Solving AI Planning Problems with SAT

Jussi Rintanen

EPCL, Dresden, November 2013

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.

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.

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.

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.

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].

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

$\sim$ 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 $I : X \to \{0, 1\}$ (a valuation of X)
- goals G (a set of literals over X)

The planning problem

An action $a = \langle p, e \rangle$ is applicable in state s iff $s \models p$.
The successor state $s' = \mathit{exec}_a(s)$ 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
$\mathit{exec}_{a_n}(\mathit{exec}_{a_{n-1}}(\cdots \mathit{exec}_{a_2}(\mathit{exec}_{a_1}(I)) \cdots)) \models G$?

Encoding of Actions as Formulas
for Sequential Plans

Let $x@t$ be propositional variables for $t \in \{0, \dots, T\}$ 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 $x \in X$

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

Choice between actions $a_1, \dots, a_m$ expressed by the formula

$$\mathcal{R}@t = E_{a_1}@t \lor \cdots \lor E_{a_m}@t.$$

Reduction of Planning to SAT

Kautz and Selman, ECAI'92

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

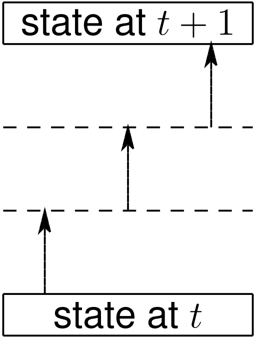
A plan of length T exists iff

$$\Phi_T = I@0 \wedge \bigwedge_{t=0}^{T-1} \mathcal{R}@t \wedge G@T$$

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 $\mathit{exec}_{a_1}(\mathit{exec}_{a_2}(s)) = \mathit{exec}_{a_2}(\mathit{exec}_{a_1}(s))$.

$\forall$-step plans: encoding

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

$$x@(t+1) \leftrightarrow ((x@t \wedge \neg a_1@t \wedge \cdots \wedge \neg a_k@t) \vee a'_1@t \vee \cdots \vee a'_{k'}@t)$$

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

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

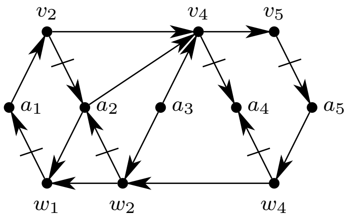
and

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

This encoding is quadratic due to the interference clauses.

$\forall$-step plans: linear encoding
Rintanen et al. 2006 [RHN06]

Action a with effect l disables all actions with precondition $\bar{l}$, 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.

$\exists$-step plans
Dimopoulos et al. 1997 [DNK97]

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.

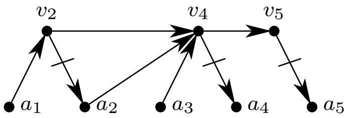
Several compact encodings exist [RHN06].
Fewer time steps are needed than with $\forall$-step plans. Sometimes only half as many.

$\exists$-step plans: linear encoding

Rintanen et al. 2006 [RHN06]

Choose an arbitrary fixed ordering of all actions $a_1, \ldots, a_n$.

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



This is needed for every literal.

Disabling graphs
Rintanen et al. 2006 [RHN06]

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.
$\Longrightarrow$ No tests for validity of orderings needed during SAT solving.

Scheduling the SAT Tests

The planning problem is reduced to SAT tests for

$$
\begin{aligned}
\Phi_0 &= I@0 \wedge G@0 \\
\Phi_1 &= I@0 \wedge \mathcal{R}@0 \wedge G@1 \\
\Phi_2 &= I@0 \wedge \mathcal{R}@0 \wedge \mathcal{R}@1 \wedge G@2 \\
\Phi_3 &= I@0 \wedge \mathcal{R}@0 \wedge \mathcal{R}@1 \wedge \mathcal{R}@2 \wedge G@3 \\
&\vdots \\
\Phi_u &= I@0 \wedge \mathcal{R}@0 \wedge \mathcal{R}@1 \wedge \cdots \mathcal{R}@(u-1) \wedge G@u
\end{aligned}
$$

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

Planning with SAT

Introduction
Encodings
Solver Calls
Sequantial Strategy
Parallel Strategy A
Parallel Strategy B
Parallel Strategy C
Summary
SAT solving
Invariants
Conclusion
References

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.

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.

The sequential strategy

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

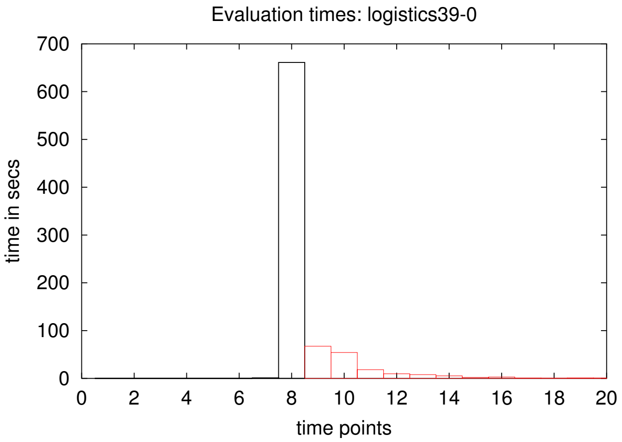
- Complete satisfiability test for t before proceeding with $t+1$.
- This is breadth-first search / iterative deepening.
- Guarantees minimality of horizon length.
- Slow.

The sequential strategy

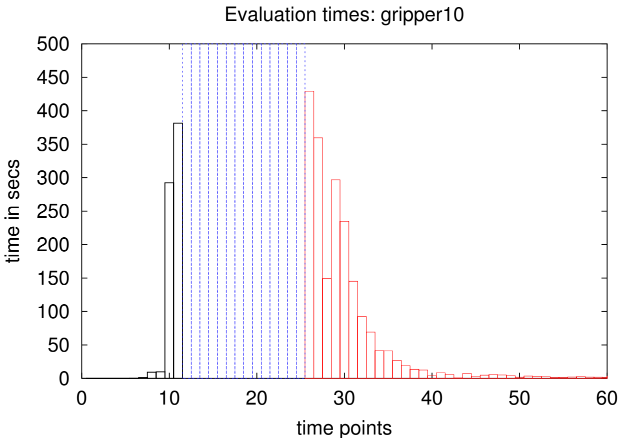
1 2 3 4 5 6 7 8 9 ...

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

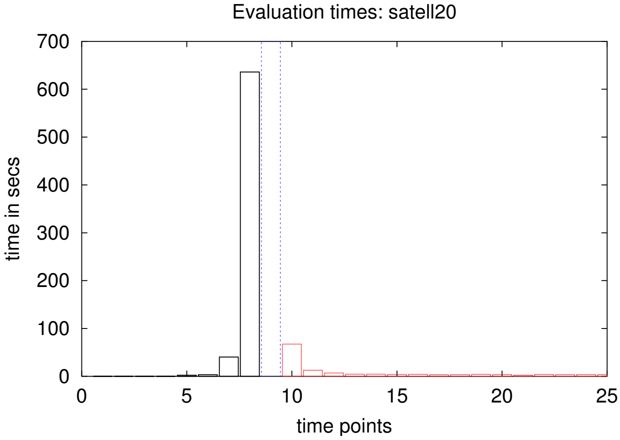
Some runtime profiles



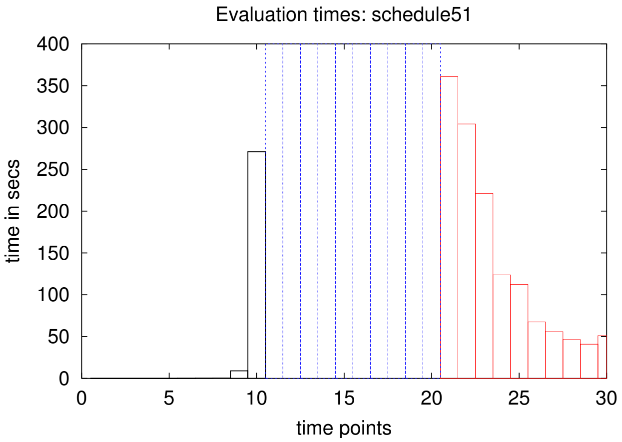
Some runtime profiles



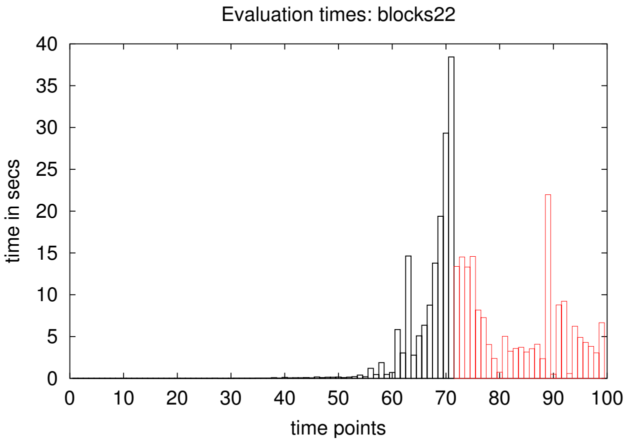
Some runtime profiles



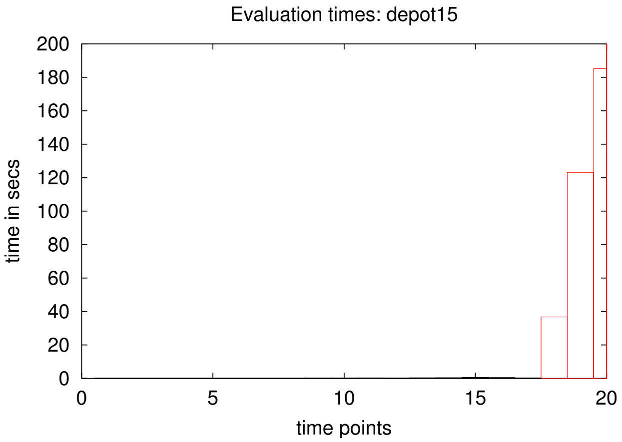
Some runtime profiles



Some runtime profiles



Some runtime profiles



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 $n = 20$.

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 $n = 20$.

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 $n = 20$.

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 $n = 20$.

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 $n = 20$.

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 $n = 20$.

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 $n = 20$.

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 $n = 20$.

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 $n = 20$.

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 $n = 20$.

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 $n = 20$.

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 $n = 20$.

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 $n = 20$.

n processes/threads
Algorithm A [Rin04b, Zar04]

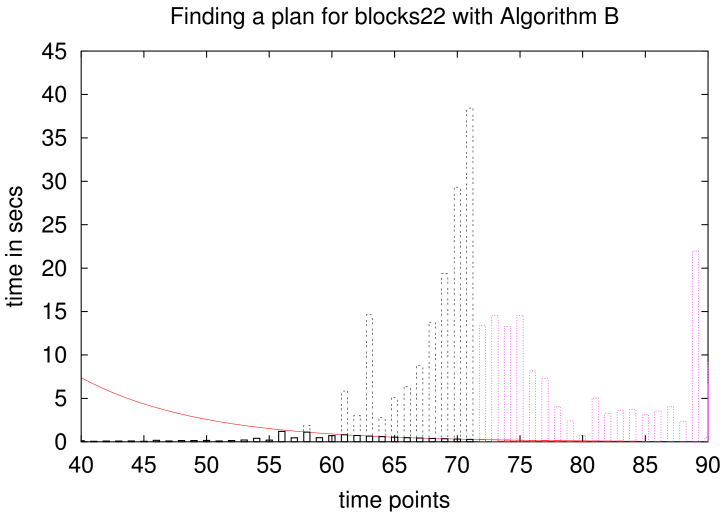
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 ...

- 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 $n = 20$.

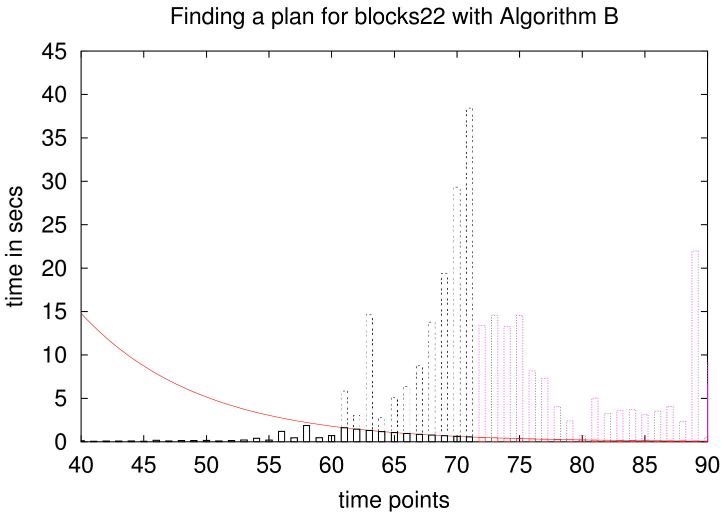
SAT solving at different rates

- 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.
- $\Longrightarrow$ SAT solving at variable rates.

Geometric rates
Algorithm B [Rin04b]

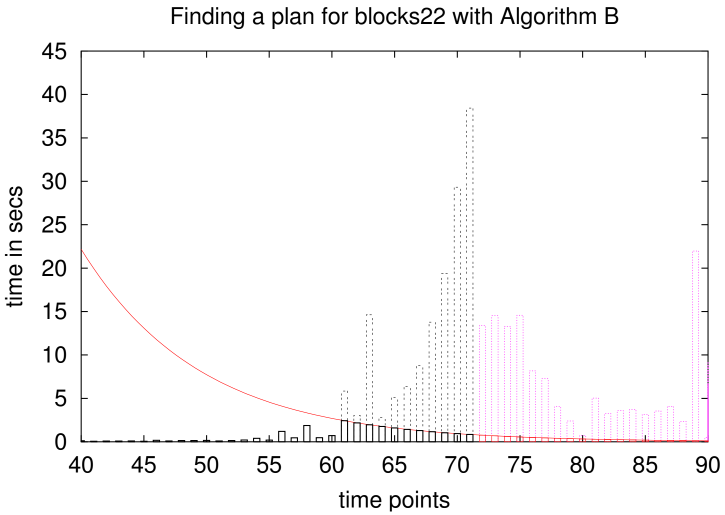


Geometric rates
Algorithm B [Rin04b]

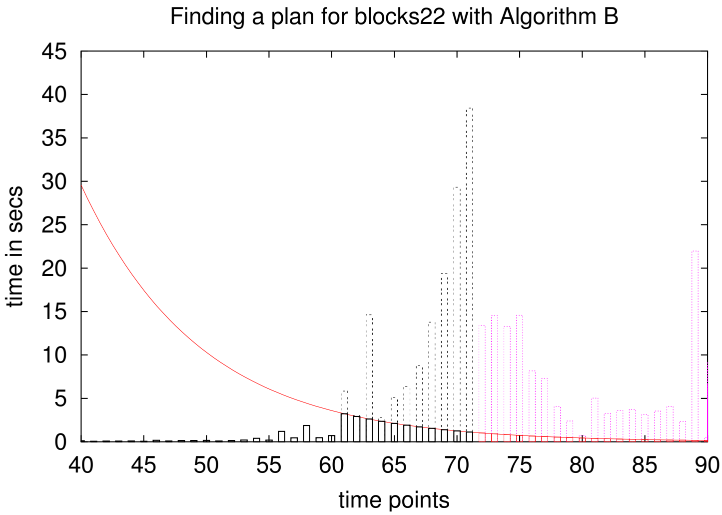


Geometric rates

Algorithm B [Rin04b]

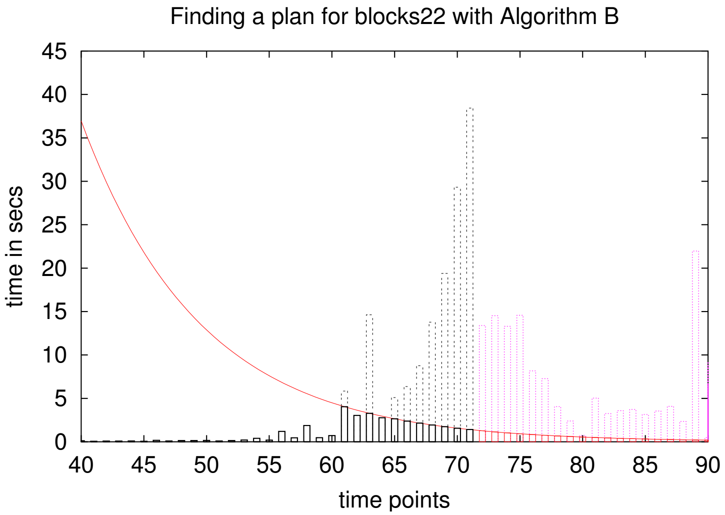


Geometric rates
Algorithm B [Rin04b]

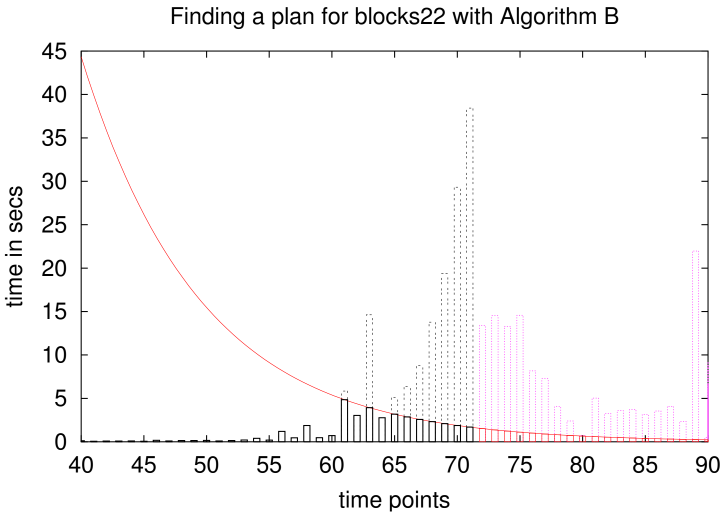


Geometric rates

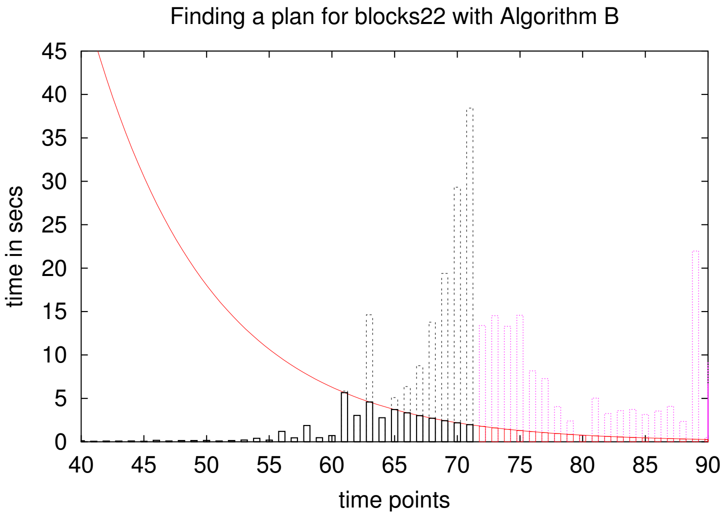
Algorithm B [Rin04b]



Geometric rates
Algorithm B [Rin04b]

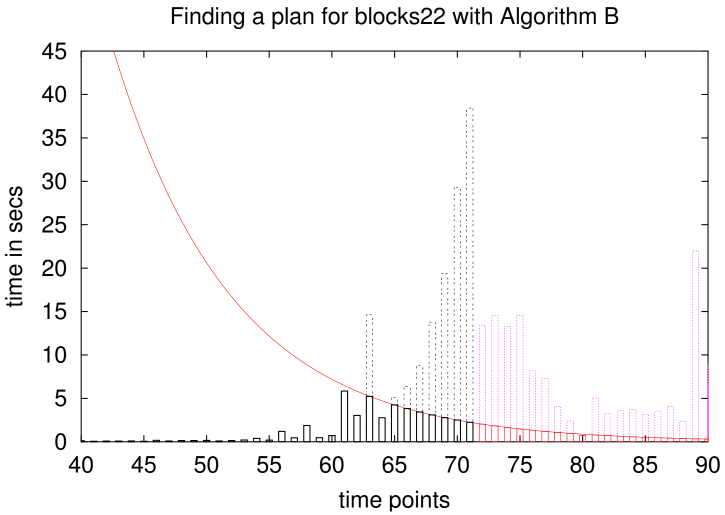


Geometric rates
Algorithm B [Rin04b]



Geometric rates

Algorithm B [Rin04b]



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

algorithm	reference	comment
sequential	[KS92, KS96]	slow, guarantees min. horizon
binary search	[SS07]	length upper bound needed
n processes	[Rin04b, Zar04]	fast, more memory needed
geometric	[Rin04b]	fast, more memory needed
exponential	Rintanen 2012	fast, still more memory needed

SAT solvers

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.

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 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 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:

	$t-8$	$t-7$	$t-6$	$t-5$	$t-4$	$t-3$	$t-2$	$t-1$	t
x	0	0	0		1		1	1	**1**
action 1	0	0	0			0	0	0	
action 2	0	0		0			0		
action 3	0	0	0	0		0	0		
action 4	0	0			0	0			

Planning-specific heuristic for CDCL

Case 1: goal/subgoal x has no support yet

Value of a state variable x at different time points:

	$t-8$	$t-7$	$t-6$	$t-5$	$t-4$	$t-3$	$t-2$	$t-1$	t
x	0	0	0		1		1	1	$\mathbf{1}$
action 1	0	0	0			0	0	0	
action 2	0	0		0			0		
action 3	0	0	0	0		0	0		
action 4	0	0			0	0			

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:

	$t-8$	$t-7$	$t-6$	$t-5$	$t-4$	$t-3$	$t-2$	$t-1$	t
x	0	0	0		1		1	1	**1**
action 1	0	0	0			0	0	0	
action 2	0	0		0			0		
action 3	0	0	0	0		0	0		
action 4	0	0			0	0			

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:

	$t-8$	$t-7$	$t-6$	$t-5$	$t-4$	$t-3$	$t-2$	$t-1$	t
x	0	0	0		1		1	1	**1**
action 1	0	0	0			0	0	0	
action 2	0	0		0			0		
action 3	0	0	0	0		0	0		
action 4	0	0			0	0			

Choose action 2 or 4 at $t-6$ as the next CDCL decision variable.

Planning-specific heuristic for CDCL

Case 2: goal/subgoal x already has support

Goal/subgoal is already made true at $t-4$ by action 4 at $t-5$.

	$t-8$	$t-7$	$t-6$	$t-5$	$t-4$	$t-3$	$t-2$	$t-1$	t
x	0	0	0		1		1	1	**1**
action 1	0	0	0			0	0	0	
action 2	0	0		0			0		
action 3	0	0	0	0		0	0		
action 4	0	0		1	0	0			

Use precondition literals of action 4 as new subgoals at $t-5$.

Planning-specific heuristic for CDCL

Case 2: goal/subgoal x already has support

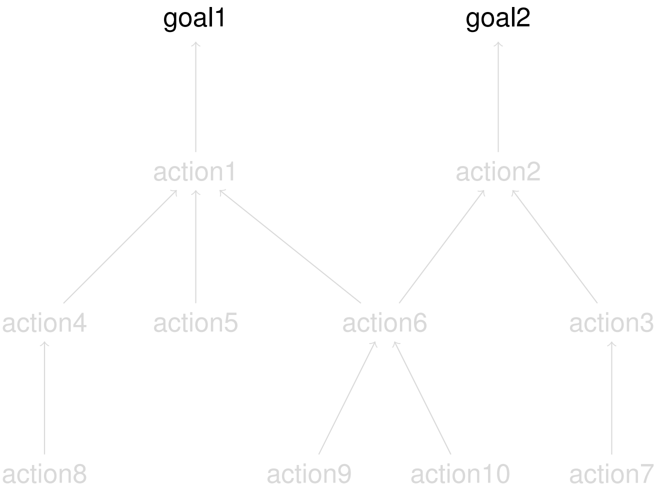
Goal/subgoal is already made true at $t-4$ by action 4 at $t-5$.

	$t-8$	$t-7$	$t-6$	$t-5$	$t-4$	$t-3$	$t-2$	$t-1$	t
x	0	0	0		1		1	1	$\mathbf{1}$
action 1	0	0	0			0	0	0	
action 2	0	0		0			0		
action 3	0	0	0	0		0	0		
action 4	0	0		1	0	0			

Use precondition literals of action 4 as new subgoals at $t-5$.

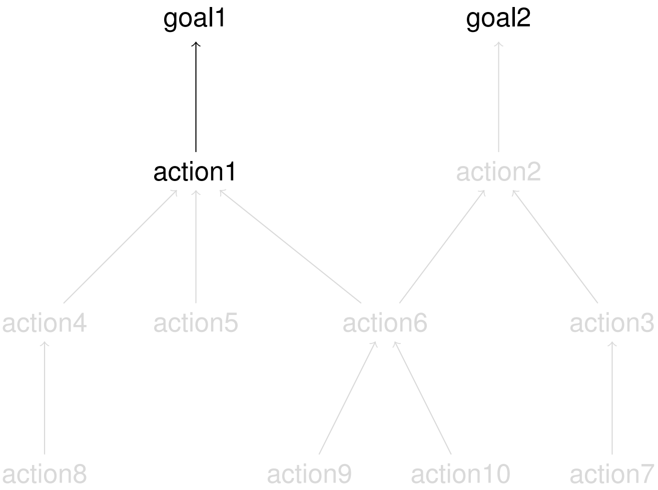
The variable selection scheme

Version 1: strict depth-first search



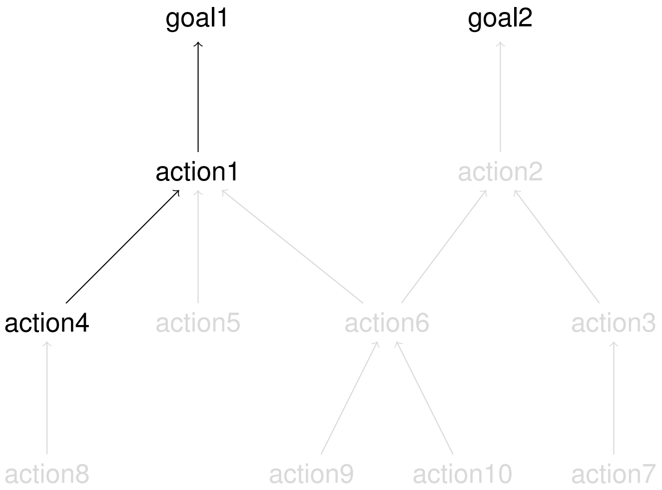
The variable selection scheme

Version 1: strict depth-first search



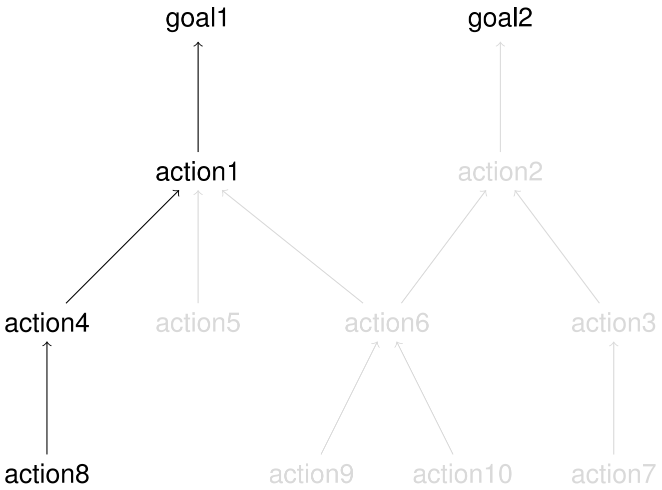
The variable selection scheme

Version 1: strict depth-first search



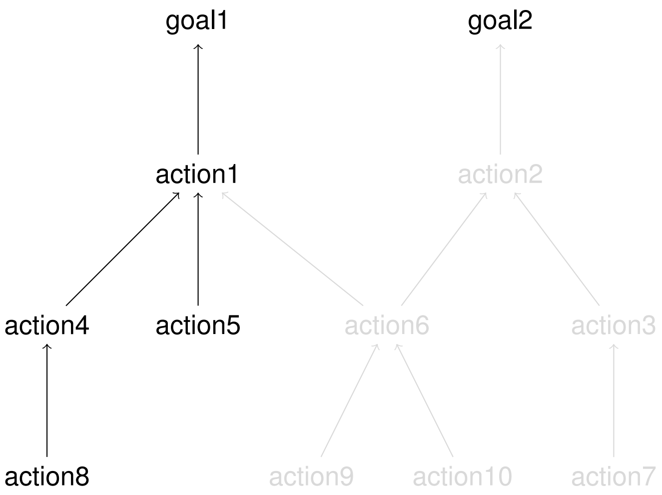
The variable selection scheme

Version 1: strict depth-first search



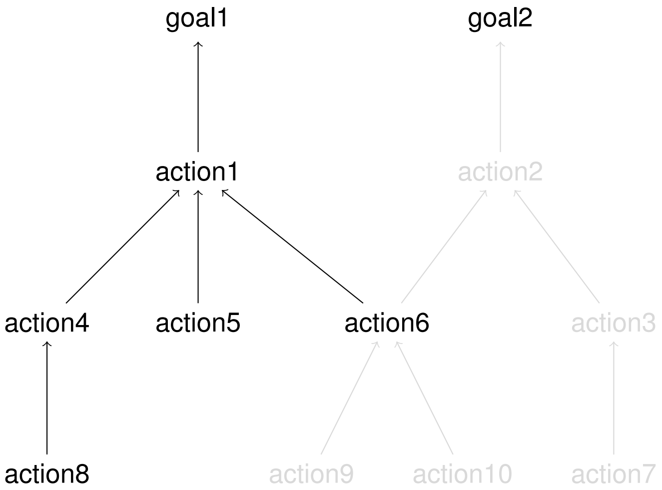
The variable selection scheme

Version 1: strict depth-first search



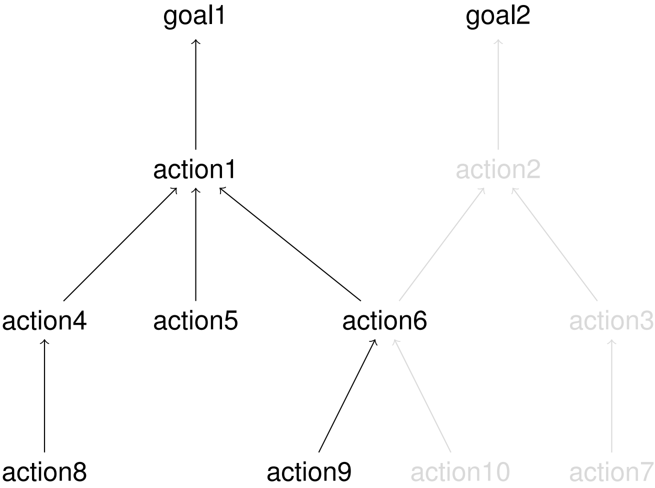
The variable selection scheme

Version 1: strict depth-first search

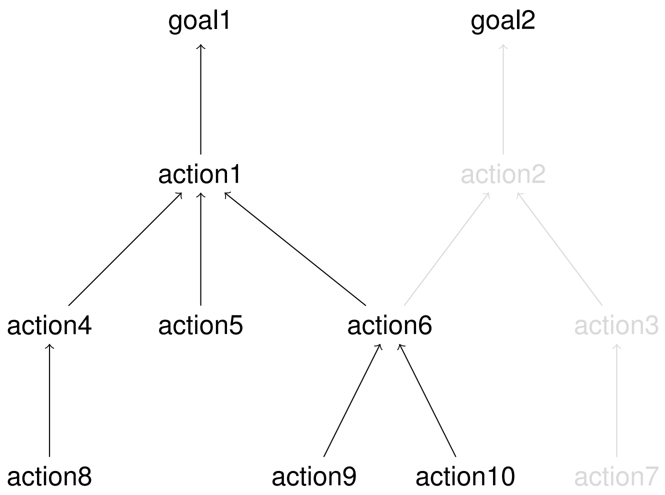


The variable selection scheme

Version 1: strict depth-first search

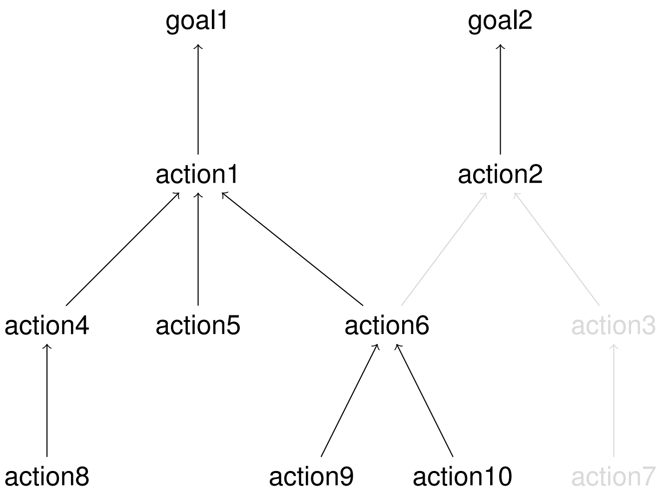


The variable selection scheme
Version 1: strict depth-first search

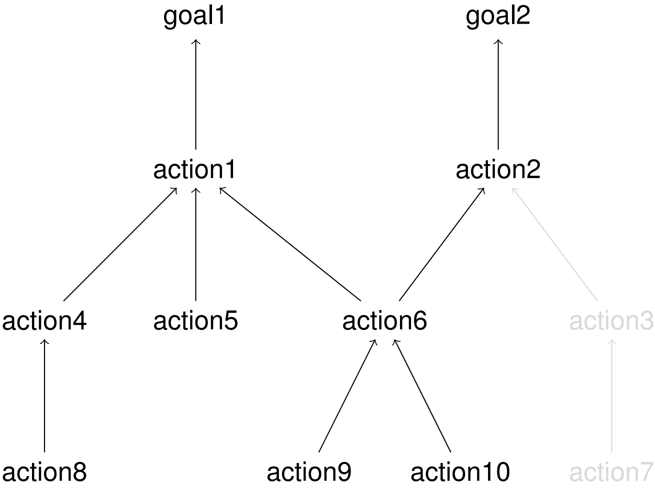


The variable selection scheme

Version 1: strict depth-first search

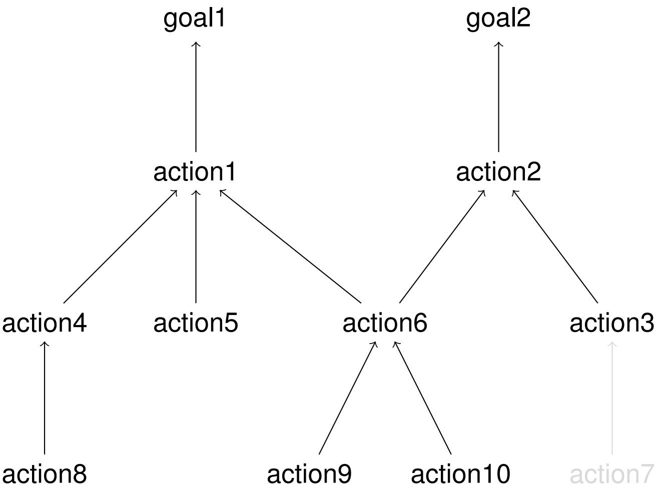


The variable selection scheme
Version 1: strict depth-first search



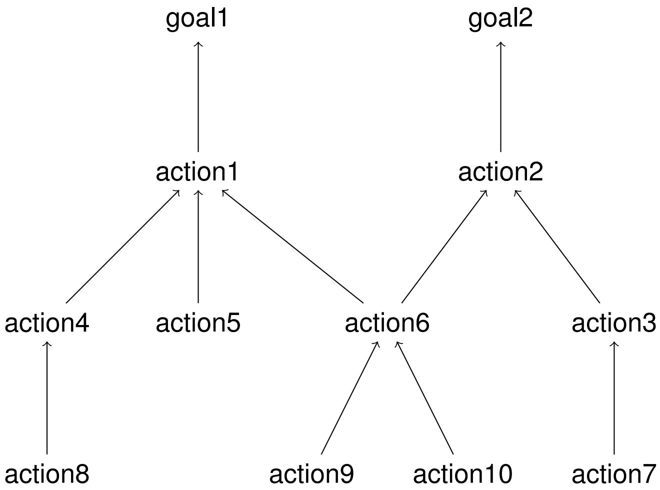
The variable selection scheme

Version 1: strict depth-first search



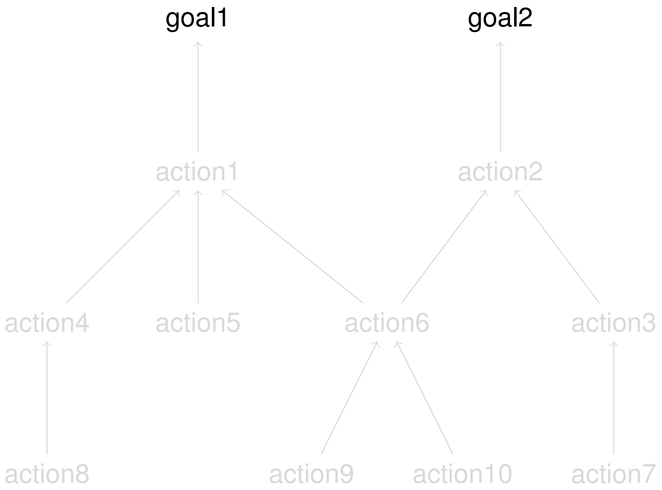
The variable selection scheme

Version 1: strict depth-first search



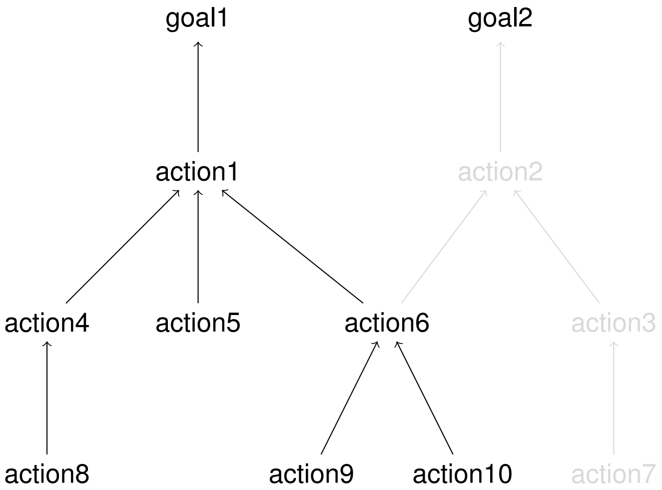
The variable selection scheme

Version 2: undirectional action selection, with VSIDS-style weights



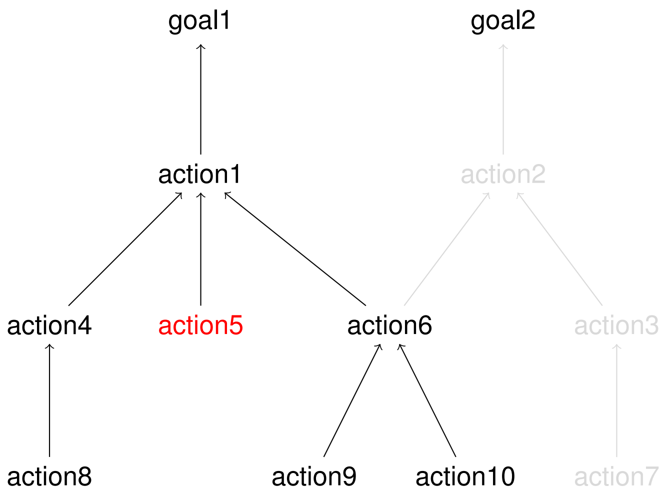
The variable selection scheme

Version 2: undirectional action selection, with VSIDS-style weights



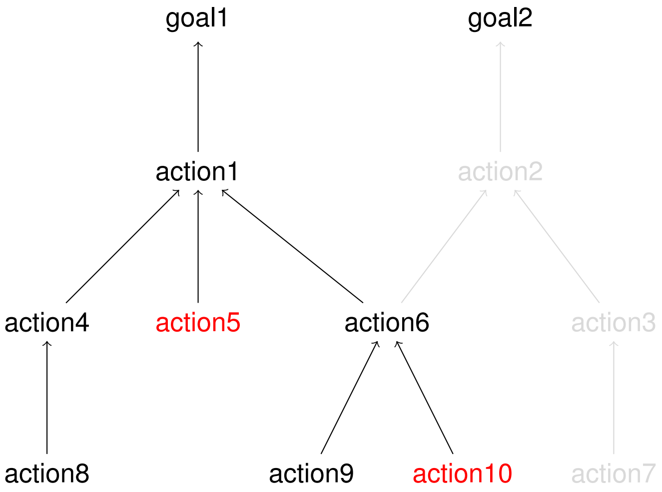
The variable selection scheme

Version 2: undirectional action selection, with VSIDS-style weights



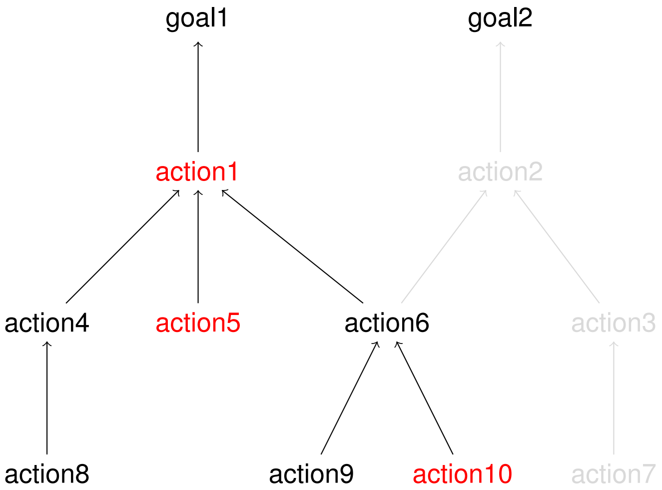
The variable selection scheme

Version 2: undirectional action selection, with VSIDS-style weights



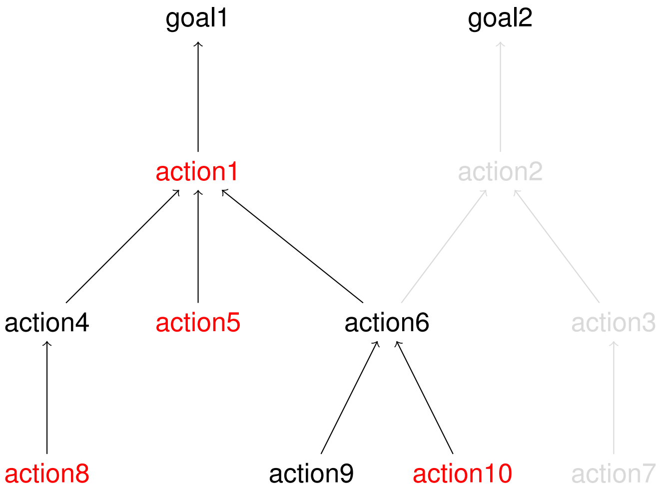
The variable selection scheme

Version 2: undirectional action selection, with VSIDS-style weights



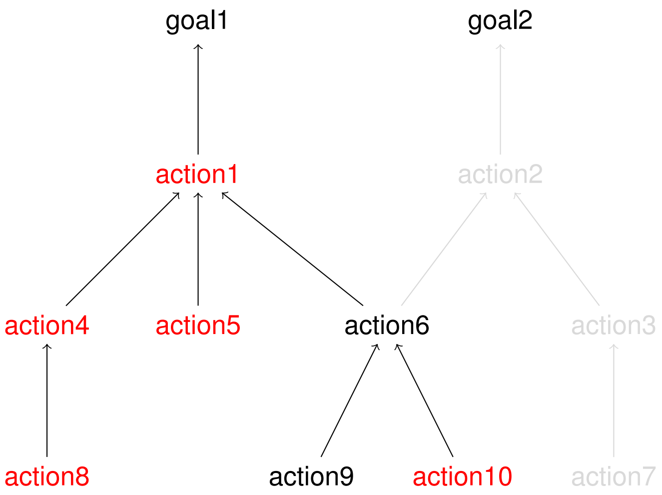
The variable selection scheme

Version 2: undirectional action selection, with VSIDS-style weights



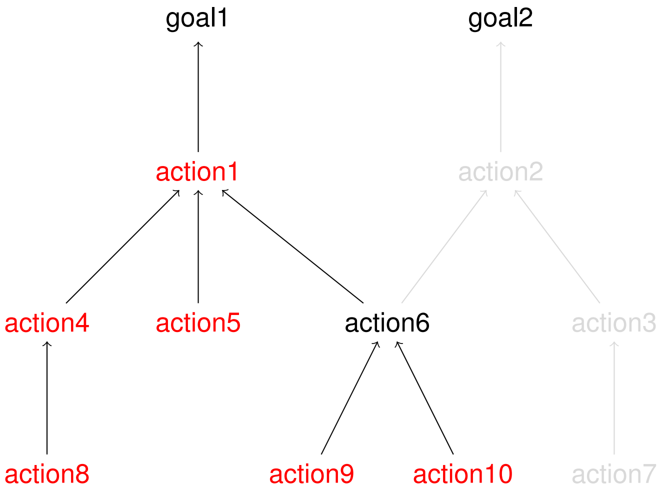
The variable selection scheme

Version 2: undirectional action selection, with VSIDS-style weights



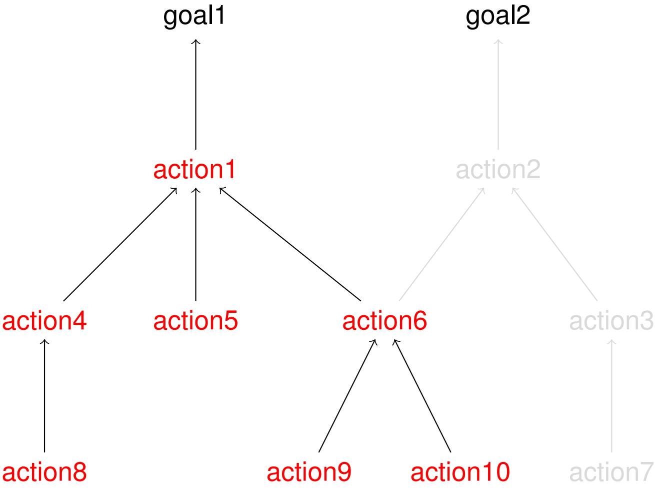
The variable selection scheme

Version 2: undirectional action selection, with VSIDS-style weights



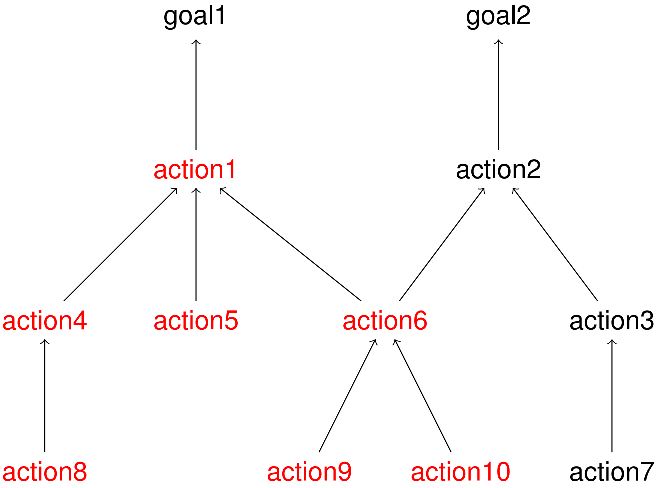
The variable selection scheme

Version 2: undirectional action selection, with VSIDS-style weights



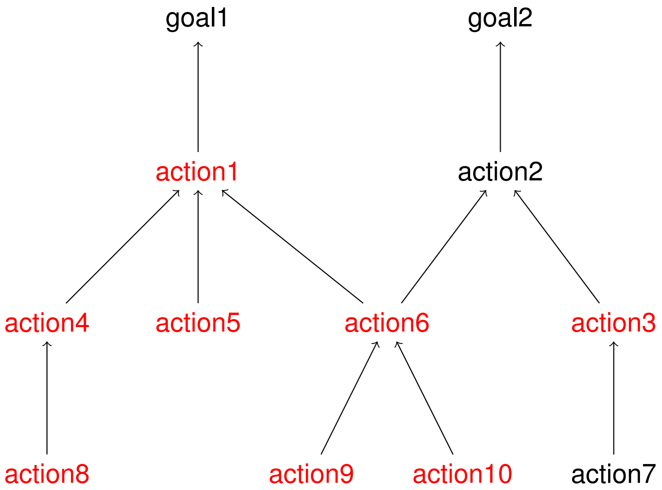
The variable selection scheme

Version 2: undirectional action selection, with VSIDS-style weights



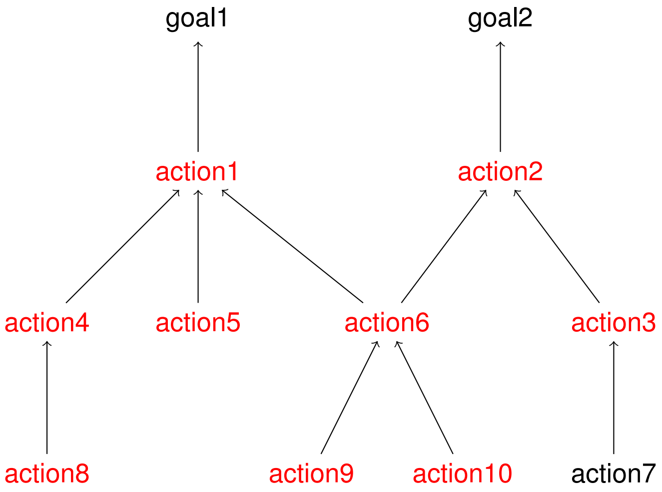
The variable selection scheme

Version 2: undirectional action selection, with VSIDS-style weights



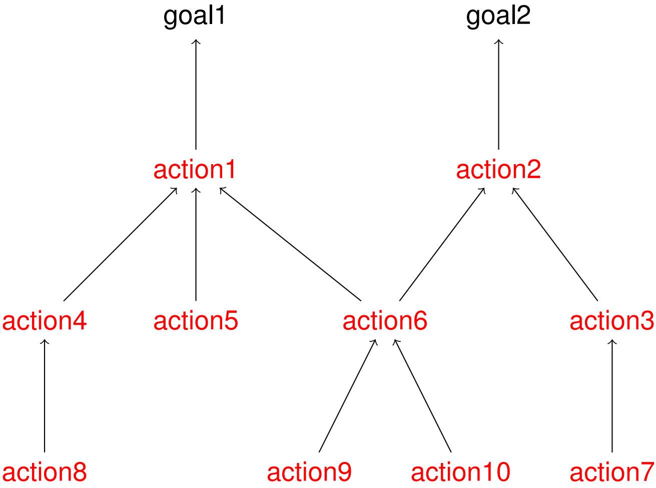
The variable selection scheme

Version 2: undirectional action selection, with VSIDS-style weights



The variable selection scheme

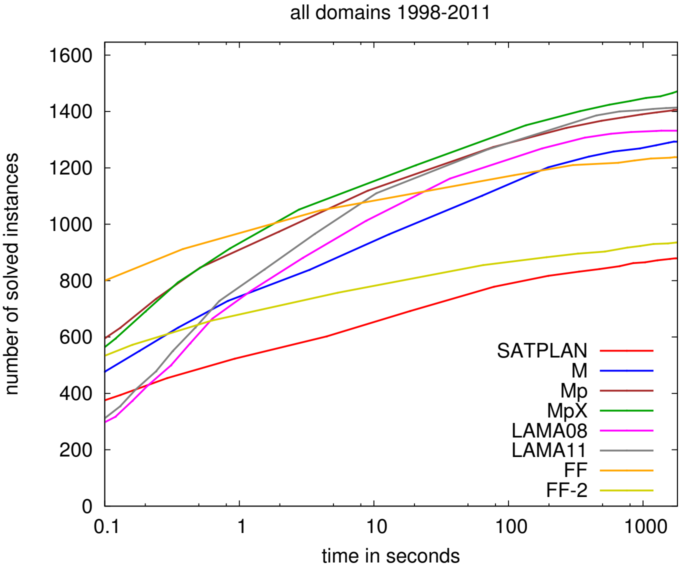
Version 2: undirectional action selection, with VSIDS-style weights



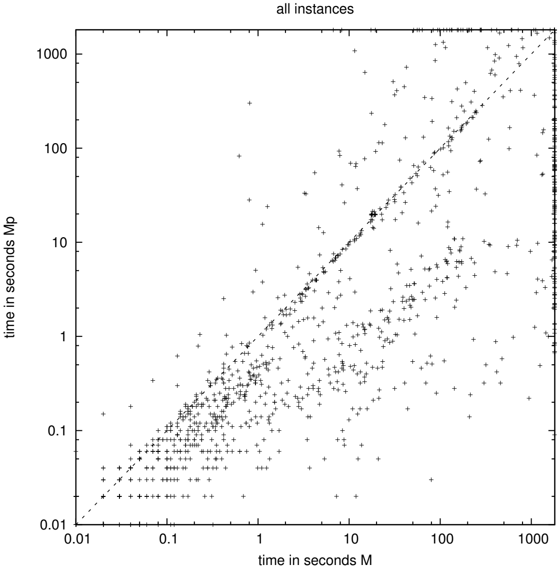
Impact on planner performance

- 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



Impact on planner performance
Planning competition problems



Impact on planner performance
Other problems

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

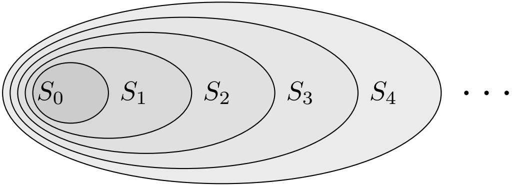
- hard combinatorial (e.g. graph) problems [PMB11], and
- hard (and easy) random problems [Rin04a].

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.

Reachability and 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

Definition

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

$$\mathit{regr}_a(\phi) = \{l \in \phi | \bar{l} \notin e\} \cup p$$

This is the well-known backward chaining step: what has to be true before a is taken to guarantee that ϕ 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 ϕ

$$\{s \in S | s \models \mathit{regr}_a(\phi)\} = \{s \in S | \mathit{app}_a(s) \models \phi\}$$

where S is the set of all states.

The Algorithm

[BF97, Rin98, Rin08]

PROCEDURE invariants(X, I, A, n);
C := $\{x \in X | I \models x\} \cup \{\neg x | x \in X, I \not\models x\}$;
REPEAT
    C' := C;
    FOR EACH $a \in A$ *AND* $c \in C$ s.t. $C' \cup \{\textit{regr}_a(\neg c)\} \in$ SAT *DO*
        C := $C \backslash \{c\}$;
        IF $|lits(c)| < n$ *THEN*
            BEGIN *(* Add weaker clauses. *)*
                C := $C \cup \{c \vee x \mid x \in X\} \cup \{c \vee \neg x \mid x \in X\}$;
            END
    END DO
UNTIL $C = C'$;
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.

References I

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 *Lecture Notes in Computer Science*, pages 193–207. Springer-Verlag, 1999.

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

Blai Bonet and Héctor Geffner.
Planning as heuristic search.
Artificial Intelligence, 129(1-2):5–33, 2001.

Stephen A. Cook.
The complexity of theorem-proving procedures.
In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971.

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, pages 169–181. Springer-Verlag, 1997.

References II

M. K. Ganai.
Propelling SAT and SAT-based BMC using careset.
In *Formal Methods in Computer-Aided Design (FMCAD), 2010*, pages 231–238. IEEE, 2010.

Malay K. Ganai.
DPLL-based SAT solver using with application-aware branching, July 2011.
patent US 2011/0184705 A1; filed August 31, 2010; provisional application January 26, 2010.

Alban Grastien, Anbulagan, Jussi Rintanen, and Elena Kelareva.
Diagnosis of discrete-event systems using satisfiability algorithms.
In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI-07)*, pages 305–310. AAAI Press, 2007.

Henry Kautz and Bart Selman.
Planning as satisfiability.
In Bernd Neumann, editor, *Proceedings of the 10th European Conference on Artificial Intelligence*, pages 359–363. John Wiley & Sons, 1992.

Henry Kautz and Bart Selman.
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*, pages 1194–1201. AAAI Press, 1996.

References III

Aldo Porco, Alejandro Machado, and Blai Bonet.
Automatic polytime reductions of NP problems into a fragment of STRIPS.
In *ICAPS 2011. Proceedings of the Twenty-First International Conference on Automated Planning and Scheduling*, pages 178–185. AAAI Press, 2011.

Jussi Rintanen and Alban Grastien.
Diagnosability testing with satisfiability algorithms.
In Manuela Veloso, editor, *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 532–537. AAAI Press, 2007.

Jussi Rintanen, Keijo Heljanko, and Ilkka Niemelä.
Planning as satisfiability: parallel plans and algorithms for plan search.
Artificial Intelligence, 170(12-13):1031–1080, 2006.

Jussi Rintanen.
A planning algorithm not based on directional search.
In A. G. Cohn, L. K. Schubert, and S. C. Shapiro, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR '98)*, pages 617–624. Morgan Kaufmann Publishers, 1998.

Jussi Rintanen.
Complexity of planning with partial observability.
In Shlomo Zilberstein, Jana Koehler, and Sven Koenig, editors, *ICAPS 2004. Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling*, pages 345–354. AAAI Press, 2004.

References IV

Jussi Rintanen.
Evaluation strategies for planning as satisfiability.
In Ramon López de Mántaras and Lorenza Saitta, editors, *ECAI 2004. Proceedings of the 16th European Conference on Artificial Intelligence*, pages 682–687. IOS Press, 2004.

Jussi Rintanen.
Regression for classical and nondeterministic planning.
In Malik Ghallab, Constantine D. Spyropoulos, and Nikos Fakotakis, editors, *ECAI 2008. Proceedings of the 18th European Conference on Artificial Intelligence*, pages 568–571. IOS Press, 2008.

Jussi Rintanen.
Heuristics for planning with SAT.
In David Cohen, editor, *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, pages 414–428. Springer-Verlag, 2010.

Jussi Rintanen.
Planning with specialized SAT solvers.
In *Proceedings of the 25th AAAI Conference on Artificial Intelligence (AAAI-11)*, pages 1563–1566. AAAI Press, 2011.

References V

Jussi Rintanen.
Planning as satisfiability: heuristics.
Artificial Intelligence, 193:45–86, 2012.

Silvia Richter and Matthias Westphal.
The LAMA planner: guiding cost-based anytime planning with landmarks.
Journal of Artificial Intelligence Research, 39:127–177, 2010.

B. Selman, H. Levesque, and D. Mitchell.
A new method for solving hard satisfiability problems.
In *Proceedings of the 11th National Conference on Artificial Intelligence*, pages 46–51, 1992.

Matthew Streeter and Stephen F. Smith.
Using decision procedures efficiently for optimization.
In *ICAPS 2007. Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling*, pages 312–319. AAAI Press, 2007.

Emmanuel Zarpas.
Simple yet efficient improvements of SAT based bounded model checking.
In Alan J. Hu and Andrew K. Martin, editors, *Formal Methods in Computer-Aided Design: 5th International Conference, FMCAD 2004, Austin, Texas, USA, November 15-17, 2004. Proceedings*, number 3312 in Lecture Notes in Computer Science, pages 174–185. Springer-Verlag, 2004.