathcal{O}(x_1, \ldots, x_n)$ where $\mathcal{O}$ is an operator name and x_i are variable symbols, $\mathit{pre}(o)$ are the operator preconditions, $\mathit{add}(o)$ are the add effects, and $\mathit{del}(o)$ the delete effects. By grounding Π over Σ we obtain the set of propositions P that characterise problem states. For example, in a blocks-world problem we can have two block objects A and B, and a binary predicate $\texttt{On}$, one grounding of which is the proposition $\texttt{On}(A, B)$. A planning problem is posed in terms of a starting state $s_0 \subseteq P$, a goal $\mathcal{G} \subseteq P$, and a small set of domain operators.

An action a is a ground operator having a set of ground preconditions $\mathit{pre}(a)$, add effects $\mathit{add}(a)$, and delete effects $\mathit{del}(a)$. The contents of each of those sets are made up of elements from P. An action a can be executed at a state $s \subseteq P$ when $\mathit{pre}(a) \subseteq s$. We denote $\mathcal{A}(s)$ the set of actions that can be executed at state s. When $a \in \mathcal{A}(s)$ is executed at s the resultant state is $(s \cup \mathit{add}(a)) \backslash \mathit{del}(a)$. Actions cannot both add and delete the same proposition – i.e., $\mathit{add}(a) \cap \mathit{del}(a) \equiv \emptyset$. Usually any two actions a_1 and a_2 are permitted to be executed instantaneously in parallel at any state s provided any serial execution of the actions is valid and achieves the same outcome. When two actions cannot be executed in parallel we say they *conflict*.

A state s is a *goal state* iff $\mathcal{G} \subseteq s$. A plan is a prescription of non-conflicting actions to each of n time steps. We say that a plan solves a planning problem when executing all the actions at each step starting from s_o achieves a goal state. A plan is optimal iff no other plan can achieve the goal in a shorter number of time steps.

Plangraph

The *plangraph* is a datastructure that was devised for the efficient GRAPHPLAN framework of planning in STRIPS domains with restrictions on where negated propositions occur in the problem specification (Blum & Furst 1997). The plangraph captures necessary conditions for goal achievement, and is computationally cheap to build, taking polynomial time and space in the number of propositions and actions.

Given a planning problem with actions $\mathcal{A}$ and propositions P, the plangraph is a directed *layered* graph. For each time step t up to the planning horizon n, the graph has two layers labelled with t: one action layer and one propositional layer. There is also a final propositional layer for the goal. The propositional layer with timestamp t contains a set of propositions such that elements in the powerset of this are states which might be reachable in t time steps from s_0. Similarly, an action layer with timestamp t contains the set of actions which might be executable t steps into a plan.

In more detail, the first layer has a vertex for each proposition in s_0. An action layer at time t, has a vertex for each action $a \in \mathcal{A}$ such that the preconditions of a appear in the propositional layer at time t, as well as an artificial action $noop_p$ for each proposition p in the propositional layer

at time t. The action $noop_p$ has the precondition p and an add effect p and represents the fact that the truth value of proposition p is maintained. The add effects of each action in the action layer at time t are added to the proposition layer at time $t+1$. This means that the size of each layer of the plangraph monotonically increases until all reachable ground facts are present. There is an arc from each action to each of its preconditions. There is an arc labelled *positive* from each action a to the add effects of a and an arc labelled *negative* to each of the delete effects of a.

The plangraph gives necessary (but not sufficient) conditions for *reachability*. That is, any state reachable from s_0 in t steps is characterised by propositions at layer t. Similarly, the t'th step of a plan can only include actions at layer t. While maintaining this important property, state-dependent *mutex* relations between actions and propositions are exploited to make the plangraph leaner. Actions a_1 and a_2 at t are *mutex* if they are *conflicting*, or if a precondition of a_1 is mutex with a precondition of a_2. Also, two propositions p_1 and p_2 at time t are mutex if all ways of achieving them together are mutex. That is, p_1 and p_2 are mutex if there is no action adding both p_1 and p_2, and every action which adds p_1 is mutex with every action which adds p_2. While constructing the plangraph, we can omit any action a at t, if there is a pair of propositions in its precondition that are mutex at t.

Another type of processing that is performed to further prune the plangraph is called *neededness analysis*. Here, for planning horizon n, any action at time $n-1$ which has a goal proposition as a delete effect, or which does not achieve any goal proposition, is marked as not-needed. Further vertices can be marked as not-needed by working backwards through the layers and labelling vertices according to the following two rules: (a) An action at time t is only *needed* if it has a needed proposition as an add effect, and (b) A proposition at time t is only *needed* if it is part of the precondition of a needed action. Any vertices (and neighbouring arcs) that are not-needed can be safely removed from the plangraph.

Our Approach

Along the same lines as BLACKBOX, our approach first builds the plangraph as described above. It then proceeds to compile the constraints implied by the graph – including action and proposition mutex – into a CNF according to axiom schemata given below. Unlike the case of BLACKBOX-like planners, the encoding we propose for the n-step problem uses a lifted representation of actions, exploits this by factoring constraints relating to actions, and tightens action mutex constraints from the plangraph to ensure compactness of representation. Given our CNF encoding of the problem, a sound and complete SAT solver then proceeds to search for a satisfying valuation for the CNF, or otherwise proves that none exists. In the case of no solution, the plangraph is extended and the process repeats until a plan is found or a preset timeout is reached. In the case that a satisfying valuation is found, this has a direct interpretation as a solution plan for the problem at hand.

The key distinguishing feature of our approach is our use of a lifted representation of actions. In all recent SAT-based planners the number of Boolean variables in the CNF, and hence its overall size, is dominated by the number of actions in the problem at hand. Kautz & Selman (1992) proposed "operator-splitting", a lifted representation of actions to reduce the number of Boolean variables in the CNF and hence its overall size. In particular, they proposed that operators taking multiple arguments be encoded in terms of the conjunct of several predicates that take no more than one argument. For example, writing $\texttt{Move}[n](x)$ for the proposition that says "the n'th argument of action Move is x", we have that action $\texttt{Move}(A,B,C)$ from blocks-world can be represented by the conjunct $\texttt{Move}[1](A) \wedge \texttt{Move}[2](B) \wedge \texttt{Move}[3](C)$. Ernst, Millstein, & Weld (1997) used this split action representation in MEDIC for their direct-encoding of the linear planning problem. By factoring constraints, for example expressing the action precondition $\texttt{Move}(A,B,C) \rightarrow \texttt{Clear}(A)$ with $Move[1](A) \rightarrow \texttt{Clear}(A)$, they achieved very compact representations of the n-step problem.[2] Ernst, Millstein, & Weld (1997) did not extend their work to planning with a parallel action semantics, citing interference as the primary reason for this. In more detail, parallel execution of 2 actions $\texttt{Move}(A,B,E)$ and $\texttt{Move}(C,D,E)$ is represented with splitting by $\texttt{Move}[1](A) \wedge \texttt{Move}[2](B) \wedge \texttt{Move}[1](C) \wedge \texttt{Move}[2](D) \wedge \texttt{Move}[3](E)$ which implies 4 different instantiations of Move – Those are, $\texttt{Move}(A,D,E)$, $\texttt{Move}(A,B,E)$, $\texttt{Move}(C,B,E)$, and $\texttt{Move}(C,D,E)$. Below, we propose axiom schemata that provides for a compact factored encoding with a split operator representation that, unlike MEDIC, admits a significant amount of action parallelism in practice.

Axiom Schemata

Our approach uses the following propositional axiom schemata to generate a CNF encoding based on the n-step plangraph. We assign a Boolean variable p^t to each state proposition p occurring at a time t in the plangraph. We also assign a Boolean variable $o^t[i]$ to each instantiation of an operator argument $o[i]$ at each time t. For example, $\texttt{Move}^4[1](A)$ says that the first argument of an operator Move at time step 4 is instantiated with A. In the remainder of this section, we will omit the time annotation of variables in a formula if all variables in that formula are from the same time step. We write Φ_a to denote a conjunct of variables that expresses the full instantiation of an action a. $\Phi_a(p)$ denotes the fragment of Φ_a that is relevant to proposition p. For example, if $\Phi_a \equiv \texttt{Move}(A,B,C)$ which has 3 preconditions: $\texttt{Clear}(A)$, $\texttt{Clear}(C)$ and $\texttt{On}(A,B)$, and 4 effects: $\texttt{Clear}(B)$, $\neg\texttt{Clear}(C)$, $\neg\texttt{On}(A,B)$ and $\texttt{On}(A,C)$, then we have

$\Phi_a(\texttt{Clear}(B)) = \texttt{Move}[2](B)$,
$\Phi_a(\texttt{Clear}(C)) = \texttt{Move}[3](C)$,
$\Phi_a(\texttt{On}(A,B)) = \texttt{Move}[1](A) \wedge \texttt{Move}[2](B)$,
$\Phi_a(\texttt{On}(A,C)) = \texttt{Move}[1](A) \wedge \texttt{Move}[3](C)$, etc.

Our encoding includes the usual *Starting State* and *Goal*

[2] In the case of a flat (resp. lifted) representation of actions, every instance of action Move whose first argument is A yields an additional clause $\texttt{Move}(A,x,y) \rightarrow \texttt{Clear}(A)$.

Axioms. In particular, we have a unit clause containing p_i^0 for every $p_i \in s_0$. Also, for each $p_i \in \mathcal{G}$ we have unit clause containing p_i^n. For each time step t in the n-step plangraph, we have the following clauses in the CNF encoding:

1. *Action Precondition Axioms*: For each action a, we have a set of clauses to ensure that if a is executed then all its preconditions $p \in \mathit{pre}(a)$ hold:

$$\Phi_a(p) \rightarrow p$$

For instance, in the blocks-world we have the precondition constraint:

$$\texttt{Move}[1](A) \wedge \texttt{Move}[2](B) \rightarrow \texttt{On}(A, B)$$

2. *Action Effect Axioms*: For each action a, we have a set of clauses to ensure that if a is executed then each of its effects p holds:

$$\Phi_a(p) \rightarrow p \quad \text{if } p \in \mathit{add}(a), \text{ or}$$
$$\Phi_a(p) \rightarrow \neg p \quad \text{if } p \in \mathit{del}(a)$$

For instance, we have the following clauses for action $\texttt{Move}(A, B, C)$:

$$\begin{array}{lcl} \texttt{Move}[2](B) & \rightarrow & \texttt{Clear}(B) \\ \texttt{Move}[3](C) & \rightarrow & \neg\texttt{Clear}(C) \\ \texttt{Move}[1](A) \wedge \texttt{Move}[2](B) & \rightarrow & \neg\texttt{On}(A, B) \\ \texttt{Move}[1](A) \wedge \texttt{Move}[3](C) & \rightarrow & \texttt{On}(A, C) \end{array}$$

3. *Operator Grounding Restriction Axioms*: For each pair of arguments $(o[i], o[j])$ of an operator o, we have clauses asserting that instantiations to these arguments occur in the plangraph. Below, the notation $\texttt{Dom}(o[i])$ denotes the set of valid assignments to the ith argument of o in the plangraph. We write $\texttt{Dom}(o[j] \mid o[i](X))$ to denote the set of valid assignments to the jth argument of o given its ith argument is X:

$$\textstyle\bigwedge_{X \in \texttt{Dom}(o[i])} o[i](X) \rightarrow o[j] \in \texttt{Dom}(o[j] \mid o[i](X)) \text{ , and}$$
$$\textstyle\bigwedge_{Y \in \texttt{Dom}(o[j])} o[j](Y) \rightarrow o[i] \in \texttt{Dom}(o[i] \mid o[j](Y))$$

For instance, we have the following clauses for actions $\texttt{Move}(A, B, C)$, $\texttt{Move}(A, B, D)$, and $\texttt{Move}(B, C, D)$:

$$\begin{array}{lcl} \texttt{Move}[1](A) & \rightarrow & \texttt{Move}[2](B) \vee \texttt{Move}[2](C) \\ \texttt{Move}[1](B) & \rightarrow & \texttt{Move}[2](C) \\ \texttt{Move}[1](A) & \rightarrow & \texttt{Move}[3](C) \vee \texttt{Move}[3](D) \\ \texttt{Move}[2](B) & \rightarrow & \texttt{Move}[1](A) \\ \text{etc.} & & \end{array}$$

4. *Propositional Mutex Axioms*: For propositions p_1 and p_2 that are mutex, we have a clause:

$$\neg p_1 \vee \neg p_2$$

5. *Inter-Operator Mutex Axioms (Conflict Exclusion)*: For each pair of actions (a_1, a_2) that instantiate distinct operators, we have clauses prohibiting their parallel execution in either of the following cases:

a. If one action has a delete effect p that is either a precondition or an add effect of the other, we have the clause below. Here, without a loss of generality, it could be that $t_1 = t_2$ or $t_1 = t_2 + 1$.

$$\neg\Phi_{a_1}(p^{t_1}) \vee \neg\Phi_{a_2}(p^{t_2})$$

b. If there exists a pair of mutex propositions (p_1, p_2) such that $\Phi_{a_1} \rightarrow p_1$ and $\Phi_{a_2} \rightarrow p_2$:

$$\neg\Phi_{a_1}(p_1) \vee \neg\Phi_{a_2}(p_2)$$

When there are multiple ground predicates causing the same action mutex, we select for the smallest arity first. Dealing with case (a), suppose we are in the version of blocks-world with a gripper – i.e., blocks can be `picked-up` and `put-down`. Then we have the axiom:

$$\neg\texttt{pick-up}[1](A) \vee \neg\texttt{put-down}[1](A)$$

Dealing with case (b), consider the mutex propositions $\texttt{On}(A, B)$ and $\texttt{On}(A, C)$. Supposing we can either use the gripper or a `Move` operator to manipulate blocks, then we have an axiom:

$$\neg\texttt{Move}[1](A) \vee \neg\texttt{Move}[3](B) \vee$$
$$\neg\texttt{put-down}[1](A) \vee \neg\texttt{put-down}[2](C)$$

6. *Backwards Chaining Axioms (Frame Axioms)*: For each proposition p that occurs in successive layers of the plangraph, we include the following clause to ensure that if p is true at time t then it is true at time $(t-1)$ or there is an action a executed at time t that has p as an add effect. Below we write $\texttt{Make}(p)$ for the set of actions that have p as an add effect:

$$(p^t \rightarrow p^{t-1}) \vee \bigvee_{a \in \texttt{Make}(p^t)} \Phi_a(p^t)$$

This axiom was originally used by the GraphPlan encoding (Kautz, McAllester, & Selman 1996). It is also equivalent to the positive half of the *explanatory frame axioms* in (Ernst, Millstein, & Weld 1997). Again taking an example from the blocks-world, we have the constraint:

$$(\texttt{Clear}^2(B) \rightarrow \texttt{Clear}^1(B)) \vee \texttt{Move}^1[2](B)$$

7. *Intra-Operator Mutex Axioms (Conflict Exclusion)*: In using operator splitting to ensure a small number of variables in our encoding, we must prohibit certain parallel instantiations of the same operator that interfere. In particular, action instantiations a and b interfere iff their simultaneous execution (i.e. $\Phi_a \wedge \Phi_b$) according to Schema 2 implies a positive effect not in $\mathit{add}(a) \cup \mathit{add}(b)$. We also prohibit two instantiations of the same operator if they are mutex in the plangraph.

a. *Full Exclusion*: If all distinct instantiations of an operator o either interfere or conflict, then we exclude all parallel instantiations as follows. For every i and for every pair $(o[i](X), o[i](Y))$ where $X \neq Y$ we have the clause:[3]

$$\neg o[i](X) \vee \neg o[i](Y)$$

b. *Relaxed Exclusion*: If *Full Exclusion* is not required, for every pair of actions a and b that cannot be instantiated in parallel, due either to interference or conflict, we have the clause:

$$\neg\Phi_a \vee \neg\Phi_b$$

[3] Although our encoding would be smaller were we to only included clauses *for one* i, we find that in practice, having the additional binary clauses greatly increases the runtime performance of SAT procedures.

Localised Subsumption Checking

It should be noted that in practice we reduce the number of clauses generated by Schemata 5 and 7b by checking for subsumption. Checking is local in the sense that it is performed in isolation on clauses generated either for a particular pair of operators in the case of Schema 5, or for a single operator in the case of Schema 7b. Moreover, in the case of Schema 7b we remove redundant literals from clauses before we test for subsumption.[4] Here a literal is redundant if, when it is removed from the clause, the result does not form a constraint that prohibits a valid execution or parallel execution of non-interfering actions.

Making these ideas more concrete, consider again the example of the `Move` action from blocks-world. We have that two distinct instantiations of `Move` conflict if their first argument is equal and one of their last two arguments differ. Our clause reduction and subsumption checking step exploits the case where action mutex and interference is dependent on a small fraction of the action arguments. Continuing the example, in our approach the set of clauses:

$$\neg(\texttt{Move}[1](A) \wedge \texttt{Move}[2](B) \wedge \texttt{Move}[3](C)) \vee \neg(\texttt{Move}[1](A) \wedge \texttt{Move}[2](B) \wedge \texttt{Move}[3](D));$$
$$\neg(\texttt{Move}[1](A) \wedge \texttt{Move}[2](E) \wedge \texttt{Move}[3](C)) \vee \neg(\texttt{Move}[1](A) \wedge \texttt{Move}[2](E) \wedge \texttt{Move}[3](D));$$
$$\neg(\texttt{Move}[1](A) \wedge \texttt{Move}[2](F) \wedge \texttt{Move}[3](C)) \vee \neg(\texttt{Move}[1](A) \wedge \texttt{Move}[2](F) \wedge \texttt{Move}[3](D));$$
$$\ldots$$

Is captured using the single clause:

$$\neg\texttt{Move}[1](A) \vee \neg\texttt{Move}[3](C) \vee \neg\texttt{Move}[3](D)$$

Discussion

A major factor that drove the development of our approach was the problematic size of CNF encodings of decision problems posed in planning. We have addressed this problem by using a lifted representation of actions with a restricted parallel execution semantics. The restrictions we place on parallel executions are the only detail at this point that requires further discussion. Axiom Schema 5 is non-restrictive, capturing exactly the conflict between instantiations of distinct operators according to the domain action physics. Thus, in the LOGISTICS domain for example, our approach is able to execute a `Drive` and a `Load` action in parallel if it is valid for the step-optimal case. The restrictions we propose are enforced by Schema 7b. They only apply to parallel instantiations of the same operator. Axioms along the lines of Schema 5 cannot be used in this case because: (1) Interference caused by the lifted representation of actions may be the reason for the exclusion constraint rather than conflict, and (2) The exclusion relation cannot be summarised in terms of the action parameters that participate in a single conflict, because we may inadvertently prohibit legal parallel executions. For example, the mutex relation between $\texttt{Move}(A, B, C)$ and $\texttt{Move}(D, E, C)$ cannot be summarised (as Schema 5 would have it) in terms of the propositional facts $\texttt{Clear}^t(C)$ (precondition) and $\texttt{Clear}^{t+1}(C)$ (delete effect) as the clause thus generated, i.e. $\neg\texttt{Move}[3](C)$, would prohibit any blocks being moved to C. More generally, the encoding we propose only tightens constraints on parallel instantiations, thus our approach never has to plan for more steps than a step-optimal linear planner.

[4] Recall that a clause generated by 7b captures a mutex or interference relation between two instantiations of the same operator.

Experimental Results

Here we provide details of our experimental comparison of the SAT-based planner we developed, called PARA-L, with those of SATPLAN-06, and the post-serialisation techniques 1-LIN (Rintanen, Heljanko, & Niemelä 2006) and R$\exists$-STEP (Wehrle & Rintanen 2007). Our evaluation is on IPC-5 domains ROVERS, STORAGE, TPP, PIPESWORLD, and IPC-3 domains DEPOTS, DRIVERLOG, FREECELL, and ZENO(TRAVEL). Of these, STORAGE, TPP, DEPOTS, DRIVERLOG, and ZENO are *transportation domains*.

Using a linear query strategy $n = 1, 5, .., 30$, Table 1 outlines the performance of the different approaches. Table 3 presents results for problems from Table 1 in the case we use the query-strategy $n = 1, 2, .., 30$. In Table 1, for each domain and solver there is one row reporting statistics for the most difficult instance solved. We occasionally include an additional row to aid our discussion. All experiments were performed on a cluster of 32 computers, each with two AMD Opteron 252 2.6GHz processors, and each processor with 2GB of RAM. In all cases satisfiability of CNFs was obtained using the solver RSAT (Pipatsrisawat & Darwiche 2007) with a 3000 second cutoff for each CNF. RSAT won gold medals for the SAT+UNSAT and UNSAT problems of the Industrial Category of the 2007 International SAT Competition.

First, summarising the composition of the CNFs generated by PARA-L, we find that proposition mutex, operator mutex, and backwards chaining axioms account for most of the clauses. Proposition and operator mutex comprise the majority (usually between 75% and 80%) of clauses. Schema 7a is used extensively for STORAGE, FREECELL and PIPESWORLD, otherwise accounting for less than 1% of clauses. Operator grounding restrictions (Schema 3) account for less than 5% of clauses for most problems. Again, STORAGE, FREECELL and PIPESWORLD are the exceptions, where Schema 3 accounts for between 10% to 15% of the clauses.

Overall, the CNFs PARA-L generates at its solution horizon are much smaller than those generated by SATPLAN-06 for its solution horizon. Looking at Table 1, our encoding of the 20 step decision problem for ROVERS-20 comprises 856 thousand clauses, compared with 2.3 million clauses in the 10 step encoding by SATPLAN-06. More generally, our approach generates CNFs at its solution horizon that have substantially fewer variables compared to CNFs other approaches produce at their respective solution horizons. Looking again at Table 1, our encoding uses 4.5 thousand Boolean variables for STORAGE-17, SATPLAN-06 uses 25.8 thousand, and the post-serialisation techniques both use 49.1 thousand.

Compared to SATPLAN-06, PARA-L dominates in terms of finding a plan quickly and efficiently. In the case of

	PARA-L					SATPLAN-06					1-LIN					R∃-STEP				
Problem	c	v	*sec.*	n	$\#a$	c	v	*sec.*	n	$\#a$	c	v	*sec.*	n	$\#a$	c	v	*sec.*	n	$\#a$
freecell-4	140	6.4	23.9	15	38	8082.1	22.2	1080.6	15	36	490.5	130	54	15	36	491.3	130	75.6	15	34
freecell-6	360.1	12.9	20.7	20	50	M					1187.8	318	202.9	15	46	1189	318	566.9	15	46
freecell-7	575.9	18.7	1952.5	25	61	M					T					1996.9	522.4	3826.6	25	57
freecell-9	620	19.3	4705.7	25	63	M					T					T				
pipesworld-9	76.3	3.3	1.2	15	23	17582.9	41.9	640.8	15	30	901.8	229.6	153.1	15	27	902.5	229.6	80.2	15	27
pipesworld-12	145.4	5.8	38.7	20	24	T					896.2	207.2	1781	20	28	897.6	207.2	3173.3	20	32
pipesworld-14	183.2	6.4	32.8	20	32	T					T					T				
pipesworld-15	258.8	8.8	3082.8	25	38	T					771.1	169.2	2778.5	20	34	T				
depots-9	307.2	11.1	1558.4	25	68	T					365.9	47	5	20	88	365.9	47	4.9	20	82
depots-12	495.6	15.6	2716.7	30	81	T					637.7	84.4	3.2	20	107	643.2	84.4	2.1	20	112
depots-15	530.2	15.3	128.2	25	78	21470.5	83.1	4692	25	103	638.5	85.5	2.9	15	103	715.4	94.9	6.5	20	125
depots-20	729.7	18	185.8	25	93	T					724.9	102	2.5	15	147	731.8	102	5.2	15	140
storage-17	160.5	4.5	622.1	15	34	4715.6	25.8	3130.5	15	29	223.5	49.1	1.2	10	42	225.2	49.1	1.1	10	33
storage-20	496.4	10.8	4302.4	20	42	T					445.9	96.8	3.2	10	43	448.8	96.8	1.5	10	40
storage-23	1226.8	20.9	5795.8	25	65	M					938.9	200.6	12.5	10	52	944.7	200.6	5.2	10	53
storage-24	T					M					1520.8	320.6	6004.8	15	78	T				
driverlog-15	359.4	5.1	151.6	15	61	1025.6	31.2	46.1	15	69	131.2	39.7	0.3	10	49	133	39.7	0.2	10	51
driverlog-19	T					M					701.1	212.7	1328.8	15	151	706.6	212.7	1171.4	15	148
driverlog-20	M					M					T					1139.8	345.7	3457.5	15	199
TPP-21	175.3	7.2	57.9	15	115	4983.4	68.8	52.7	15	151	1124.3	292.2	2.5	10	161	1126	292.2	2	10	159
TPP-28	M					M					3363.5	875.7	21.1	10	221	3367.4	875.7	32.8	10	220
TPP-30	M					M					4466.3	1163	39.9	10	252	4471.2	1163	26.8	10	256
zenotravel-14	234.3	2.8	0.53	10	44	14954.1	55.3	21.1	10	69	125.9	32.7	0.3	5	33	126.9	32.7	0.2	5	33
zenotravel-20	T					M					1888.2	487.9	6317.5	15	185	1892.3	487.9	5134.1	15	185
rovers-20	1495.1	12.3	4426.7	20	88	2346.1	33.4	3.3	10	98	239.2	63	0.9	10	114	240.3	63	0.4	10	116
rovers-24	T					9489.4	84.2	17.9	15	126	531.2	138	4.4	15	148	533.7	138	3.3	15	154
rovers-29	1689.3	9.4	182.2	10	70	M					292.5	77.6	0.36	5	75	296.4	77.6	0.38	5	77

Table 1: n is the number of plan steps and $\#a$ is the number of actions in the plan. For the CNF generated at horizon n, c is the number of clauses in thousands and v is the number of variables in thousands. *Sec.* gives the cumulative runtime of RSAT in seconds, including that expended in obtaining refutation proofs. T indicates that no solution is found at any of $n = 5, 10, .., 30$ given our 3000 second timeout. M indicates the solver ran out of memory.

ROVERS, we are unable to effectively exploit important action parallelism, requiring more plan steps than SATPLAN-06. For example in Table 1, for ROVERS-20 PARA-L requires 20 steps compared to the 10 needed by the other approaches. Despite this, we are able to solve more difficult instances than SATPLAN-06 (e.g. ROVERS-29 for which SATPLAN-06 runs out of memory) because we have a much more compact encoding.

Compared to post-serialisation techniques, PARA-L dominates in PIPESWORLD and FREECELL; domains where deciding plan existence is intractable (Helmert 2006). For transportation domains, and especially in DRIVERLOG, PARA-L requires a relatively long solution horizon. Moreover, although PARA-L generates a very small CNF for a given horizon, because it is unable to capture important intra-operator parallelism, and more importantly because we do not have a post-serialisation scheme, that benefit is offset by the large number of plan steps required for a solution. This observation indicates the advantage of post-serialisation in reducing planning time. Indeed, SATPLAN-06 often uses fewer plan steps than PARA-L, however this only translates into a planning time advantage in ROVERS.[5]

Essentially we find that post-serialisation approaches have the key advantage of being able to instantiate transport actions `load`, `unload`, and `relocate` on one transport (vehicle) at a single plan step because ambiguity in the outcome is resolved against a serialisation scheme.

Finally, despite having a longer solution horizon for transportation domains, the lifted encoding of PARA-L often yields plans with comparatively few actions. This is particularly noticeable in DEPOTS. Taking DEPOTS-15 for example, PARA-L for 25 steps produces a plan with 78 actions, SATPLAN-06 for 25 steps uses 103 actions, while 1-LIN and R∃-STEP for 20 steps use 116 and 125 respectively.

Taking problems from Table 1, Table 2 reports the time RSAT spends satisfying the decision problems at the solution horizon n^* against the time spent refuting plan existence at a horizon $n' = n^* - 5$. If RSAT does not yield a refutation proof for our encoding at $n = 5$, we omit that result from the table. We report $\geq 3K$ in a *sec.* entry if RSAT could not decide the satisfiability of a CNF before our timeout.

Examining Table 2, we find that overall, PARA-L is more likely to cause a timeout given an "inadequate" number of steps n' compared to 1-LIN and R∃-STEP. Thus, refu-

[5] ROVERS is a rather special case in that it has unusually many predicates (25) and operators (9), many of which have high arity. Thus, the size of a ROVERS' plangraph, and subsequently the size of our encoding, grows unusually fast for a longer horizon.

Problem	PARA-L		SATPLAN-06		1-LIN		R∃-STEP	
	UNS	SAT	UNS	SAT	UNS	SAT	UNS	SAT
freecell-4	$\frac{10}{0.04}$	$\frac{15}{23.9}$	$\frac{10}{195}$	$\frac{15}{886}$	$\frac{10}{1.74}$	$\frac{15}{52.15}$	$\frac{10}{0.76}$	$\frac{15}{74.67}$
freecell-6	$\frac{15}{14.6}$	$\frac{20}{5.5}$	M		$\frac{10}{1.83}$	$\frac{15}{201}$	$\frac{10}{1.55}$	$\frac{15}{565}$
freecell-7	$\frac{20}{1836.9}$	$\frac{25}{115.5}$	M		T		T	
freecell-9	$\frac{20}{\geq 3K}$	$\frac{25}{1700}$	M		T		T	
pipesworld-9	$\frac{10}{0.32}$	$\frac{15}{0.82}$	$\frac{10}{366}$	$\frac{15}{274}$	$\frac{10}{117}$	$\frac{15}{35.56}$	$\frac{10}{60.86}$	$\frac{15}{19.09}$
pipesworld-12	$\frac{15}{4.4}$	$\frac{20}{34.3}$	T		$\frac{15}{1068}$	$\frac{20}{711}$	$\frac{15}{1599}$	$\frac{20}{1571}$
pipesworld-14	$\frac{15}{0.3}$	$\frac{20}{32.4}$	T		T		T	
pipesworld-15	$\frac{20}{123}$	$\frac{25}{2959}$	T		$\frac{15}{183}$	$\frac{20}{2595}$	T	
depots-9	$\frac{20}{0.73}$	$\frac{25}{1557.6}$	T		$\frac{15}{0.33}$	$\frac{20}{4.49}$	$\frac{15}{1.12}$	$\frac{20}{1.9}$
depots-12	$\frac{25}{2443}$	$\frac{30}{272.9}$	T		$\frac{15}{0.44}$	$\frac{20}{2.42}$	$\frac{15}{0.74}$	$\frac{20}{1.23}$
depots-15	$\frac{20}{9.47}$	$\frac{25}{118.5}$	$\frac{20}{\geq 3K}$	$\frac{25}{1686}$	$\frac{10}{0.57}$	$\frac{15}{1.86}$	$\frac{15}{1.8}$	$\frac{20}{3.6}$
depots-20	$\frac{20}{26.9}$	$\frac{25}{158.7}$	T		$\frac{10}{0.5}$	$\frac{15}{1.85}$	$\frac{10}{1.4}$	$\frac{15}{2.81}$
storage-17	$\frac{10}{0.54}$	$\frac{15}{621.6}$	$\frac{10}{\geq 3K}$	$\frac{15}{130}$	$\frac{5}{0.13}$	$\frac{10}{1.06}$	$\frac{5}{0.12}$	$\frac{10}{1.02}$
storage-20	$\frac{15}{\geq 3K}$	$\frac{20}{1300}$	T		$\frac{5}{0.46}$	$\frac{10}{2.74}$	$\frac{5}{0.32}$	$\frac{10}{1.15}$
storage-23	$\frac{20}{\geq 3K}$	$\frac{25}{2750}$	M		$\frac{5}{0.49}$	$\frac{10}{12.03}$	$\frac{5}{0.45}$	$\frac{10}{4.71}$
storage-24	$\frac{15}{250}$	T	M		$\frac{10}{\geq 3K}$	$\frac{15}{4.07}$	T	
driverlog-15	$\frac{10}{9.47}$	$\frac{15}{151.5}$	$\frac{10}{1.1}$	$\frac{15}{45}$	$\frac{5}{0.04}$	$\frac{10}{0.21}$	$\frac{5}{0.04}$	$\frac{10}{0.19}$
driverlog-19	$\frac{15}{6.1}$	T	M		$\frac{10}{3.14}$	$\frac{15}{1325}$	$\frac{10}{5.68}$	$\frac{15}{1165}$
driverlog-20	$\frac{10}{1}$	T	M		T		$\frac{10}{\geq 3K}$	$\frac{15}{457}$
TPP-21	$\frac{10}{0.0}$	$\frac{15}{57.9}$	$\frac{10}{0.0}$	$\frac{15}{52.69}$	$\frac{5}{1.37}$	$\frac{10}{2.18}$	$\frac{5}{0.33}$	$\frac{10}{1.63}$
TPP-28	$\frac{10}{0.12}$	M	M		$\frac{5}{1.05}$	$\frac{10}{20.0}$	$\frac{5}{1.07}$	$\frac{10}{31.66}$
TPP-30	$\frac{10}{0.17}$	M	M		$\frac{5}{1.38}$	$\frac{10}{38.5}$	$\frac{5}{1.42}$	$\frac{10}{25.38}$
zenotravel-14	$\frac{5}{0.03}$	$\frac{10}{0.5}$	$\frac{5}{5.28}$	$\frac{10}{15.82}$	NA	$\frac{5}{0.27}$	NA	$\frac{5}{0.214}$
zenotravel-20	$\frac{5}{0.06}$	M	M		$\frac{10}{\geq 3K}$	$\frac{15}{2314}$	$\frac{10}{\geq 3K}$	$\frac{15}{204}$
rovers-20	$\frac{15}{\geq 3K}$	$\frac{20}{1417}$	$\frac{5}{0.0}$	$\frac{10}{3.25}$	$\frac{5}{0.13}$	$\frac{10}{0.79}$	$\frac{5}{0.14}$	$\frac{10}{0.29}$
rovers-24	$\frac{10}{0.6}$	T	$\frac{10}{3.09}$	$\frac{15}{14.81}$	$\frac{10}{2.91}$	$\frac{15}{0.13}$	$\frac{10}{2.49}$	$\frac{15}{0.69}$
rovers-29	$\frac{5}{0.05}$	$\frac{10}{472}$	M		NA	$\frac{5}{0.35}$	NA	$\frac{5}{0.38}$

Table 2: Runtime comparison of satisfaction versus refutation in the form $\frac{n}{sec}$.

tation appears to be a bigger problem for PARA-L when compared to other suboptimal SAT encodings. In examining reasons for this, we have already seen that for some domains post-serialisation techniques have a clear advantage over PARA-L in terms of the number of planning steps required. This is primarily due to their ability to plan for conflicting executions in parallel. For example, using a post-serialisation scheme, conflicting instantiations of `Lift`, `Move`, and `Drop` from STORAGE can be executed in parallel at a single plan step. In Table 2 we see this effect in terms of the size (and complexity) of problems PARA-L requests RSat produce refutation proofs for. For example, taking results for PARA-L in STORAGE-20 we have that RSat has to refute plan exists for an $n' = 20$ step problem. Here, post-serialisation techniques have $n' = 5$. On the other hand, when PARA-L does not have to plan for a longer horizon than other approaches, we see that it has a clear advantage in terms of the cost of refutation. For example, in PIPESWORLD we have that refutation performed for PARA-L by RSAT is relatively cheap. This observation is emphasized by the results for PIPESWORLD in Table 3.

Conclusions and Outlook

We have introduced a compact and efficient SAT encoding for planning with a restricted form of parallel action semantics. Along the lines of SATPLAN-06, MAXPLAN and post-serialisation SAT-based techniques, we leverage the latest technology from the SAT community to address planning by solving the corresponding decision problem. Our approach exploits plangraph analysis and is the first to use a lifted representation of actions with a parallel semantics. We have shown that our approach compares favourably in many respects with two state-of-the-art SAT-based planning techniques 1-LIN and R∃-STEP, and is generally faster and more efficient than the step-optimal planner SATPLAN-06; a winning entry from the optimal track of IPC-5. Overall, we dominate in hard domains such as FREECELL and PIPESWORLD. Moreover, our encoding is compact and requires comparatively few Boolean variables. On the downside, our proposed restrictions on action parallelism, due to our lifted representation of actions, means we often require more plan steps compared to other parallel approaches.

Because refutation is such an expensive component of SAT-based planning, future work should examine the effectiveness of query strategies from (Rintanen 2004) and (Streeter & Smith 2007) to mitigate this problem for our encoding. Future work should also evaluate how reliant our encoding is on plangraph analysis. In particular, the use of lifted representations of actions under parallel execution semantics should be evaluated in the case of direct-encodings of planning-as-SAT with a post-serialisation scheme. Finally, we would like to explore the direction of adding variables to our lifted encoding to mitigate the problem of interference, and thus allow more action parallelism and consequently a shorter planning horizon.

Acknowledgements

Thanks to Jussi Rintanen for useful discussions. NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

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

Chen, Y.; Xing, Z.; and Zhang, W. 2007. Long-distance mutual exclusion for propositional planning. In Proc. *IJCAI*.

Dimopoulos, Y.; Nebel, B.; and Koehler, J. 1997. Encoding planning problems in nonmonotonic logic programs. In Proc. *ECP*.

Ernst, M.; Millstein, T.; and Weld, D. S. 1997. Automatic sat-compilation of planning problems. In Proc. *IJCAI*.

Gerevini, A.; Dimopoulos, Y.; Haslum, P.; and Saetti, A. 2006. *5th International Planning Competition*.

	PARA-L					SATPLAN-06					1-LIN					R∃-STEP				
Problem	c	v	*sec.*	n	$\#a$	c	v	*sec.*	n	$\#a$	c	v	*sec.*	n	$\#a$	c	v	*sec.*	n	$\#a$
freecell-4	125.4	5.7	1249.5	14	33	8008.1	222.4	12583	15	36	425.1	113.5	50.6	13	34	425.9	113.5	113.2	13	33
freecell-6	253.5	9.3	66	16	48	M					1108.6	279.8	157.2	14	46	1109.8	279.8	144.5	14	45
freecell-7	484.5	16.6	16596	25	61	M					2075.4	542.7	27776	26	59	1996.9	522.4	22785	25	57
freecell-9	584.5	18.2	10026	24	58	M					T					T				
pipesworld-9	69.5	2.8	7.5	13	19	11429	28.1	708.2	11	22	661.4	171.4	143.9	11	21	662	171.4	114.8	11	21
pipesworld-12	117.5	4.7	68.5	17	28	13562	30	7771.5	16	28	717	167.3	2337.5	16	32	718.4	167.3	2206.4	16	28
pipesworld-14	183.1	6.4	50.2	20	32	T					T					T				
pipesworld-15	209.9	7	404.6	21	34	T					771.1	169.2	10709.5	20	34	T				
depots-9	307.2	11.1	2938.4	25	68	7150.4	43.5	8421.4	24	85	365.9	47	8.4	20	88	370.4	47	8.9	20	82
depots-12	425.6	13.4	6462.6	27	78	T					637.7	84.4	7.9	20	107	643.2	84.4	6.6	20	112
depots-15	503.7	14.5	6263.6	24	76	17585	69.7	9387.6	22	93	638.5	85.5	6.6	15	103	644.6	85.5	7.6	18	107
depots-20	585.3	14.3	418.6	21	79	16837	62	7172.2	16	94	676.6	95.3	5.5	14	131	683.5	95.3	7.6	14	126
storage-17	147.5	4.1	1820	14	32	3713.3	20.9	12845	13	35	178.9	39.8	21.5	8	33	180.5	39.8	21.6	8	31
storage-20	470.8	10.1	10711	19	46	T					401.4	87.6	288.5	9	39	404.3	87.6	200.2	9	37
storage-23	31672	1226.9	27215	25	65	M					938.9	200.6	3860.4	10	52	944.7	200.6	3392.1	10	53
storage-24	T					M					1115.5	238	6304.9	11	58	1121.6	238	6392.5	11	58
driverlog-15	221.6	4.6	50.8	14	48	560.4	181.6	48.75	11	54	118.1	35.8	0.6	9	51	119.9	35.8	0.5	9	51
driverlog-19	T					M					701.1	212.7	7648.4	15	151	706.6	212.7	7490.4	15	148
driverlog-20	M					M					906.2	279.4	11421	12	174	913.6	279.4	11953	12	177
TPP-21	175.3	7.2	3120.3	15	115	2665.5	43.8	3202.4	12	145	1124.3	292.2	5.4	10	161	1126.1	292.2	4.9	10	159
TPP-28	M					M					3363.5	875.7	31.8	10	221	3367.4	875.7	53.7	10	220
TPP-30	M					M					4466.3	1163	51.92	10	252	4471.2	1163	37.9	10	256
zenotravel-14	234	2.8	0.86	10	44	4650.3	19.1	65.8	6	42	100.8	26.2	0.38	4	33	101.8	26.2	0.35	4	34
zenotravel-20	T					M					1888	488	30183	20	185	1640	423	24310	13	151
rovers-20	1295.2	10.8	20744	18	83	2346.1	33.4	25.6	10	98	191.4	50.8	2.8	8	97	192.5	50.8	2.9	8	104
rovers-24	T					7855.7	69.9	95.8	13	122	389.7	102.1	7.3	11	123	392.3	102.1	6.64	11	126
rovers-29	1689.3	9.4	298.1	10	70	3053.7	30.8	11.2	6	73	292.5	77.6	0.74	5	75	296.4	77.6	0.78	5	77

Table 3: Presents 1-step ramp-up results for problems from Table 1.

Helmert, M. 2006. New complexity results for classical planning benchmarks. In *ICAPS*.

Hoffmann, J., and Edelkamp, S. 2005. The deterministic part of IPC-4: an overview. *Journal of Artificial Intelligence Research* 24:519–579.

Kautz, H., and Selman, B. 1992. Planning as satisfiability. In Proc. *ECAI*.

Kautz, H., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic, and stochastic search. In Proc. *AAAI*.

Kautz, H. A., and Selman, B. 1999. Unifying SAT-based and graph-based planning. In Proc. *IJCAI*.

Kautz, H.; McAllester, D.; and Selman, B. 1996. Encoding plans in propositional logic. In Proc. *KR*.

Kautz, H. A.; Selman, B.; and Hoffmann, J. 2006. SatPlan: Planning as satisfiability. In Abstracts *IPC-05*.

Kautz, H. A. 2006. Deconstructing planning as satisfiability. In Proc. *AAAI*.

Pipatsrisawat, K., and Darwiche, A. 2007. Rsat 2.0: Sat solver description. Technical Report D–153, Automated Reasoning Group, Computer Science Department, UCLA.

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. 2004. Evaluation strategies for planning as satisfiability. In *ECAI*.

Ryan, L. 2003. *Efficient Algorithms for Clause-Learning SAT Solvers*.

Streeter, M., and Smith, S. 2007. Using decision procedures efficiently for optimization. In *ICAPS*.

Wehrle, M., and Rintanen, J. 2007. Planning as satisfiability with relaxed e-step plans. In Proc. *AAI*.