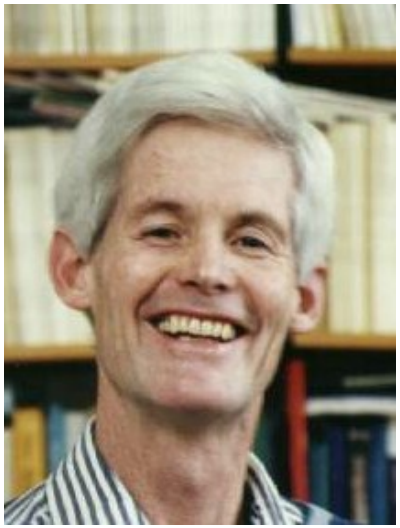
# Planning and Satisfiability

Jussi Rintanen

SAT-SMT School, Trento, June 2012

# Reduction of poly-time NDTMs to SAT

Cook [Coo71]



- proof of NP-hardness of SAT
- Test if an NP Turing machine accepts its input.
- Idea:
  - propositional variables for every tape cell, TM state and R/W head location, at every time point
  - clauses that describe how configuration can change between two consecutive time points
  - unit clauses specifying the initial configuration and final configurations

# 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.

# Same with better SAT solvers and encodings

Kautz and Selman 1996 [KS96]

- better SAT solvers: tableau (Crawford & Auton 1993), Walksat [SKC94, SKC96]
- better encodings: several actions per time point (parallel plans)
- Earlier planners (Graphplan, UCPOP) often dramatically outperformed

At this stage, stochastic local search seemed much more promising than systematic algorithms such as DPLL. IJCAI'95 had a panel *Systematic versus stochastic constraint satisfaction*, with arguments for and against systematic algorithms.

# Same with fully automated translation into SAT

Kautz and Selman 1998, 1999 [KS99]

- better SAT solvers: satz [LA97], rel_sat [BS97].
- satz enhanced with randomized restarts [GSK98]
- $n$-valued variables, e.g. location, represented as $n$ binary variables at($L_1$), at($L_2$), ..., at($L_n$).
- Invariants $\neg$at($L_1$)$\vee\neg$at($L_2$), $\neg$at($L_1$)$\vee\neg$at($L_3$), ... critical for fast SAT solving.
- Kautz and Selman translated planning graphs of Graphplan [BF97] into SAT, because they contain the relevant invariants.

# Same with fully automated translation into SAT

Kautz and Selman 1998, 1999 [KS99]

- better SAT solvers: satz [LA97], rel_sat [BS97].
- satz enhanced with randomized restarts [GSK98]
- $n$-valued variables, e.g. location, represented as $n$ binary variables at($L_1$), at($L_2$), ..., at($L_n$).
- Invariants $\neg$at($L_1$)$\vee\neg$at($L_2$), $\neg$at($L_1$)$\vee\neg$at($L_3$), ... critical for fast SAT solving.
- Kautz and Selman translated planning graphs of Graphplan [BF97] into SAT, because they contain the relevant invariants.

# Significance

- Planning the first “real” application of SAT in 1992 (along with graph-coloring and other combinatorial problems.)
- Later, same ideas applied to other reachability problems:
  - verification and validation (BMC [BCCZ99])
  - DES diagnosability testing [RG07] and diagnosis [GARK07]
- Substantial increase in interest in SAT and SAT solving in the 1992-1998 period, mostly in the AI community, before the emergence of the current generation of CDCL solvers following Chaff [MMZ+01].

# Significance

- Planning the first “real” application of SAT in 1992 (along with graph-coloring and other combinatorial problems.)
- Later, same ideas applied to other reachability problems:
  - verification and validation (BMC [BCCZ99])
  - DES diagnosability testing [RG07] and diagnosis [GARK07]
- Substantial increase in interest in SAT and SAT solving in the 1992-1998 period, mostly in the AI community, before the emergence of the current generation of CDCL solvers following Chaff [MMZ$^+$01].

# Planning
## What to do to achieve your objectives?

- Which actions to take to achieve your objectives?
- Number of agents
  - single agent, perfect information: s-t-reachability in succinct graphs
  - + nondeterminism/adversary: and-or tree search
  - + partial observability: and-or search in the space of beliefs

Time

- asynchronous or instantaneous actions (integer time, unit duration)
- rational/real time, concurrency

Objective

- Reach a goal state.
- Maximize probability of reaching a goal state.
- Maximize (expected) rewards.
- temporal goals (e.g. LTL)

# Planning
## What to do to achieve your objectives?

- Which actions to take to achieve your objectives?
- Number of agents
  - single agent, perfect information: s-t-reachability in succinct graphs
  - + nondeterminism/adversary: and-or tree search
  - + partial observability: and-or search in the space of beliefs

Time

- asynchronous or instantaneous actions (integer time, unit duration)
- rational/real time, concurrency

Objective

- Reach a goal state.
- Maximize probability of reaching a goal state.
- Maximize (expected) rewards.
- temporal goals (e.g. LTL)

# Planning
## What to do to achieve your objectives?

- Which actions to take to achieve your objectives?
- Number of agents
  - single agent, perfect information: s-t-reachability in succinct graphs
  - + nondeterminism/adversary: and-or tree search
  - + partial observability: and-or search in the space of beliefs

  Time
  - asynchronous or instantaneous actions (integer time, unit duration)
  - rational/real time, concurrency

  Objective
  - Reach a goal state.
  - Maximize probability of reaching a goal state.
  - Maximize (expected) rewards.
  - temporal goals (e.g. LTL)

# Domain-dependent vs. domain-independent planning





## Domain-independent planning

- representation in a general-purpose language (PDDL, ...)
- solution fully automatic, with a general-purpose planner

## Domain-dependent planning

- problem-specific (handcrafted) representation
- problem-specific (handcrafted) constraints
- problem-specific (handcrafted) heuristics

The latter more laborious, but a very good investment when solving a problem that has value in itself.

# 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 $t \geq 0$ 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
- $$x@(t+1) \leftrightarrow \top \text{ if } x \in e,$$
  $$x@(t+1) \leftrightarrow \bot \text{ if } \neg x \in e, \text{ and}$$
  $$x@(t+1) \leftrightarrow x@t \text{ otherwise}$$

  for all $x \in X$.

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

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

# 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**

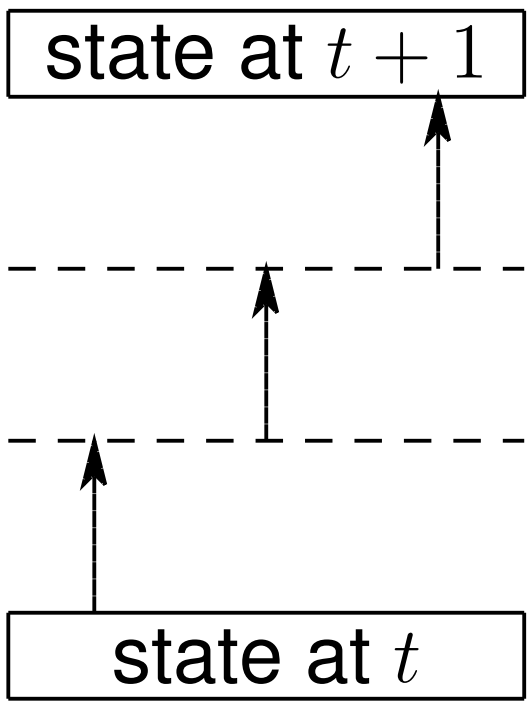
*A plan of length $T$ exists iff*

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

*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 ($\forall$-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 $exec_{a_1}(exec_{a_2}(s)) = exec_{a_2}(exec_{a_1}(s))$.*

# $\forall$-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

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

and

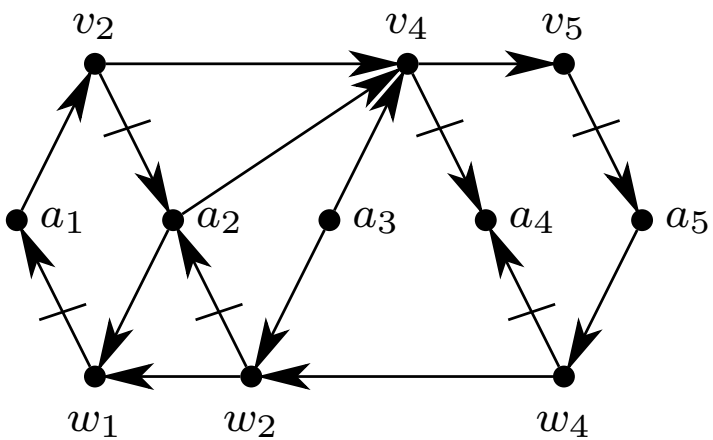
$\neg(a@t \wedge a'@t)$ for all $a$ and $a'$ that interfere.

This encoding is quadratic due to the interference clauses.

# ∀-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.

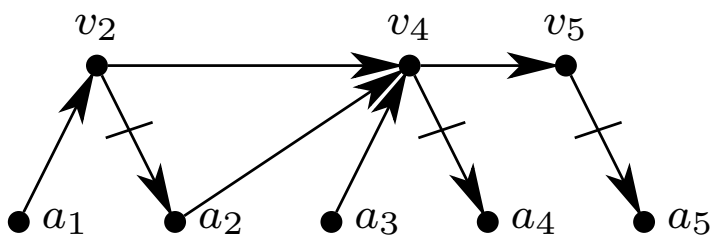
# $\exists$-step plans: linear encoding

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

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 ...

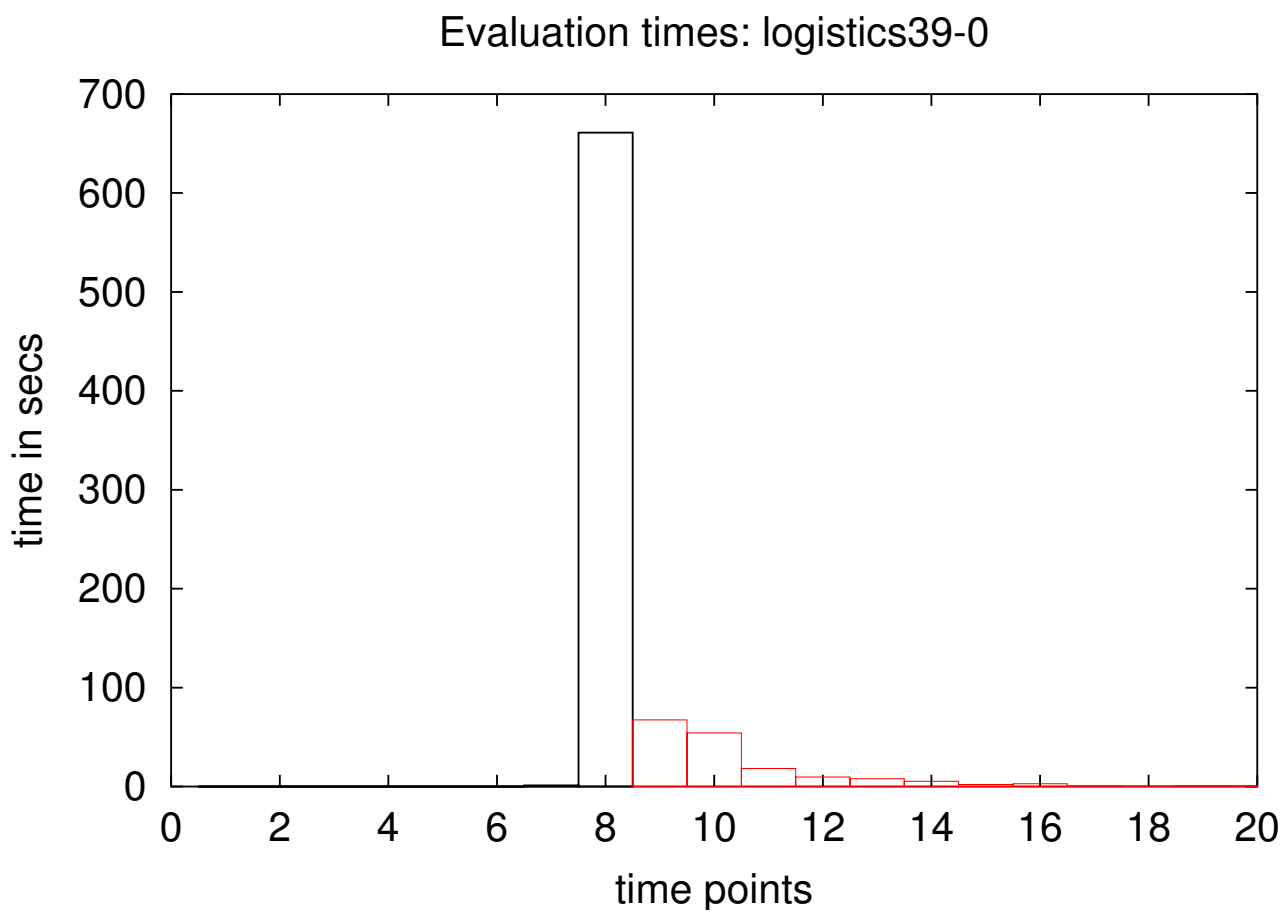
- Complete satisfiability test for $t$ before proceeding with $t + 1$.
- This is breadth-first search / iterative deepening.
- Guarantees minimality of horizon length.
- Slow.

# The sequential strategy

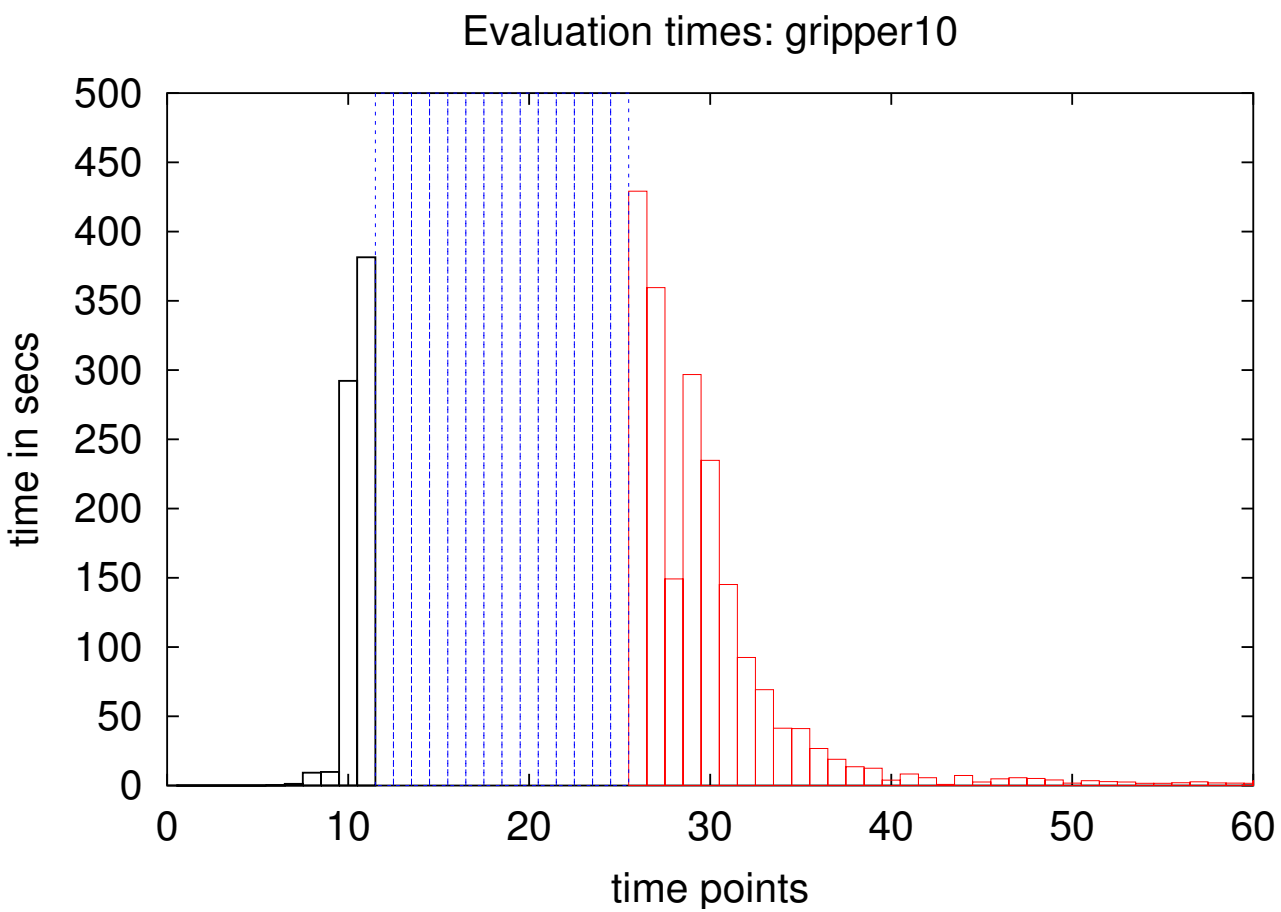
1 2 3 4 5 6 7 8 9 ...

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

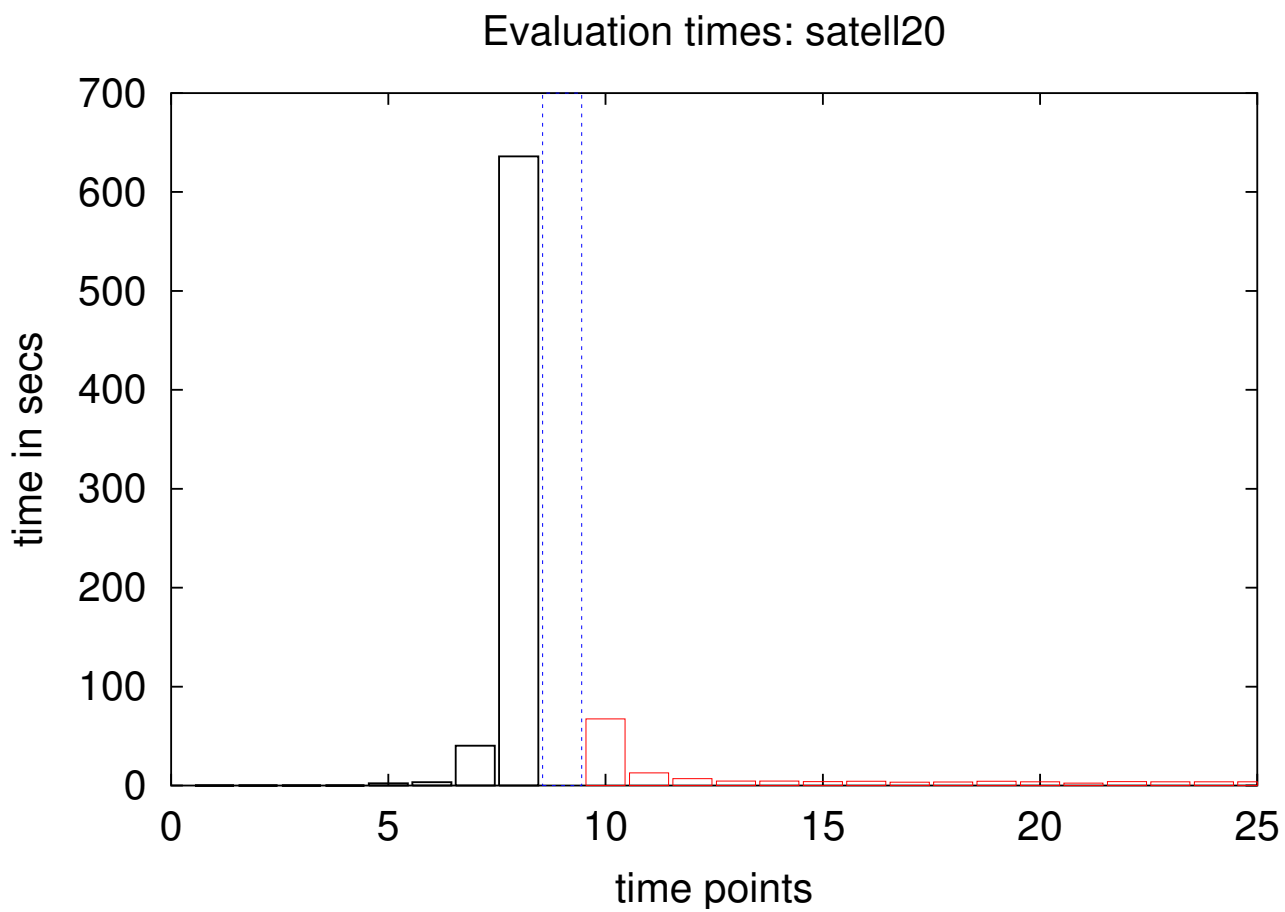
# Some runtime profiles



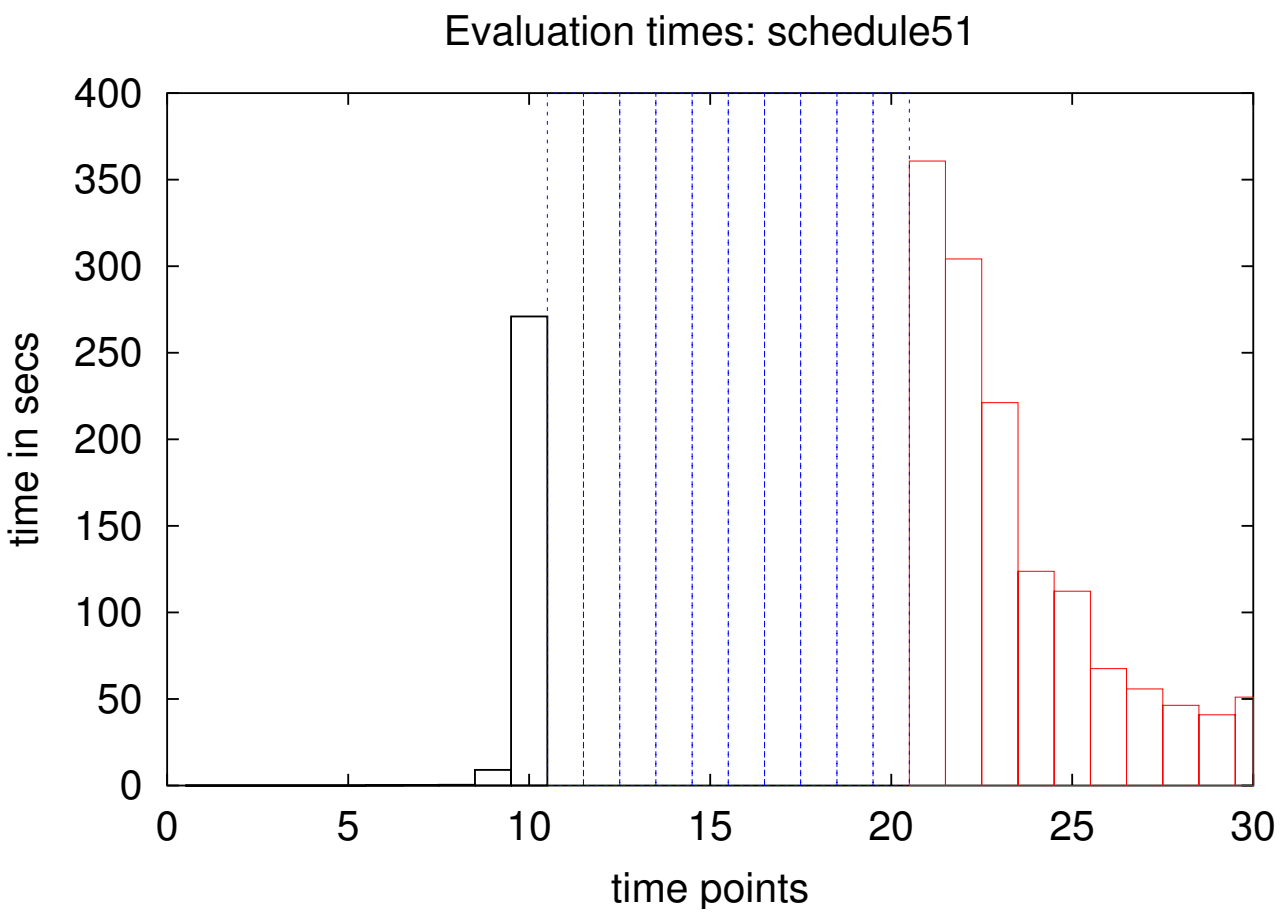
# Some runtime profiles



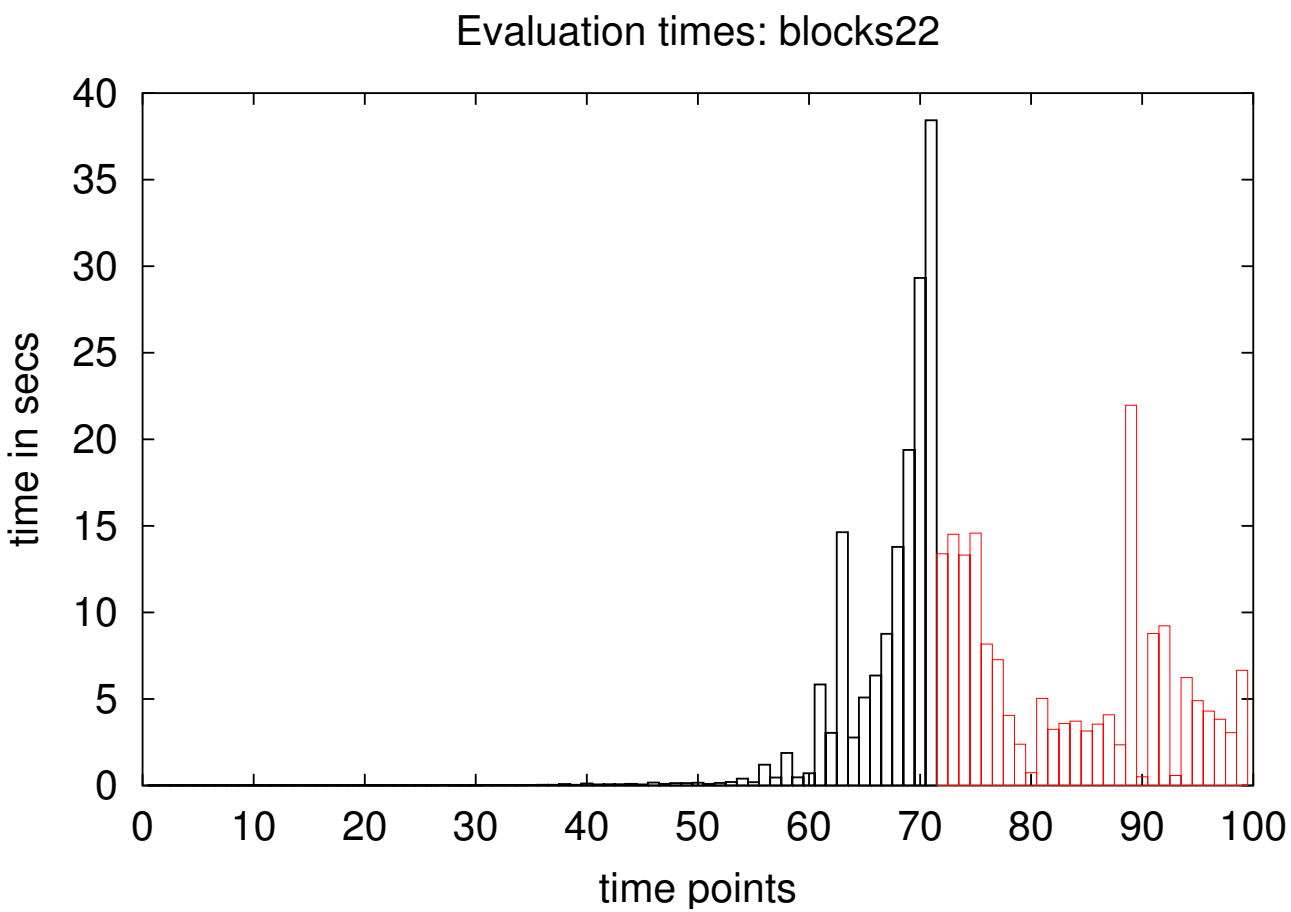
# Some runtime profiles



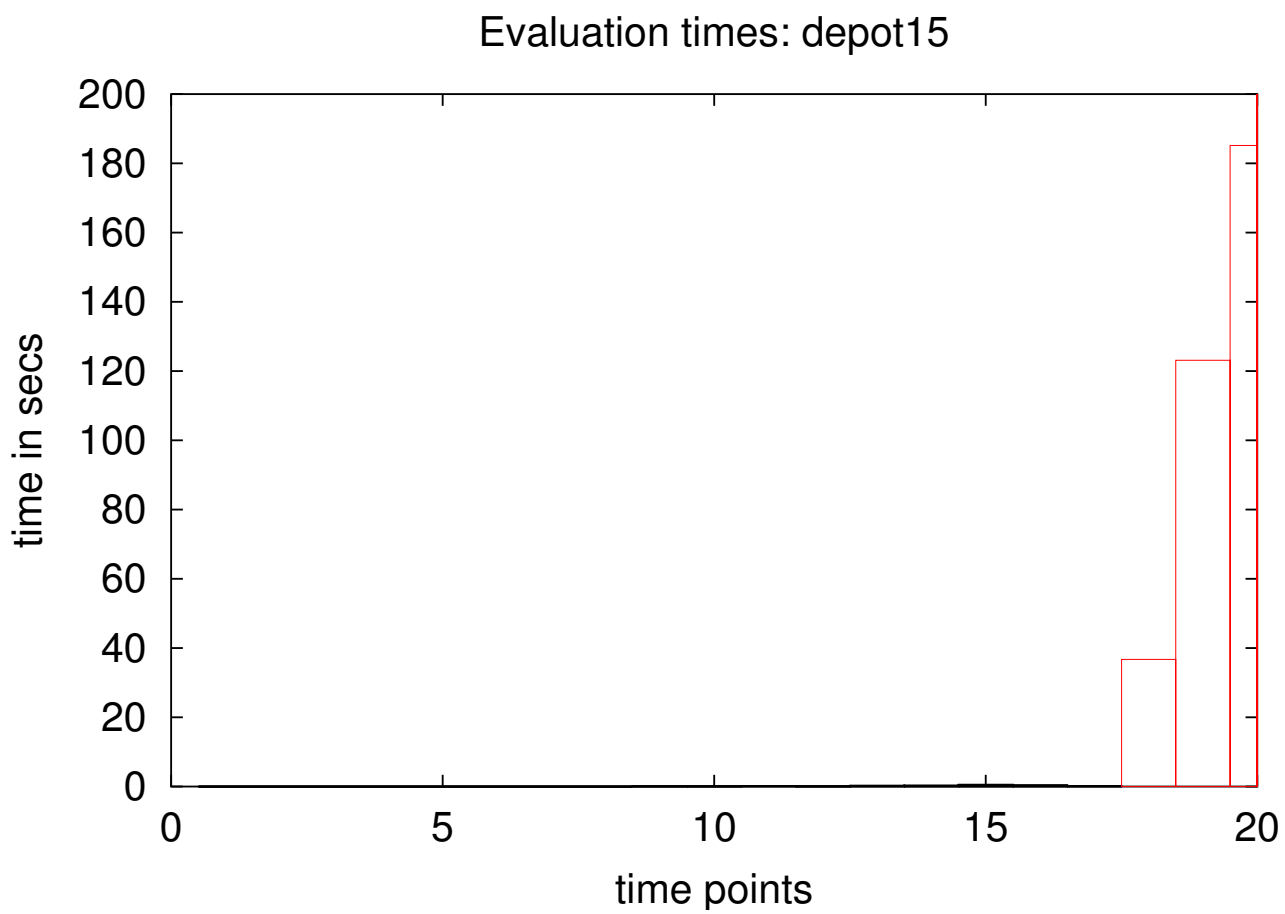
# Some runtime profiles



# 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]

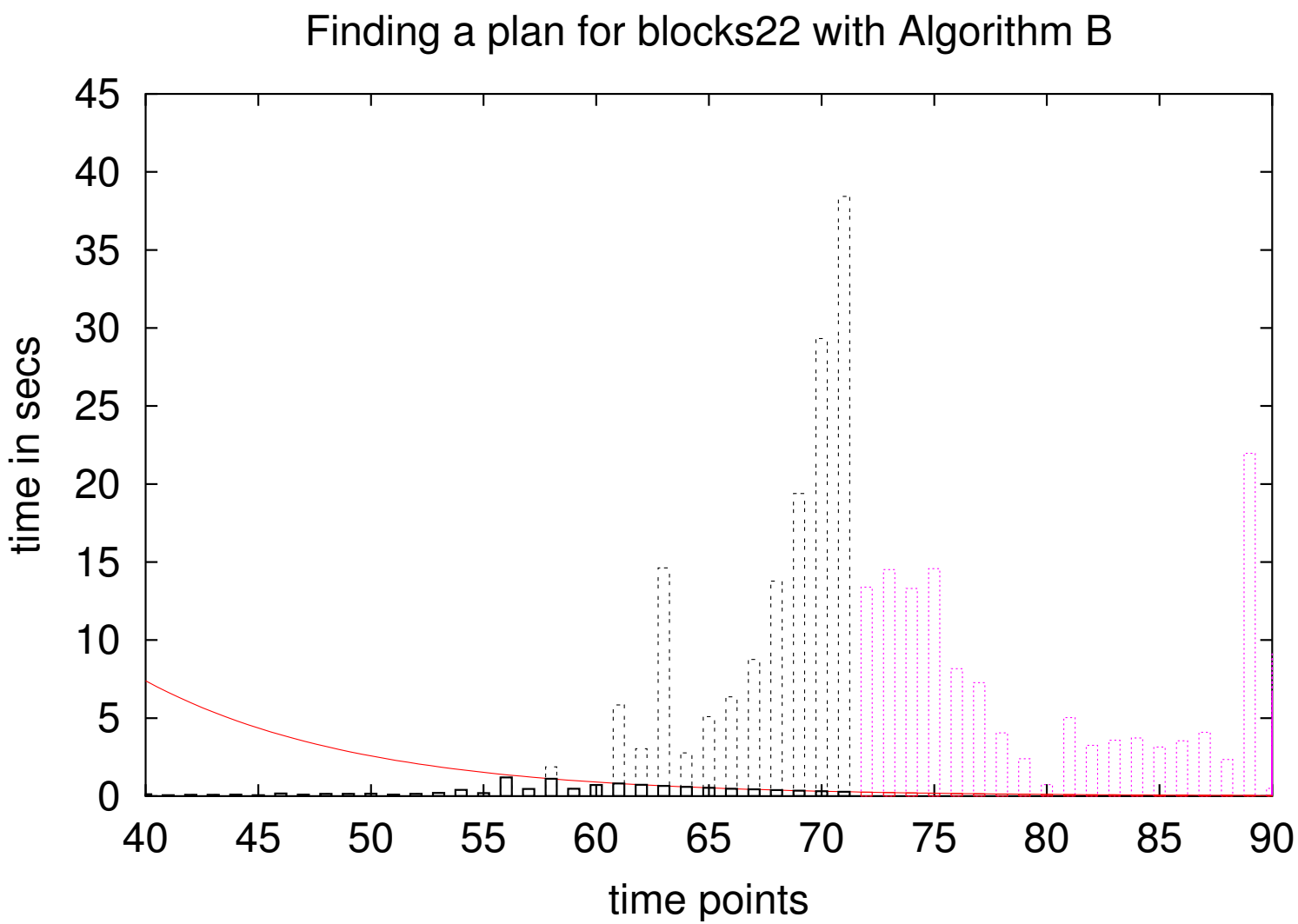
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 ...

- 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 $n = 20$.

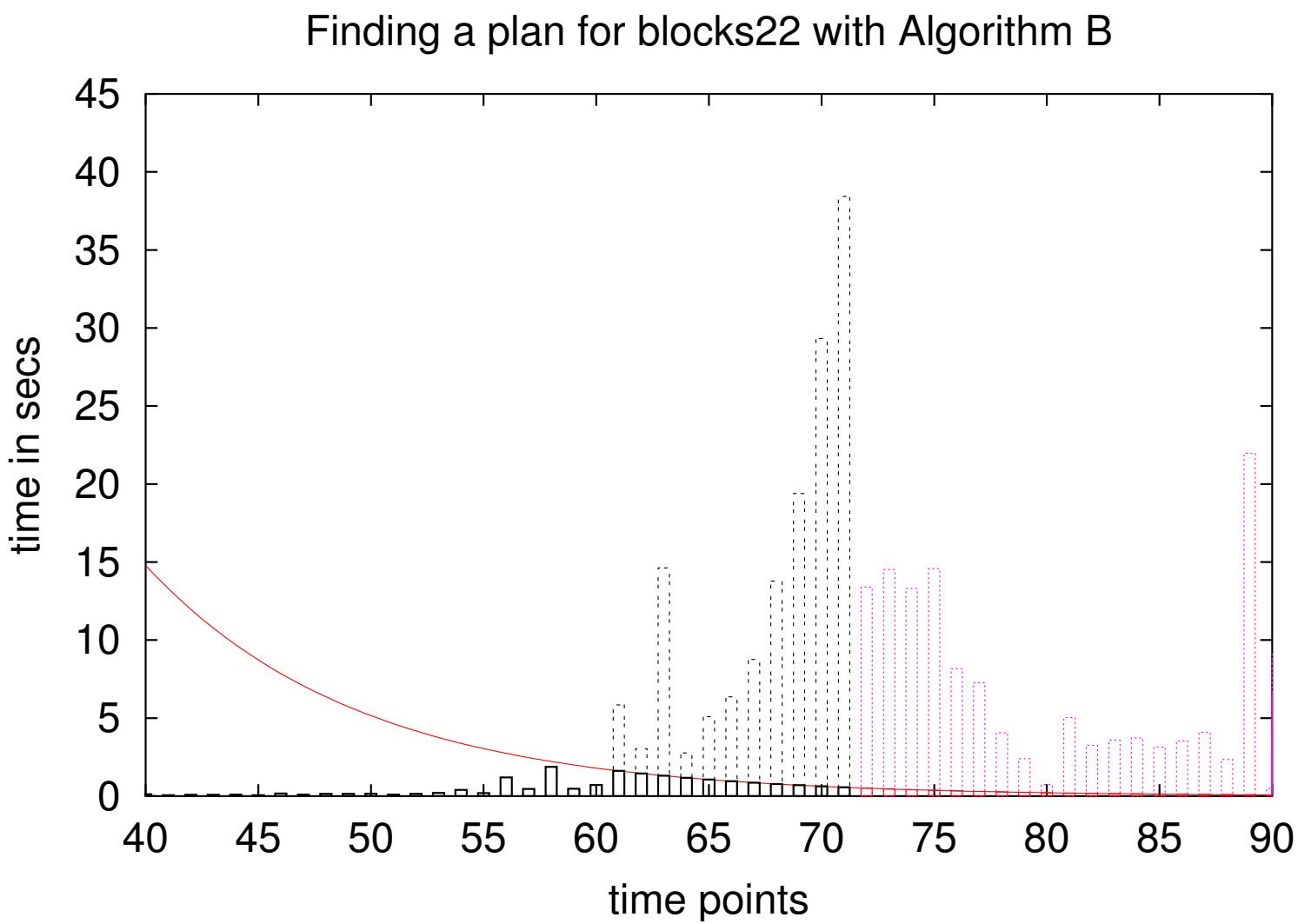
# SAT solving at different rates

- 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.

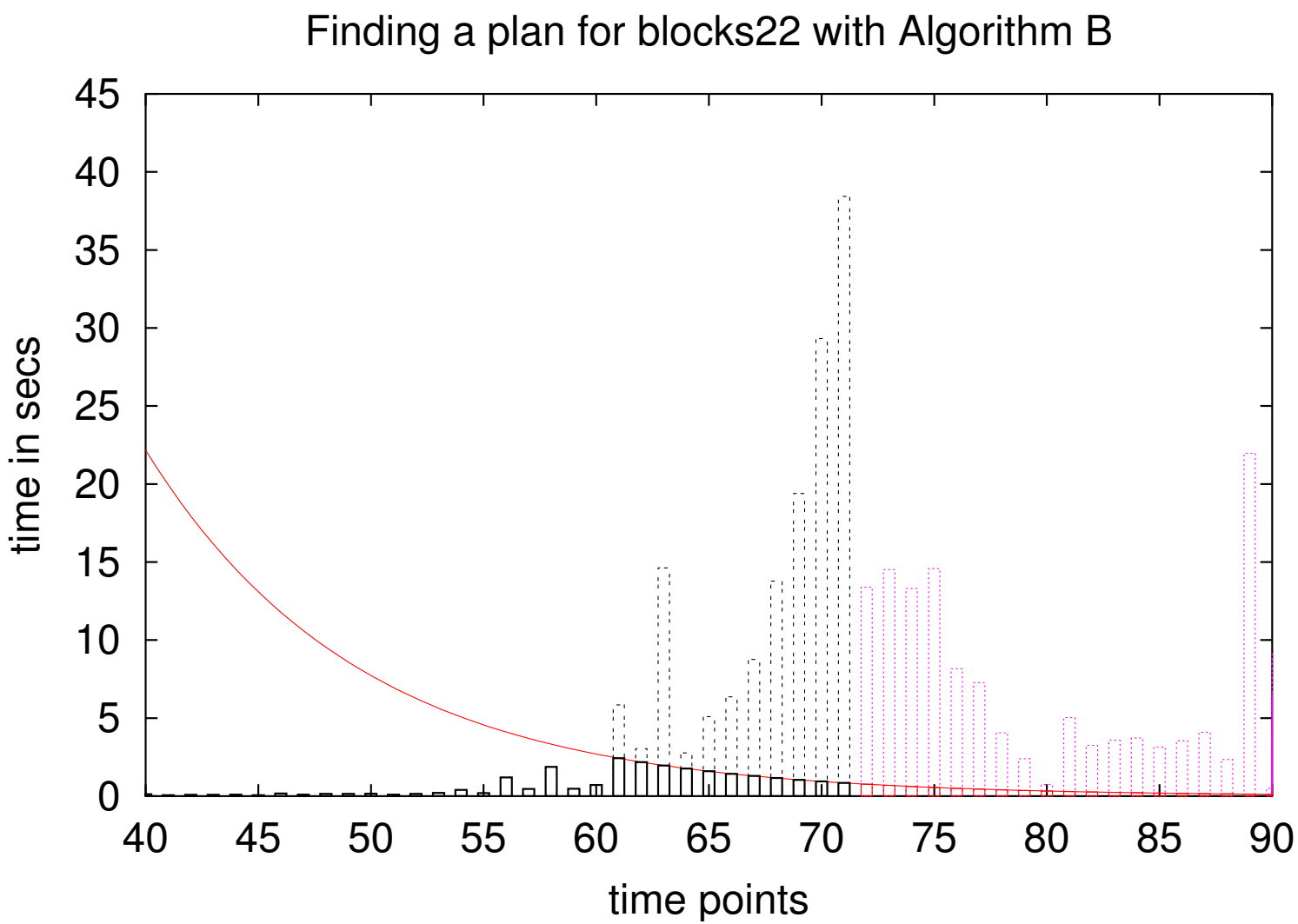
# Geometric rates
## Algorithm B [Rin04b]



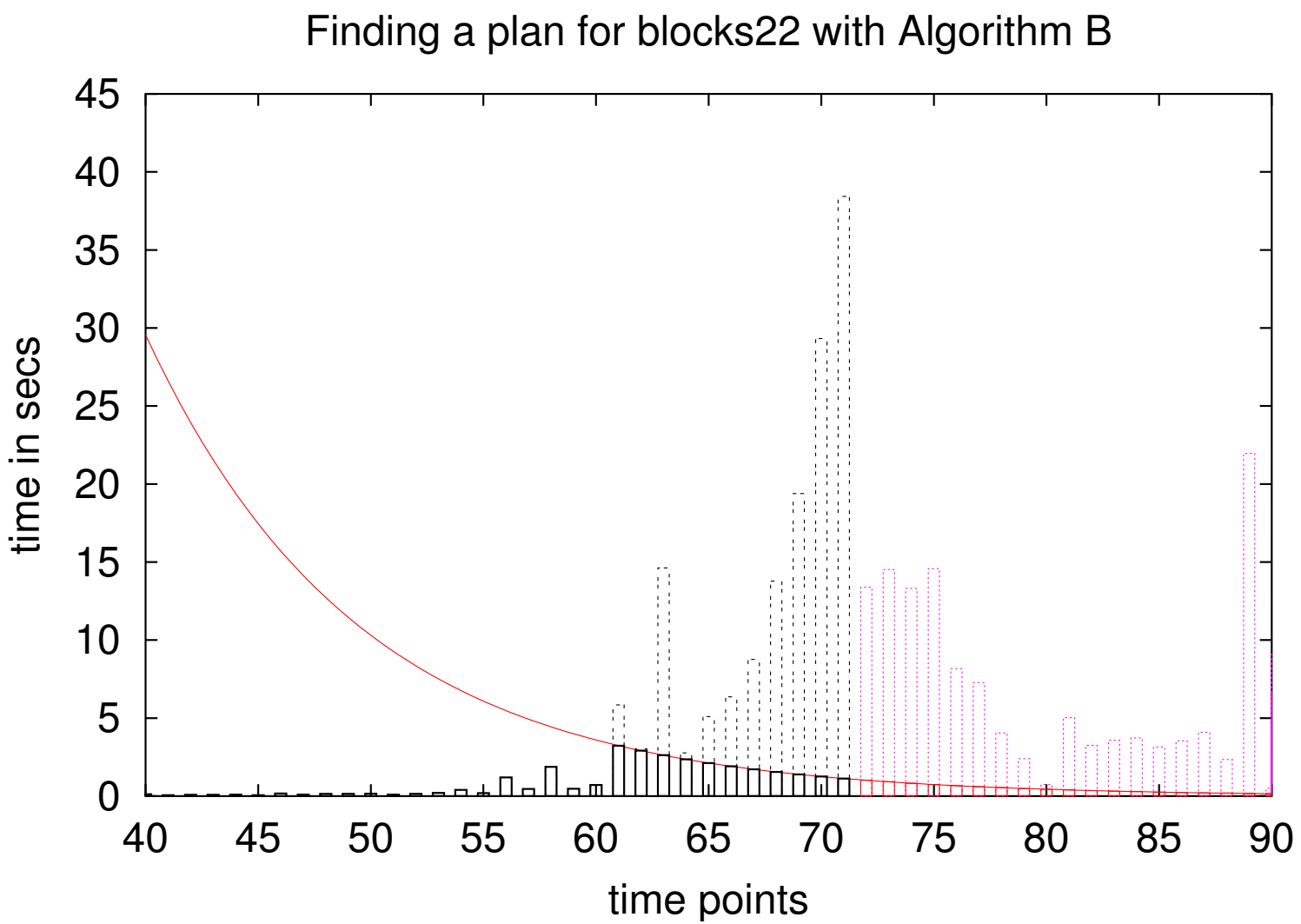
# Geometric rates
## Algorithm B [Rin04b]



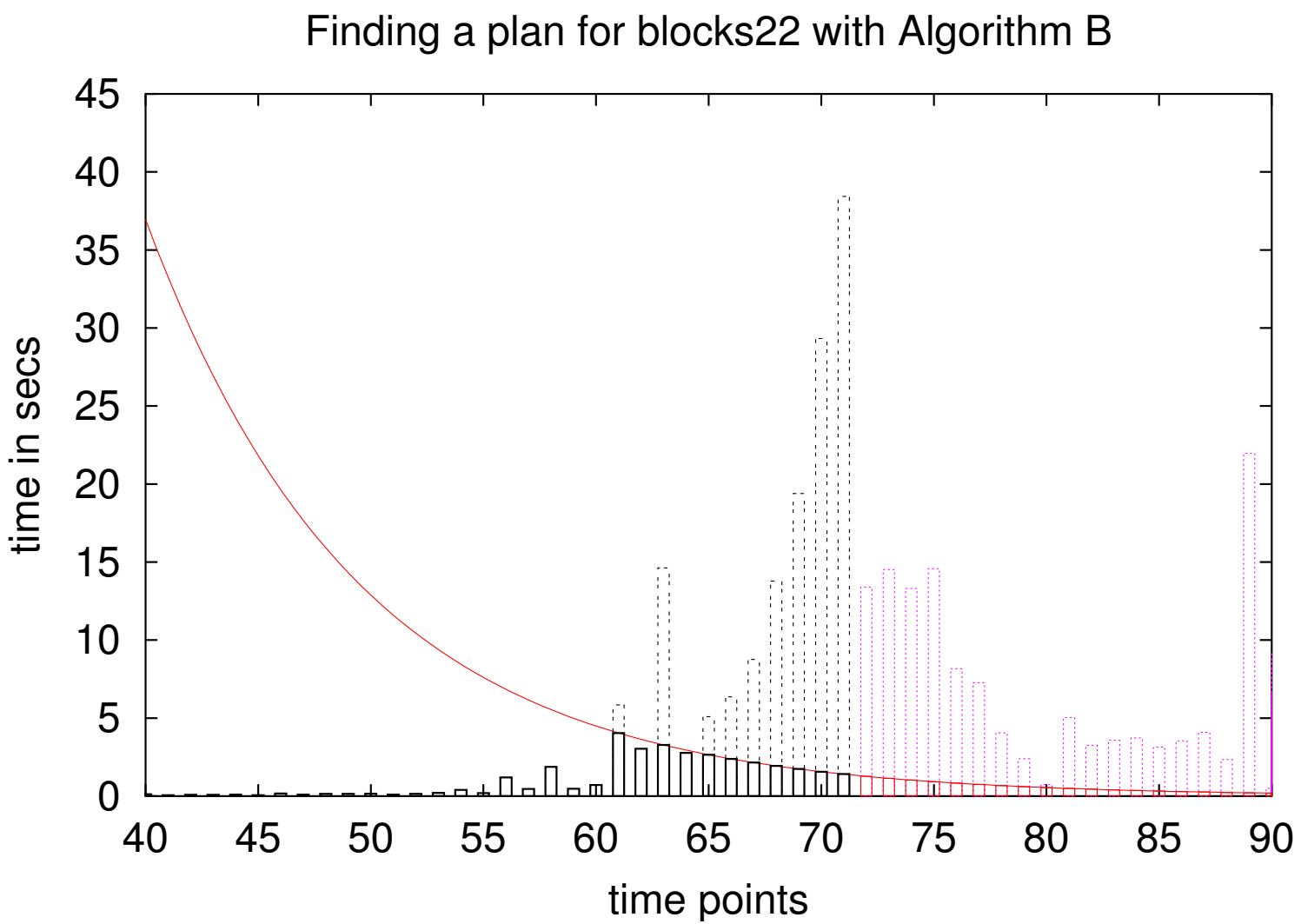
# Geometric rates
## Algorithm B [Rin04b]



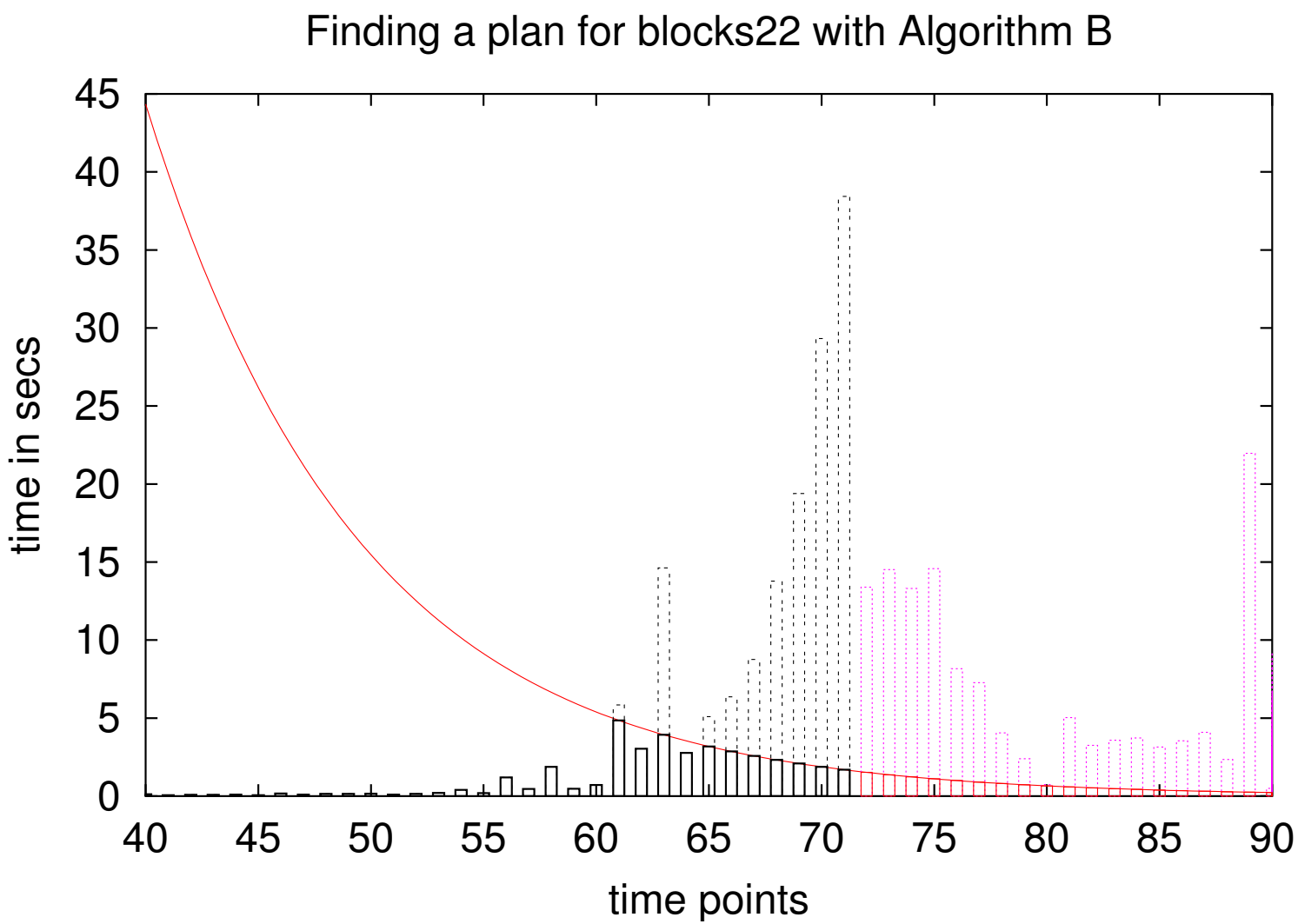
# Geometric rates
## Algorithm B [Rin04b]



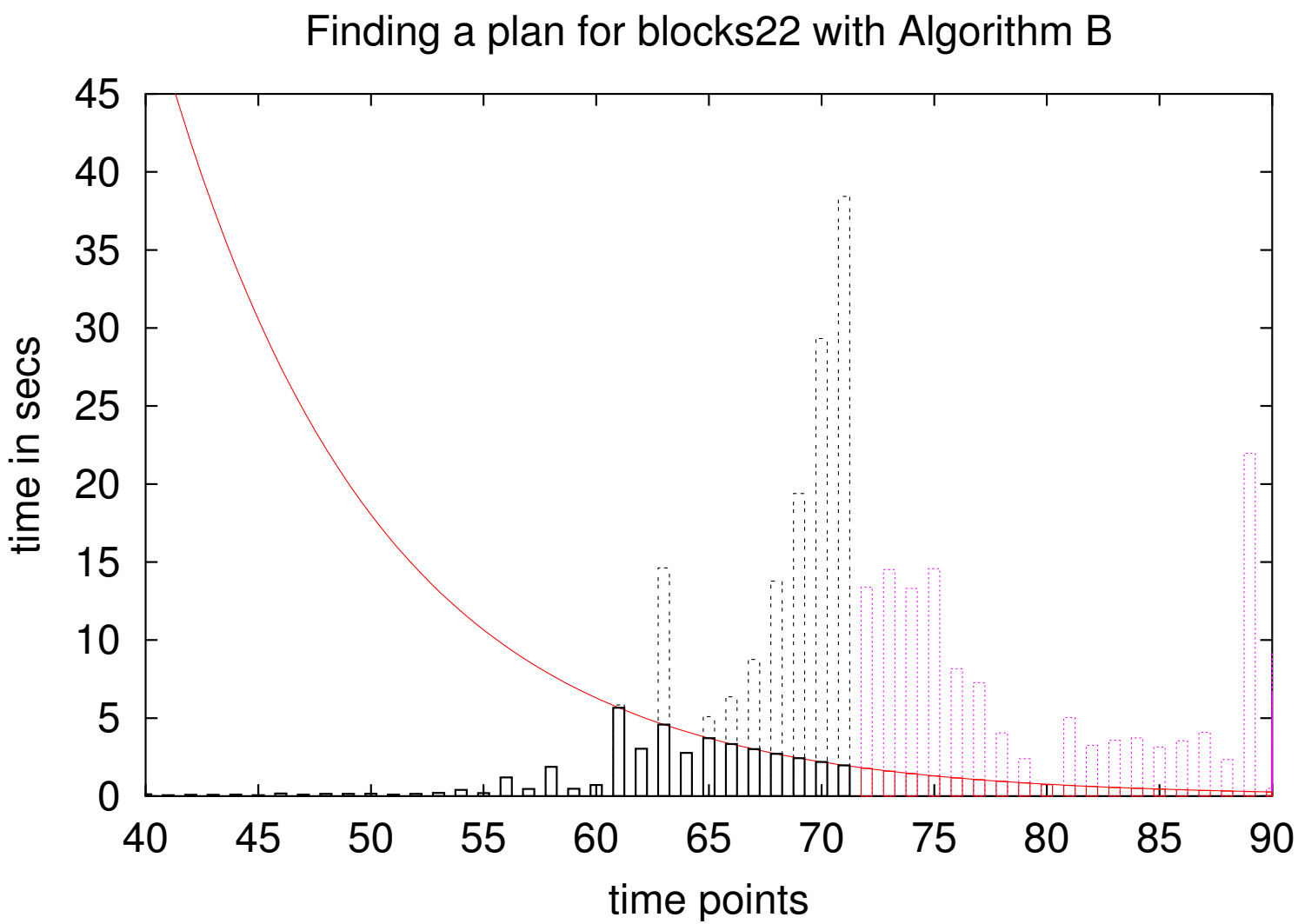
# Geometric rates
## Algorithm B [Rin04b]



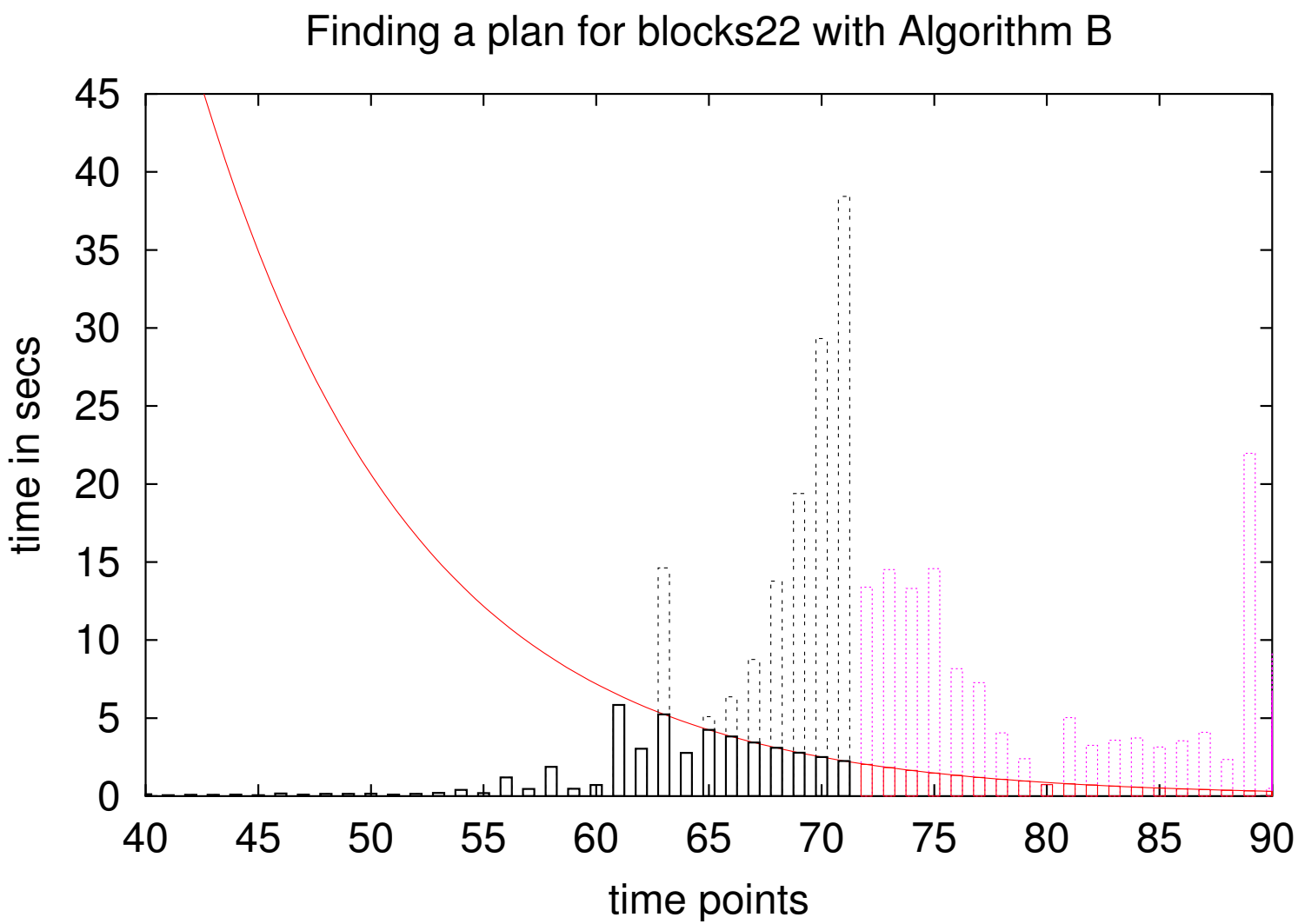
# Geometric rates
## Algorithm B [Rin04b]



# 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.

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, Rin12b]

- 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, Rin12b]

- 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, Rin12b]

- 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, Rin12b]

- 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 | **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

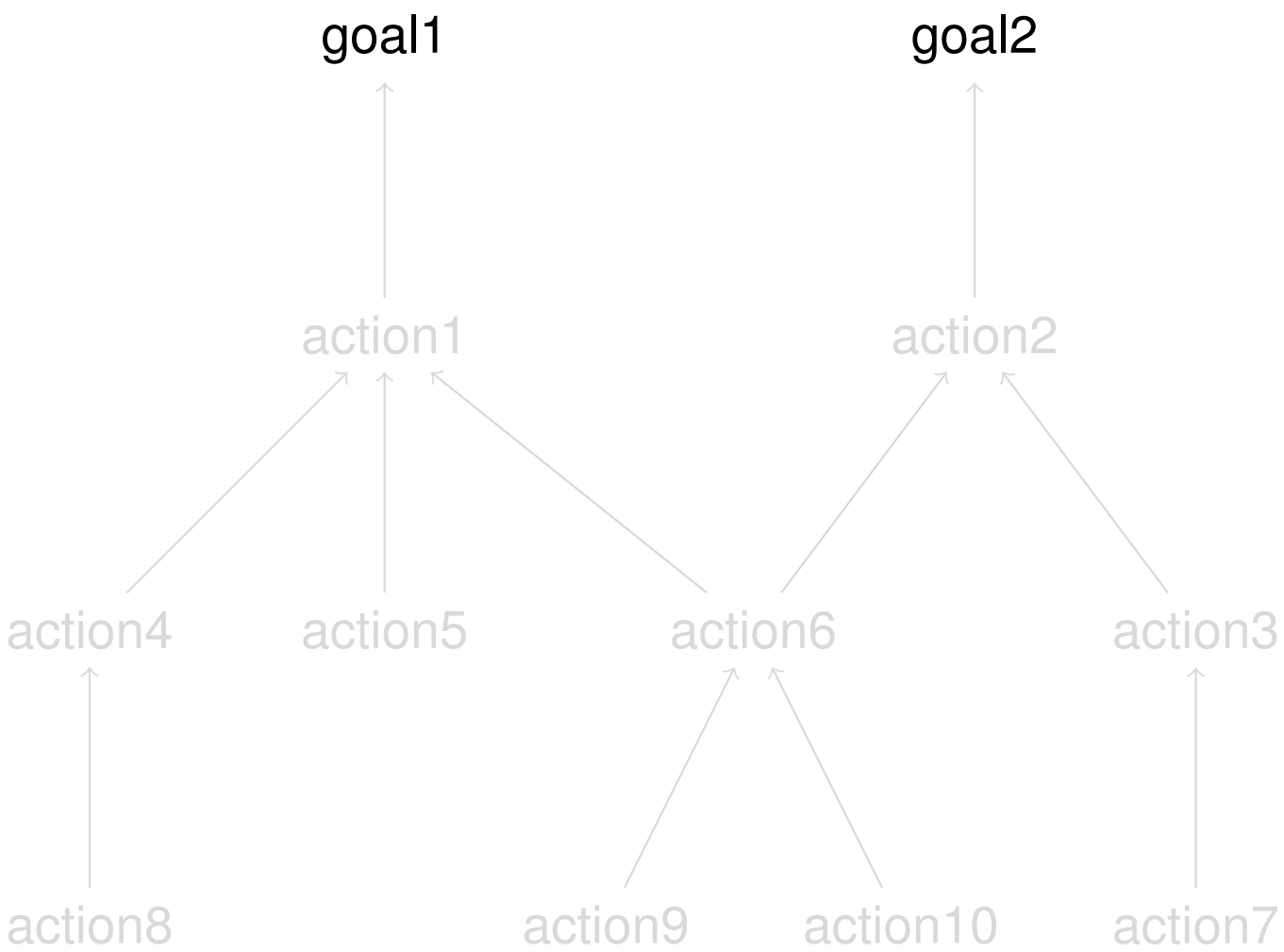
Goal/subgoal is already made true at $t-4$ by action 4 at $t-5$.

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

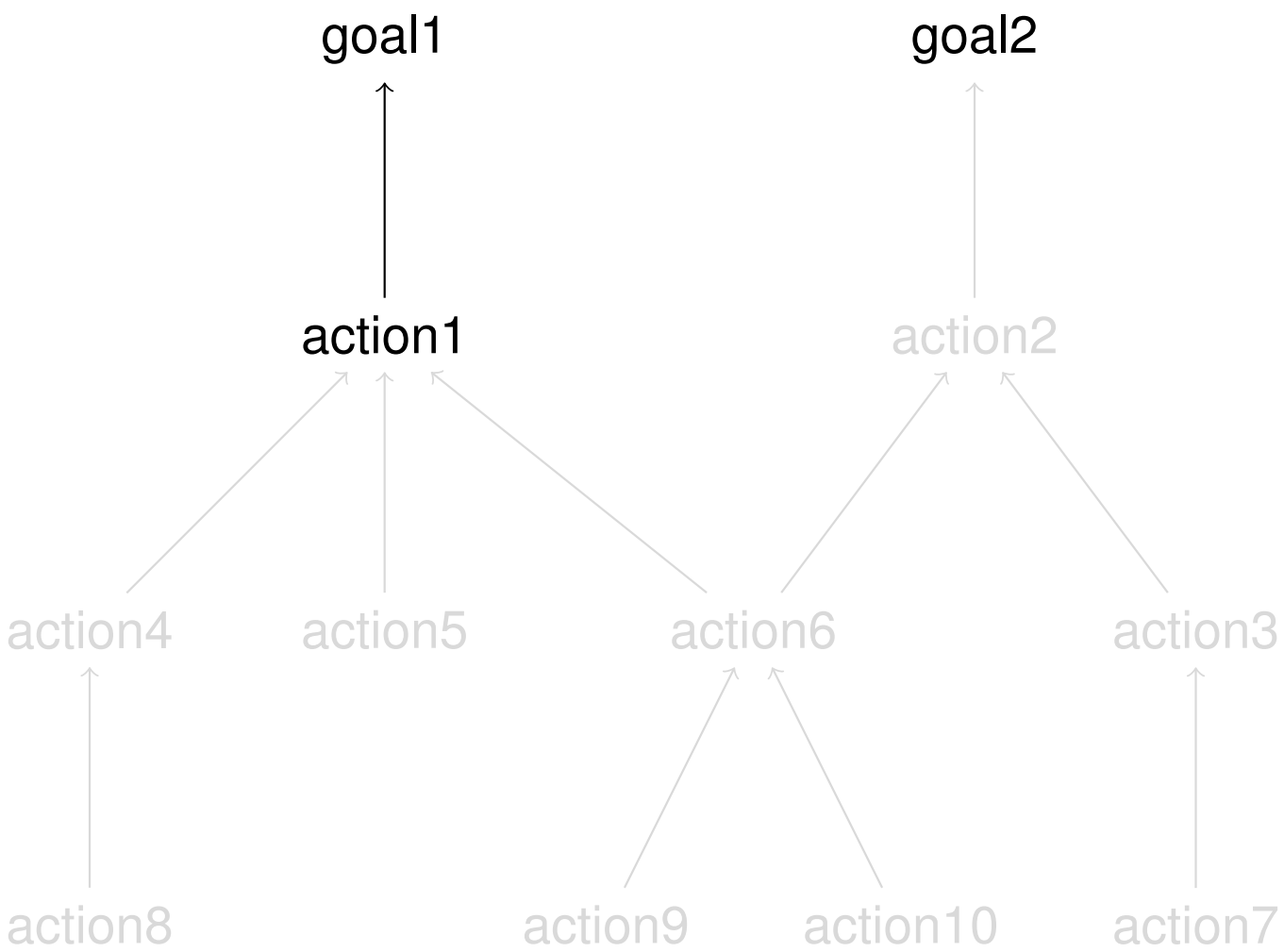
# The variable selection scheme

## Version 1: strict depth-first search



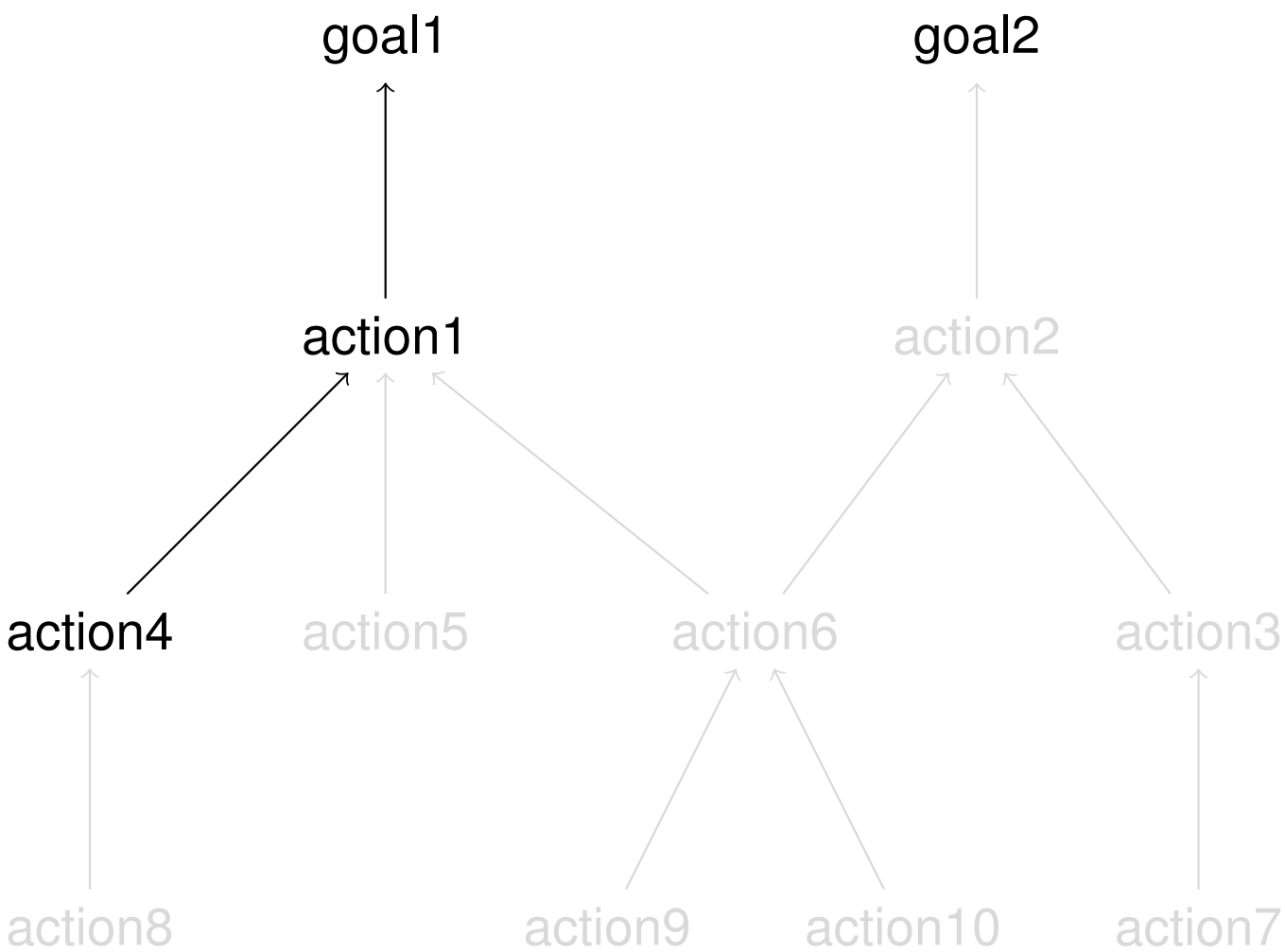
# The variable selection scheme

## Version 1: strict depth-first search



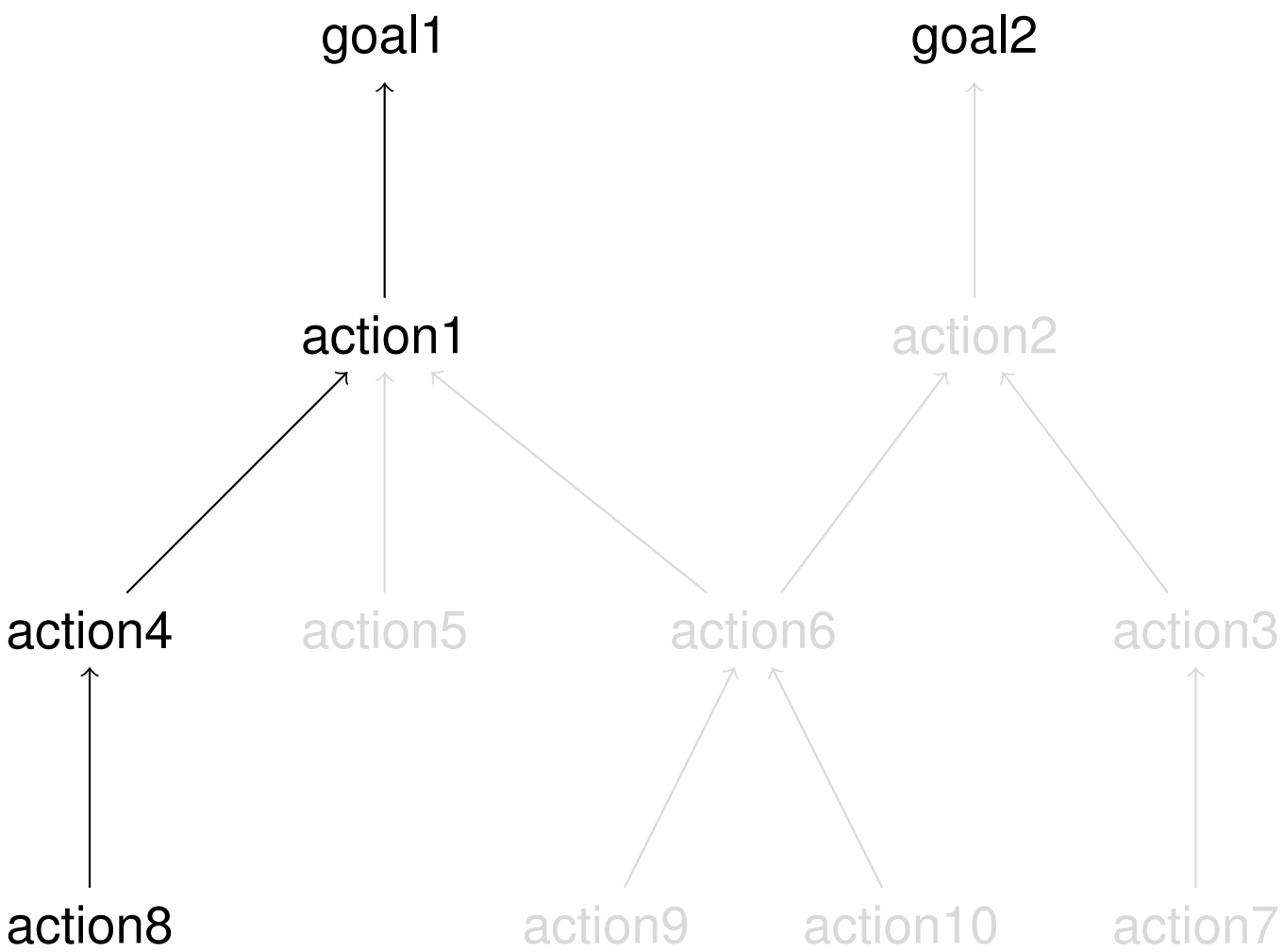
# The variable selection scheme

## Version 1: strict depth-first search



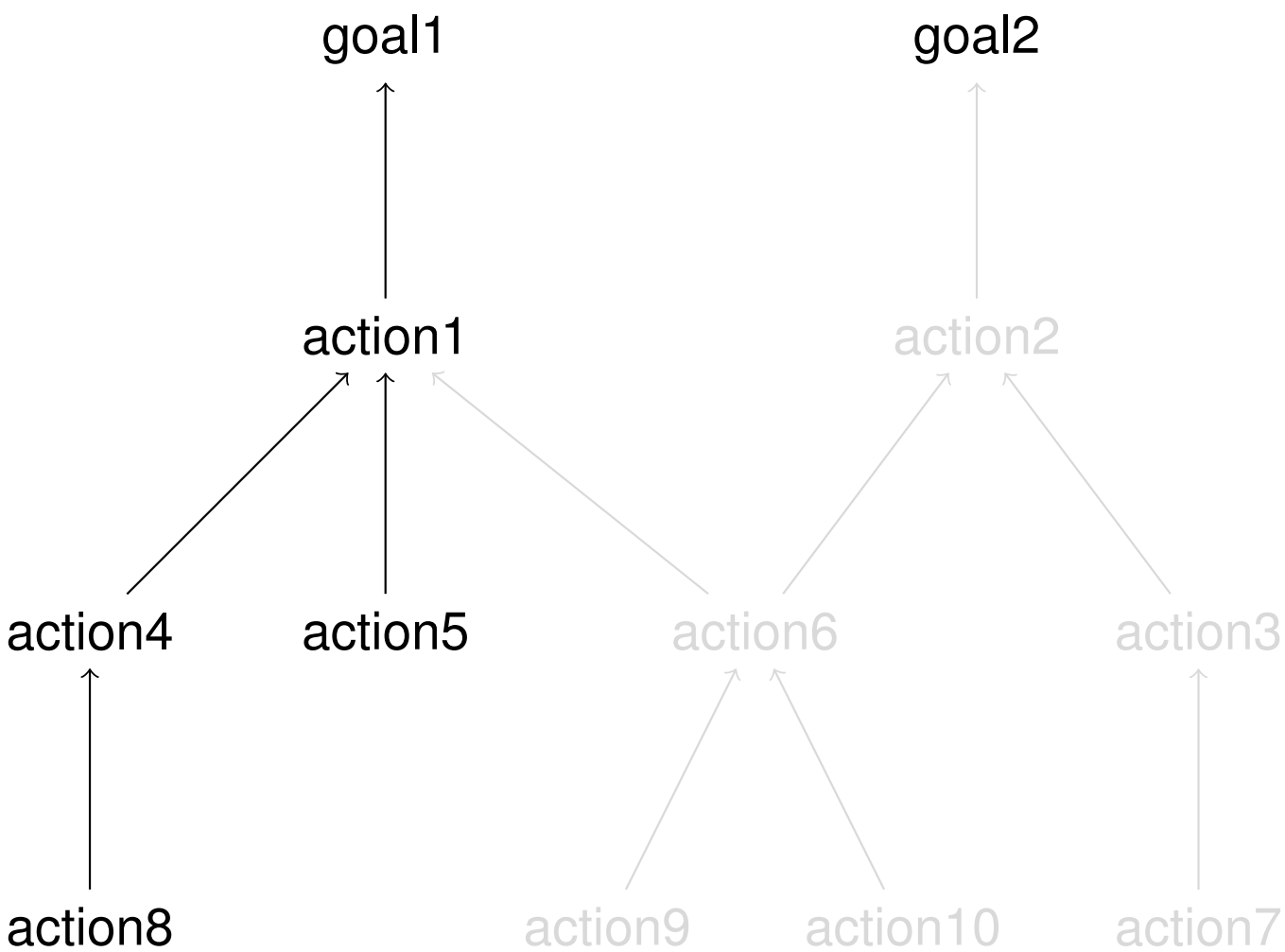
# The variable selection scheme

## Version 1: strict depth-first search



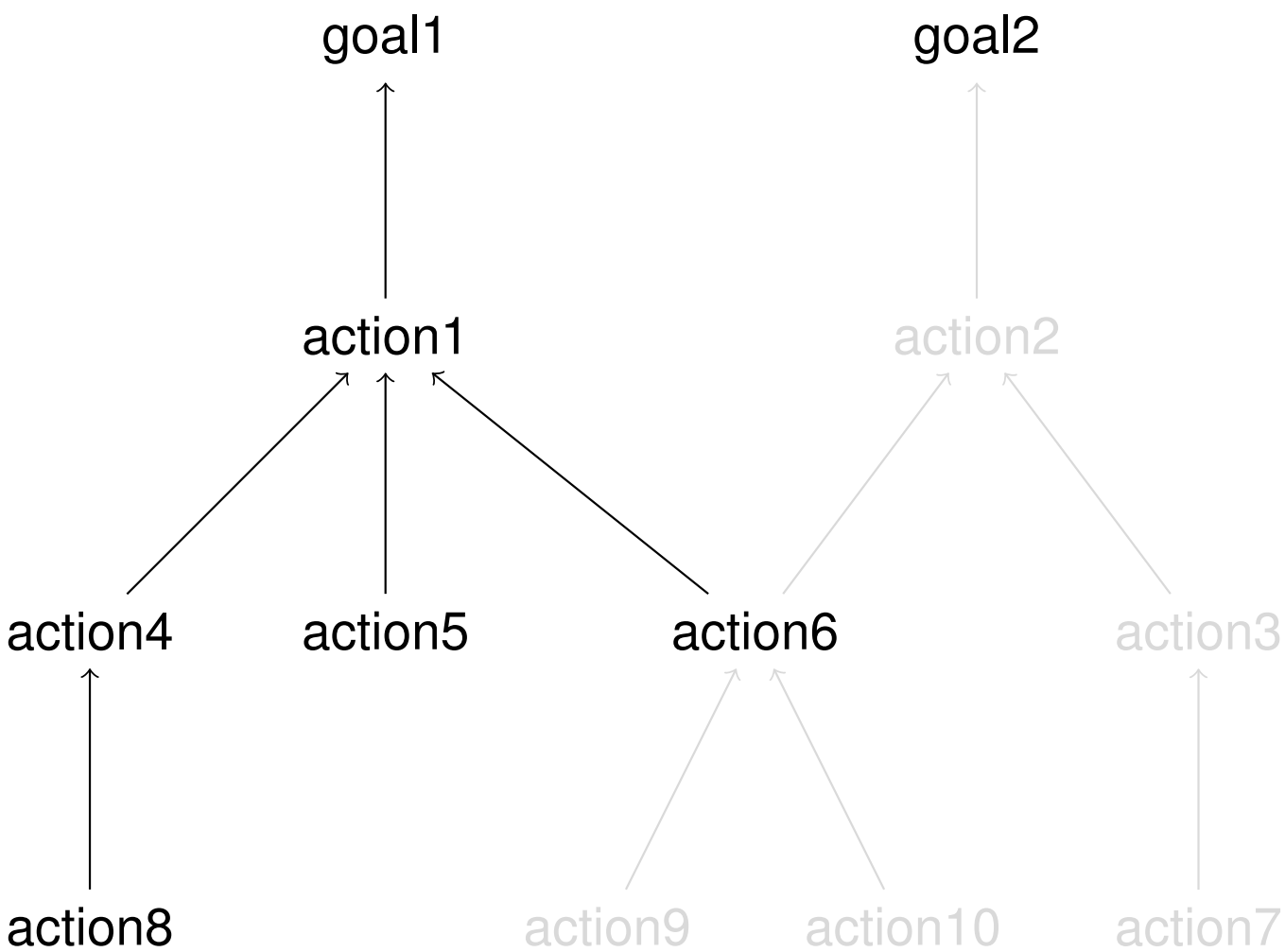
# The variable selection scheme

## Version 1: strict depth-first search



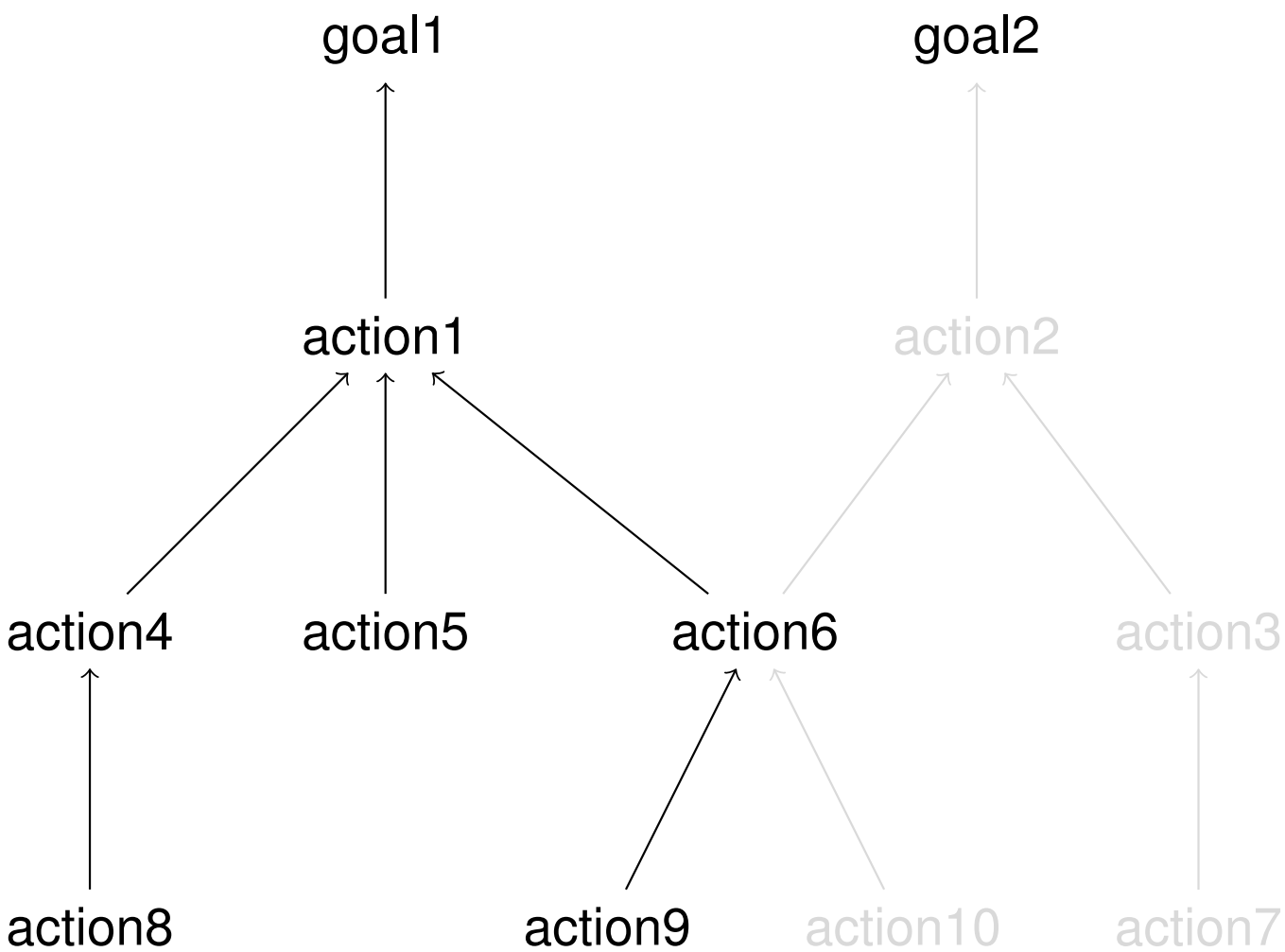
# The variable selection scheme

## Version 1: strict depth-first search



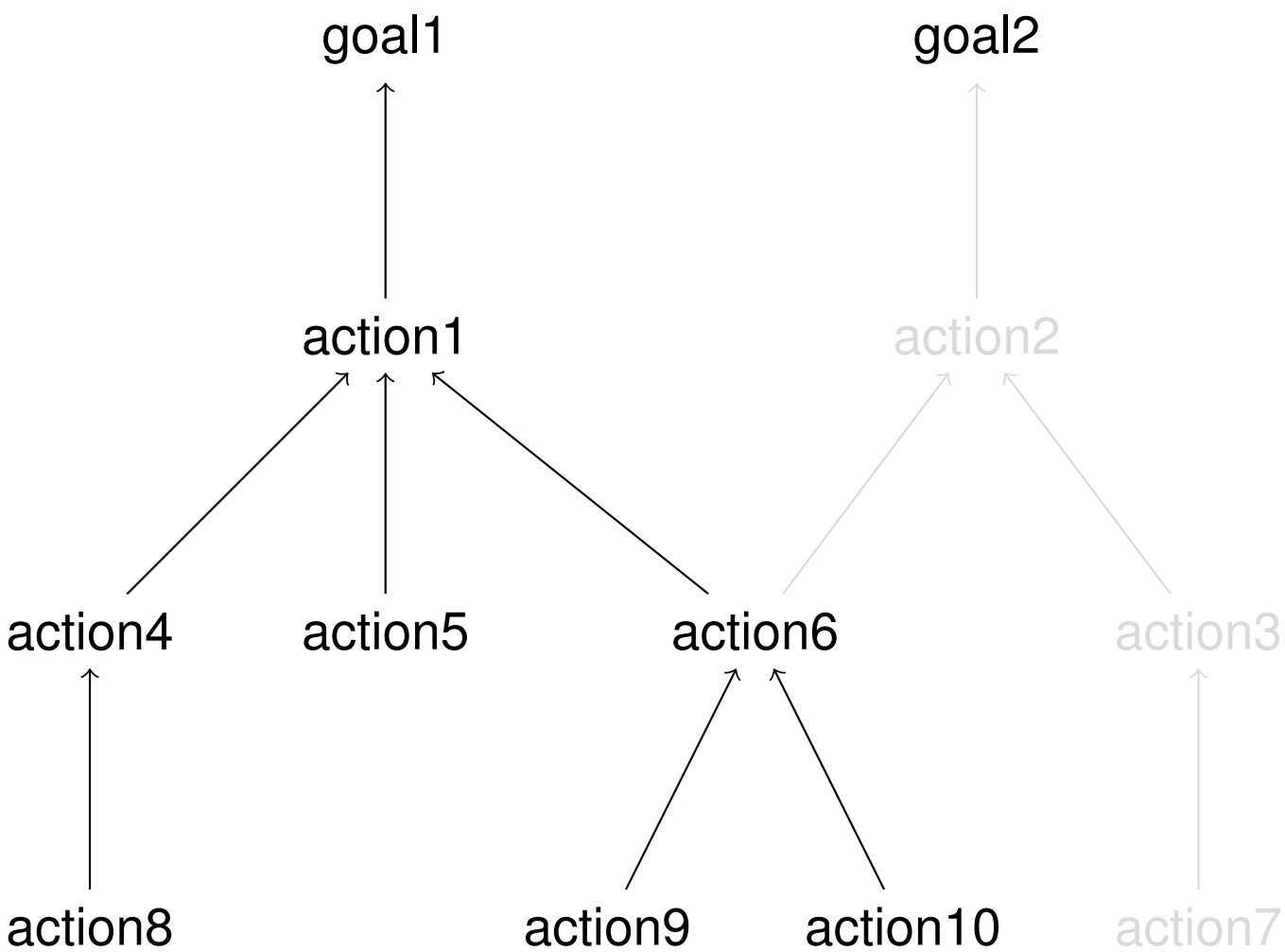
# The variable selection scheme

## Version 1: strict depth-first search



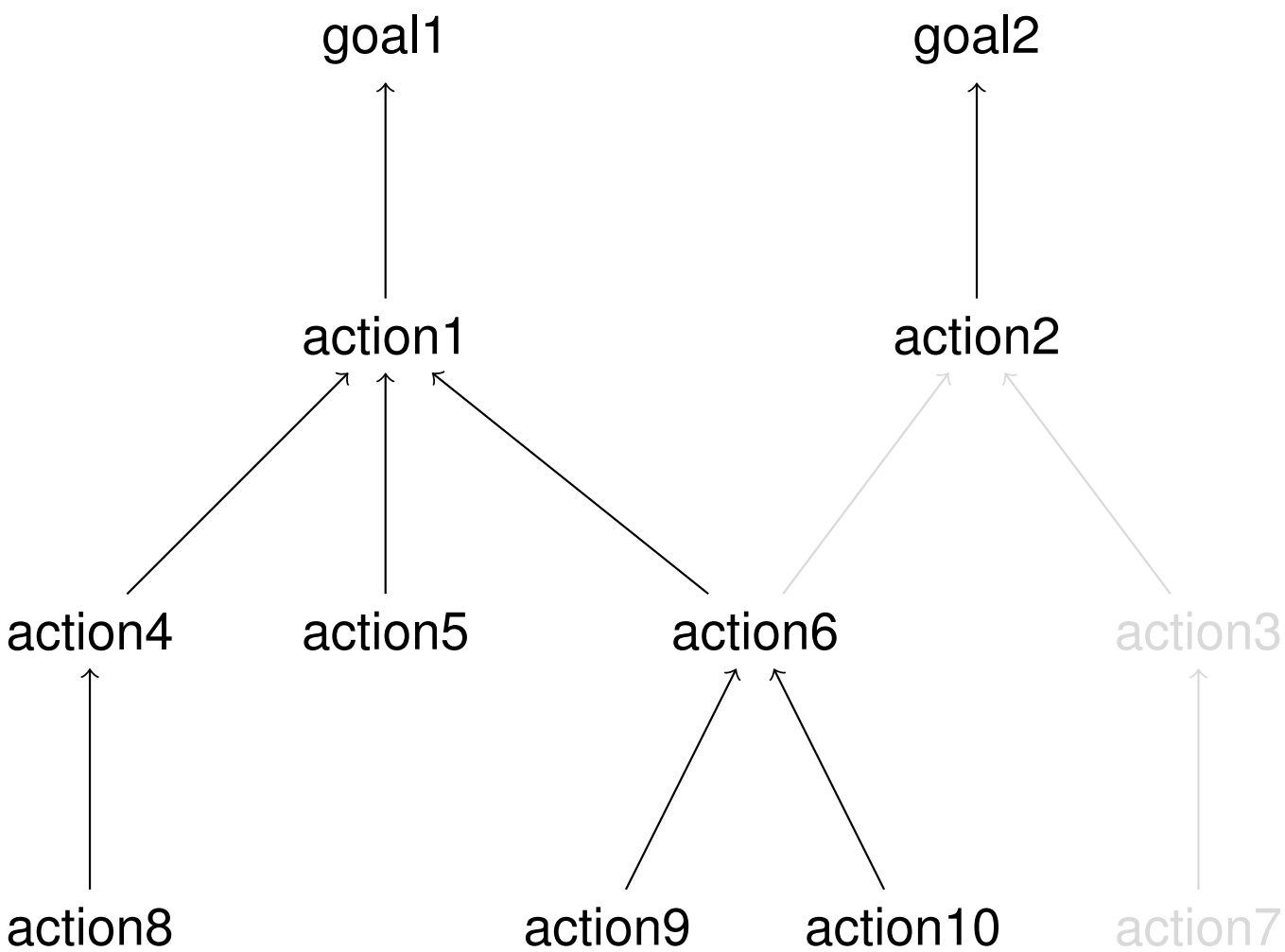
# The variable selection scheme

## Version 1: strict depth-first search



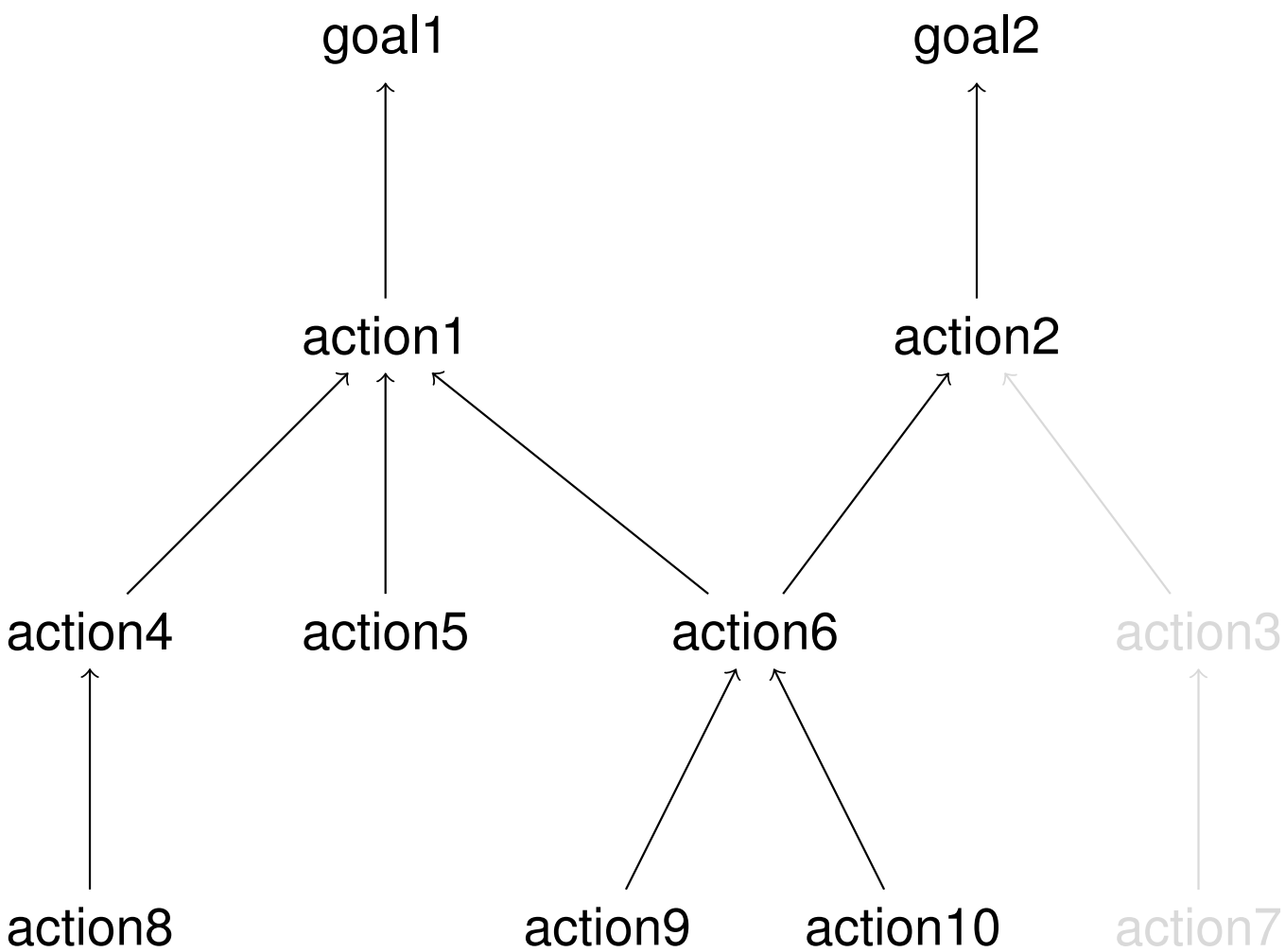
# The variable selection scheme

## Version 1: strict depth-first search



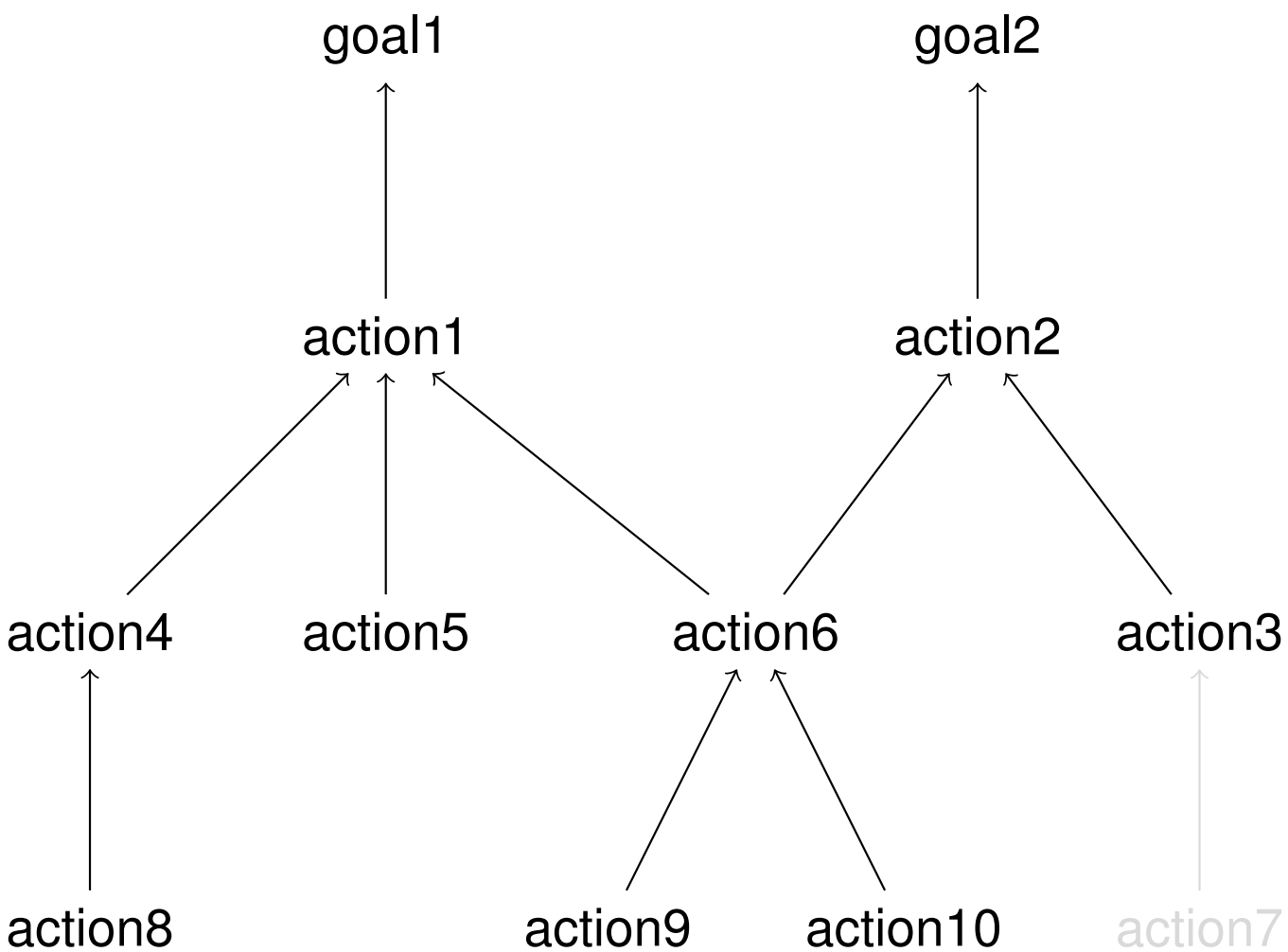
# The variable selection scheme

## Version 1: strict depth-first search



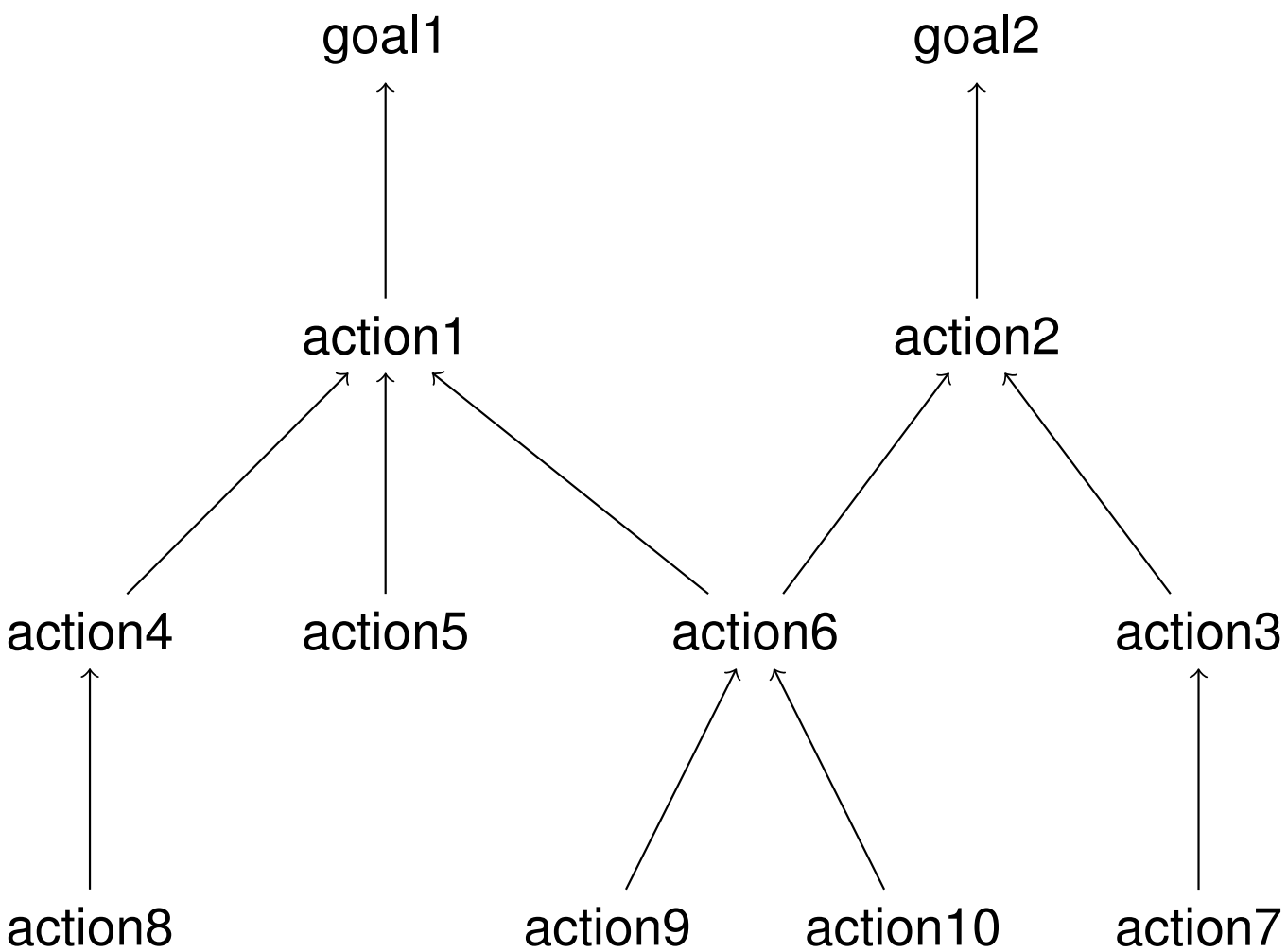
# The variable selection scheme

## Version 1: strict depth-first search



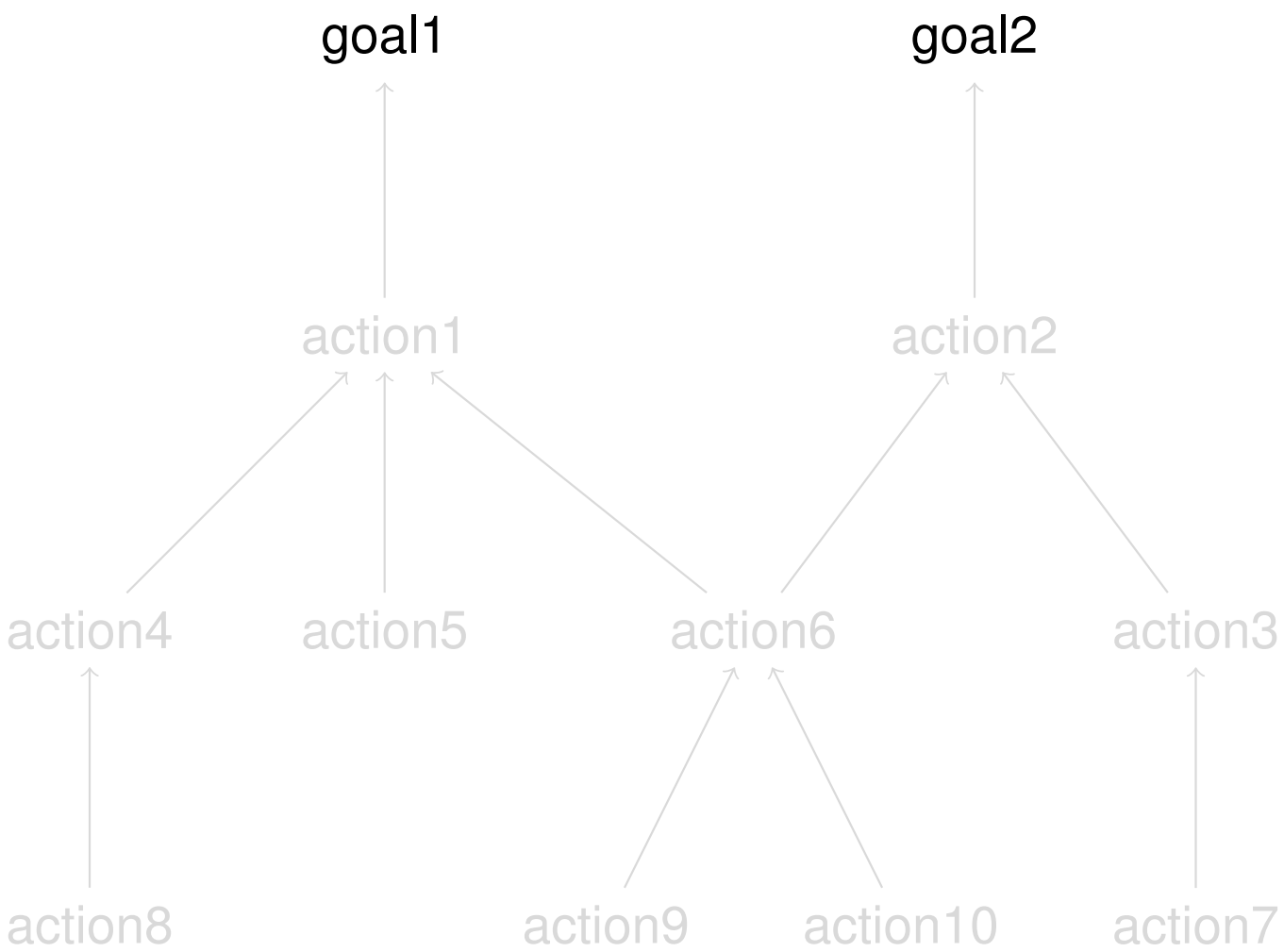
# The variable selection scheme

## Version 1: strict depth-first search



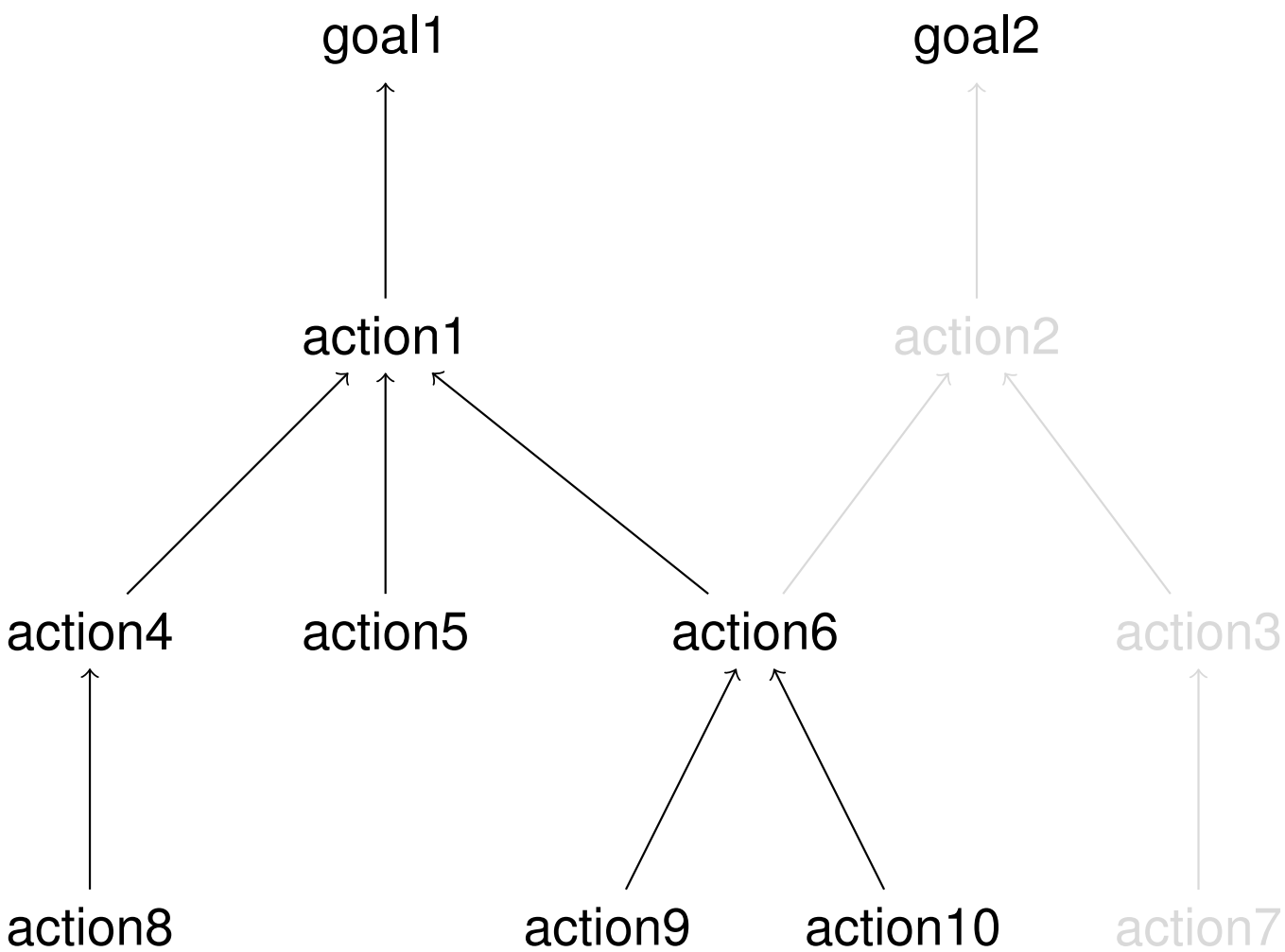
# The variable selection scheme

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



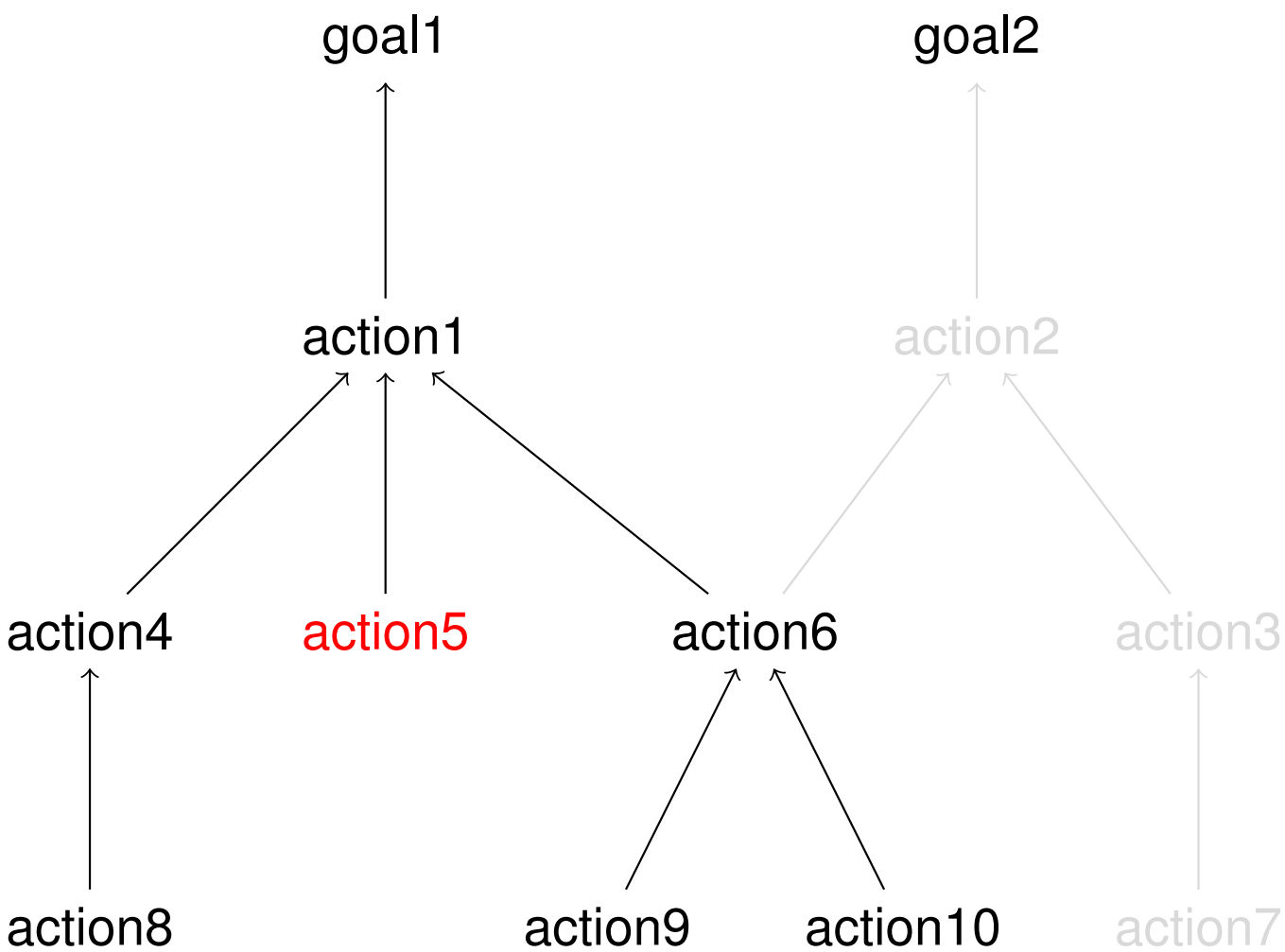
# The variable selection scheme

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



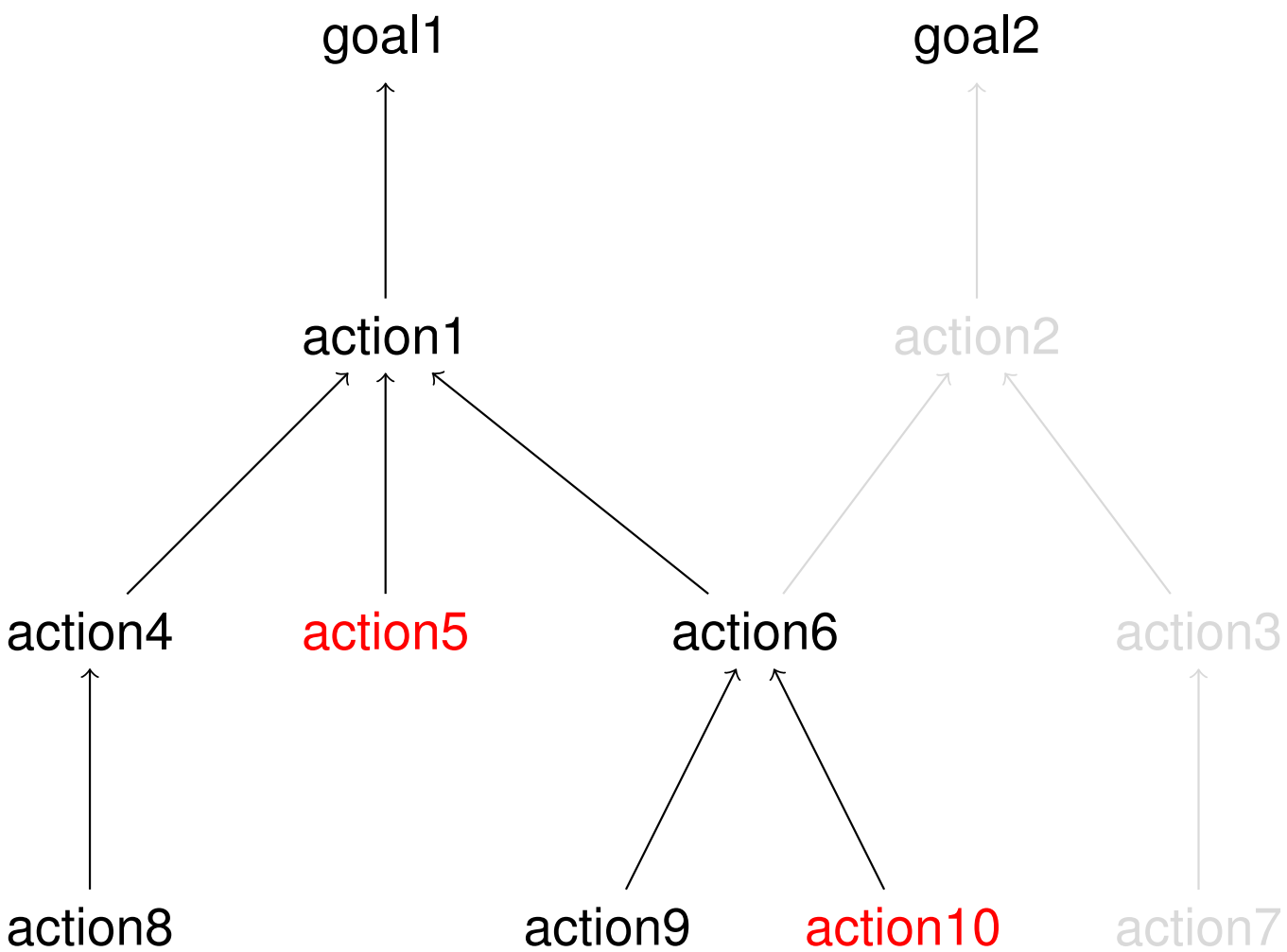
# The variable selection scheme

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



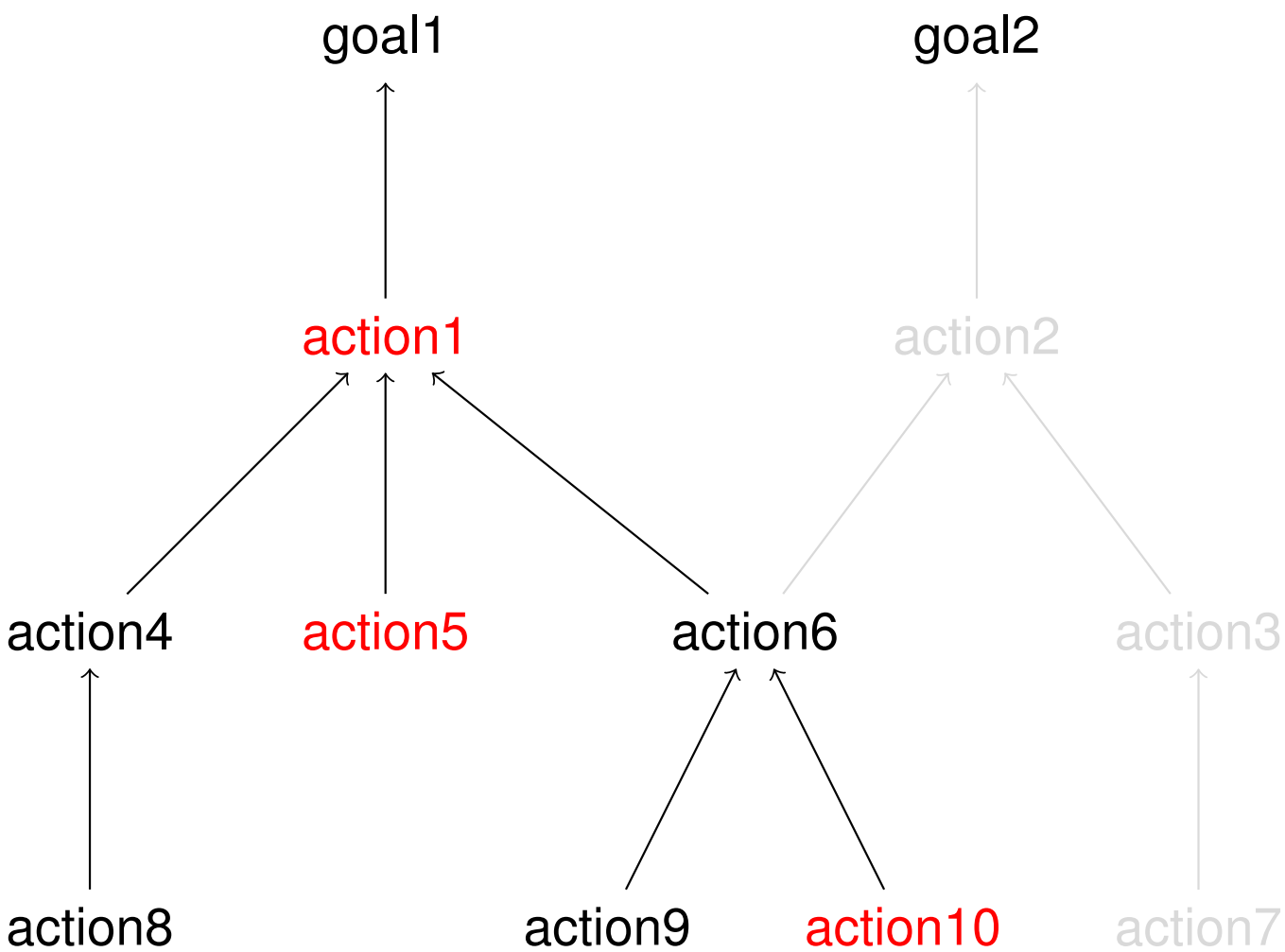
# The variable selection scheme

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



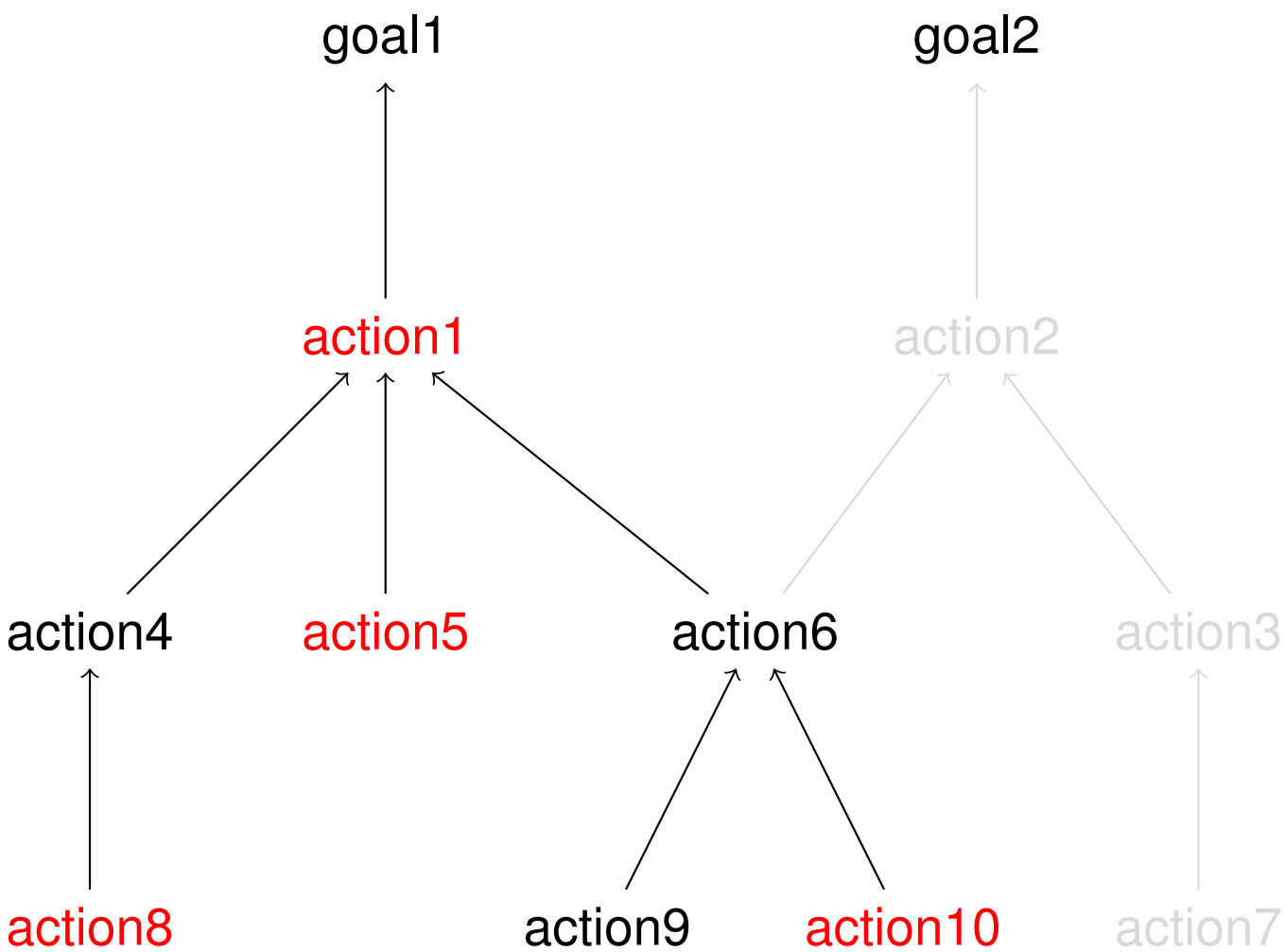
# The variable selection scheme

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



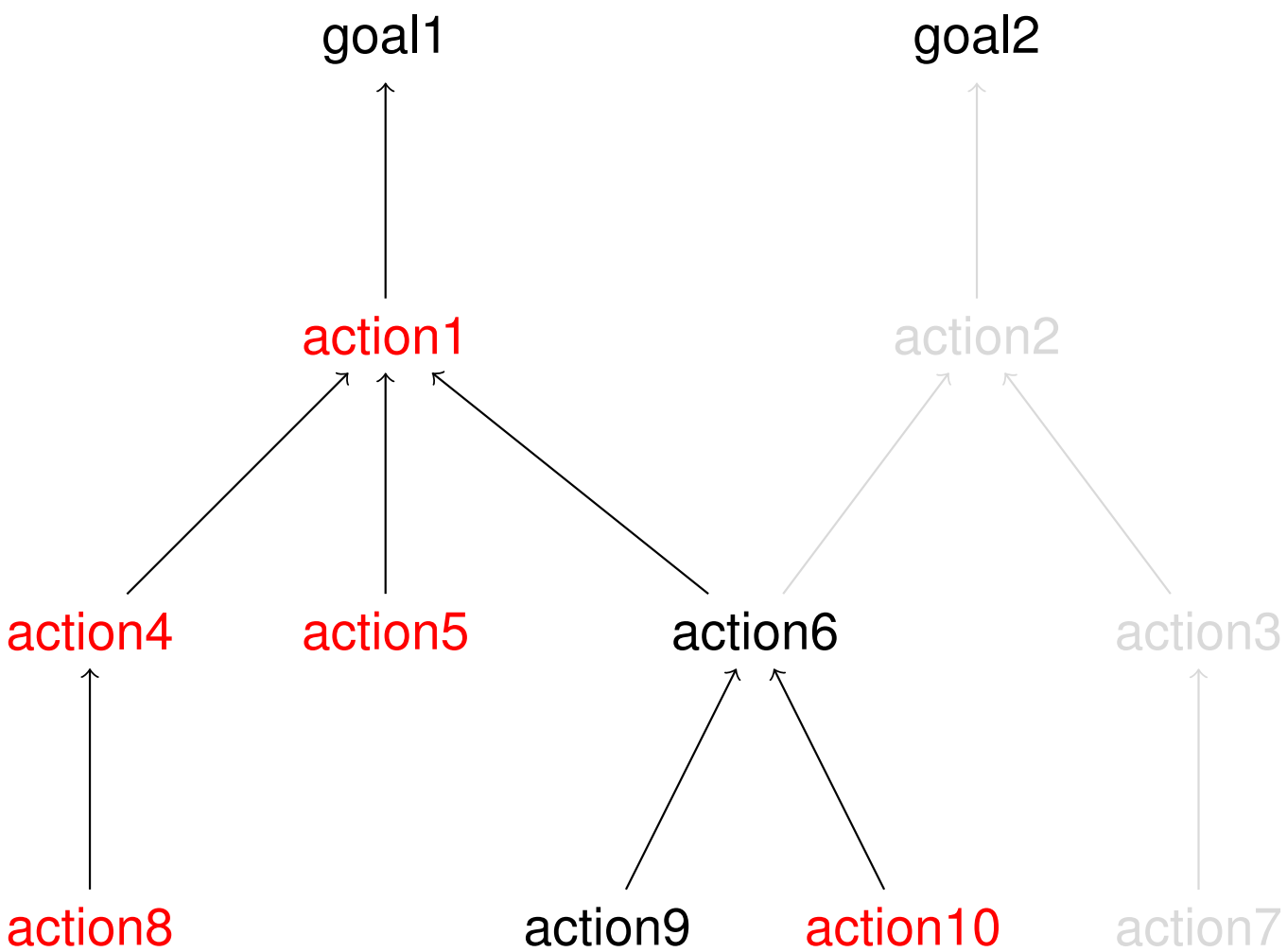
# The variable selection scheme

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



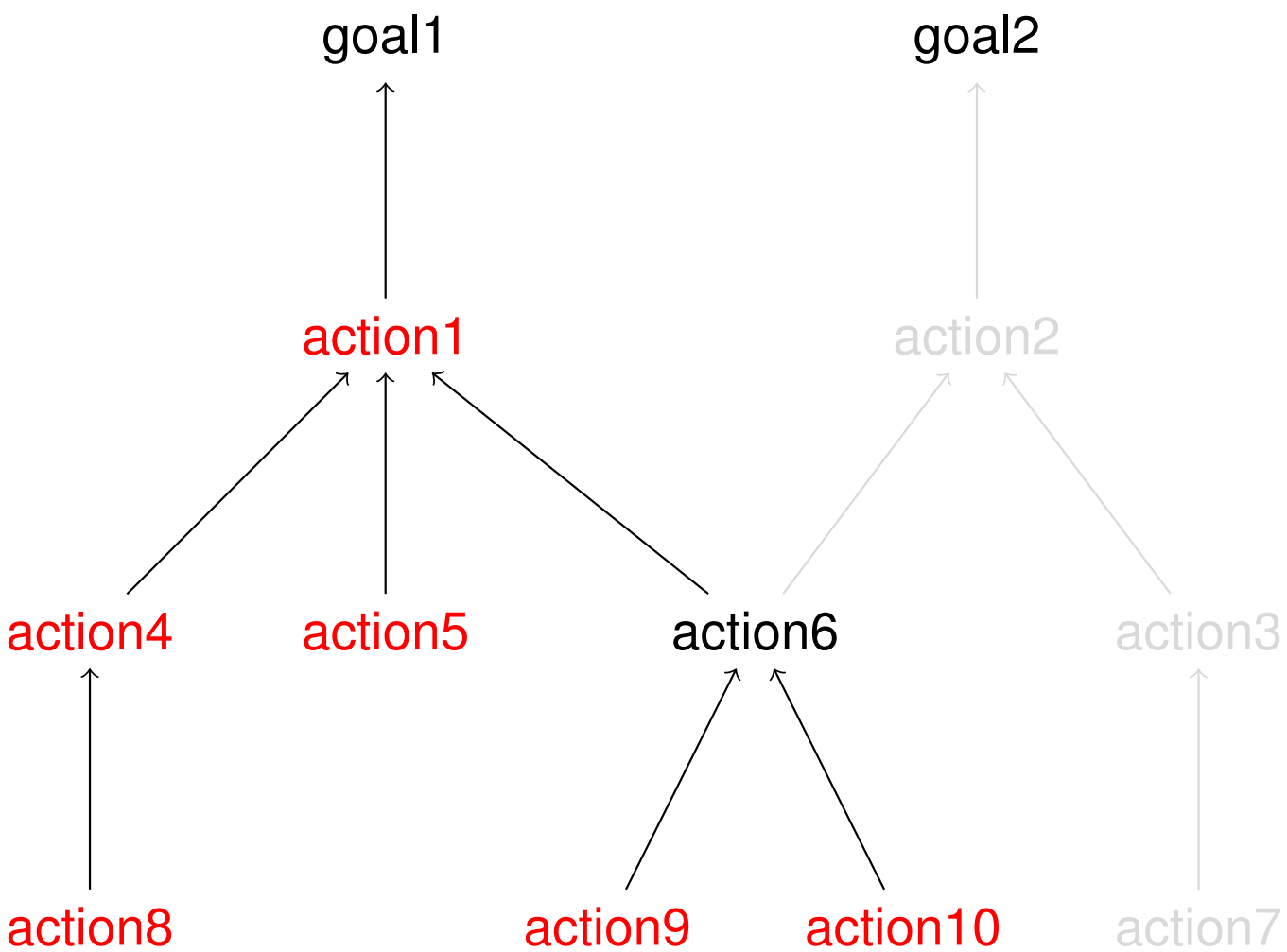
# The variable selection scheme

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



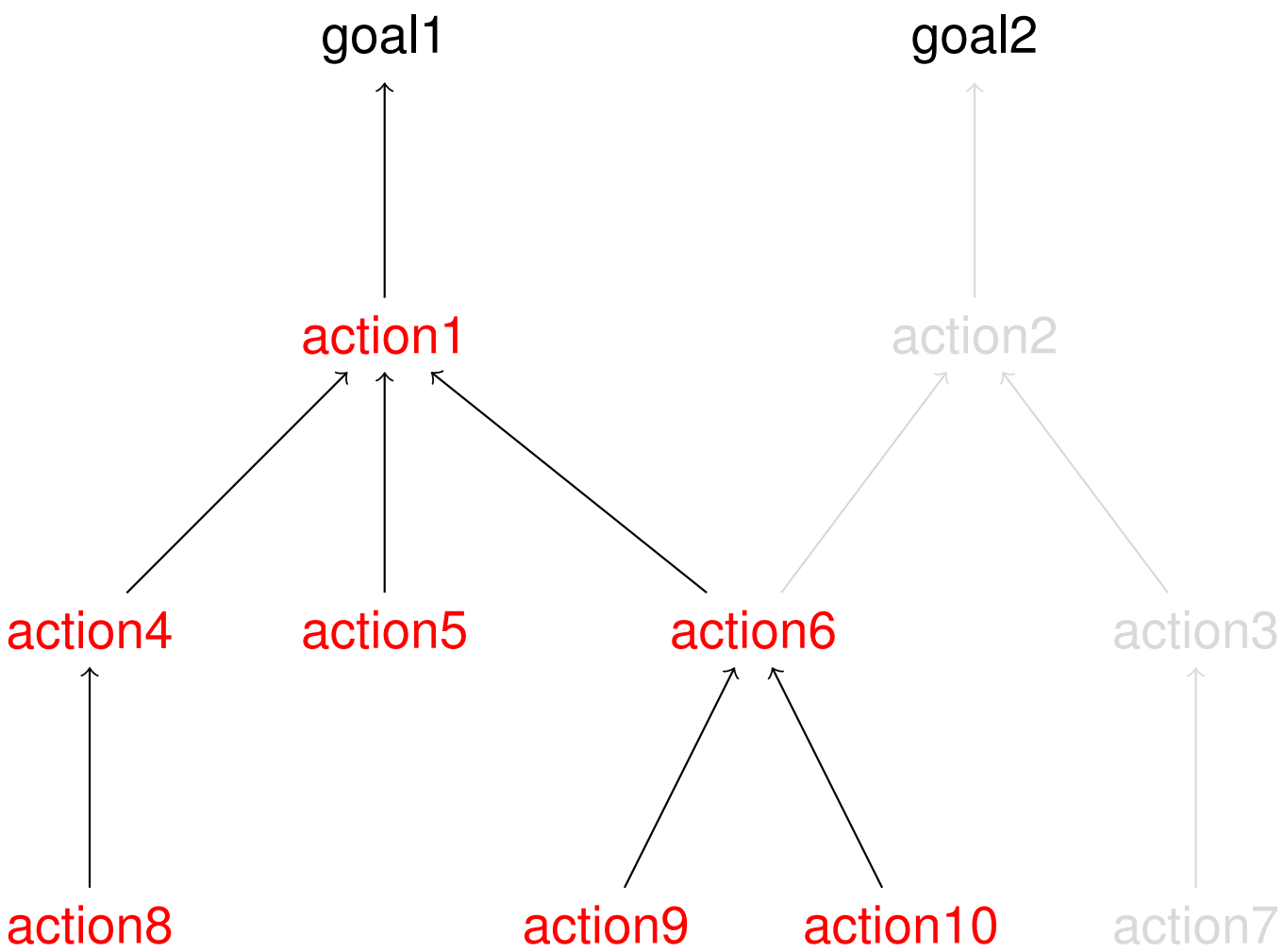
# The variable selection scheme

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



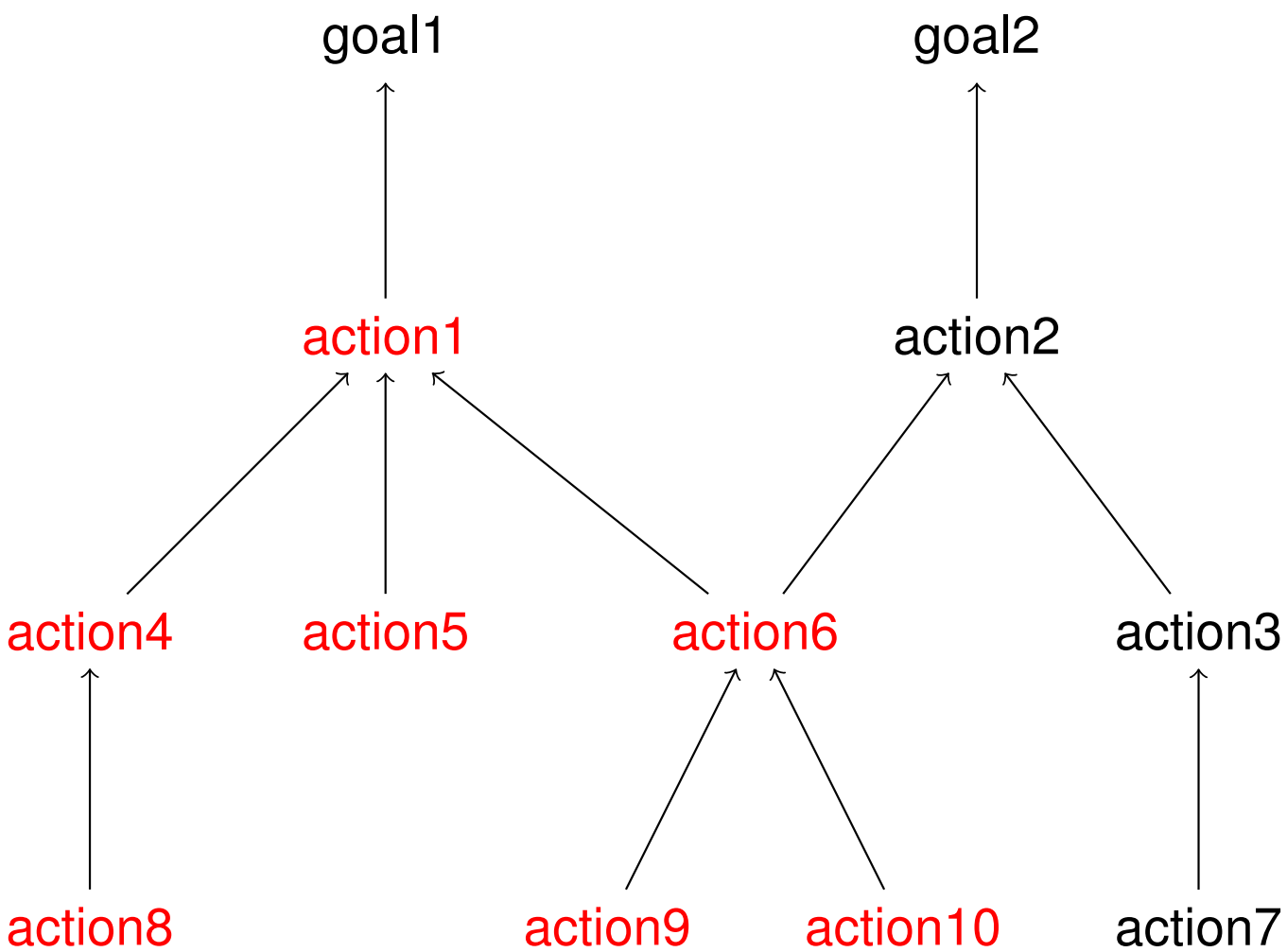
# The variable selection scheme

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



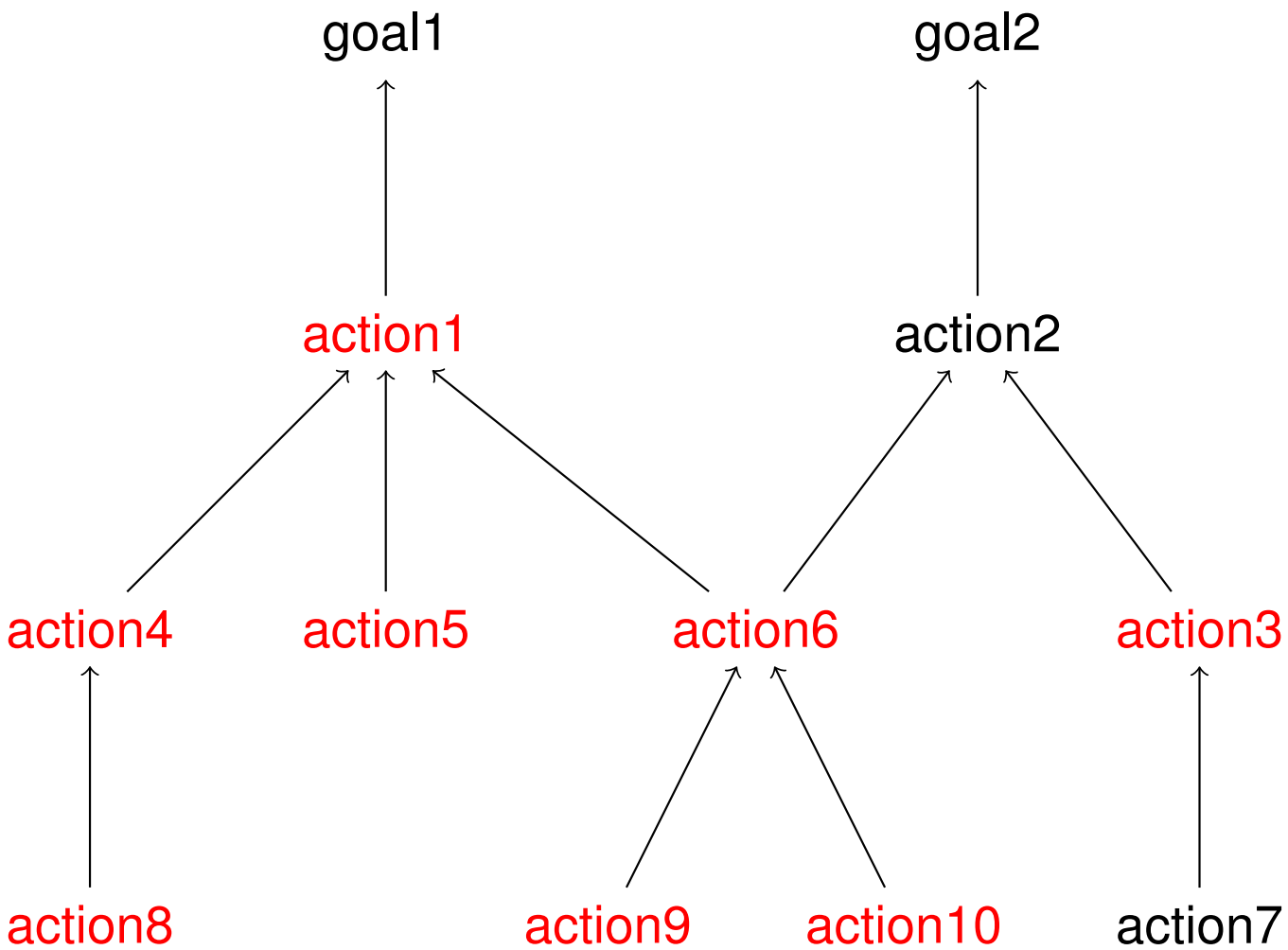
# The variable selection scheme

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



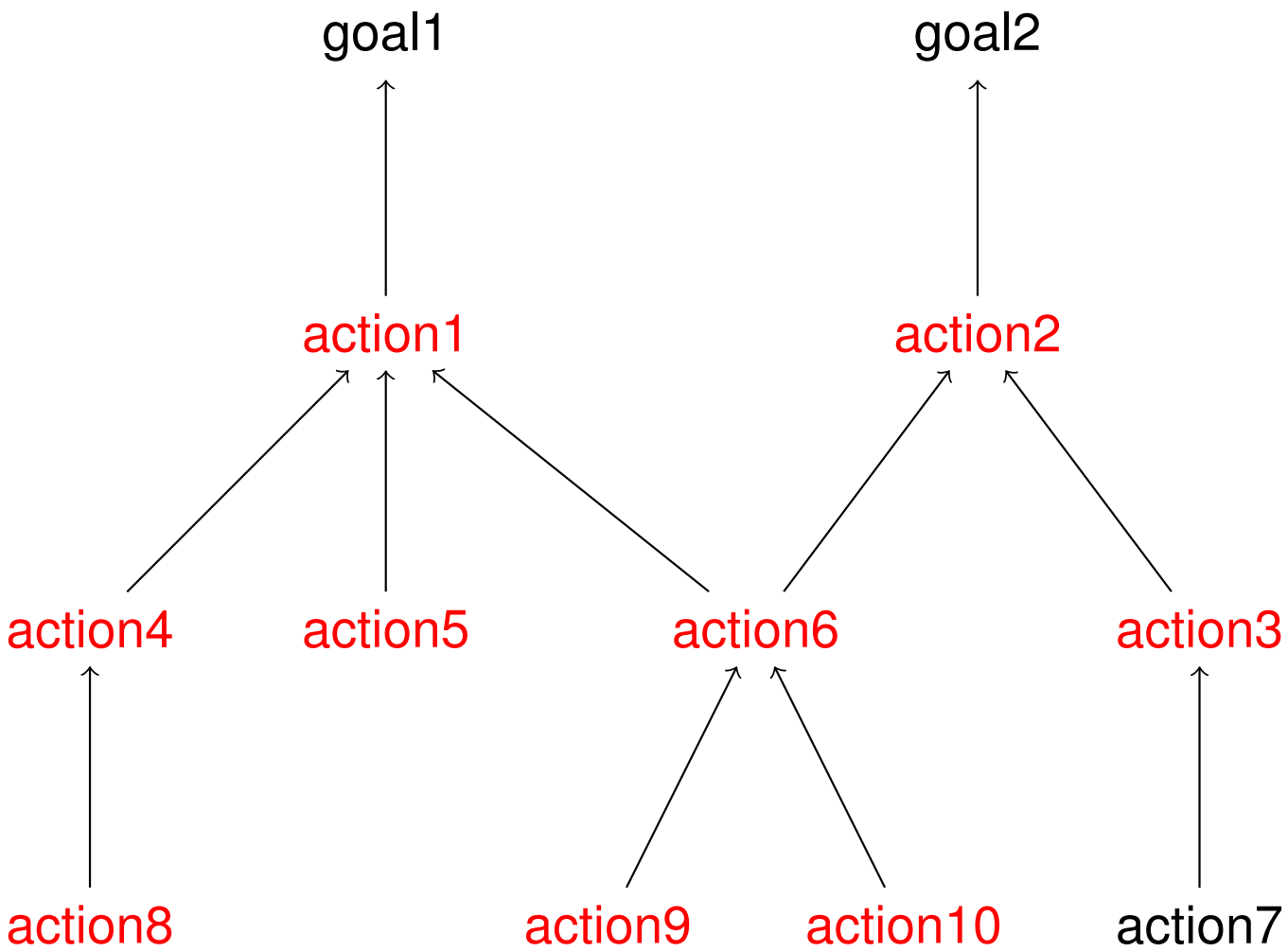
# The variable selection scheme

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



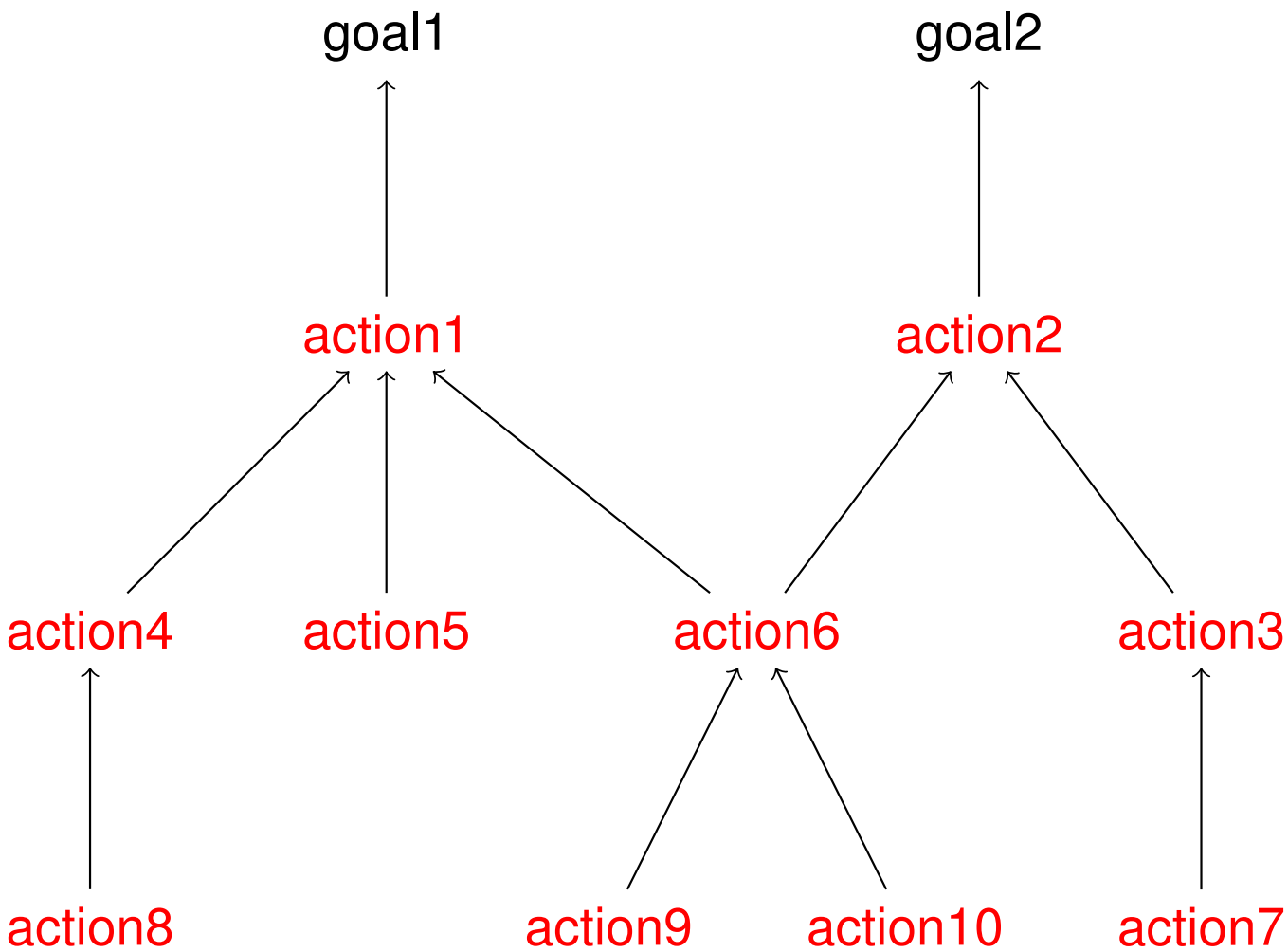
# The variable selection scheme

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



# The variable selection scheme

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

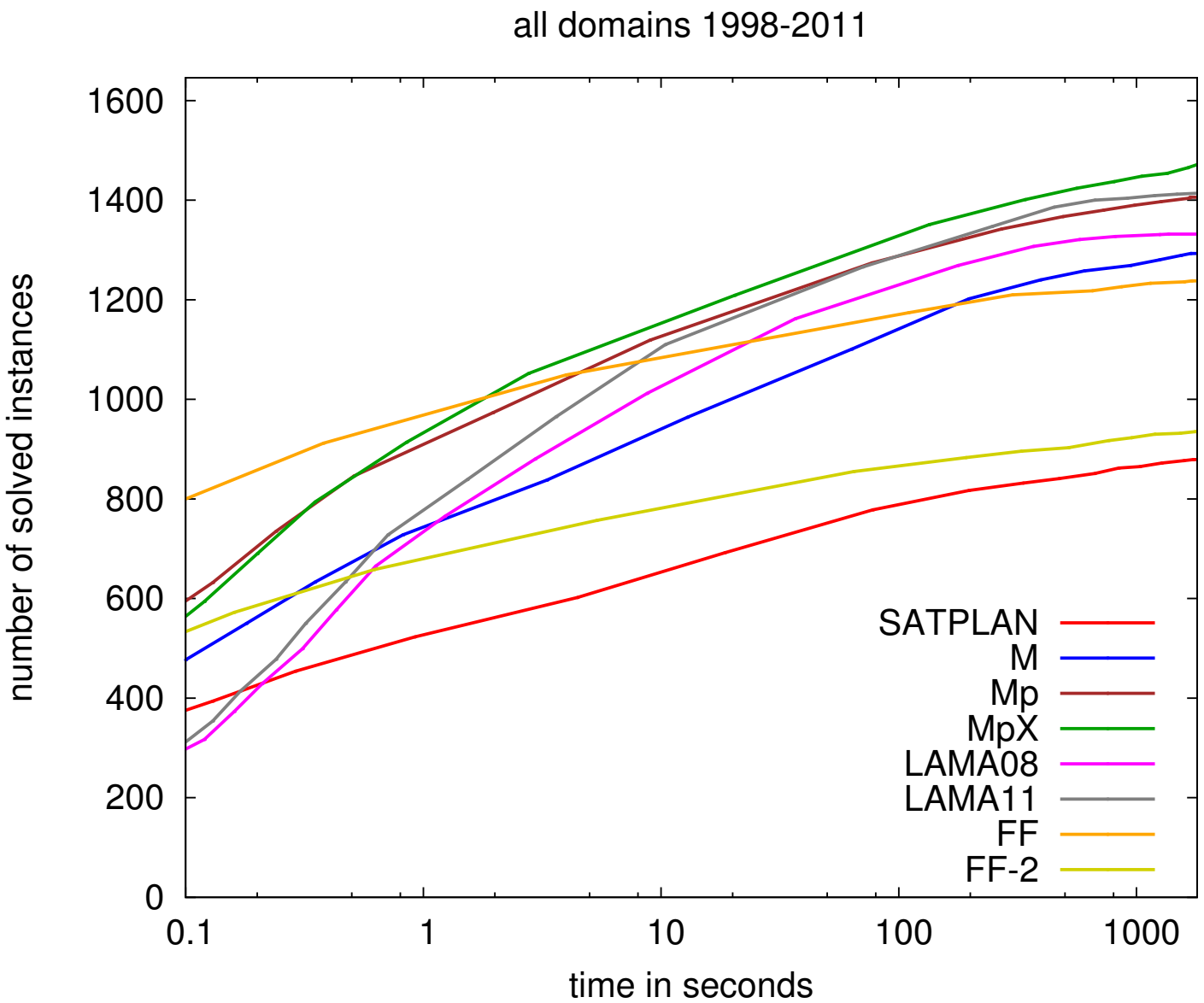


# Impact on planner performance

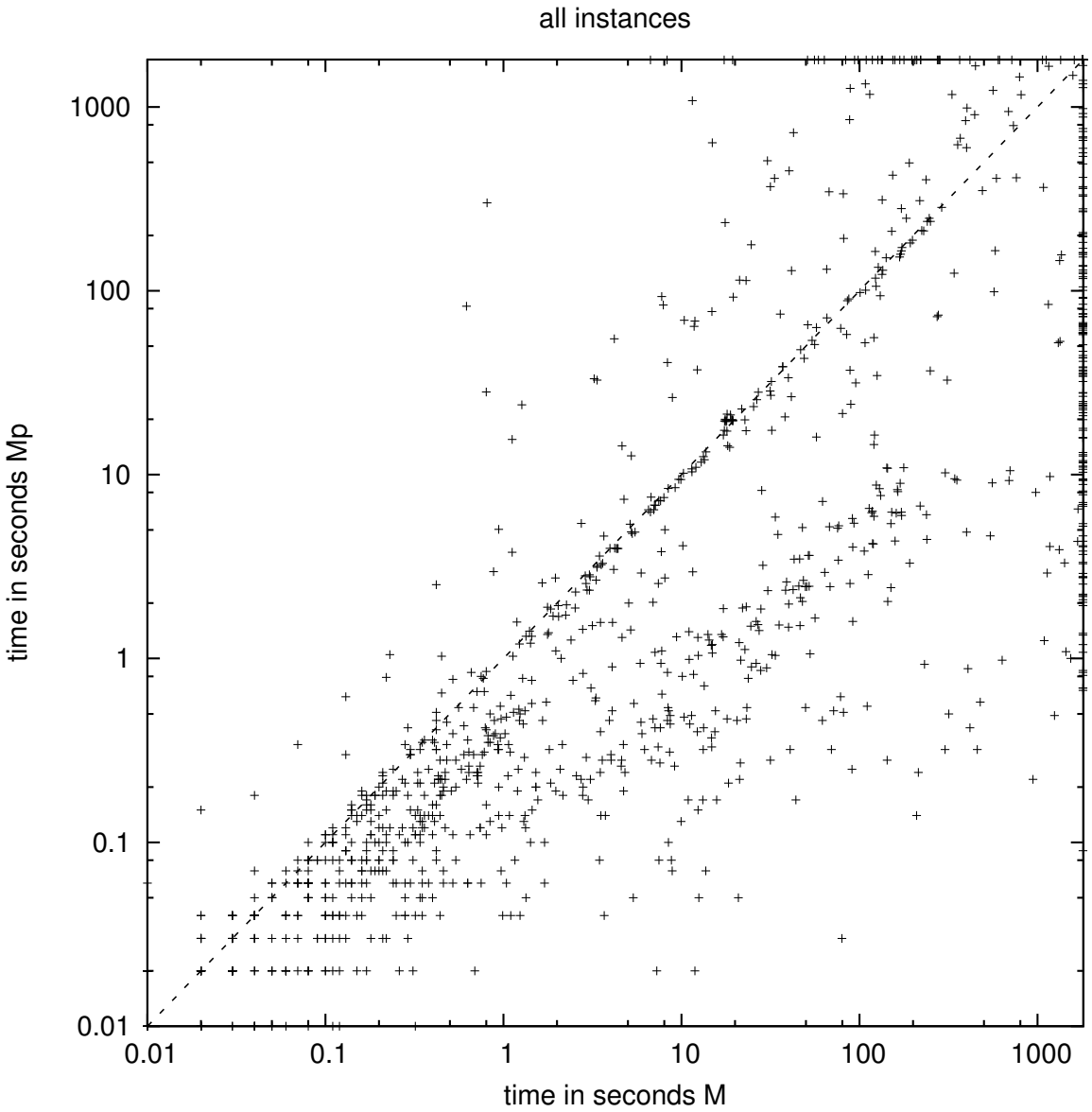
- Outperforms VSIDS with almost all benchmark problems the planning community is using.
- Worse than VSIDS with small, hard, combinatorial problems.
- Ganai [Gan10, Gan11] reports good performance of a different heuristic with partly similar flavor, for BMC.

# 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.

# Other issues with current SAT solvers

- 1-UIP sometimes learns very long clauses, with hundreds of thousands or millions of literals. Does not make sense.
- Compact clause set representations possible: parameterize recurring 2-literal clauses w.r.t. time [Rin12a].
- General-purpose SAT preprocessing not well suited to very large SAT instances representing planning and other reachability problems:
  - SAT instances extremely large: millions of variables, tens of millions of clauses
  - Structure of $I@0 \wedge \bigwedge_{t=0}^{T-1} \mathcal{R}@t \wedge G@T$ completely ignored; specialized preprocessors can do better.
  - Should reuse preprocessing results for $\Phi_t$ with $\Phi_{t+1}$.
- Lots of possibilities to engineer better solvers!

# More general problems

| problem | complexity | reducible to |
|---|---|---|
| classical/deterministic planning | NP | SAT [KS92] |
| stochastic planning | PSPACE | SSAT [ML99] |
| nondeterministic planning | PSPACE | QBF |
| ND planning w/o observability | PSPACE | QBF [Rin99, Rin07] |
| temporal planning | NP | SMT($\mathbb{N}, \mathbb{Q}$) |

(Complexities given for polynomial execution lengths. For unrestricted lengths they are much higher: EXP-complete, EXPSPACE-complete, unsolvable [Lit97, Rin04a, MHC03].)

# Planning with QBF and SSAT
[ML99]



$$\overbrace{\exists a_1@1 a_2@1 a_3@1}^{agent@1}\ \overbrace{\forall r_1@1 r_2@1}^{randomness@1}\ \overbrace{\exists a_1@2 a_2@2 a_3@2}^{agent@2}\ \overbrace{\forall r_1@2 r_2@2}^{randomness@2}\ \cdots \Phi$$

- Nondeterministic/stochastic randomness, full observability
- With SSAT, replace $\forall$ by the stochastic quantifier $\mathcal{R}$.
- Plans are implicit in the AND-OR tree constructed by QBF/SSAT solver.

# Encoding of Nondeterminism
## Example



$a_1 = \langle \neg x, \overbrace{(y|w)}^{p} \wedge \overbrace{(x|z)}^{q} \rangle$ and $a_2 = \langle \neg y, \overbrace{(\overbrace{(y|z)}^{s}|x)}^{r} \rangle$ translate to

$$x@0 \rightarrow x@1 \quad (\neg x@0 \wedge x@1) \rightarrow (a_1@0 \wedge q@0) \vee (a_2@0 \wedge \neg r@0)$$
$$y@0 \rightarrow y@1 \quad (\neg y@0 \wedge y@1) \rightarrow (a_1@0 \wedge p@0) \vee (a_2@0 \wedge r@0 \wedge s@0))$$
$$z@0 \rightarrow z@1 \quad (\neg z@0 \wedge z@1) \rightarrow (a_1@0 \wedge \neg q@0) \vee (a_2@0 \wedge r@0 \wedge \neg s@0)$$
$$w@0 \rightarrow w@1 \quad (\neg w@0 \wedge w@1) \rightarrow (a_1@0 \wedge \neg p@0)$$

$$a_1@0 \rightarrow \neg x@0$$
$$(a_1@0 \wedge p@0) \rightarrow y@1 \qquad (a_1@0 \wedge q@0) \rightarrow x@1$$
$$(a_1@0 \wedge \neg p@0) \rightarrow w@1 \qquad (a_1@0 \wedge \neg q@0) \rightarrow z@1$$
$$a_2@0 \rightarrow \neg y@0$$
$$(a_2@0 \wedge r@0 \wedge s@0) \rightarrow y@1 \qquad (a_2@0 \wedge \neg r@0) \rightarrow x@1$$
$$(a_2@0 \wedge r@0 \wedge \neg s@0) \rightarrow z@1$$

# Planning without Observations and Branching
[Rin07, Rin99]



- One action sequence to reach the goals no matter what.
- equivalent to: reset / synchronization sequences
- $\Sigma_2^p$-complete, easily expressible as $\exists\forall\exists$-QBF

$$\exists \overbrace{a_1@0a_2@0a_3@0}^{actions@0} \cdots \overbrace{a_1@(T-1)a_2@(T-1)a_3@(T-1)}^{actions@T-1}$$
$$\forall \overbrace{r_1@0r_2@0r_1@1r_2@1 \cdots r_1@Tr_2@T}^{randomness}$$
$$\exists \overbrace{x_1@0x_2@0x_1@1x_2@1 \cdots x_1@Tx_2@T}^{state\ sequence}$$
$$\Phi$$

(For $\forall\exists$-QBF encodings see [Rin07].)

# Solvers for QBF and SSAT



- Important research area
- For many types of problems, current QBF and SSAT solvers not very competitive compared to specialized planning algorithms.
- Standard QBF-CDCL leads to blind generate-and-test, which is often not very effective.

# Conclusion

- Close correspondence between the space of SAT + extensions and the space of planning problems.
- SAT (including SMT) an established framework, very much competitive with (and sometimes much stronger than) other state-space search methods.
- Generalizations such as QBF and SSAT much less so.

# 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.

Roberto J. Bayardo, Jr. and Robert C. Schrag.
Using CSP look-back techniques to solve real-world SAT instances.
In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97) and 9th Innovative Applications of Artificial Intelligence Conference (IAAI-97)*, pages 203–208, 1997.

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

# References II


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.

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.

Carla P. Gomes, Bart Selman, and Henry Kautz.
Boosting combinatorial search through randomization.
In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97) and 9th Innovative Applications of Artificial Intelligence Conference (IAAI-97)*, pages 431–437. AAAI Press, 1998.

# References III


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.

Henry Kautz and Bart Selman.

Unifying SAT-based and graph-based planning.

In Thomas Dean, editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 318–325. Morgan Kaufmann Publishers, 1999.

Chu Min Li and Anbulagan.

Heuristics based on unit propagation for satisfiability problems.

In Martha Pollack, editor, *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 366–371. Morgan Kaufmann Publishers, August 1997.

Michael L. Littman.

Probabilistic propositional planning: Representations and complexity.

In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97) and 9th Innovative Applications of Artificial Intelligence Conference (IAAI-97)*, pages 748–754. AAAI Press, 1997.

# References IV




Omid Madani, Steve Hanks, and Anne Condon.
On the undecidability of probabilistic planning and related stochastic optimization problems.
*Artificial Intelligence*, 147(1–2):5–34, 2003.


Stephen M. Majercik and Michael L. Littman.
Contingent planning under uncertainty via probabilistic satisfiability.
In *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI-99) and the 11th Conference on Innovative Applications of Artificial Intelligence (IAAI-99)*, pages 549–556. AAAI Press, 1999.


Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik.
Chaff: engineering an efficient SAT solver.
In *Proceedings of the 38th ACM/IEEE Design Automation Conference (DAC'01)*, pages 530–535. ACM Press, 2001.


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, 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.

# References V




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


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.


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.
Asymptotically optimal encodings of conformant planning in QBF.
In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI-07)*, pages 1045–1050. AAAI Press, 2007.

# References VI


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.

Jussi Rintanen.

Engineering efficient planners with SAT.
In *ECAI 2012. Proceedings of the 20th European Conference on Artificial Intelligence*. IOS Press, 2012.
to appear.

Jussi Rintanen.

Planning as satisfiability: Heuristics, 2012.
accepted for publication on November 2, 2011.

Silvia Richter and Matthias Westphal.

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

# References VII




Bart Selman, Henry A. Kautz, and Bram Cohen.
Noise strategies for improving local search.
In *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI-2004) and the 16th Conference on Innovative Applications of Artificial Intelligence (IAAI-2004)*, pages 337–343. AAAI Press, 1994.


Bart Selman, Henry A. Kautz, and Bram Cohen.
Local search strategies for satisfiability testing.
*DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 25:521–531, 1996.


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 Mark Boddy, Maria Fox, and Sylvie Thiébaux, editors, *ICAPS 2007. Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling*, pages 312–319, 2007.

# References VIII



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.