

Principles of AI Planning

14. Strong nondeterministic planning

Bernhard Nebel and Robert Mattmüller

Albert-Ludwigs-Universität Freiburg

January 24th, 2012

Principles of AI Planning

January 24th, 2012 — 14. Strong nondeterministic planning

1 Concepts

2 Algorithms

3 Summary

Strong planning

We will consider the simplest case of nondeterministic planning by restricting attention to **strong plans**.

1 Concepts

- Strong plans
- Images
- Weak preimages
- Strong preimages

Strong plans

Recall the definition of strong plans:

Definition (strong plan)

Let S be the set of states of a planning task Π . Then a **strong plan** for Π is a function $\pi : S_\pi \rightarrow O$ for some subset $S_\pi \subseteq S$ such that

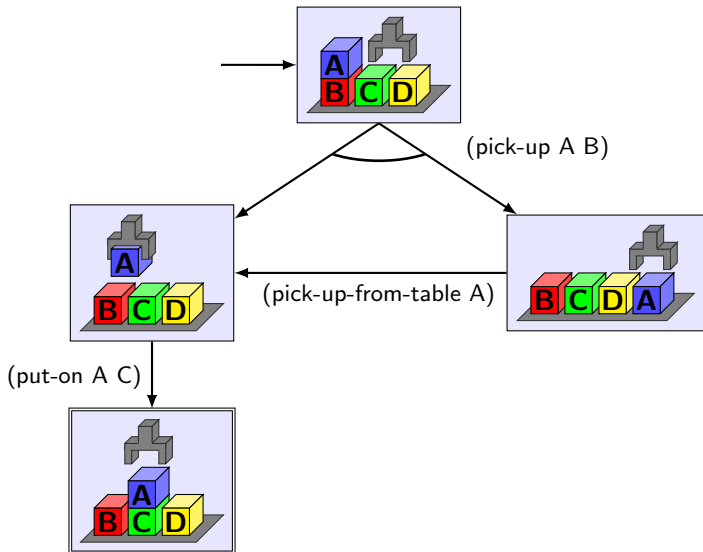
- ▶ $\pi(s)$ is applicable in s for all $s \in S_\pi$,
- ▶ $S_\pi(s_0) \subseteq S_\pi \cup S_\star$ (π is closed),
- ▶ $S_\pi(s') \cap S_\star \neq \emptyset$ for all $s' \in S_\pi(s_0)$ (π is proper), and
- ▶ there is no state $s' \in S_\pi(s_0)$ such that s' is reachable from s' following π in a strictly positive number of steps (π is acyclic).

Strong plans

Execution of a strong plan

1. Determine the current state s .
2. If s is a goal state then terminate.
3. Execute action $\pi(s)$.
4. Repeat from first step.

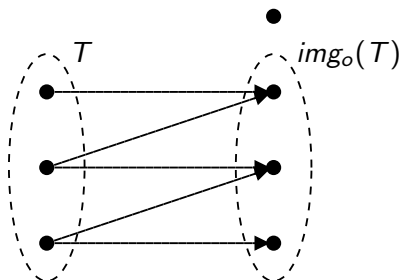
Strong plans



Images

Image

The **image** of a set T of states with respect to an operator o is the set of those states that can be reached by executing o in a state in T .



Images

Definition (image of a state)

$$img_o(s) = \{s' \in S \mid s \xrightarrow{o} s'\} = app_o(s)$$

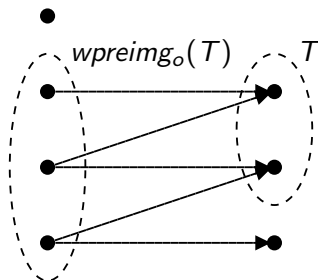
Definition (image of a set of states)

$$img_o(T) = \bigcup_{s \in T} img_o(s)$$

Weak preimages

Weak preimage

The **weak preimage** of a set T of states with respect to an operator o is the set of those states from which a state in T can be reached by executing o .



Weak preimages

Definition (weak preimage of a state)

$$wpreimg_o(s') = \{s \in S \mid s \xrightarrow{o} s'\}$$

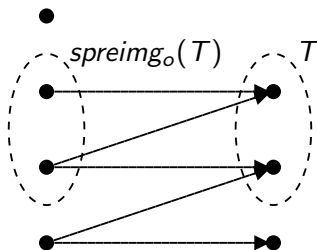
Definition (weak preimage of a set of states)

$$wpreimg_o(T) = \bigcup_{s \in T} wpreimg_o(s).$$

Strong preimages

Strong preimage

The **strong preimage** of a set T of states with respect to an operator o is the set of those states from which a state in T is always reached when executing o .



Strong preimages

Definition (strong preimage of a set of states)

$$\text{spreimg}_o(T) = \{s \in S \mid \exists s' \in T : s \xrightarrow{o} s' \wedge \text{img}_o(s) \subseteq T\}$$

2 Algorithms

- Regression
- Efficient implementation of regression
- Progression

Algorithms for strong planning

1. Dynamic programming (backward)

Compute operator/distance/value for a state based on the operators/distances/values of its all successor states.

1.1 Zero actions needed for goal states.

1.2 If states with i actions to goals are known, states with $\leq i + 1$ actions to goals can be easily identified.

Automatic reuse of plan suffixes already found.

2. Heuristic search (forward)

Strong planning can be viewed as AND/OR graph search.

OR nodes: Choice between operators

AND nodes: Nondeterministically reached state

Heuristic AND/OR search algorithms:

AO*, Proof Number Search, ...

Dynamic programming

Planning by dynamic programming

If for all successors of state s with respect to operator o a plan exists, assign operator o to s .

Base case $i = 0$: In goal states there is nothing to do.

Inductive case $i \geq 1$: If there is $o \in O$ such that for all $s' \in \text{img}_o(s)$, the state s' is a goal state or $\pi(s')$ was assigned in an earlier iteration, then assign $\pi(s) = o$, unless $\pi(s)$ was already assigned earlier.

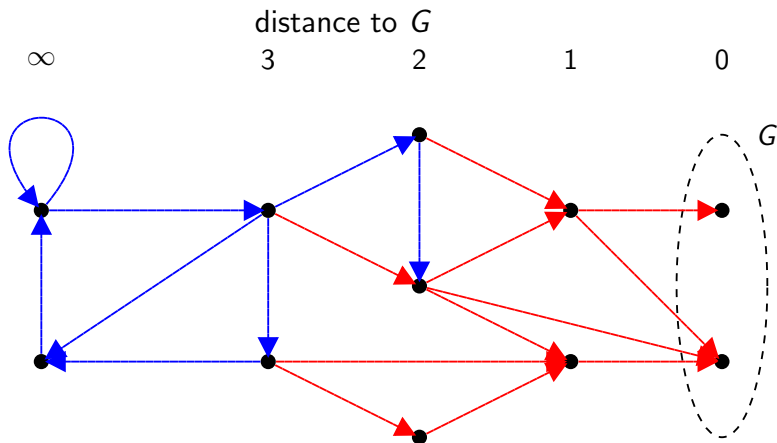
Backward distances

If s is assigned a value on iteration $i \geq 1$, then the **backward distance** of s is i .

The dynamic programming algorithm essentially computes the **backward distances** of states.

Backward distances

Example



Backward distances

Definition (backward distance sets)

Let G be a set of states and O a set of operators.

The **backward distance sets** D_i^{bwd} for G and O consist of those states for which there is a guarantee of reaching a state in G with at most i operator applications using operators in O :

$$D_0^{bwd} := G$$

$$D_i^{bwd} := D_{i-1}^{bwd} \cup \bigcup_{o \in O} \text{preimg}_o(D_{i-1}^{bwd}) \text{ for all } i \geq 1$$

Backward distances

Definition (backward distance)

Let G be a set of states and O a set of operators, and let $D_0^{bwd}, D_1^{bwd}, \dots$ be the backward distance sets for G and O . Then the **backward distance** of a state s for G and O is

$$\delta_G^{bwd}(s) = \min\{i \in \mathbb{N} \mid s \in D_i^{bwd}\}$$

(where $\min \emptyset = \infty$).

Strong plans based on distances

Let $\Pi = \langle V, I, O, \gamma \rangle$ be a nondeterministic planning task with state set S and goal states S_\star .

Extraction of a strong plan from distance sets

1. Let $S' \subseteq S$ be those states having a finite backward distance for $G = S_\star$ and O .
2. Let $s \in S'$ be a state with distance $i = \delta_G^{bwd}(s) \geq 1$.
3. Assign to $\pi(s)$ any operator $o \in O$ such that $img_o(s) \subseteq D_{i-1}^{bwd}$. Hence o decreases the backward distance by at least one.

Then π is a strong plan for \mathcal{T} iff $I \in S'$.

Question: What is the **worst-case** runtime of the algorithm?

Question: What is the **best-case** runtime of the algorithm if most states have a finite backward distance?

Making the algorithm a logic-based algorithm

- ▶ An algorithm that represents the states **explicitly** stops being feasible at about 10^8 or 10^9 states.
- ▶ For planning with bigger transition systems **structural properties** of the transition system have to be taken advantage of.
- ▶ As before, representing state sets as **propositional formulae** (or **BDDs**) often allows taking advantage of the structural properties: a formula (or BDD) that represents a set of states or a transition relation that has certain regularities may be very small in comparison to the set or relation.
- ▶ In the following, we will present an algorithm using a boolean-formula representation (without going into the details of how to implement it using BDDs).

Making the algorithm a logic-based algorithm

Remark: The following algorithm assumes a propositional representation of the state space as opposed to a finite-domain representation. We have already seen how to translate an FDR encoding into a propositional encoding in Chapter 9 (cf. definition of the “induced propositional planning task”).

Therefore, for the rest of the present section, we will assume without loss of generality that all $v \in V$ are propositional variables with domain $\mathcal{D}_v = \{0, 1\}$.

Breadth-first search with progression and state sets (deterministic case)

Progression breadth-first search

def bfs-progression(V, I, O, γ):

goal := formula-to-set(γ)

reached := $\{I\}$

loop:

if $reached \cap goal \neq \emptyset$:

return solution found

new-reached := $reached \cup \bigcup_{o \in O} img_o(reached)$

if $new-reached = reached$:

return no solution exists

reached := *new-reached*

\rightsquigarrow This can easily be transformed into a **regression** algorithm.

Breadth-first search with regression and state sets (deterministic case)

Regression breadth-first search

def bfs-regression(V, I, O, γ):

init := I

reached := *formula-to-set*(γ)

loop:

if *init* \in *reached*:

return solution found

new-reached := *reached* \cup $\bigcup_{o \in O} \text{wpreimg}_o(\text{reached})$

if *new-reached* = *reached*:

return no solution exists

reached := *new-reached*

- ▶ This algorithm is very similar to the dynamic programming algorithm for the nondeterministic case!

Breadth-first search with regression and state sets (nondeterministic case)

Regression breadth-first search

def bfs-regression(V, I, O, γ):

init := I

reached := *formula-to-set*(γ)

loop:

if *init* \in *reached*:

return solution found

new-reached := *reached* $\cup \bigcup_{o \in O} spreimg_o(\textit{reached})$

if *new-reached* = *reached*:

return no solution exists

reached := *new-reached*

- ▶ How do we define *spreimg* with logic (or BDD) operations?

Transition formula for nondeterministic operators

Let V be the set of state variables and $V' := \{v' \mid v \in V\}$ a set of primed copies of the variables in V . Intuition:

- ▶ Variables in V describe the **current state** s .
- ▶ Variables in V' describe the **next state** s' .

We would like to define a formula $\tau_V(o)$ that describes the transitions labeled with o between states s (over V) and s' (over V') in terms of V and V' .

Transition formula for nondeterministic operators

The formula $\tau_V(o)$ must express

- ▶ the conditions for **applicability** of o ,
- ▶ how o **changes** state variables, and
- ▶ which state variables o **does not change**.

A significant difficulty lies in the third requirement because **different variables** may be affected depending on nondeterministic choices.

Transition formula for nondeterministic operators

$\tau_V(o)$ for deterministic operators $o = \langle \chi, e \rangle$

$$\begin{aligned} \tau_V(o) = & \chi \wedge \bigwedge_{v \in V} ((EPC_v(e) \vee (v \wedge \neg EPC_{\neg v}(e))) \leftrightarrow v') \\ & \wedge \bigwedge_{v \in V} \neg(EPC_v(e) \wedge EPC_{\neg v}(e)) \end{aligned}$$

Assume that $e = \bigwedge_{a \in A} a \wedge \bigwedge_{d \in D} \neg d$ for $A = \{a_1, \dots, a_k\}$ and $D = \{d_1, \dots, d_l\}$ with $A \cap D = \emptyset$. Then this becomes simpler.

$\tau_V(o)$ for STRIPS operators $o = \langle \chi, \bigwedge_{a \in A} a \wedge \bigwedge_{d \in D} \neg d \rangle$

$$\tau_V(o) = \chi \wedge \bigwedge_{a \in A} a' \wedge \bigwedge_{d \in D} \neg d' \wedge \bigwedge_{v \in V \setminus (A \cup D)} (v \leftrightarrow v')$$

Transition formula for nondeterministic operators

For nondeterministic operators $o = \langle \chi, \{e_1, \dots, e_n\} \rangle$ with corresponding add and delete lists A_i and D_i of e_i such that $A_i \cap D_i = \emptyset$, $i = 1, \dots, n$, we get:

$\tau_V(o)$ for nondeterministic operators $o = \langle \chi, \{e_1, \dots, e_n\} \rangle$

$$\tau_V(o) = \chi \wedge \bigvee_{i=1}^n \left(\bigwedge_{a \in A_i} a' \wedge \bigwedge_{d \in D_i} \neg d' \wedge \bigwedge_{v \in V \setminus (A_i \cup D_i)} (v \leftrightarrow v') \right)$$

Example

Let $V = \{a, b\}$, $V' = \{a', b'\}$, and $o = \langle \neg a, \{a, a \wedge \neg b\} \rangle$. Then

$$\tau_V(o) = \neg a \wedge \left((a' \wedge (b \leftrightarrow b')) \vee (a' \wedge \neg b') \right).$$

Computing strong preimages

Definition (substitution)

Let φ, t_1, \dots, t_n be propositional formulas and v_1, \dots, v_n atomic propositions.

We denote the formula obtained from φ by simultaneous replacement of all variables v_i by the corresponding formulas t_i , $i = 1, \dots, n$, by $\varphi[t_1, \dots, t_n/v_1, \dots, v_n]$.

Computing strong preimages

Definition (existential abstraction)

Let φ be a propositional formula and v_1, \dots, v_n be atomic propositions. Then the **existential abstraction** of φ wrt. v_1, \dots, v_n is recursively defined as follows:

$$\begin{aligned}\exists v. \varphi &:= \varphi[\top/v] \vee \varphi[\perp/v] \\ \exists v_1 \dots \exists v_n. \varphi &:= \exists v_1 \dots \exists v_{n-1}. (\varphi[\top/v_n] \vee \varphi[\perp/v_n])\end{aligned}$$

For a set of variables $V = \{v_1, \dots, v_n\}$ we use the abbreviation

$$\exists V. \varphi := \exists v_1 \dots \exists v_n. \varphi.$$

Note: Even with intermediate formula simplifications this can lead to an exponential blowup. BDDs can be useful here.

Computing strong preimages

Strong preimages

$$\begin{aligned}
 \text{spreimg}_o(T) &= \{s \in S \mid \exists s' \in T : s \xrightarrow{o} s' \wedge \text{img}_o(s) \subseteq T\} \\
 &= \{s \in S \mid (\exists s' \in S : s \xrightarrow{o} s' \wedge s' \in T) \wedge \\
 &\quad \{s' \in S \mid s \xrightarrow{o} s'\} \subseteq T\} \\
 &= \{s \in S \mid (\exists s' \in S : s \xrightarrow{o} s' \wedge s' \in T) \wedge \\
 &\quad (\forall s' \in S : s \xrightarrow{o} s' \Rightarrow (s' \in T))\} \\
 &= \{s \in S \mid (\exists s' \in S : s \xrightarrow{o} s' \wedge s' \in T) \wedge \\
 &\quad (\neg \exists s' \in S : s \xrightarrow{o} s' \wedge s' \notin T)\}
 \end{aligned}$$

Computing strong preimages with boolean function operations

$$\text{spreimg}_o(T) = \{s \in S \mid (\exists s' \in S : s \xrightarrow{o} s' \wedge s' \in T) \wedge (\neg \exists s' \in S : s \xrightarrow{o} s' \wedge s' \notin T)\}$$

Strong preimages with boolean functions

For formula φ characterizing set T of strongly backward-reached states:

$$\text{spreimg}_o(\varphi) = (\exists V'. (\tau_V(o) \wedge \varphi[v'_1, \dots, v'_n / v_1, \dots, v_n])) \wedge (\neg \exists V'. (\tau_V(o) \wedge \neg \varphi[v'_1, \dots, v'_n / v_1, \dots, v_n]))$$

We can use this regression formula for efficient **symbolic** regression search. BDDs support all necessary operations (atomic propositions, \neg , \wedge , \vee , substitution, \exists , ...).

Computing strong preimages with boolean function operations

Example

Let $V = \{a, b\}$, $V' = \{a', b'\}$, and

$$o = \langle \neg a, \{a, a \wedge \neg b\} \rangle, \quad \text{i.e.,}$$

$$\tau_V(o) = \neg a \wedge \left((a' \wedge (b \leftrightarrow b')) \vee (a' \wedge \neg b') \right).$$

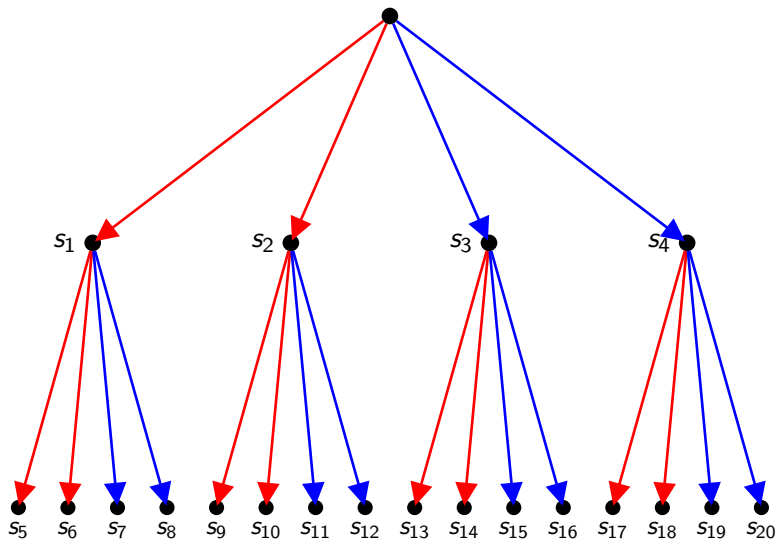
Moreover, let $\varphi = a$. Then

$$\begin{aligned} \text{spreimg}_o(\varphi) &= \exists a' \exists b'. \left(\neg a \wedge \left((a' \wedge (b \leftrightarrow b')) \vee (a' \wedge \neg b') \right) \wedge a' \right) \wedge \\ &\quad \neg \exists a' \exists b'. \left(\neg a \wedge \left((a' \wedge (b \leftrightarrow b')) \vee (a' \wedge \neg b') \right) \wedge \neg a' \right) \\ &\equiv \neg a \end{aligned}$$

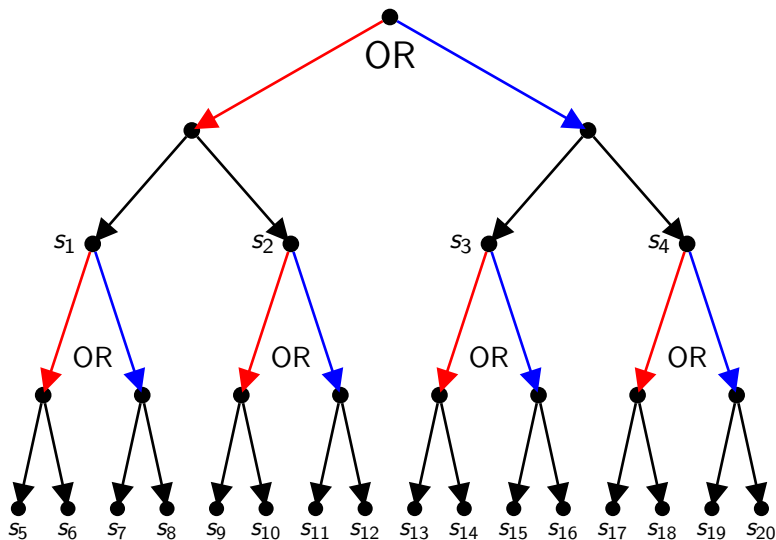
Progression Search

- ▶ We saw a generalization of **regression search** to strong planning.
- ▶ However, this search is **uninformed** (breadth-first search).
- ▶ Is there an **analogue to A* search** for strong planning?
- ▶ Yes: **AO* search**
 - ▶ **Progression** search (like A*)
 - ▶ Guided by a **heuristic** (like A*)
 - ▶ Guaranteed **optimality** (under certain conditions, like A*)

AND/OR search



AND/OR search



Progression Search

- ▶ We describe AO* on a graph representation **without intermediate nodes**, i.e., as in the first figure.
- ▶ There are different variants of AO*, depending on whether the graph that is being searched is an AND/OR **tree**, an AND/OR **DAG**, or a general, possibly **cyclic**, AND/OR graph.
- ▶ The graphs we want to search, $\mathcal{T}(\Pi)$, are in general cyclic.
- ▶ However, AO* becomes a bit more involved when dealing with cycles, so we only discuss AO* under the assumption of acyclicity and leave the generalization to cyclic state spaces as an exercise.

AO* Search

- ▶ The search is over $\mathcal{T}(\Pi)$.
- ▶ For ease of presentation, we do not distinguish between states of $\mathcal{T}(\Pi)$ and search nodes.
- ▶ Also, for ease of presentation, we do not handle the case that no strong plan exists.

AO* Search

Definition (solution graph)

A **solution graph** for a nondeterministic transition system

$\mathcal{T} = \langle S, L, T, s_0, S_\star \rangle$ is an acyclic subgraph of \mathcal{T} (viewed as a graph),

$\mathcal{T}' = \langle S', L, T' \rangle$, such that

- ▶ $s_0 \in S'$,
- ▶ for each $s' \in S' \setminus S_\star$, there is exactly one label $l \in L$ s.t.
 - ▶ T' contains at least one outgoing transition from s' labeled with l ,
 - ▶ T' contains all outgoing transitions from s' labeled with l
(and S' contains the states reached via such transitions),
 - ▶ T' contains no outgoing transitions from s' labeled with any $\tilde{l} \neq l$, and
- ▶ every directed path in \mathcal{T}' terminates at a goal state.

AO* Search

Conceptually, there are three graphs/transition systems:

- ▶ The induced transitions system $\mathcal{T} = \mathcal{T}(\Pi)$, which only exists as a mathematical object, but is in general not made explicit completely during AO* search,
- ▶ The current portion of \mathcal{T} explicitly represented by the search algorithm, \mathcal{T}_e , and
- ▶ The current portion of \mathcal{T}_e considered by the algorithm as the cheapest/best current **partial solution graph**, \mathcal{T}_p .

AO* Search

Definition (partial solution graph)

A **partial solution graph** for a nondeterministic transition system $\mathcal{T} = \langle S, L, T, s_0, S_\star \rangle$ is an acyclic subgraph of \mathcal{T} (viewed as a graph), $\mathcal{T}_p = \langle S_p, L, T_p \rangle$, s.t.

- ▶ $s_0 \in S_p$,
- ▶ for each $s' \in S_p$ **that is not an unexpanded leaf node in \mathcal{T}_p** there is exactly one label $l \in L$ such that
 - ▶ T_p contains at least one outgoing transition from s' labeled with l ,
 - ▶ T_p contains all outgoing transitions from s' labeled with l (and S_p contains the states reached via such transitions),
 - ▶ T_p contains no outgoing transitions from s' labeled with any $\tilde{l} \neq l$, and
- ▶ every directed path in \mathcal{T}_p terminates at a **goal state or an unexpanded leaf node in \mathcal{T}_p** .

AO* Search

Definition (cost of a partial solution graph)

Let $h : S \rightarrow \mathbb{N} \cup \{\infty\}$ be a heuristic function for the state space S of \mathcal{T} , and let $\mathcal{T}_p = \langle S_p, L, T_p \rangle$ be a partial solution graph.

The **cost labeling** of \mathcal{T}_p is the solution to the following system of equations over the states S_p of \mathcal{T}_p :

$$f(s) = \begin{cases} 0 & \text{if } s \text{ is a goal state} \\ h(s) & \text{if } s \text{ is an unexpanded non-goal} \\ 1 + \max_{s \xrightarrow{o} s'} f(s') & \text{for the unique outgoing action} \\ & o \text{ of } s \text{ in } \mathcal{T}_p, \text{ otherwise.} \end{cases}$$

The cost of \mathcal{T}_p is the cost labeling of its root.

AO* search keeps track of a **cheapest** partial solution graph by **marking** for each expanded state s an outgoing action o **minimizing** $1 + \max_{s \xrightarrow{o} s'} f(s')$.

AO* Search

Procedure ao-star

def ao-star(\mathcal{T}):

let \mathcal{T}_e initially consist of the initial state s_0 .

while \mathcal{T}_p has unexpanded non-goal node:

expand unexpanded non-goal node s of \mathcal{T}_p

add new successor states to \mathcal{T}_e

for all new states s' added to \mathcal{T}_e :

$$f(s') \leftarrow h(s')$$

$Z \leftarrow s$ and its ancestors in \mathcal{T}_e along marked actions.

while Z is not empty:

remove from Z a state s w/o descendant in Z .

$$f(s) \leftarrow \min_{o \text{ applicable in } s} (1 + \max_{s \xrightarrow{o} s'} f(s')).$$

mark the best outgoing action for s

(this may implicitly change \mathcal{T}_p).

return an optimal solution graph.

AO* Search

Correctness (proof sketch)

- ▶ Solution graphs directly correspond to strong plans.
- ▶ Algorithm eventually terminates (finite number of possible node expansions).
- ▶ Acyclicity guarantees that extraction of \mathcal{T}_ρ and dynamic programming back-propagation of f values always terminates.
- ▶ Marking makes sure that existing solutions are eventually marked.

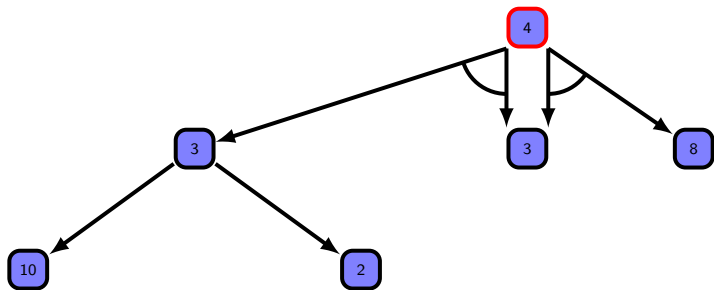
AO* Search

Details

- ▶ Pseudocode omits **bookkeeping of solved states** (can improve performance).
- ▶ **Choice of unexpanded non-goal node of best partial solution graph is unspecified.**
 - ▶ Correctness/optimality not affected.
 - ▶ One possibility: choose node with lowest cost estimate.
 - ▶ Alternative: expand several nodes simultaneously.
- ▶ Algorithm can be extended to deal with **cycles in the AND/OR graph.**

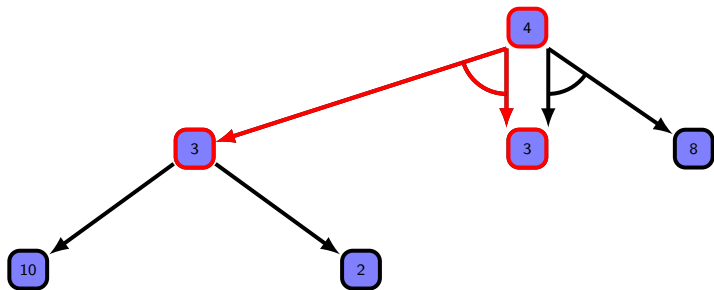
AO* Search

Example



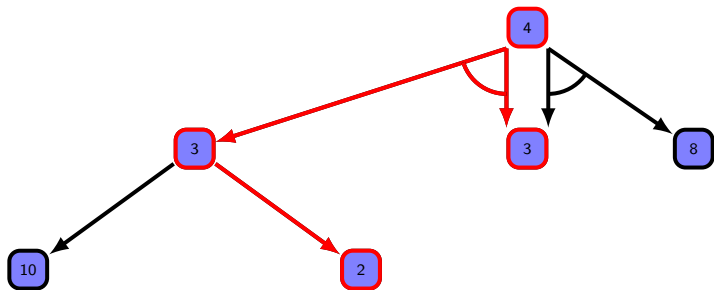
AO* Search

Example



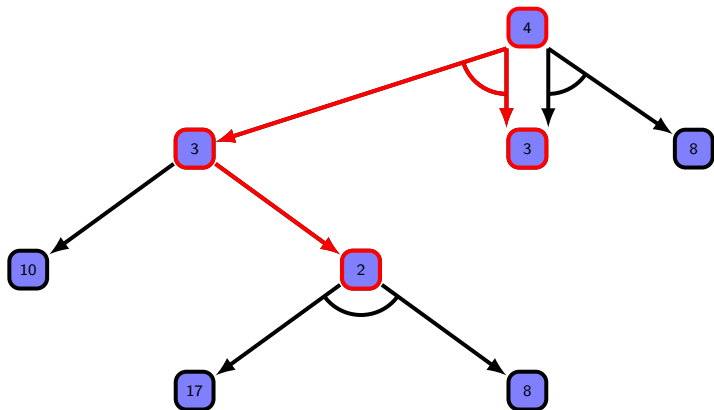
AO* Search

Example



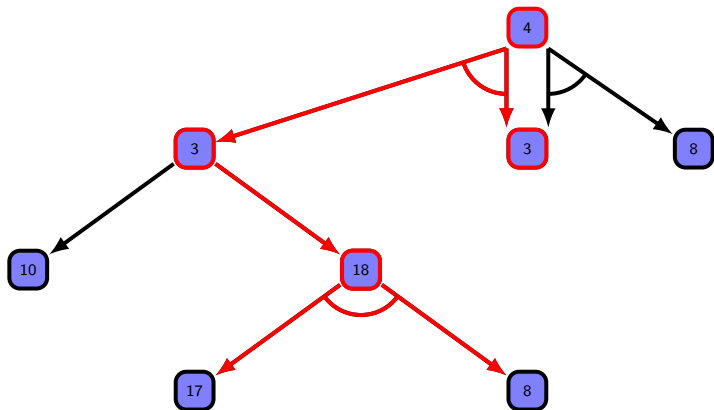
AO* Search

Example



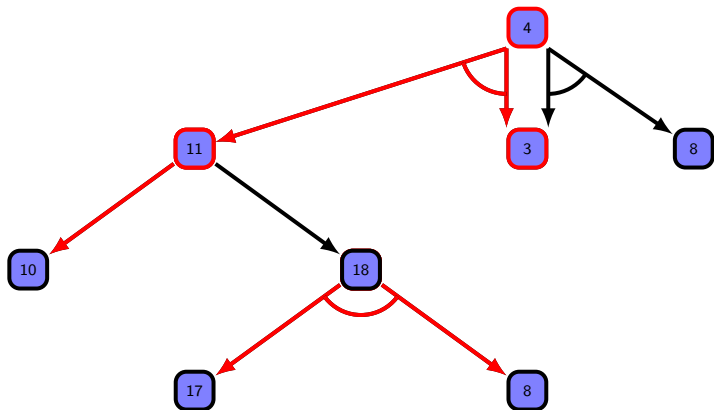
AO* Search

Example



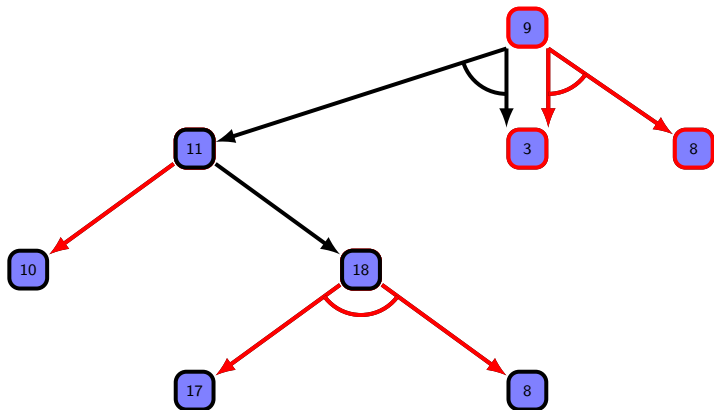
AO* Search

Example



AO* Search

Example



AO* Search

Heuristic Evaluation Function

- ▶ Desirable: **informative, domain-independent heuristic** to initialize cost estimates.
- ▶ Heuristic should **estimate (strong) goal distances**.
- ▶ Heuristic does **not necessarily** have to be **admissible** (unless we seek optimal solutions).
- ▶ We can adapt many heuristics we already know from classical planning (details omitted).

3 Summary

Summary

- ▶ We have considered the special case of nondeterministic planning where
 - ▶ planning tasks are **fully observable** and
 - ▶ we are interested in **strong plans**.
- ▶ We have introduced important concepts also relevant to other variants of nondeterministic planning such as
 - ▶ **images** and
 - ▶ **weak and strong preimages**.
- ▶ We have discussed some basic classes of algorithms:
 - ▶ **backward induction** by dynamic programming, and
 - ▶ **forward search** in AND/OR graphs.