Solving AI Planning Problems with SAT

Jussi Rintanen

EPCL, Dresden, November 2013
Solving the AI planning problem with SAT algorithms

Novelty: planning earlier viewed as a deduction problem

Idea:
- propositional variables for every state variable for every time point
- clauses that describe how state can change between two consecutive time points
- unit clauses specifying the initial state and goal states

Test material for local search algorithm GSAT [SLM92]

Resulting SAT problems that could be solved had up to 1000 variables and 15000 clauses.
Solving the AI planning problem with SAT algorithms

Novelty: planning earlier viewed as a deduction problem

Idea:

- propositional variables for every state variable for every time point
- clauses that describe how state can change between two consecutive time points
- unit clauses specifying the initial state and goal states

Test material for local search algorithm GSAT [SLM92]

Resulting SAT problems that could be solved had up to 1000 variables and 15000 clauses.
Solving the AI planning problem with SAT algorithms

Novelty: planning earlier viewed as a deduction problem

Idea:
- propositional variables for every state variable for every time point
- clauses that describe how state can change between two consecutive time points
- unit clauses specifying the initial state and goal states

Test material for local search algorithm GSAT [SLM92]

Resulting SAT problems that could be solved had up to 1000 variables and 15000 clauses.
Reduction of AI Planning to SAT
Kautz and Selman 1992 [KS92]

• Solving the AI planning problem with SAT algorithms
• Novelty: planning earlier viewed as a deduction problem
• Idea:
  • propositional variables for every state variable for every time point
  • clauses that describe how state can change between two consecutive time points
  • unit clauses specifying the initial state and goal states
• Test material for local search algorithm GSAT [SLM92]
• Resulting SAT problems that could be solved had up to 1000 variables and 15000 clauses.
Planning one of the first “real” applications for SAT (others: graph-coloring, test pattern generation, …)

Later, same ideas applied to other reachability problems:
- computer-aided verification (Bounded Model-Checking [BCCZ99])
- DES diagnosability testing [RG07] and diagnosis [GARK07]

SAT and related methods currently a leading approach to solving state space reachability problems in AI and other areas of CS.

Overlooked connection: the encoding is very close to Cook’s reduction from P-time Turing machines to SAT in his proof of NP-hardness of SAT [Coo71].
Significance

- Planning one of the first “real” applications for SAT (others: graph-coloring, test pattern generation, ...)
- Later, same ideas applied to other reachability problems:
  - computer-aided verification (Bounded Model-Checking [BCCZ99])
  - DES diagnosability testing [RG07] and diagnosis [GARK07]
- SAT and related methods currently a leading approach to solving state space reachability problems in AI and other areas of CS.
- Overlooked connection: the encoding is very close to Cook’s reduction from P-time Turing machines to SAT in his proof of NP-hardness of SAT [Coo71].
Classical (Deterministic, Sequential) Planning

- succinct s-t-reachability problem for graphs

- states and actions expressed in terms of state variables
- single initial state, that is known
- all actions deterministic
- actions taken sequentially, one at a time
- a goal state (expressed as a formula) reached in the end

Deciding whether a plan exists is PSPACE-complete. With a polynomial bound on plan length, NP-complete.
Formalization

A problem instance in (classical) planning consists of the following.

- set $X$ of state variables
- set $A$ of actions $\langle p, e \rangle$ where
  - $p$ is the precondition (a set of literals over $X$)
  - $e$ is the effects (a set of literals over $X$)
- initial state $I : X \rightarrow \{0, 1\}$ (a valuation of $X$)
- goals $G$ (a set of literals over $X$)
An action $a = \langle p, e \rangle$ is applicable in state $s$ iff $s \models p$. The successor state $s' = \text{exec}_a(s)$ is defined by

- $s' \models e$
- $s(x) = s'(x)$ for all $x \in X$ that don't occur in $e$.

**Problem**

Find $a_1, \ldots, a_n$ such that

$\text{exec}_{a_n}(\text{exec}_{a_{n-1}}(\cdots \text{exec}_{a_2}(\text{exec}_{a_1}(I)) \cdots)) \models G$?
Let \( x@t \) be propositional variables for \( t \in \{0, \ldots, T\} \) and \( x \in X \).

\( a = \langle p, e \rangle \) is mapped to \( E_a@t \) which is the conjunction of

- \( l@t \) for all \( l \in p \), and
- for all \( x \in X \)

\[
\begin{align*}
x@&(t + 1) \leftrightarrow \top \text{ if } x \in e, \\
x@&(t + 1) \leftrightarrow \bot \text{ if } \neg x \in e, \text{ and} \\
x@&(t + 1) \leftrightarrow x@t \text{ otherwise.}
\end{align*}
\]

Choice between actions \( a_1, \ldots, a_m \) expressed by the formula

\[
R@t = E_{a_1}@t \lor \cdots \lor E_{a_m}@t.
\]
Define

- $I_{@0}$ as $\bigwedge(\{x_{@0} | x \in X, I(x) = 0\} \cup \{\neg x_{@0} | x \in X, I(x) = 1\})$, and
- $G_{@T}$ as $\bigwedge_{l \in G} l_{@T}$

**Theorem**

*A plan of length $T$ exists iff*

$$\Phi_T = I_{@0} \land \bigwedge_{t=0}^{T-1} R_{@t} \land G_{@T}$$

*is satisfiable.*
Parallel Plans: Motivation

- Don’t represent all intermediate states of a sequential plan.
- Ignore relative ordering of consecutive actions.
- Reduced number of explicitly represented states $\Rightarrow$ smaller formulas $\Rightarrow$ easier to solve
Parallel plans (∀-step plans)
Kautz and Selman 1996

Allow actions $a_1 = \langle p_1, e_1 \rangle$ and $a_2 = \langle p_2, e_2 \rangle$ in parallel whenever they don’t interfere, i.e.
- both $p_1 \cup p_2$ and $e_1 \cup e_2$ are consistent, and
- both $e_1 \cup p_2$ and $e_2 \cup p_1$ are consistent.

**Theorem**

If $a_1 = \langle p_1, e_1 \rangle$ and $a_2 = \langle p_1, e_1 \rangle$ don’t interfere and $s$ is a state such that $s \models p_1$ and $s \models p_2$, then
$\text{exec}_{a_1}(\text{exec}_{a_2}(s)) = \text{exec}_{a_2}(\text{exec}_{a_1}(s))$. 
∀-step plans: encoding

Define $R^\forall @t$ as the conjunction of

$$x@ (t + 1) \leftrightarrow ((x@t \land \neg a_1@t \land \cdots \land \neg a_k@t) \lor a'_1@t \lor \cdots \lor a'_k, @t)$$

for all $x \in X$, where $a_1, \ldots, a_k$ are all actions making $x$ false, and $a'_1, \ldots, a'_k$ are all actions making $x$ true, and

$$a@t \rightarrow l@t \text{ for all } l \text{ in the precondition of } a,$$

and

$$\neg(a@t \land a'@t) \text{ for all } a \text{ and } a' \text{ that interfere}.$$

This encoding is quadrastrict due to the interference clauses.
Action \( a \) with effect \( l \) disables all actions with precondition \( \overline{l} \), except \( a \) itself.

This is done in two parts: disable actions with higher index, disable actions with lower index.

This is needed for every literal.
Allow actions $\{a_1, \ldots, a_n\}$ in parallel if they can be executed in at least one order.

- $\bigcup_{i=1}^{n} p_i$ is consistent.
- $\bigcup_{i=1}^{n} e_i$ is consistent.
- There is a total ordering $a_1, \ldots, a_n$ such that $e_i \cup p_j$ is consistent whenever $i \leq j$: disabling an action earlier in the ordering is allowed.

Several compact encodings exist [RHN06]. Fewer time steps are needed than with $\forall$-step plans. Sometimes only half as many.
Choose an **arbitrary fixed ordering** of all actions $a_1, \ldots, a_n$.

Action $a$ with effect $l$ disables all **later** actions with precondition $\overline{l}$.

This is needed for every literal.
Define a **disabling graph** with actions as nodes and with an arc from \( a_1 \) to \( a_2 \) if \( p_1 \cup p_2 \) and \( e_1 \cup e_2 \) are consistent and \( e_1 \cup p_2 \) is inconsistent.

The test for valid execution orderings can be limited to strongly connected components (SCC) of the disabling graph.

In many structured problems all SCCs are singleton sets. \( \implies \) No tests for validity of orderings needed during SAT solving.
The planning problem is reduced to SAT tests for

\[
\Phi_0 = I@0 \land G@0 \\
\Phi_1 = I@0 \land R@0 \land G@1 \\
\Phi_2 = I@0 \land R@0 \land R@1 \land G@2 \\
\Phi_3 = I@0 \land R@0 \land R@1 \land R@2 \land G@3 \\
\vdots \\
\Phi_u = I@0 \land R@0 \land R@1 \land \cdots \land R@(u - 1) \land G@u
\]

where \( u \) is the maximum possible plan length.

Q: How to schedule these tests?

How this is done has much more impact on planner performance than e.g. encoding details!
The sequential strategy

- Complete satisfiability test for $t$ before proceeding with $t + 1$.
- This is breadth-first search / iterative deepening.
- Guarantees minimality of horizon length.
- Slow.
The sequential strategy

1 2 3 4 5 6 7 8 9 ...

- Complete satisfiability test for $t$ before proceeding with $t + 1$.
- This is breadth-first search / iterative deepening.
- Guarantees minimality of horizon length.
- Slow.
The sequential strategy

- Complete satisfiability test for $t$ before proceeding with $t + 1$.
- This is breadth-first search / iterative deepening.
- Guarantees minimality of horizon length.
- Slow.
The sequential strategy

- Complete satisfiability test for $t$ before proceeding with $t + 1$.
- This is breadth-first search / iterative deepening.
- Guarantees minimality of horizon length.
- Slow.
**The sequential strategy**

1 2 3 4 5 6 7 8 9 ...

- Complete satisfiability test for $t$ before proceeding with $t + 1$.
- This is breadth-first search / iterative deepening.
- Guarantees minimality of horizon length.
- Slow.
The sequential strategy

- Complete satisfiability test for $t$ before proceeding with $t + 1$.
- This is breadth-first search / iterative deepening.
- Guarantees minimality of horizon length.
- Slow.
The sequential strategy

1 2 3 4 5 6 7 8 9 ...

- Complete satisfiability test for $t$ before proceeding with $t + 1$.
- This is breadth-first search / iterative deepening.
- Guarantees minimality of horizon length.
- Slow.
The sequential strategy

1 2 3 4 5 6 7 8 9 ...

- Complete satisfiability test for $t$ before proceeding with $t + 1$.
- This is breadth-first search / iterative deepening.
- Guarantees minimality of horizon length.
- Slow.
The sequential strategy

1 2 3 4 5 6 7 8 9 ...

- Complete satisfiability test for $t$ before proceeding with $t + 1$.
- This is breadth-first search / iterative deepening.
- Guarantees minimality of horizon length.
- Slow.
The sequential strategy

1 2 3 4 5 6 7 8 9 ...

- Complete satisfiability test for $t$ before proceeding with $t + 1$.
- This is breadth-first search / iterative deepening.
- Guarantees minimality of horizon length.
- Slow.
Some runtime profiles
Some runtime profiles

Evaluation times: gripper10

- Planning with SAT
- Introduction
- Encodings
- Solver Calls
  - Sequential Strategy
  - Parallel Strategy A
  - Parallel Strategy B
  - Parallel Strategy C
- Summary
- SAT solving
- Invariants
- Conclusion
- References
Some runtime profiles

Evaluation times: satell20

- Time in secs
- Time points
Some runtime profiles

Evaluation times: schedule51

<table>
<thead>
<tr>
<th>time in secs</th>
<th>0</th>
<th>50</th>
<th>100</th>
<th>150</th>
<th>200</th>
<th>250</th>
<th>300</th>
<th>350</th>
<th>400</th>
</tr>
</thead>
<tbody>
<tr>
<td>time points</td>
<td>0</td>
<td>5</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>25</td>
<td>30</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diagram showing some runtime profiles with time in seconds on the y-axis and time points on the x-axis.
Some runtime profiles
Some runtime profiles

Evaluation times: depot15
n processes/threads
Algorithm A [Rin04b, Zar04]

- Generalization of the previous: $n$ simultaneous SAT processes; when process $t$ finishes, start process $t + n$.
- Gets past hard UNSAT formulas if $n$ high enough.
- Worst case: $n$ times slower than the sequential strategy.
- Higher memory requirements.
- Skipping lengths is OK: 10 20 30 40 50 60 70 80 90 100 ... 
- We have successfully used $n = 20$. 

Generalization of the previous: \( n \) simultaneous SAT processes; when process \( t \) finishes, start process \( t + n \).

- Gets past hard UNSAT formulas if \( n \) high enough.
- Worst case: \( n \) times slower than the sequential strategy.
- Higher memory requirements.
- Skipping lengths is OK: 10 20 30 40 50 60 70 80 90 100 ...
- We have successfully used \( n = 20 \).
Generalization of the previous: \( n \) simultaneous SAT processes; when process \( t \) finishes, start process \( t + n \).

- Gets past hard UNSAT formulas if \( n \) high enough.
- Worst case: \( n \) times slower than the sequential strategy.
- Higher memory requirements.
- Skipping lengths is OK: 10 20 30 40 50 60 70 80 90 100 ... 
- We have successfully used \( n = 20 \).
Algorithm A [Rin04b, Zar04]

$\text{Generalization of the previous: } n \text{ simultaneous SAT processes; when process } t \text{ finishes, start process } t + n.$

$\text{Gets past hard UNSAT formulas if } n \text{ high enough.}$

$\text{Worst case: } n \text{ times slower than the sequential strategy.}$

$\text{Higher memory requirements.}$

$\text{Skipping lengths is OK: } 10 \ 20 \ 30 \ 40 \ 50 \ 60 \ 70 \ 80 \ 90 \ 100 \ \ldots$

$\text{We have successfully used } n = 20.$
$n$ processes/threads
Algorithm A [Rin04b, Zar04]

- Generalization of the previous: $n$ simultaneous SAT processes; when process $t$ finishes, start process $t + n$.
- Gets past hard UNSAT formulas if $n$ high enough.
- Worst case: $n$ times slower than the sequential strategy.
- Higher memory requirements.
- Skipping lengths is OK: 10 20 30 40 50 60 70 80 90 100 ... 
- We have successfully used $n = 20$. 
$n$ processes/threads

Algorithm A [Rin04b, Zar04]

Generalization of the previous: $n$ simultaneous SAT processes; when process $t$ finishes, start process $t + n$.

- Gets past hard UNSAT formulas if $n$ high enough.
- Worst case: $n$ times slower than the sequential strategy.
- Higher memory requirements.
- Skipping lengths is OK: 10 20 30 40 50 60 70 80 90 100 ...
- We have successfully used $n = 20$. 
$n$ processes/threads
Algorithm A [Rin04b, Zar04]

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 ...

- Generalization of the previous: $n$ simultaneous SAT processes; when process $t$ finishes, start process $t + n$.
- Gets past hard UNSAT formulas if $n$ high enough.
- Worst case: $n$ times slower than the sequential strategy.
- Higher memory requirements.
- Skipping lengths is OK: 10 20 30 40 50 60 70 80 90 100 ...
- We have successfully used $n = 20$. 

$\begin{array}{c}
\text{Planning with SAT} \\
\text{Introduction} \\
\text{Encodings} \\
\text{Solver Calls} \\
\text{Parallel Strategy A} \\
\text{Parallel Strategy B} \\
\text{Parallel Strategy C} \\
\text{Summary} \\
\text{SAT solving} \\
\text{Invariants} \\
\text{Conclusion} \\
\text{References}
\end{array}$
$n$ processes/threads
Algorithm A [Rin04b, Zar04]

Generalization of the previous: $n$ simultaneous SAT processes; when process $t$ finishes, start process $t + n$.

- Gets past hard UNSAT formulas if $n$ high enough.
- Worst case: $n$ times slower than the sequential strategy.
- Higher memory requirements.
- Skipping lengths is OK: 10 20 30 40 50 60 70 80 90 100 ...
- We have successfully used $n = 20$. 
Generalization of the previous: \( n \) simultaneous SAT processes; when process \( t \) finishes, start process \( t + n \).
- Gets past hard UNSAT formulas if \( n \) high enough.
- Worst case: \( n \) times slower than the sequential strategy.
- Higher memory requirements.
- Skipping lengths is OK: 10 20 30 40 50 60 70 80 90 100 ...
- We have successfully used \( n = 20 \).
Generalization of the previous: \( n \) simultaneous SAT processes; when process \( t \) finishes, start process \( t + n \).

- Gets past hard UNSAT formulas if \( n \) high enough.
- Worst case: \( n \) times slower than the sequential strategy.
- Higher memory requirements.
- Skipping lengths is OK: 10 20 30 40 50 60 70 80 90 100 ...
- We have successfully used \( n = 20 \).
\[ n \] processes/threads
Algorithm A [Rin04b, Zar04]

Generalization of the previous: \( n \) simultaneous SAT processes; when process \( t \) finishes, start process \( t + n \).

- Gets past hard UNSAT formulas if \( n \) high enough.
- Worst case: \( n \) times slower than the sequential strategy.
- Higher memory requirements.
- Skipping lengths is OK: 10 20 30 40 50 60 70 80 90 100 ...
- We have successfully used \( n = 20 \).
Generalization of the previous: $n$ simultaneous SAT processes; when process $t$ finishes, start process $t + n$.

- Gets past hard UNSAT formulas if $n$ high enough.
- Worst case: $n$ times slower than the sequential strategy.
- Higher memory requirements.
- Skipping lengths is OK: 10 20 30 40 50 60 70 80 90 100 ...
- We have successfully used $n = 20$. 

$n$ processes/threads
Algorithm A [Rin04b, Zar04]
Algorithm A [Rin04b, Zar04]

\( n \) processes/threads

- Generalization of the previous: \( n \) simultaneous SAT processes; when process \( t \) finishes, start process \( t + n \).
- Gets past hard UNSAT formulas if \( n \) high enough.
- Worst case: \( n \) times slower than the sequential strategy.
- Higher memory requirements.
- Skipping lengths is OK: 10 20 30 40 50 60 70 80 90 100 ... 
- We have successfully used \( n = 20 \).
$n$ processes/threads
Algorithm A [Rin04b, Zar04]

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 ...

- Generalization of the previous: $n$ simultaneous SAT processes; when process $t$ finishes, start process $t + n$.
- Gets past hard UNSAT formulas if $n$ high enough.
- Worst case: $n$ times slower than the sequential strategy.
- Higher memory requirements.
- Skipping lengths is OK: 10 20 30 40 50 60 70 80 90 100 ...
- We have successfully used $n = 20$. 

SAT solving at different rates

- With the previous algorithm, choosing $n$ may be tricky: sometimes big difference e.g. between $n = 10$ and $n = 11$.
- Best to have a high $n$, but focus on the first SAT instances.
- $\implies$ SAT solving at variable rates.
Finding a plan for blocks22 with Algorithm B
Finding a plan for blocks22 with Algorithm B
Finding a plan for blocks22 with Algorithm B [Rin04b]
Finding a plan for blocks22 with Algorithm B
Finding a plan for blocks22 with Algorithm B

Geometric rates
Algorithm B [Rin04b]
Finding a plan for blocks22 with Algorithm B
Finding a plan for blocks22 with Algorithm B
Finding a plan for blocks22 with Algorithm B
Exponential length increase

- Previous strategies restrictive when plans are very long (200, 500, 1000 steps or more).
- Why not exponential steps to cover very long plans?
- Works surprisingly well! (...as long as you have enough memory...)
- Dozens of previously unsolved instances solved.
- Large slow-downs uncommon (but depends on SAT heuristics being used and type of problems).

1 2 4 8 16 32 64 128 256 512 1024 2048 4096 8192
## Scheduling the SAT Tests: Summary

<table>
<thead>
<tr>
<th>algorithm</th>
<th>reference</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequential</td>
<td>[KS92, KS96]</td>
<td>slow, guarantees min. horizon</td>
</tr>
<tr>
<td>binary search</td>
<td>[SS07]</td>
<td>length upper bound needed</td>
</tr>
<tr>
<td>n processes</td>
<td>[Rin04b, Zar04]</td>
<td>fast, more memory needed</td>
</tr>
<tr>
<td>geometric</td>
<td>[Rin04b]</td>
<td>fast, more memory needed</td>
</tr>
<tr>
<td>exponential</td>
<td>Rintanen 2012</td>
<td>fast, still more memory needed</td>
</tr>
</tbody>
</table>
General-purpose SAT solvers (RSAT, Precosat, Lingeling) work very well with
- short plans (< 10) with lots of actions in parallel, and
- small but hard problems.

Other problems more challenging for general-purpose solvers.
- long plans
- lots of actions and state variables

This is so especially when compared to planners that use explicit state-space search driven by heuristics [BG01, RW10].
Planning-specific heuristics

[Rin10, Rin11, Rin12]

- How to match the performance of explicit state-space search when solving *large but “easy”* problems?
- Planning-specific heuristics for SAT solving [Rin10]
- Observation: both $I$ and $G$ are needed for unsatisfiability. (“set of support” strategies)
- Idea: fill in “gaps” in the current partial plan.
- Force SAT solver to emulate backward chaining:
  1. Start from a top-level goal literal.
  2. Go to the *latest* preceding time where the literal is *false*.
  3. Choose an action to change the literal from *false* to *true*.
  4. Use the action variable as the CDCL decision variable.
  5. If such action there already, do the same with its preconditions.
How to match the performance of explicit state-space search when solving large but “easy” problems?

Planning-specific heuristics for SAT solving [Rin10]

Observation: both \( I \) and \( G \) are needed for unsatisfiability. ("set of support" strategies)

Idea: fill in "gaps" in the current partial plan.

Force SAT solver to emulate backward chaining:

1. Start from a top-level goal literal.
2. Go to the latest preceding time where the literal is false.
3. Choose an action to change the literal from false to true.
4. Use the action variable as the CDCL decision variable.
5. If such action there already, do the same with its preconditions.
How to match the performance of explicit state-space search when solving large but “easy” problems?

Planning-specific heuristics for SAT solving [Rin10]

Observation: both $I$ and $G$ are needed for unsatisfiability. (“set of support” strategies)

Idea: fill in “gaps” in the current partial plan.

Force SAT solver to emulate backward chaining:

1. Start from a top-level goal literal.
2. Go to the latest preceding time where the literal is false.
3. Choose an action to change the literal from false to true.
4. Use the action variable as the CDCL decision variable.
5. If such action there already, do the same with its preconditions.
Planning-specific heuristics [Rin10, Rin11, Rin12]

- How to match the performance of explicit state-space search when solving large but “easy” problems?
- Planning-specific heuristics for SAT solving [Rin10]
- Observation: both \( I \) and \( G \) are needed for unsatisfiability. (“set of support” strategies)
- Idea: fill in “gaps” in the current partial plan.
- Force SAT solver to emulate backward chaining:
  1. Start from a top-level goal literal.
  2. Go to the latest preceding time where the literal is false.
  3. Choose an action to change the literal from false to true.
  4. Use the action variable as the CDCL decision variable.
  5. If such action there already, do the same with its preconditions.
## Planning-specific heuristic for CDCL

**Case 1: goal/subgoal \( x \) has no support yet**

Value of a state variable \( x \) at different time points:

<table>
<thead>
<tr>
<th></th>
<th>( t-8 )</th>
<th>( t-7 )</th>
<th>( t-6 )</th>
<th>( t-5 )</th>
<th>( t-4 )</th>
<th>( t-3 )</th>
<th>( t-2 )</th>
<th>( t-1 )</th>
<th>( t )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x  )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>action 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>action 2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>action 3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>action 4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
**Planning-specific heuristic for CDCL**

Case 1: goal/subgoal $x$ has no support yet

Value of a state variable $x$ at different time points:

<table>
<thead>
<tr>
<th></th>
<th>$t-8$</th>
<th>$t-7$</th>
<th>$t-6$</th>
<th>$t-5$</th>
<th>$t-4$</th>
<th>$t-3$</th>
<th>$t-2$</th>
<th>$t-1$</th>
<th>$t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>action 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>action 2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>action 3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>action 4</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Actions that can make $x$ true.
Planning-specific heuristic for CDCL
Case 1: goal/subgoal $x$ has no support yet

Value of a state variable $x$ at different time points:

<table>
<thead>
<tr>
<th></th>
<th>$t - 8$</th>
<th>$t - 7$</th>
<th>$t - 6$</th>
<th>$t - 5$</th>
<th>$t - 4$</th>
<th>$t - 3$</th>
<th>$t - 2$</th>
<th>$t - 1$</th>
<th>$t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>action 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>action 2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>action 3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>action 4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Actions that can make $x$ true at $t - 5$. 
Planning-specific heuristic for CDCL
Case 1: goal/subgoal $x$ has no support yet

Value of a state variable $x$ at different time points:

<table>
<thead>
<tr>
<th></th>
<th>$t-8$</th>
<th>$t-7$</th>
<th>$t-6$</th>
<th>$t-5$</th>
<th>$t-4$</th>
<th>$t-3$</th>
<th>$t-2$</th>
<th>$t-1$</th>
<th>$t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>action 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>action 2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>action 3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>action 4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Choose action 2 or 4 at $t-6$ as the next CDCL decision variable.
Goal/subgoal is already made true at $t - 4$ by action 4 at $t - 5$.

<table>
<thead>
<tr>
<th></th>
<th>$t - 8$</th>
<th>$t - 7$</th>
<th>$t - 6$</th>
<th>$t - 5$</th>
<th>$t - 4$</th>
<th>$t - 3$</th>
<th>$t - 2$</th>
<th>$t - 1$</th>
<th>$t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>action 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>action 2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>action 3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>action 4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Use precondition literals of action 4 as new subgoals at $t - 5$. 
Goal/subgoal is already made true at $t - 4$ by action 4 at $t - 5$.

<table>
<thead>
<tr>
<th></th>
<th>$t - 8$</th>
<th>$t - 7$</th>
<th>$t - 6$</th>
<th>$t - 5$</th>
<th>$t - 4$</th>
<th>$t - 3$</th>
<th>$t - 2$</th>
<th>$t - 1$</th>
<th>$t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>action 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>action 2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>action 3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>action 4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Use preconditions literals of action 4 as new subgoals at $t - 5$. 
The variable selection scheme
Version 1: strict depth-first search

Diagram:

- Goal 1
  - Action 1
    - Action 4
    - Action 8
  - Action 5
  - Action 9
  - Action 10
- Goal 2
  - Action 2
  - Action 6
  - Action 3
  - Action 7
The variable selection scheme
Version 1: strict depth-first search
The variable selection scheme
Version 1: strict depth-first search

Diagram:
- Goal 1:
  - Action 1
  - Action 4
  - Action 8
- Goal 2:
  - Action 2
  - Action 5
  - Action 9
  - Action 10
  - Action 3
  - Action 6
  - Action 7
The variable selection scheme
Version 1: strict depth-first search
The variable selection scheme

Version 1: strict depth-first search
The variable selection scheme
Version 1: strict depth-first search
The variable selection scheme
Version 1: strict depth-first search

- goal1
  - action1
    - action4
    - action5
      - action8
    - action9
      - action10
  - goal2
    - action2
      - action3
      - action7
The variable selection scheme
Version 1: strict depth-first search
The variable selection scheme
Version 1: strict depth-first search
The variable selection scheme
Version 1: strict depth-first search

- goal1
  - action1
    - action4
    - action8
  - action5
    - action9

- goal2
  - action2
    - action6
    - action10
  - action3
    - action7
The variable selection scheme
Version 1: strict depth-first search

```
  goal1
    /   \
   /     \   \
action1  action2
       /       \
    /         \   
  action4  action5  action6
          /       / \
        /         /   \
action8  action9  action10
```
The variable selection scheme
Version 1: strict depth-first search
The variable selection scheme
Version 2: undirectional action selection, with VSIDS-style weights
The variable selection scheme
Version 2: undirectional action selection, with VSIDS-style weights

```
<table>
<thead>
<tr>
<th>goal1</th>
</tr>
</thead>
<tbody>
<tr>
<td>action1</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>action4</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>action8</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>action5</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>action9</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>action10</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>goal2</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>action2</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>action6</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>action3</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>action7</td>
</tr>
</tbody>
</table>
```
The variable selection scheme
Version 2: undirectional action selection, with VSIDS-style weights
The variable selection scheme
Version 2: undirectional action selection, with VSIDS-style weights
The variable selection scheme
Version 2: undirectional action selection, with VSIDS-style weights

Diagram:
- Goal 1
  - Action 1
    - Action 4
    - Action 5
      - Action 8
  - Action 6
    - Action 9
      - Action 10
- Goal 2
  - Action 2
    - Action 3
      - Action 7
The variable selection scheme
Version 2: undirectional action selection, with VSIDS-style weights
The variable selection scheme
Version 2: undirectional action selection, with VSIDS-style weights
The variable selection scheme
Version 2: undirectional action selection, with VSIDS-style weights

Diagram:
- Goal 1
  - Action 1
    - Action 4
    - Action 5
      - Action 8
  - Action 6
    - Action 9
    - Action 10
- Goal 2
  - Action 2
  - Action 3
    - Action 7
The variable selection scheme
Version 2: undirectional action selection, with VSIDS-style weights
The variable selection scheme
Version 2: undirectional action selection, with VSIDS-style weights

- Goal 1
  - Action 1
    - Action 4
    - Action 5
    - Action 8
  - Action 6
    - Action 9
- Goal 2
  - Action 2
    - Action 3
    - Action 10
  - Action 7
The variable selection scheme
Version 2: undirectional action selection, with VSIDS-style weights
The variable selection scheme
Version 2: undirectional action selection, with VSIDS-style weights

![Diagram of variable selection scheme]
The variable selection scheme
Version 2: undirectional action selection, with VSIDS-style weights
Impact on planner performance

- Outperforms VSIDS with almost all benchmark problems the planning community is using.
- Worse than VSIDS with small, hard, combinatorial problems.
- Ganai [Gan10, Gan11] reports good performance of a different heuristic with partly similar flavor, for BMC.
Impact on planner performance
Planning competition problems

![Graph showing the impact on planner performance over all domains from 1998 to 2011. The x-axis represents time in seconds, ranging from 0.1 to 1000, and the y-axis represents the number of solved instances, ranging from 0 to 1600. The graph compares performance across different solvers: SATPLAN, M, Mp, MpX, LAMA08, LAMA11, FF, and FF-2.](image)
Impact on planner performance
Planning competition problems

![Graph showing the impact on planner performance with time in seconds on the x-axis and all instances on the y-axis. The graph plots time in seconds Mp against time in seconds M for different planning problems.]
Impact on planner performance

Other problems

VSIDS et al. continue to be the best heuristic for SAT-based planning e.g. with

- hard combinatorial (e.g. graph) problems [PMB11], and
- hard (and easy) random problems [Rin04a].

Research goal: combine the strengths of both types of heuristics.
**Invariants**

- **Invariants** represent dependencies between state variables.
- Dependencies arise naturally: representation of $n$-valued variables as Boolean values when $n > 2$.
- Dependencies are not always easy to detect manually.
- Dependencies can be critical for the efficiency search methods other than explicit state-space search, including SAT-based methods. (Early SAT-based planners used hand-crafted invariants, later invariants extracted from planning graphs [BF97], and now specialized algorithms.)
- Need for fast polynomial-time algorithms for finding invariants.
Inductive invariant algorithms compute a sequence of sets of formulas $C_0, C_1, C_2, \ldots$ which approximate the sequence $S_0, S_1, S_2, \ldots$ of sets of states that are reachable by taking $0, 1, 2, \ldots$ actions.

Each $C_i$ approximates from above the set $S_i$.

Level of approximation can typically be tuned by tuning the accuracy of approximate SAT tests.
Definition of Regression

Let $\phi$ be a goal (a set of literals) and $a = \langle p, e \rangle$ an action. Regression of $\phi$ w.r.t. $a = \langle p, e \rangle$ is

$$regr_a(\phi) = \{ l \in \phi | \overline{l} \notin e \} \cup p$$

This is the well-known backward chaining step: what has to be true before $a$ is taken to guarantee that $\phi$ is true afterwards.

This operation can be generalized to arbitrarily complex actions, and the operation coincides with the preimage operation defined for arbitrary transition relations in the BDD context.

Theorem

For any action $a$ and set $\phi$

$$\{ s \in S | s \models regr_a(\phi) \} = \{ s \in S | app_a(s) \models \phi \}$$

where $S$ is the set of all states.
The Algorithm
[BF97, Rin98, Rin08]

1: \textit{PROCEDURE} invariants($X, I, A, n$);
2: \hspace{0.5cm} $C := \{ x \in X \mid I \models x \} \cup \{ \neg x \mid x \in X, I \not\models x \}$;
3: \textit{REPEAT}
4: \hspace{1cm} $C' := C$;
5: \hspace{1cm} \textbf{FOR EACH} $a \in A \textbf{ AND } c \in C$ \textbf{s.t.} $C' \cup \{ \text{regr}_a(\neg c) \} \in \text{SAT}$ \textbf{DO}
6: \hspace{1.5cm} $C := C \setminus \{c\}$;
7: \hspace{1.5cm} \textbf{IF} $|\text{lits}(c)| < n$ \textbf{THEN}
8: \hspace{2cm} \textbf{BEGIN} \hspace{0.5cm} (* \textit{Add weaker clauses.} \ *)
9: \hspace{2.5cm} $C := C \cup \{ c \lor x \mid x \in X \} \cup \{ c \lor \neg x \mid x \in X \}$;
10: \hspace{2cm} \textbf{END}
11: \hspace{1cm} \textbf{END DO}
12: \hspace{1cm} \textbf{UNTIL} $C = C'$;
13: \hspace{1cm} \textit{RETURN} $C$;

(Easy to plug in regression and preimage operations for more complex definitions of actions.)
Conclusion

- Improvements in all components of SAT-based planners:
  - encodings (compact linear size, much faster)
  - solver scheduling (trade-off optimality vs. low runtimes)
  - SAT solver algorithms and implementations (CDCL, watched literals, ...)
  - SAT solver heuristics tuned for large and easy problems

- Generic SAT algorithms still a promising source of further progress.
Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu.
Symbolic model checking without BDDs.
In W. R. Cleaveland, editor, Tools and Algorithms for the Construction and Analysis of
Systems, Proceedings of 5th International Conference, TACAS’99, volume 1579 of

Avrim L. Blum and Merrick L. Furst.
Fast planning through planning graph analysis.

Blai Bonet and Héctor Geffner.
Planning as heuristic search.

Stephen A. Cook.
The complexity of theorem-proving procedures.
In Proceedings of the Third Annual ACM Symposium on Theory of Computing, pages

Yannis Dimopoulos, Bernhard Nebel, and Jana Koehler.
Encoding planning problems in nonmonotonic logic programs.
In S. Steel and R. Alami, editors, Recent Advances in AI Planning. Fourth European
Conference on Planning (ECP’97), number 1348 in Lecture Notes in Computer Science,
M. K. Ganai.
Propelling SAT and SAT-based BMC using careset.

Malay K. Ganai.
DPLL-based SAT solver using with application-aware branching, July 2011.

Alban Grastien, Anbulagan, Jussi Rintanen, and Elena Kelareva.
Diagnosis of discrete-event systems using satisfiability algorithms.

Henry Kautz and Bart Selman.
Planning as satisfiability.

Henry Kautz and Bart Selman.
Pushing the envelope: planning, propositional logic, and stochastic search.
References III

Aldo Porco, Alejandro Machado, and Blai Bonet.
Automatic polytime reductions of NP problems into a fragment of STRIPS.

Jussi Rintanen and Alban Grastien.
Diagnosability testing with satisfiability algorithms.

Jussi Rintanen, Keijo Heljanko, and Ilkka Niemelä.
Planning as satisfiability: parallel plans and algorithms for plan search.

Jussi Rintanen.
A planning algorithm not based on directional search.

Jussi Rintanen.
Complexity of planning with partial observability.
References IV

Jussi Rintanen.
Evaluation strategies for planning as satisfiability.

Jussi Rintanen.
Regression for classical and nondeterministic planning.

Jussi Rintanen.
Heuristics for planning with SAT.

Jussi Rintanen.
Planning with specialized SAT solvers.
References V

Jussi Rintanen.
Planning as satisfiability: heuristics.

Silvia Richter and Matthias Westphal.
The LAMA planner: guiding cost-based anytime planning with landmarks.

B. Selman, H. Levesque, and D. Mitchell.
A new method for solving hard satisfiability problems.

Matthew Streeter and Stephen F. Smith.
Using decision procedures efficiently for optimization.

Emmanuel Zarpas.
Simple yet efficient improvements of SAT based bounded model checking.