In this chapter, we will consider the simplest case of nondeterministic planning by restricting attention to strong plans.

**Definition (strong plan)**

Let $S$ be the set of states of a planning task $\Pi$. Then a strong plan for $\Pi$ 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 $s \in S_\pi$,
- $S_\pi(s_0) \subseteq S_\pi \cup S_*$ ($\pi$ is closed),
- $S_\pi(s') \cap S_* \neq \emptyset$ for all $s' \in S_\pi(s_0)$ ($\pi$ is proper), and
- there is no state $s' \in S_\pi(s_0)$ such that $s'$ is reachable from $s'$ following $\pi$ in a strictly positive number of steps ($\pi$ 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.

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 \( T \).

\[
img_o(T) = \{ s' \in S | s \xrightarrow{o} s' \} = app_o(s)
\]

\[
img_o(T) = \bigcup_{s \in T} img_o(s)
\]
**Weak preimages**

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 $o$.

**Definition (weak preimage of a state)**

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

**Definition (weak preimage of a set of states)**

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

---

**Strong preimages**

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 $o$.

**Definition (strong preimage of a set of states)**

$$spreimg_o(T) = \{ s \in S | \exists s' \in T : s \xrightarrow{o} s' \land img_o(s) \subseteq T \}$$
Algorithms

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 \( \pi(s) \) is still undefined and 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 \( \pi(s) = o \).

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.

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.
   - Zero actions needed for goal states.
   - 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**: Choice between effects
   - Heuristic AND/OR search algorithms: \( AO^* \), Proof Number Search, . . .
Backward distances

**Definition (backward distance sets)**
Let \( G \) be a set of states and \( O \) a set of operators. The **backward distance sets** \( D_{i}^{\text{bwd}} \) 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 \( O \):

\[
\begin{align*}
D_{0}^{\text{bwd}} & := G \\
D_{i}^{\text{bwd}} & := D_{i-1}^{\text{bwd}} \cup \bigcup_{o \in O} \text{spreimg}_{o}(D_{i-1}^{\text{bwd}}) \quad \text{for all } i \geq 1
\end{align*}
\]

January 29th, 2018 B. Nebel, R. Mattmüller – AI Planning 17 / 58

**Strong plans based on distances**

Let \( \Pi = (V, I, O, \gamma) \) 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 \( i = \delta_{G}^{\text{bwd}}(s) \geq 1 \).
3. Assign to \( \pi(s) \) any operator \( o \in O \) such that \( \text{img}_{o}(s) \subseteq D_{i-1}^{\text{bwd}} \). Hence \( o \) decreases the backward distance by at least one.

Then \( \pi \) 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?

January 29th, 2018 B. Nebel, R. Mattmüller – AI Planning 19 / 58

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).

January 29th, 2018 B. Nebel, R. Mattmüller – AI Planning 20 / 58
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 \( \mathbb{D}_v = \{0, 1\} \).

This can easily be transformed into a regression algorithm.

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

Progression breadth-first search

\[
\text{def } \text{bfs-progression}(V, I, O, \gamma):
\]
\[
\begin{align*}
goal & := \text{formula-to-set}(\gamma) \\
reached & := \{I\} \\
\text{loop:} & \quad \text{if } reached \cap goal \neq \emptyset: \\
& \quad \quad \text{return solution found} \\
& \quad \quad \text{new-reached} := reached \cup \bigcup_{o \in O} \text{img}_o(reached) \\
& \quad \quad \text{if } new-reached = reached: \\
& \quad \quad \quad \text{return no solution exists} \\
& \quad \quad reched := new-reached \\
\end{align*}
\]

Remark: Do you recognize the assignments \( D_{0}^{\text{bwd}} := G \) and \( D_{i}^{\text{bwd}} := D_{i-1}^{\text{bwd}} \cup \bigcup_{o \in O} \text{spreimg}_o(D_{i-1}^{\text{bwd}}) \) for \( i \geq 1 \)?

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

Regression breadth-first search

\[
\text{def } \text{bfs-regression}(V, I, O, \gamma):
\]
\[
\begin{align*}
\text{init} & := I \\
\reached & := \text{formula-to-set}(\gamma) \\
\text{loop:} & \quad \text{if } \text{init} \in \text{reached}: \\
& \quad \quad \text{return solution found} \\
& \quad \quad \text{new-reached} := \text{reached} \cup \bigcup_{o \in O} \text{wpreimg}_o(\text{reached}) \\
& \quad \quad \text{if } \text{new-reached} = \text{reached}: \\
& \quad \quad \quad \text{return no solution exists} \\
& \quad \quad \text{reached} := \text{new-reached} \\
\end{align*}
\]

This algorithm is very similar to the dynamic programming algorithm for the nondeterministic case!
Regression breadth-first search

**def** bfs-regression($V$, $I$, $O$, $\gamma$):

- **init** := $I$
- **reached** := $\gamma$

**loop**:
- if **init** $\equiv$ **reached**:
  - return solution found
- **new-reached** := **reached** $\vee \bigvee_{o \in O} spreimgsymb_o(reached)$
- if **new-reached** $\equiv$ **reached**:
  - return no solution exists
- **reached** := **new-reached**

How do we define $spreimgsymb$ with logic (or BDD) operations?

Symbolic strong preimage computation

Let $\varphi$ be a logic formula and $[[\varphi]] = \{s \in S | s \models \varphi\}$.

**We want**: a symbolic preimage operation $spreimgsymb$ such that if $\psi = spreimgsymb_o(\varphi)$, then $[[\psi]] = \{s \in S | s \models \psi\} = spreimg_o([[\varphi]])$.

In other words, we want the following diagram to commute:

```
Ψ ← spreimgsymb_o(·) → Φ
       \downarrow
       \downarrow
[[Ψ]] ← spreimg_o(·) → [[Φ]]
```

Transition formula for nondeterministic operators

Let $V$ be the set of state variables and $V' := \{v' | v \in V\}$ 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

Let $\phi$ be a propositional formula and $V_1, \ldots, V_n$ be atomic propositions. Then the existential abstraction of $\phi$ wrt. $V_1, \ldots, V_n$ is recursively defined as follows:

$$\exists V. \phi := \phi[T/V] \lor \phi[\bot/V]$$

For a set of variables $V = \{V_1, \ldots, V_n\}$ we use the abbreviation

$$\exists V. \phi := \exists V_1 \ldots \exists V_n. \phi$$

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

**Strong preimages**

Let $V$ be a set of propositions, and $\mathcal{O}$ be a set of operations. Then, for a set $T$ of propositions,

$$\text{spreimg}_\mathcal{O}(T) = \{ s \in S \mid \exists s' \in T : s \xrightarrow{\mathcal{O}} s' \land \text{img}_\mathcal{O}(s) \subseteq T \}$$

Moreover, let $\mathcal{O}$ be a set of boolean functions. Then,

$$\text{spreimg}_\mathcal{O}(T) = \{ s \in S \mid \exists s' \in S : s \xrightarrow{\mathcal{O}} s' \land s' \in T \} \land$$

$$\{ s' \in S : s \xrightarrow{\mathcal{O}} s' \land s' \in T \} \land$$

$$\{ s \in S : s \xrightarrow{\mathcal{O}} T \} \land$$

$$\{ s \in S : s \xrightarrow{\mathcal{O}} (s' \land s' \in T) \}$$

Computing strong preimages with boolean function operations

Let $\mathcal{O}$ be a set of boolean functions. Then,

$$\text{spreimg}_\mathcal{O}(T) = \{ s \in S \mid (\exists s' \in S : s \xrightarrow{\mathcal{O}} s' \land s' \in T) \land$$

$$\neg \exists s' \in S : s \xrightarrow{\mathcal{O}} s' \land (s' \in T) \}$$

**Strong preimages with boolean functions**

For formula $\varphi$ characterizing set $T$ of strongly backward-reached states:

$$\text{spreimgsymb}_\mathcal{O}(\varphi) = (\exists V'.(\tau_V(o) \land \varphi[v_1, \ldots, v_n / v_1, \ldots, v_n]) \land$$

$$\neg \exists V'.(\tau_V(o) \land \neg \varphi[v_1, \ldots, v_n / v_1, \ldots, v_n]))$$

We can use this regression formula for efficient symbolic regression search. BDDs support all necessary operations (atomic propositions, $\neg$, $\land$, $\lor$, substitution, $\exists$, $\ldots$).

**Example**

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

$${\mathcal{O}} = \langle \lnot a, \{a, a \land \lnot b\} \rangle$$

i.e.,

$$\tau_V(o) = \lnot a \land \left( (a' \land (b \leftrightarrow b')) \lor (a' \land \lnot b') \right).$$

Moreover, let $\varphi = a$. Then

$$\text{spreimgsymb}_\mathcal{O}(\varphi) = \exists a' \exists b'. \left( \lnot a \land \left( (a' \land (b \leftrightarrow b')) \lor (a' \land \lnot b') \right) \land \neg a' \land b' \land \left( (a' \land (b \leftrightarrow b')) \lor (a' \land \lnot b') \right) \land \neg a' \right) \equiv \lnot a$$

Computing strong preimages with boolean function operations

**Theorem**

The previous definition of the symbolic preimage operator makes the following diagram commute:
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*)

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 $T(\Pi)$.

For ease of presentation, we do not distinguish between states of $T(\Pi)$ and search nodes.

Also, for ease of presentation, we do not handle the case that no strong plan exists.

AO* Search

Conceptually, there are three graphs/transition systems:

- The induced transitions system $T = T(\Pi)$, which only exists as a mathematical object, but is in general not made explicit completely during AO* search,

- The current portion of $T$ explicitly represented by the search algorithm, $T_e$, and

- The current portion of $T_e$ considered by the algorithm as the cheapest/best current partial solution graph, $T_p$.

AO* Search

Definition (solution graph)

A solution graph for a nondeterministic transition system $T = \langle S, L, T, s_0, S_\ast \rangle$ is an acyclic subgraph of $T$ (viewed as a graph), $T' = \langle S', L, T' \rangle$, such that

- $S_0 \in S'$,

- for each $s' \in S' \setminus S_\ast$, 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 $T'$ terminates at a goal state.

AO* Search

Definition (partial solution graph)

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

- $S_0 \in S_p$,

- for each $s' \in S_p$ that is not an unexpanded leaf node in $T_p$ 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 $T_p$ terminates at a goal state or an unexpanded leaf node in $T_p$. 
**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 \( \mathcal{T}_p = (S_p, L, T_p) \) 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 \( \mathcal{T}_p \):

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

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 \( 1 + \max_{s \rightarrow s'} f(s') \).

**Correctness (proof sketch)**

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

---

**AO* Search**

**Procedure ao-star**

\[
def \text{ao-star}\left(\mathcal{T}\right) := \\
\text{let } \mathcal{T}_e \text{ and } \mathcal{T}_p \text{ initially consist of the initial state } s_0. \\
\text{while } \mathcal{T}_p \text{ has unexpanded non-goal node:} \\
\quad \text{expand an unexpanded non-goal node } s \text{ of } \mathcal{T}_p \\
\quad \text{add new successor states to } \mathcal{T}_e \\
\text{for all } \text{new states } s' \text{ added to } \mathcal{T}_e: \\
\quad f(s') \leftarrow h(s') \\
\quad Z \leftarrow s \text{ and its ancestors in } \mathcal{T}_e \text{ along marked actions.} \\
\text{while } Z \text{ is not empty:} \\
\quad \text{remove from } Z \text{ a state } s \text{ w/o descendant in } Z. \\
\quad f(s) \leftarrow \min_o \text{applicable in } s \left(1 + \max_{s \rightarrow s'} f(s')\right). \\
\quad \text{mark the best outgoing action for } s \text{ (this may implicitly change } \mathcal{T}_p). \\
\text{return } \text{an optimal solution graph.}
\]

**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

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).
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.