# Heuristics for Planning with SAT and Expressive Action Definitions

**Jussi Rintanen**
The Australian National University, Canberra, Australia

### Abstract

We present the first effective SAT heuristics for planning with expressive planning languages such as ADL. Recently, SAT heuristics for STRIPS planning have been introduced. In this work we show that the basic ideas in the heuristic can be generalized to actions with conditional effects but without disjunction, and that disjunction requires a more fundamental analysis of the STRIPS heuristic, which, despite complications, will still lead to a natural heuristic which can be implemented efficiently. The experimental analysis shows substantial and systematic improvements over the state of the art in planning with SAT with ADL.

## Introduction

Conditional effects and disjunctivity (alternatives) in preconditions are an important feature of expressive language for planning. They naturally arise as a way of compactly encoding action sets, and are also unavoidable in planning with observational incompleteness where restriction to unconditional actions makes it impossible to express a problem.

Many implementations of classical planning limit to STRIPS, and most of those that venture beyond, reduce all actions to either STRIPS or to actions without disjunction. Although a feasible strategy for many of the standard benchmarks, the lack of full support for disjunctivity and conditional effects is still problematic. This has been the case especially when translating other complex combinatorial problems into planning, where the reductive approach often leads to impractically large planning problems, making planners which do not natively support full PDDL/ADL unusable.

For (forward) state-space search, however, implementing the general form of actions is trivial: computing the successor state of a given state with respect to a given action, which may contain disjunctive (pre)conditions and conditional effects, can be done in linear time by a very simple and efficient algorithm. Complications arise not from the search method itself, but from other components of planner, for example the implementation of heuristics. Most of the research papers on heuristics published in the recent years restrict to STRIPS only (for an exception see (Rintanen 2006)), and this has consequently meant that many planning algorithm implementations similarly bypass the conceptual difficulties of handling more expressive planning languages.

For SAT-based planning, as far as general-purpose SAT solvers are used, general ADL actions do not pose any difficulty: disjunctivity and conditionality can be represented as arbitrary propositional formulas, which can be easily and efficiently translated into CNF: the Tseitin transformation and its enhancements (Tseitin 1962; Jackson and Sheridan 2005; Manolios and Vroon 2007) avoid the worst-case exponential size increase inherent in naive CNF translations.

This paper addresses the problem of defining planning-specific SAT search heuristics in the context of expressive languages such as PDDL, in which problems with disjunctivity and conditionality do arise. Simple heuristics have recently been demonstrated to lead to an impressive performance for planning with SAT (Rintanen a2010; b2010). The simplicity of the heuristics is at least partly due to the restriction to STRIPS actions only. In this work we show that the simplicity and elegance generalize to conditional effects without disjunction, and further extensions allow covering general actions with arbitrary forms of disjunction. The main technical challenges are posed by disjunction: backward-chaining with disjunctive preconditions is more complex than with conjunctive preconditions. We propose effective solutions to problems arising in this setting. Our experimental analysis shows that the generalization leads to an equally impressive speedup as in the STRIPS case, and the resulting planner outperforms earlier ones with the ADL problems from the planning competitions.

The structure of the paper is as follows. In the next section we describe the background of the work in planning. Then we show how actions without disjunction but with conditional effects can be handled by heuristics similar to the STRIPS case. The core results of the paper address the disjunctivity in goals, conditions and preconditions by defining heuristics for the most general form of actions. This is followed by an evaluation of the proposed techniques with standard benchmark problems, and a discussion of the related and future work.

## Background

We use a propositional logic with some finite set $A$ of atomic propositions, the constants *true* $\top$ and *false* $\bot$, and formulas that are constructed from $A$, $\top$ and $\bot$ with $\neg$, $\vee$ and

$\wedge$. We assume all formulas to be in Negation Normal Form. This means that $\neg$ only occurs in front of atomic formulas. Hence all formulas are formed from positive or negative literals with $\wedge$ and $\vee$. Literals are atomic propositions $a \in A$ or their negations $\neg a$. The complement $\overline{l}$ of a literal $l$ is defined by $\overline{a} = \neg a$ and $\overline{\neg a} = a$.

## Formalization of Planning

We use the propositional logic to formalize a general planning language with disjunctive preconditions and conditional effects. We use a finite set of propositional variables $A$ (which is often related to the state variables of a planning problem). A *valuation* $v : A \to \{0, 1\}$ is a function (a subset of the Cartesian product $A \times \{0, 1\}$) that maps propositional variables to truth-values 0 or 1. A valuation may be *partial*, meaning that $v(a)$ may not be defined for some $a \in A$ (i.e., neither $(a, 0) \in v$ nor $(a, 1) \in v$), and otherwise the valuation is complete. We sometimes write $v(l)$ for negative literals $l = \neg a$, which means $1 - v(a)$.

A state is a valuation of a set $A$ of state variables. An action is a pair $\langle p, c\rangle$ where $p$ is a propositional formula over $A$ and $c$ is a set $\{\phi_1 \rhd l_1, \phi_2 \rhd l_2, \ldots, \phi_n \rhd l_n\}$ of conditional effects where the $\phi_i$ are formulas and $l_i$ are literals, and for any $l$ there is at most $\phi \rhd l$ with $l$ on the right hand side. An action can be taken in a state $s$ if $s \models p$. The action has the effect $l$ if there is $\phi \rhd l \in c$ and $s \models \phi$. An effect $l$ is *unconditional* if $\phi = \top$ for all $\phi \rhd l \in c$. An action is a *STRIPS action* if the precondition is a conjunction of (positive) literals and all effects are unconditional.

## Reduction to STRIPS

Many of the implemented planning systems avoid the problems stemming from disjunctivity and conditional effects by reducing everything to STRIPS actions. This is a feasible strategy when the number of conditional effects in each action is very small and when the disjunctions are close to being in the Disjunctive Normal Form (DNF). When these properties, however, are not satisfied, there may be an impractically big increase in the size of the action set. Disjunctivity from the actions can be easily removed by the following sequence of transformations.

1. Every formula is translated into DNF.
2. Conditional effects $(\phi_1 \vee \cdots \vee \phi_n) \rhd l$ are split to $n$ conditional effects $\phi_1 \rhd l, \ldots, \phi_n \rhd l$ without disjunction.

   Similarly, actions $\langle p_1 \vee \cdots \vee p_n, c\rangle$ are split to $n$ actions $\langle p_1, c\rangle, \ldots, \langle p_n, c\rangle$ without disjunction.
3. Non-STRIPS actions $\langle p, \{\phi_1 \rhd l_1, \ldots, \phi_n \rhd l_n\}\rangle$ are reduced to STRIPS actions.

   Generate every subset $\{(\phi'_1, l'_1), \ldots, (\phi'_{n'}, l'_{n'})\}$ of $\{(\phi_1, l_1), \ldots, (\phi_n, l_n)\}$. Each subset represents one possible way the action could be executed. We create an action without conditional effects for each such subset. The action is simply $\langle p \wedge \phi'_1 \wedge \cdots \wedge \phi'_{n'}, \{\top \rhd l'_1, \ldots, \top \rhd l'_{n'}\}\rangle$.

Steps 1 and 3 can both increase the size of the action description by an exponential. This exponential increase, which does indeed occur with practical planning problems, sometimes turns an easily solvable problem instance to an unsolvable one, often simply because the resulting action set is far too large to fit in the main memory.

Some planners perform the first two steps to eliminate disjunction, but not the third step and do handle non-disjunctive conditional effects inside the planning algorithm.

## Translation into SAT

The early translations of planning into SAT restricted to STRIPS (Kautz and Selman 1996). Even some recent works on planning with SAT make the same restrictions because they are based on planning graphs (Blum and Furst 1997) which do not have a natural generalization to disjunctive and conditional actions. How actions with disjunctions and conditional effects are efficiently translated into SAT is well known (Rintanen, Heljanko, and Niemelä 2006). We outline the translation for completeness.

A problem instance is determined by a set $A$ of state variables, a set $O$ of actions, an initial state $I$, and a goal $G$ (a propositional formula). The translation is parameterized by a horizon length $T \geq 0$. The propositional variables that are needed are time-tagged variables $a@t$ for every state variable $a \in A$ and $o@t$ for every action $o \in O$ (we use the action description itself as its "name"). When we have a formula $\phi$ with variables without time-tags, we can construct the formula $\phi@t$ by tagging all variables with $@t$.

The translation of actions into SAT is as follows. For every action $o = \langle p, c\rangle$ we have the following formulas for $t \in \{0, \ldots, T-1\}$. For the precondition $p$ we have

$$o@t \to p@t$$

for expressing the precondition, and for all $\phi \rhd l \in c$

$$(o@t \wedge \phi@t) \to l@(t+1)$$

for the effects. For unconditional effects this simplifies to $o@t \to l@(t+1)$.

For expressing the conditions under which a state variable $a \in A$ does not change its value we have *frame axioms*

$$(a@t \wedge \neg a@(t+1)) \to ((o_1 \wedge \phi_1) \vee \cdots \vee (o_n \wedge \phi_n))$$

for all $t \in \{0, \ldots, T-1\}$, where $o_1, \ldots, o_n$ are all the actions that can make $a$ false, and $\phi_1, \ldots, \phi_n$ are the respective antecedents of conditional effects, that is, $o_i = \langle p_i, c_i\rangle$ and $\phi_i \rhd \neg a \in c_i$ for all $i \in \{1, \ldots, n\}$. There is an analogous frame axiom for the change from false to true.

Under different definitions of (parallel or sequential) plans, different combinations of simultaneous actions are permitted. Whenever the above formulas don't preclude an unallowed combination, additional formulas are required to encode the constraints. For all main notions of plans there are linear-size encodings of these constraints (Rintanen, Heljanko, and Niemelä 2006).

Finally, we have $a@0$ for every $a \in A$ such that $I(a) = 1$ and $\neg a@0$ for every $a \in A$ such that $I(a) = 0$, and $G@T$.

The translation is linear in the product of $T$ with the sum of sizes of $A$, $O$ and $G$, and it can be easily translated into CNF by efficient linear-time (and consequently linear-size) translations (Tseitin 1962; Jackson and Sheridan 2005; Manolios and Vroon 2007).

```
 1: PROCEDURE support(G, O, T, v)
 2: empty the stack;
 3: push all l ∈ G into the stack;
 4: C := ∅;
 5: WHILE the stack is non-empty DO
 6:   pop l@t from the stack;
 7:   t' := t − 1;
 8:   found := 0;
 9:   REPEAT
10:     IF v(o@t') = 1 FOR SOME o ∈ O with l ∈ eff(o)
11:     THEN
12:       FOR ALL l' ∈ prec(o) DO push l'@t' into the stack;
13:       found := 1;
14:     ELSE IF v(l@t') = 0 THEN
15:       o := any o ∈ O such that l ∈ eff(o) and v(o@t') ≠ 0;
16:       C := C ∪ {o@t'};
17:       FOR ALL l' ∈ prec(o) DO push l'@t' into the stack;
18:       found := 1;
19:     t' := t' − 1;
20:   UNTIL found = 1 OR t' < 0;
21: END WHILE
22: RETURN C;
```

Figure 1: Computation of the heuristic for STRIPS

## The Heuristic for STRIPS

Rintanen presented a heuristic for SAT algorithms that limits to STRIPS (Rintanen a2010). In STRIPS all top-level goals and action preconditions are (conjunctions of) literals, and the backward chaining part of the heuristic can be done one literal at the time. The algorithm for computing the heuristic is given in Figure 1.

The algorithm starts from a top-level goal literal, and goes backward step by step until it finds a time point in which either an action makes the literal true, or the literal is false. In the first case the algorithm pushes the preconditions of the action in the stack and proceeds with these literals as the new goals. In the second case there are one or more actions that can turn the literal from false to true. One of these actions is chosen (e.g. the first one) and included in the set $C$ of candidate decision variables. This computation is repeated until the stack is empty or a termination condition is reached, e.g. the cardinality of $C$ reaches a predefined bound, which could be 1 or more. Additional heuristics can be incorporated in the basic algorithm (Rintanen b2010). We ignore them here as they are orthogonal to the issues with disjunctive and conditional actions.

## Conjunctive Conditional Effects

A formula is conjunctive if it is a conjunction of one or more literals and constants $\top$ or $\bot$. An action $(p, e)$ is *conjunctive*, if $p$ is conjunctive and for every $\phi \rhd l$ the condition $\phi$ is conjunctive.

The definition of heuristics for STRIPS by Rintanen (a2010) extends to conjunctive actions more easily than to arbitrary actions. Instead of considering literals to be achieved by actions, we consider them to be achieved by specific conditional effects $\phi \rhd l$ of actions. When translating a planning problem into a propositional formula, we introduce a propositional variable for every conditional effect (effects $\phi \rhd l_1$ and $\phi \rhd l_2$ of one action may use the same propositional variable, because these conditional effects must always take place together.)

For example, the action $\langle \phi, \{a \rhd b, d \rhd e\}\rangle$ is translated into a formula with the two auxiliary variables $x_1$ and $x_2$ for the two conditional effects as follows.

$$
\begin{array}{l}
o@t \rightarrow \phi@t \\
o@t \wedge a@t \rightarrow x_1@t \\
x_1@t \rightarrow b@(t+1) \\
x_1@t \rightarrow a@t \\
x_1@t \rightarrow o@t \\
\\
o@t \wedge d@t \rightarrow x_2@t \\
x_2@t \rightarrow e@(t+1) \\
x_2@t \rightarrow d@t \\
x_2@t \rightarrow o@t
\end{array}
$$

With the $x_i$ variables the frame axioms

$$(b@t \wedge \neg b@(t+1)) \rightarrow x_1@t \vee \cdots$$

can be trivially turned to clauses.

The framework of Rintanen (a2010) can be used for conjunctive actions by replacing the use of action variables by the auxiliary variables $x_i$: instead of finding an action that supports a subgoal, we find a (conditional) effect that supports the subgoal. In the procedure in Figure 1 the role of the preconditions as new subgoals is taken by formulas $\text{prec}(x) = p \wedge \phi$, where $p$ is the precondition of the action and $\phi$ is the antecedent of the conditional effect which supports a literal $l$ which is the effect of the conditional effect corresponding to the variable $x$. We similarly define $\text{eff}(x) = \{l\}$.

The general (disjunctive) heuristic in the next section directly builds on the construction in this section. Later we denote the set of all $x$ variables by $X$.

## Heuristics for Disjunctive Problems

The main problem to solve when extending the heuristic to disjunctive problems is the determination of which new subgoals to satisfy to make a disjunctive (sub)goal true. Similarly to the conjunctive case (with single literals as subgoals), we have to determine the earliest time points in which parts of a disjunctive goal can become true, and then identify one or more actions that can achieve this.

The first idea, which does not work, is to identify the earliest time point at which the whole formula could be true. However, it is possible that not all literals in the formula can be made true at that time point: some of them may have to be made true earlier. Hence we will be following a different approach.

1. Determine which literals cannot be made true (this is the literals that are false at the time when the formula should be true.)
2. Determine which literals in the formula are made true by actions already in the partial plan, possibly at a much earlier time point.

```
1: PROCEDURE value(l, t, v);
2: IF v(l@t) = 0 THEN RETURN (false, ∅);
3: WHILE t > 0 DO
4:   t := t − 1;
5:   IF v(x@t) = 1 for some x with effect l
6:   THEN RETURN (true, {x@t});
7:   IF v(l@t) = 0 THEN
8:     x := any effect variable with effect l such that v(x@t) ≠ 0;
9:     RETURN (unknown, {x@t});
10: END DO
11: RETURN true;
```

Figure 2: Procedure for determining the value of a literal

3. Determine which additional literals should be made true to satisfy the formula. Because of the disjunctivity, there may be several alternative sets of literals for achieving this.
4. Find one action for each of the additional literals, and add it to the set of candidate decision variables. As in the STRIPS case, there may be several actions to choose from.
5. Use preconditions of the existing and candidate actions as new subgoals.

Checking whether the formula is already true and which additional literals are needed to make it true contains the NP-hard satisfiability problem, which would in the worst case, for arbitrarily large formulas, require exponential time. For this reason we perform this check incompletely. However, the incompleteness will not compromise the correctness of the overall planning algorithm. As in the STRIPS case (Rintanen a2010), the correctness of the conflict-directed clause learning (CDCL) algorithm (Moskewicz et al. 2001), and therefore of planning, is not affected by the way decision variables are chosen: any "wrong" decisions will be detected by CDCL and will lead to "right" decisions through the learning of clauses that prevent making the wrong decisions again.

Of the above 5 steps, for a given $l@t$ and the current valuation $v$, the first two are handled by the procedure value$(l, t, v)$ we will be describing next. Steps 3 to 4 are taken care of by the procedure select$(\phi, t, v)$ for a goal formula $\phi$ at time $t$. And the last step 5 is by the procedure in Figure 4 which replaces the STRIPS heuristic in Figure 1.

## Determining Tentative Values of Literals

In Figure 2 we give an algorithm for determining the tentative truth-value of a literal for a (sub)goal formula that must be *true* at time $t$. The literal is definitely *false* if it is *false* at $t$. Otherwise it is possible to make it *true*. The procedure goes backward one step at a time. If it finds an effect $x$ that makes $l$ *true*, then $l$ is considered to be *true* because it has support in the current partial plan. If $l$ is *false* at $t$ without support at $t$, then its truth-value is considered to be undetermined. Since $l$ is not *false* at $t + 1$, this suggests choosing a support for it at $t$. If this literal is chosen as a candidate

```
1: PROCEDURE select(φ, t, v)
2: CASE φ OF
3:   φ1 ∧ φ2:
4:     (b1, S1, P1) := select(φ1, t, v);
5:     (b2, S2, P2) := select(φ2, t, v);
6:     RETURN (b1 and b2, S1 ∪ S2, P1 ∪ P2);
7:   φ1 ∨ φ2:
8:     (b1, S1, P1) := select(φ1, t, v);
9:     (b2, S2, P2) := select(φ2, t, v);
10:    IF b1 = false THEN RETURN (b2, S2, P2);
11:    RETURN (b1, S1, P1);
12:  ¬a:
13:  a:
14:    CASE value(l, t, v) OF
15:      (true, S): RETURN (true, ∅, S);
16:      (false, ∅): RETURN (false, ∅, ∅);
17:      (unknown, S): RETURN (true, S, ∅);
18:  ⊥: RETURN (false, ∅, ∅);
19:  ⊤: RETURN (true, ∅, ∅);
```

Figure 3: Procedure for identifying needed literals

| $\vee$ | $U$ | $T$ | $F$ |
|---|---|---|---|
| $U$ | $U$ | $T$ | $U$ |
| $T$ | $T$ | $T$ | $T$ |
| $F$ | $U$ | $T$ | $F$ |

| $\wedge$ | $U$ | $T$ | $F$ |
|---|---|---|---|
| $U$ | $U$ | $U$ | $F$ |
| $T$ | $U$ | $T$ | $F$ |
| $F$ | $F$ | $F$ | $F$ |

| $\neg$ | |
|---|---|
| $U$ | $U$ |
| $T$ | $F$ |
| $F$ | $T$ |

Table 1: 3-valued Kleene truth-tables

to be made true, an effect for it is returned as the second component of the returned value.

**Proposition 1** *If value$(l, t, v)$ returns $(true, \{x@t\})$, then $v(x@t) = 1$ and $l \in eff(x)$.*
*value$(l, t, v)$ returns $(false, \emptyset)$ if and only if $v(l@t) = 0$.*
*If value$(l, t, v)$ returns $(unknown, \{x@t\})$, then $v(x@t)$ is undefined and $l \in eff(x)$.*

For a given effect $x$, the associated precondition is denoted by prec$(x)$. As discussed earlier in connection with non-disjunctive actions, for unconditional effects in an action $\langle p, e\rangle$ simply prec$(x) = p$, and for conditional effects $\phi \rhd l$ in $e$ prec$(x) = p \wedge \phi$.

## Determining New Subgoals

In Figure 3 we give a procedure select$(\phi, t, v)$ which computes $(b, S, P)$, where

- $b \in \{\text{true}, \text{false}, \text{unknown}\}$ is the truth-value of $\phi$ at $t$,
- $S$ is a set of literals such that making them *true* would be sufficient to guarantee the truth of $\phi$ at time $t$, when the current partial valuation is $v$, and
- $P$ are the effects that are already in the plan and support $\phi$, but which may have unsupported preconditions.

Essentially, one of the 3-valued semantics for propositional logic, expressed as truth-tables in Table 1, determine the truth-value of the formula. If it is *true*, nothing needs to

be done. If it is *unknown*, a set of literals that are (seemingly) sufficient to turn it from *unknown* to *true* is found.

**Proposition 2** *If select*$(\phi, t, v)$ *returns* $(\mathit{false}, S, P)$ *for some* $S$ *and* $P$*, then* $v \models \neg\phi@t$*.*

*If* $v \models \neg\phi@t$ *and the valuation* $v$ *is complete for propositional variables for time* $t$*, then select*$(\phi, t, v)$ *returns* $(\mathit{false}, S, P)$ *for some* $S$ *and* $P$*.*

The base case of the procedure on lines 14-17 uses value$(l, t, v)$ to determine a truth-value *true*, *false* or *unknown* for the literal in question. If the literal is true, then the truth-value together with the two sets, the empty set for the required effects and a singleton set with one already supporting effect, is returned. If the literal is *false*, then two empty sets are returned. If the literal is *unknown*, then a singleton set of one required effect and the empty set for already supporting actions is returned.

The inductive case for the conjunction on line 3 returns the truth-value of the conjunction of the two conjuncts and the union of the corresponding sets of effects, which all have to be *true* to satisfy the formula. Here the value *unknown* does not show up any more, as in the base case only the distinction between *false* and non-*false* values is made.

The inductive case for the disjunction on line 7 returns the truth-value of the disjunction and the actions/effects for one of the non-false disjuncts. An implementation of the procedure could use additional heuristics, for example to minimize the number of additional effects that need to be taken to fulfill the disjunctive subgoal. We could also prefer a *true* disjunct over an *unknown* one, to minimize the number of new effects introduced in the plan.

Notice that a literal $l$ and its complement $\bar{l}$ may both occur in a disjunctive formula, and according to the procedure, they may both have the value *unknown*, or one may be *unknown* and the other *true*. The procedure in Figure 3 to find actions or effects to satisfy a (sub)goal may choose to satisfy both $l$ and $\bar{l}$. However, at most one of them can be chosen as the next decision literal. Later, the choice of the complement of the decision literal leads either to a conflict in the CDCL algorithm, or to the use of some unrelated literal to satisfy the subgoal. Essentially, we relegate the NP-hard part of subgoal satisfaction to the CDCL algorithm.

For select$(\phi, t, v) = (\text{true}, \{x_1@t_1, \ldots, x_n@t_n\}, P)$, we can use any member of $\{x_1@t_1, \ldots, x_n@t_n\}$ as a decision variable, or proceed to compute $\{l@t | l \in \text{prec}(x) | x@t \in S\}$ and further literals (Rintanen b2010).

Figure 4 is an adaptation of the STRIPS version from Figure 1. The changes are the calls to select$(G, T, v)$ to identify the literals supporting the goal formula $G$. As the procedure may be called when the CDCL algorithm does not yet know that the subgoal $G$ is impossible to satisfy – a difference to the STRIPS case – there is a test on line 4 for this.

One of the variables returned by support$(G, O, T, v)$ is used as the next decision variable in the CDCL algorithm (Rintanen b2010). The implementation described by Rintanen (b2010) has heuristics for ordering the generation of the actions/effects (replacing the stack with a priority queue)

```
 1: PROCEDURE support(G, O, T, v)
 2: empty the stack;
 3: (b, L, P) := select(G, T, v);
 4: IF b = false OR T = 0 THEN RETURN ∅;
 5: C := L;
 6: push all l@t ∈ L ∪ P into the stack;
 7: WHILE the stack is non-empty DO
 8:   pop x@t from the stack;
 9:   (b, L, P) := select(prec(x), t, v);
10:   C := C ∪ L;
11:   push all x@t ∈ L ∪ P into the stack;
12: END WHILE
13: RETURN C;
```

Figure 4: Computation of the heuristic for ADL

and for limiting the computation to actions/effects that support one top-level goal at a time. These modifications are orthogonal to the computation of the decision variables by support$(G, O, T, v)$, and therefore fully compatible with the generalized heuristic.

## Evaluation

We have implemented the new heuristic and compared it to the VSIDS heuristic (Moskewicz et al. 2001) as well as planners that don't use SAT as their search method. The comparison is with the non-optimizing classical planning problem, in which no guarantees of the quality of the plans in terms of number of actions or costs are required.

Following Rintanen (a2010), the planner Madagascar with the VSIDS heuristic is called *M* and with the planning heuristic extended to cover full ADL is called *Mp*. The Mp planner (Rintanen b2010) is used as is except for the modified encoding and the heuristic extended to support ADL. The VSIDS version of the planner, M, is unchanged, as it handled the full language already. The SAT solver of the planner has been demonstrated to outperform best existing general-purpose SAT solvers (Rintanen a2010). These planners do not optimize the parallel plan length, as some SAT-based planners earlier did, and interleave the solution of SAT problems for several horizon lengths, giving more time to shorter horizon lengths according to a geometric series, thus avoiding the unnecessary proofs of parallel optimality (Rintanen a2010; Rintanen, Heljanko, and Niemelä 2006).

We compiled the planners with GCC 4.4.5 and ran them in a high performance 2.0 GHz Xeon E5405 cluster under Linux, with 4 GB of memory for each planner run.

We test the planners with IPC benchmarks which contain conditional effects or disjunctive preconditions (ADL). Practically all of the instances have a plan, and the considered planners are cannot effectively show that an instance has no solution plans.

To calibrate the results with the state of the art, we also tried some other planners. The YAHSP planner of Vidal, which performs best of all the non-SAT planners with the STRIPS problems from the planning competitions, does not currently support conditional effects and disjunctivity, so we could not compare to it. Of the other well-known planners

| | | Mp | M | LAMA | FF |
|---|---|---|---|---|---|
| 1998-ASSEMBLY-ADL | 24 | 23 | 18 | 24 | 24 |
| 2000-ELEVATOR-FULL | 143 | 138 | 139 | 135 | 132 |
| 2000-ELEVATOR-SIMPLE | 150 | 150 | 150 | 150 | 150 |
| 2000-SCHEDULE-ADL | 150 | 149 | 144 | 150 | 150 |
| 2002-SATELLITE-ADL | 20 | 20 | 20 | 20 | 20 |
| 2004-AIRPORT-ADL | 50 | 26 | 21 | 33 | 21 |
| 2004-OPTICAL-TELEG-ADL | 48 | 22 | 17 | 2 | 11 |
| 2004-PHILOSOPHERS-ADL | 48 | 48 | 48 | 12 | 12 |
| 2006-PATHWAYS-ADL | 30 | 30 | 30 | 29 | 16 |
| 2006-TRUCKS-ADL | 29 | 15 | 14 | 14 | 11 |
| 2008-OPENSTACKS-ADL | 30 | 15 | 14 | 30 | 30 |
| total | 722 | 636 | 615 | 599 | 577 |
| weighted score | 11 | 8.91 | 8.41 | 8.35 | 7.73 |

Table 2: Number of instances solved in 300 seconds

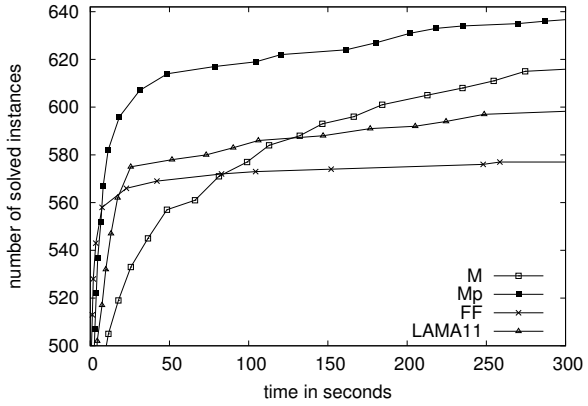


Figure 5: Number of instances solved in a given time

only LAMA and FF fully support the extended language. Of these two, LAMA[1] performs better on our problem set, as shown in Table 2. FF runs out of memory with half of the instances of Airport and Optical before starting the search. In the table we also give a "score" for each planner, based on the proportion of instances in each domain they solve.

A comparison of the number of problems solved is given in Figure 5. Figure 6 plots, on a logarithmic scale, the runtime of each instance as a dot so that the X-coordinate is for the new heuristic and the Y-coordinate for VSIDS. Figure 7 plots the respective plan sizes. Mp is usually faster, sometimes by two orders of magnitude. Plan sizes often slightly suffer, but all instances where there is a substantial difference, it is M that has produced a substantially longer plan.

As shown in Figure 8, the new heuristic substantially outperforms VSIDS with Assembly, Openstacks, Pathways and Schedule. On the other domains, the improvements are less consistent, and with Elevator-Full its performance is worse than that of VSIDS, and with Airport the runtimes are often higher although more instances are solved in 300 seconds.[2]

[1]We used the newest version of LAMA from March 1, 2011.

[2]Some of the 150 Elevator-Full instances have no solutions.

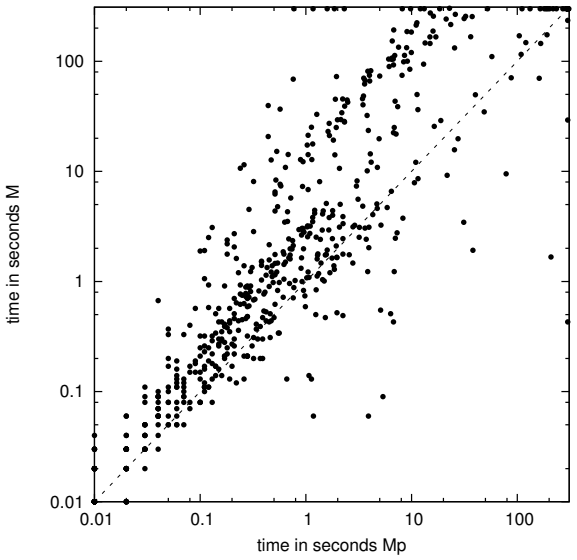


Figure 6: Comparison of runtimes by instance

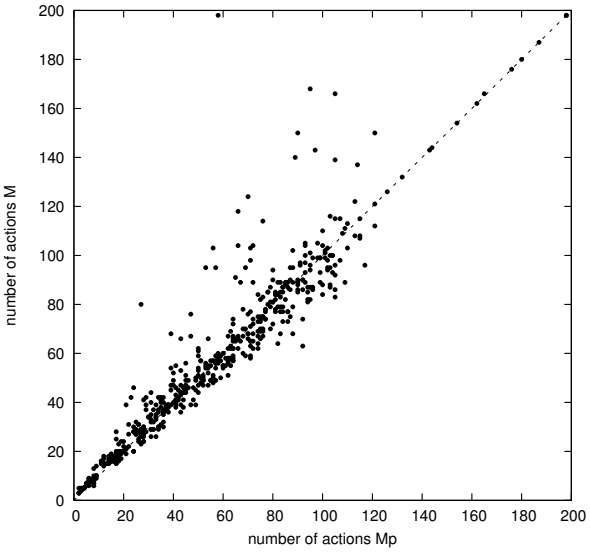


Figure 7: Comparison of plan sizes by instance

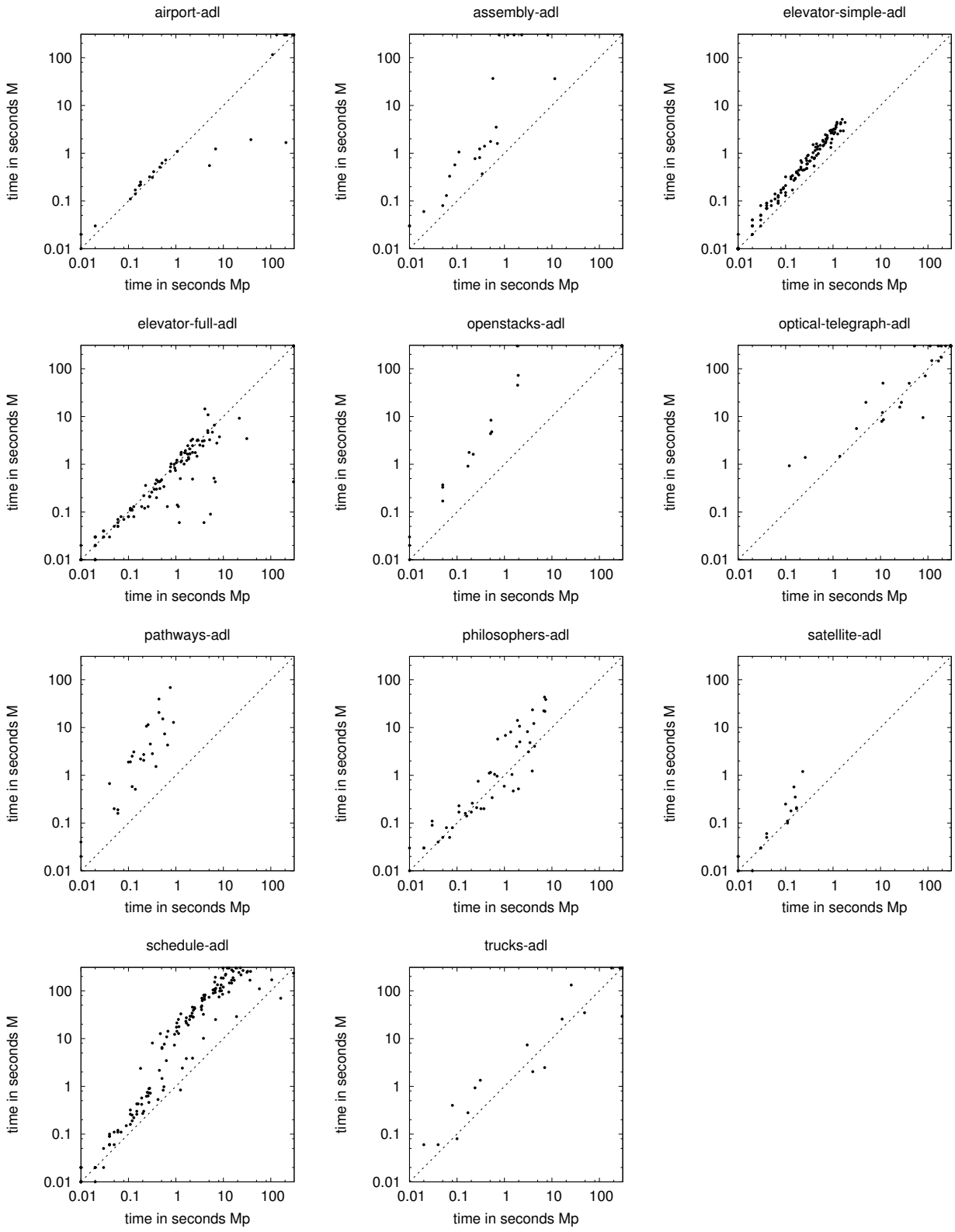


Figure 8: Comparison of runtimes for each domain

We guess that the decreased efficiency for some of the instances is because of the very complex disjunctive formulas in this domain combined with a naive implementation of the heuristic for disjunctive formulas.

## Related Work

Although no planning-specific heuristics for SAT have been presented earlier, some SAT solvers have been tailored to planning. Giunchiglia et al. (1998) limit the Davis-Putnam procedure to branch with action variables only, obtaining a performance improvement. No similar improvements have been demonstrated with modern CDCL solvers.

Application specific heuristics have rarely been shown to outperform general SAT heuristics such as VSIDS. Beame et al. (2004) give a heuristic that is very effective on a narrow class of SAT problems called pebbling formulas.

## Conclusions

We have presented the first SAT-based planning heuristic for ADL problems with conditional effects and disjunctive preconditions. The work extends the earlier STRIPS-based heuristic (Rintanen a2010). For conjunctive problems the extension is straightforward, but for disjunctive problems there are conceptual and theoretical difficulties, which for example require the use of approximate solutions to problems for which exact solutions are NP-hard.

Our experimental evaluation shows that the improvements demonstrated for the STRIPS heuristic carry over to the more general case, on average substantially improving runtimes in comparison to generic SAT heuristics, sometimes by 2 orders of magnitude or more. The results in this respects may not be surprising, but they more convincingly demonstrate that the impressive performance improvement with the STRIPS benchmarks is not specific to the simple structure of the actions in those benchmarks. A practical outcome of this work is the substantial strengthening of the ability to solve planning problems in languages more general than STRIPS.

Important topics for future research include making the search more flexible by reintroducing VSIDS style literal weighting to the heuristic, as well as preprocessing techniques that take into account the specific properties of SAT instances that represent planning.

## References


Beame, P.; Kautz, H.; and Sabharwal, A. 2004. Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligence Research* 22:319–351.

Blum, A. L., and Furst, M. L. 1997. Fast planning through planning graph analysis. *Artificial Intelligence* 90(1-2):281–300.

Giunchiglia, E.; Massarotto, A.; and Sebastiani, R. 1998. Act, and the rest will follow: Exploiting determinism in planning as satisfiability. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98) and the 10th Conference on Innovative Applications of Artificial Intelligence (IAAI-98)*, 948–953.

Jackson, P., and Sheridan, D. 2005. Clause form conversions for Boolean circuits. In Hoos, H. H., and Mitchell, D. G., eds., *Theory and Applications of Satisfiability Testing, 7th International Conference, SAT 2004, Vancouver, BC, Canada, May 10-13, 2004, Revised Selected Papers*, volume 3542 of *Lecture Notes in Computer Science*, 183–198. Springer-Verlag.

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

Manolios, P., and Vroon, D. 2007. Efficient circuit to CNF conversion. In Marques-Silva, J., and Sakallah, K. A., eds., *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing (SAT-2007)*, volume 4501 of *Lecture Notes in Computer Science*, 4–9. Springer-Verlag.

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

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. 2006. Unified definition of heuristics for classical planning. In Brewka, G.; Coradeschi, S.; Perini, A.; and Traverso, P., eds., *ECAI 2006. Proceedings of the 17th European Conference on Artificial Intelligence*, 600–604. IOS Press.

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

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

Tseitin, G. S. 1962. On the complexity of derivation in propositional calculus. In Slisenko, A. O., ed., *Studies in Constructive Mathematics and Mathematical Logic, Part 2*. New York - London: Consultants Bureau. 115–125.


These were excluded from the comparison. Further 4 instances not solved by either of our planners may be similarly unsolvable, but we could not confirm this with any of the planners.