Part III

State Space Planning

Outline of Part III

Introduction

What is State Space Planning ?

- The simplest classical planning algorithms.
- Search algorithms in which the search space is a subset of the state space:
  - Each node corresponds to a state of the world.
  - Each arc corresponds to a state transition.
  - The current plan corresponds to the current path in the seach space.

I. Forward Search

Forward Search Principle

- The forward search algorithm is nondeterministic
- The forward search algorithm is sound and complete
- The forward search algorithm takes as input the statement $P = (\mathcal{O}, s_0, g)$ of a planning problem $\mathcal{P}$. If $\mathcal{P}$ is solvable, then Forward-search($\mathcal{O}$, s_0, g) returns a solution plan. Otherwise it returns failure.
- The plan returned by each recursive invocation of the algorithm is called a partial solution because it is part of the final solution returned by the top level invocation.

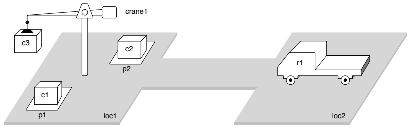
Forward Search Algorithm

Algorithm (ForwardSearch($\mathcal{O}$, s_0, g))

if *s satisfies g* **then return** *an empty plan π*
active $\leftarrow \{a \mid$ *a is a ground instance of an operator $\mathcal{O}$*
and precond(a) is true in s $\}$
if active $= \emptyset$ **then return** Failure
nondeterministically choose an action a_1 $\in$ active
$s_1 \leftarrow \gamma(s, a_1)$
$\pi \leftarrow$ `ForwardSearch`($\mathcal{O}$, s_1, g)
if $\pi \neq$ Failure **then return** $a_1 \cdot \pi$
else return Failure

Forward Search Example

Take the state s defined in figure below:



- $s_0 = \{$ attached(p1, loc1), in(c1, p1), top(c1,p1), on(c1, pallet), attached(p2,loc1), in(c2,p2), top(c2,p2), on(c2,pallet), belong(crane1,loc1), holding(crane1,c3), adjacent(loc1,loc2), adjacent(loc2,loc1), at(r1,loc2), occupied(loc2), unloaded(r1)$\}$.
- and the goal $g = \{$ at(r1,loc1), loaded(r1,c3)$\}$.
- If the ForwardSearch algorithm chooses the action $a =$ move(r1,loc2,loc1) in the first invocation and $a =$ load(crane1, loc1,c3,r1) in the second invocation producing the state s'. s' satisfies g, the execution returns:
  - $\pi = \langle$move(r1,loc2,loc1), load(cran1,loc1,c3,r1)$\rangle$

Forward Search Example

Warning

There are many other execution traces, some of which are infinite. For instance, one of them makes the following infinite sequence of choices for a:

- move(r1,loc2,loc1)
- move(r1,loc1,loc2)
- move(r1,loc2,loc1)
- move(r1,loc1,loc2)
- etc.

- In practice, you can use any classical graph search algorithms such as A*, Iterative Deepening, greedy best first search, etc.

II. Backward Search

Backward Search Principle and Algorithm

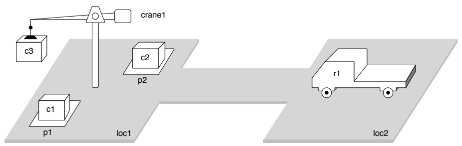
The idea is to start at the goal and apply inverses of the planning operator to produce subgoals, stopping if we produce a set of subgoals satisfied by the initial state. The backward search algorithm is also sound and complete.

Algorithm (BackwardSearch($\mathcal{O}$, s_0, g))

if s_0 *satisfies* g **then return** *an empty plan* π
revelant $\leftarrow \{a \mid$ *a is a ground instance of an operator* $\mathcal{O}$
that is revelant for g $\}$
if revelant $= \emptyset$ **then return** Failure
nondeterministically choose an action $a_1 \in$ revelant
$s_1 \leftarrow \gamma^{-1}(s, a_1)$
$\pi \leftarrow$ `BackwardSearch`$(\mathcal{O}, s_1, g)$
if $\pi \neq$ Failure **then return** $a_1 \cdot \pi$
else return Failure

Backward Search Example

Recall that the initial state is the state s:



- s_0 = { attached(p1, loc1), in(c1, p1), top(c1,p1), on(c1, pallet), attached(p2,loc1), in(c2,p2), top(c2,p2), on(c2,pallet), belong(crane1,loc1), holding(crane1,c3), adjacent(loc1,loc2), adjacent(loc2,loc1), at(r1,loc2), occupied(loc2), unloaded(r1)}.
- and the goal g = { at(r1,loc1), loaded(r1,c3)}.

Backward Search Example: First Invocation

In the first invocation of the BackwardSearch algorithm, it chooses a = load(crane1, loc1,c3,r1) and then assigns:

First Invocation

$$\begin{aligned} g \;\leftarrow\; & \gamma^{-1}(g, a) \\ = \; & (g - \text{effects}^{+}(a)) \cup \text{precond}(a) \\ = \; & (\{\text{at(r1,loc1), loaded(r1,c3)}\} - \{\text{empty(crane1), loaded(r1,c3)}\}) \\ \cup \; & \{\text{belong(crane1,loc1), holding(crane1,c3), at(r1,loc1),} \\ & \text{unloaded(r1)}\} \\ = \; & \{\text{at(r1,loc1), belong(crane1,loc1), holding(crane1,c3),} \\ & \text{unloaded(r1)}\} \end{aligned}$$

Backward Search Example: Second Invocation

In the second invocation of the BackwardSearch algorithm, it chooses $a =$ move(r1, loc2,loc1) and then assigns:

Second Invocation

$$\begin{aligned} g &\leftarrow \gamma^{-1}(g, a) \\ &= (g - \text{effects}^{+}(a)) \cup \text{precond}(a) \\ &= (\{\text{at(r1,loc1), belong(crane1,loc1), holding(crane1,c3),} \\ &\quad \text{at(r1,loc1), unloaded(r1)}\} - \{\text{at(r1,loc1), occupied(loc1)}\}) \\ &\cup \{\text{adjacent(loc2,loc1), at(r1,loc2), } \neg\text{occupied(loc1)}\} \\ &= \{\text{belong(crane1,loc1), holding(crane1,c3), unloaded(r1),} \\ &\quad \text{adjacent(loc2,loc1), at(r1,loc2), occupied(loc1)}\} \end{aligned}$$

Backward Search Example: Result

This time g is satisfied by s, so the execution trace terminates and returns the plans:

- $\pi = \langle$(move(r1,loc2,loc1), load(crane1,loc1,c3,r1)$\rangle$

Warning

Like ForwardSearch algorithm, there are many other execution traces, some of which are infinite. For instance, one of them makes the following infinite sequence of choices for a:

- load(crane1,loc1,c3,r1)
- unload(crane1,loc1,c3,r1)
- load(crane1,loc1,c3,r1)
- unload(crane1,loc1,c3,r1)
- etc.

III. STRIPS Algorithm

STRIPS Algorithm Principle

- The biggest problem of the previous approaches is how improve efficiency by reducing the size of the search space.
- STRIPS is somewhat similar to the BackwardSearch but differs from it in the following ways:
  1. In each recursive call of the STRIPS algorithm, the only subgoals eligible to be worked on are the preconditions of the last operator added to the plan. This reduce the branching factor substantially. However, it makes STRIPS incomplete.
  2. If the current state satisfies all of on operator's preconditions, STRIPS commits to executing that operator and will not backtrack over this commitment. This prune a large portion of the search space but again make STRIPS incomplete.

STRIPS Algorithm

Algorithm (STRIPS($\mathcal{O}$, s, g))

$\pi \leftarrow$ *the empty plan*
while true **do**
  if s *satisfies* g **then return** π
  revelant $\leftarrow \{a \mid$ *a is a ground instance of an operator* $\mathcal{O}$
    that is revelant for g $\}$
  if revelant $= \emptyset$ **then return** Failure
  nondeterministically choose an action $a \in$ revelant
  $\pi' \leftarrow$ `STRIPS`($\mathcal{O}$, s, *precond(a)*)
  if π' = Failure **then return** Failure
  ;; if we get here, then π' *achieves precond(a) from s*
  $s \leftarrow \gamma(s, \pi')$
  ;; s now satisfies precond(a)
  $s \leftarrow \gamma(s, a)$
  $\pi \leftarrow \pi \cdot \pi' \cdot a$
end

STRIPS Algorithm Remarks

- As an example of a case where STRIPS is incomplet, STRIPS is unabe to find a plan for on of the first problems a computer programmer learns:
  - The problem of interchanging the values of two variables
- Even for problems that STRIPS solves, it does not always find the best solution.

Sussman Anomaly

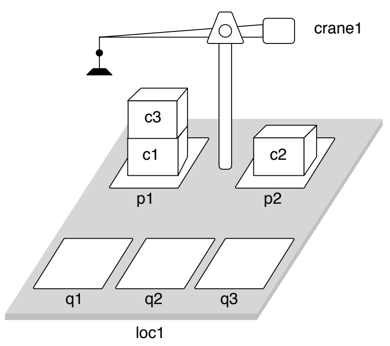


Figure 4: s_0 = { in(c3,p1), top(c3,p1), in(c1,p1), on(c3, c1), on(c1,pallet), in(c2,p2), top(c2,p2), on(c2,pallet), top(pallet,q1), top(pallet,q2), top(pallet, q3), empty(crane1) }

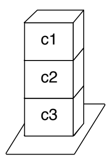


Figure 5: g = { on(c1,c2), on(c2,c3) }

STRIPS Result for the Sussman Anomaly

The shortest solutions that STRIPS can find are all similar to the following:

take(c3,loc1,crane1,c1)	
put(c3,loc1,crane1,q1)	
take(c1,loc1,crane1,p1)	
put(c1,loc1,crane1,c2)	*STRIPS has achieved on(c1,c2)*
take(c1,loc1,crane1,c2	
put(c1,loc1,crane1,p1)	
take(c2,loc1,crane1,p2)	
put(c2,loc1,crane1,c3)	*STRIPS has achieved on(c2,c3)* *but needs to reachieved on(c1,c2)*
take(c1,loc1,crane1,p1)	
put(c1,loc1,crane1,c2)	*STRIPS has now achieved both goals*

STRIPS result for the Sussman Anomaly

- STRIPS's difficulty involves deleted condition interaction.

Example

The action take(c1,loc1,crane1,c2) is necessary in order to help achieve on(c2,c3) but it deletes the previous achieved condition on(c1,c2).

- One way to find the shortest plan for Sussman anomaly is to interleave plans for different goals.

Note

This observation such as these led to the development of a technique called plan space planning, in which the planning system searches thought a space whose nodes are partial plans rather that states of the world.

To go further

Exercice

Consider the Sussman anomaly shown previously introduced slide 86. The shortest plan π_1 for achieving on(c1,c2) from the initial state is:

$$\begin{aligned}\pi_1 &= \langle \text{take(c3,loc1,crane1,c1)} \\ &\quad \text{put(c3,loc1,crane1,q1)} \\ &\quad \text{take(c1,loc1,crane1,p1)} \\ &\quad \text{put(c1,loc1,crane1,c2)}\rangle\end{aligned}$$

and the the shortest plan π_2 for achieving on(c2,c3) from the initial state is:

$$\begin{aligned}\pi_2 &= \langle \text{take(c2,loc1,crane1,p2)} \\ &\quad \text{put(c2,loc1,crane1,c3)}\rangle\end{aligned}$$

How to interleave π_1 and π_2 to find the shortest plan for the Sussman anomaly ?

Further readings

- R. Fikes and N. Nilsson
  STRIPS: A new approach to the application of theorem proving to problem solving.
  Artificial Intelligence 2(3-4):189-208, 1971
- J. Hoffmann
  FF: The fast forward planning system.
  Artificial Intelligence Magazine 22(3):57-62, 2001
- M. Helmert
  The Fast Downward Planning System.
  Journal Of Artificial Intelligence Research, Volume 26, pages 191-246, 2006