

CSC2542

State-Space Planning

Sheila McIlraith
Department of Computer Science
University of Toronto
Winter 2009

1

Acknowledgements

Some the slides used in this course are modifications of Dana Nau's lecture slides for the textbook *Automated Planning*, licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License:

<http://creativecommons.org/licenses/by-nc-sa/2.0/>

Other slides are modifications of slides developed by Malte Helmert, Bernhard Nebel, and Jussi Rintanen.

I have also used some material prepared by P@trick Haslum and Rao Kambhampati.

I would like to gratefully acknowledge the contributions of these researchers, and thank them for generously permitting me to use aspects of their presentation material.

2

Motivation

- Nearly all planning procedures are search procedures
- Different planning procedures have different search spaces
 - Two examples:
- *State-space planning*
 - Each node represents a state of the world
 - A plan is a path through the space
- *Plan-space planning*
 - Each node is a set of partially-instantiated operators, plus some constraints
 - Impose more and more constraints, until we get a plan

4

Outline

- State-space planning
 - Forward search
 - Backward search
 - Lifting
 - STRIPS
 - Block-stacking

5

Forward-search(O, s_0, g)

$s \leftarrow s_0$

$\pi \leftarrow$ the empty plan

loop

if s satisfies g then return π

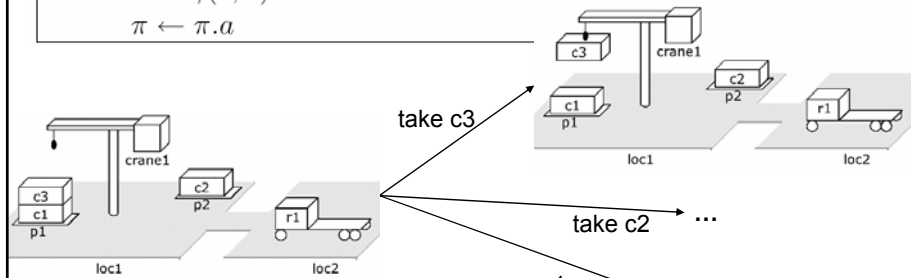
$E \leftarrow \{a \mid a \text{ is a ground instance an operator in } O,$
and $\text{precond}(a)$ is true in $s\}$

if $E = \emptyset$ then return failure

nondeterministically choose an action $a \in E$

$s \leftarrow \gamma(s, a)$

$\pi \leftarrow \pi.a$



6

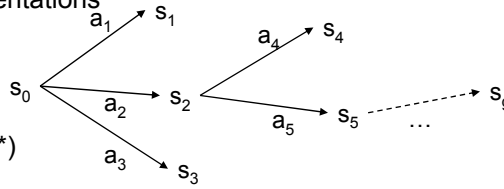
Properties

- Forward-search is *sound*
 - for any plan returned by any of its nondeterministic traces, this plan is guaranteed to be a solution
- Forward-search also is *complete*
 - if a solution exists then at least one of Forward-search's nondeterministic traces will return a solution.

7

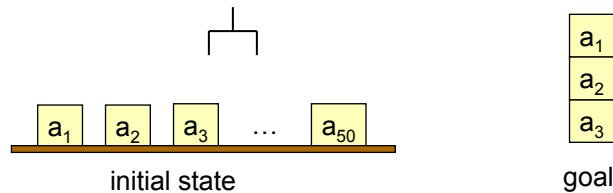
Deterministic Implementations

- Some deterministic implementations of forward search:
 - breadth-first search
 - depth-first search
 - best-first search (e.g., A*)
 - greedy search
- Breadth-first and best-first search are sound and complete
 - But they usually aren't practical, requiring too much memory
 - Memory requirement is exponential in the length of the solution
- In practice, more likely to use depth-first search or greedy search
 - Worst-case memory requirement is linear in the length of the solution
 - In general, sound but not complete
 - But classical planning has only finitely many states
 - Thus, can make depth-first search complete by doing loop-checking



8

Branching Factor of Forward Search



- Forward search can have a very large branching factor
 - Can have many applicable actions that don't progress toward goal
- Why this is bad:
 - Deterministic implementations can waste time trying lots of irrelevant actions
- Need a good heuristic function and/or pruning procedure (This will be a focus of later discussion)

9

Backward Search

- For forward search, we started at the initial state and computed state transitions
 - new state = $\gamma(s,a)$
- For backward search, we start at the goal and compute inverse state transitions
 - new set of subgoals = $\gamma^{-1}(g,a)$
- To define $\gamma^{-1}(g,a)$, must first define *relevance*:
 - An action a is relevant for a goal g if
 - a makes at least one of g 's literals true
 - $g \cap \text{effects}(a) \neq \emptyset$
 - a does not make any of g 's literals false
 - $g^+ \cap \text{effects}^-(a) = \emptyset$ and $g^- \cap \text{effects}^+(a) = \emptyset$

10

Inverse State Transitions

- If a is relevant for g , then
 - $\gamma^{-1}(g,a) = (g - \text{effects}(a)) \cup \text{precond}(a)$
- Otherwise $\gamma^{-1}(g,a)$ is undefined

- Example: suppose that
 - $g = \{\text{on}(b1,b2), \text{on}(b2,b3)\}$
 - $a = \text{stack}(b1,b2)$
- What is $\gamma^{-1}(g,a)$?

11

Backward-search(O, s_0, g)

$\pi \leftarrow$ the empty plan

loop

if s_0 satisfies g then return π

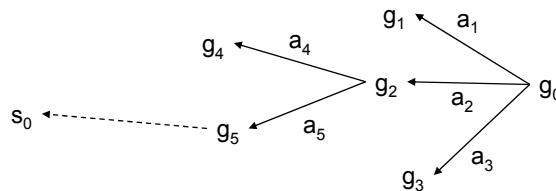
$A \leftarrow \{a \mid a \text{ is a ground instance of an operator in } O$
and $\gamma^{-1}(g, a)$ is defined}

if $A = \emptyset$ then return failure

nondeterministically choose an action $a \in A$

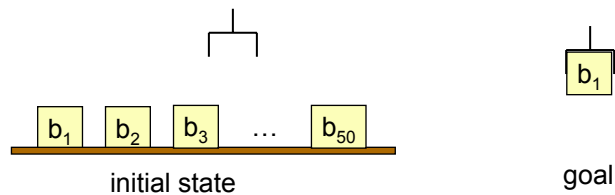
$\pi \leftarrow a.\pi$

$g \leftarrow \gamma^{-1}(g, a)$



12

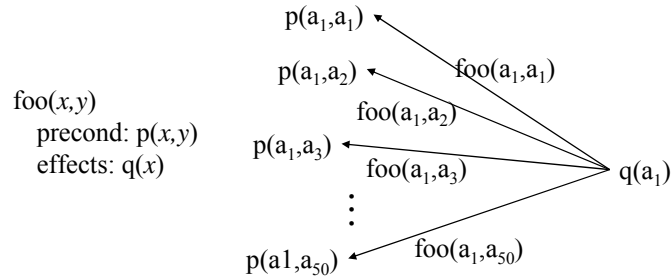
Efficiency of Backward Search



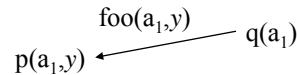
- Backward search can *also* have a very large branching factor
 - E.g., an operator o that is relevant for g may have many ground instances a_1, a_2, \dots, a_n such that each a_i 's input state might be unreachable from the initial state
- As before, deterministic implementations can waste lots of time trying all of them

13

Lifting



- Can reduce the branching factor of backward search if we *partially* instantiate the operators
 - this is called *lifting*



14

Lifted Backward Search

- More complicated than Backward-search
 - Have to keep track of what substitutions were performed
- But it has a much smaller branching factor

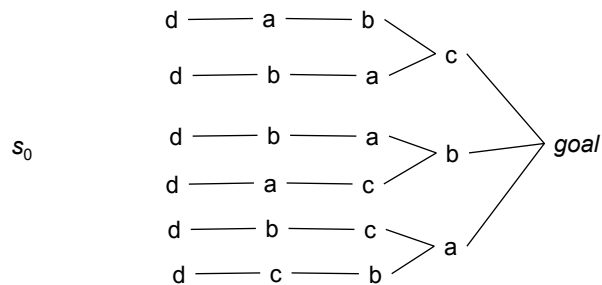
```

Lifted-backward-search( $O, s_0, g$ )
 $\pi \leftarrow$  the empty plan
loop
  if  $s_0$  satisfies  $g$  then return  $\pi$ 
   $A \leftarrow \{(o, \theta) \mid o \text{ is a standardization of an operator in } O,$ 
     $\theta \text{ is an mgu for an atom of } g \text{ and an atom of effects}^+(o),$ 
    and  $\gamma^{-1}(\theta(g), \theta(o)) \text{ is defined}\}$ 
  if  $A = \emptyset$  then return failure
  nondeterministically choose a pair  $(o, \theta) \in A$ 
   $\pi \leftarrow$  the concatenation of  $\theta(o)$  and  $\theta(\pi)$ 
   $g \leftarrow \gamma^{-1}(\theta(g), \theta(o))$ 
    
```

15

The Search Space is Still Too Large

- Lifted-backward-search generates a smaller search space than Backward-search, but it still can be quite large
 - Suppose actions a , b , and c are independent, action d must precede all of them, and there's no path from s_0 to d 's input state
 - We'll try all possible orderings of a , b , and c before realizing there is no solution
 - Plan-space planning can help with this problem



16

Pruning the Search Space

Pruning the search space can really help.

Two techniques we will discuss:

- Sound pruning using branch-and-bound heuristic search
- Domain customization that prunes actions and states

For now, just two examples:

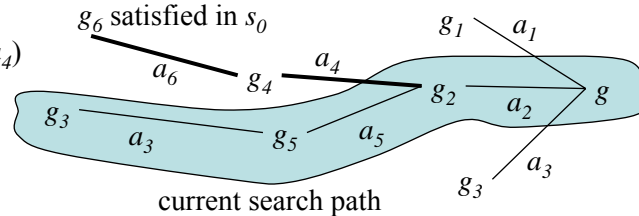
- STRIPS
- Block stacking

17

STRIPS

- $\pi \leftarrow$ the empty plan
- do a modified backward search from g
 - instead of $\gamma^{-1}(s,a)$, each new set of subgoals is just $\text{precond}(a)$
 - when you find an action that's executable in the current state, then go forward on the current search path as far as possible, executing actions and appending them to π
 - repeat until all goals are satisfied

$\pi = \langle a_6, a_4 \rangle$
 $s = \gamma(\gamma(s_0, a_6), a_4)$



18

Quick Review of Blocks World

unstack(x,y)

Pre: $\text{on}(x,y), \text{clear}(x), \text{handempty}$
 Eff: $\sim\text{on}(x,y), \sim\text{clear}(x), \sim\text{handempty},$
 $\text{holding}(x), \text{clear}(y)$

stack(x,y)

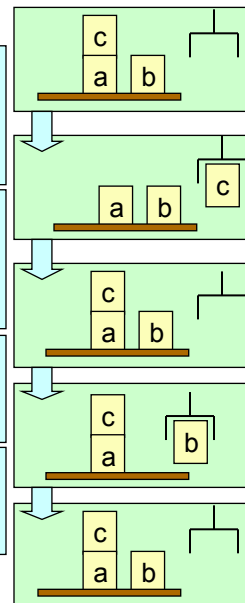
Pre: $\text{holding}(x), \text{clear}(y)$
 Eff: $\sim\text{holding}(x), \sim\text{clear}(y),$
 $\text{on}(x,y), \text{clear}(x), \text{handempty}$

pickup(x)

Pre: $\text{ontable}(x), \text{clear}(x), \text{handempty}$
 Eff: $\sim\text{ontable}(x), \sim\text{clear}(x), \sim\text{handempty}, \text{holding}(x)$

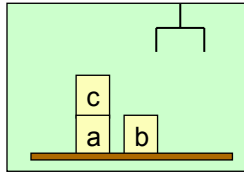
putdown(x)

Pre: $\text{holding}(x)$
 Eff: $\sim\text{holding}(x), \text{ontable}(x), \text{clear}(x), \text{handempty}$

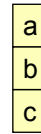


19

The Sussman Anomaly



Initial state



goal

- On this problem, STRIPS can't produce an irredundant solution
 - Try it and see

20

The Register Assignment Problem

- State-variable formulation:

Initial state: $\{\text{value}(r1)=3, \text{value}(r2)=5, \text{value}(r3)=0\}$

Goal: $\{\text{value}(r1)=5, \text{value}(r2)=3\}$

Operator: $\text{assign}(r, v, r', v')$

precond: $\text{value}(r)=v, \text{value}(r')=v'$

effects: $\text{value}(r)=v'$

- STRIPS cannot solve this problem at all

21

How to Handle Problems like These?

Several ways:

- Do something other than state-space search
 - e.g., Chapters 5–8
- Use forward or backward state-space search, with *domain-specific* knowledge to prune the search space
 - Can solve both problems quite easily this way
 - Example: block stacking using forward search

22

Domain-Specific Knowledge

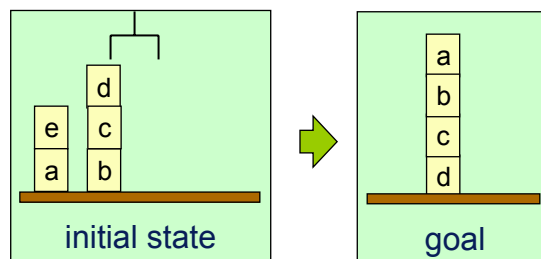
- A blocks-world planning problem $P = (O, s_0, g)$ is solvable if s_0 and g satisfy some simple consistency conditions
 - g should not mention any blocks not mentioned in s_0
 - a block cannot be on two other blocks at once
 - etc.
 - Can check these in time $O(n \log n)$
- If P is solvable, can easily construct a solution of length $O(2m)$, where m is the number of blocks
 - Move all blocks to the table, then build up stacks from the bottom
 - Can do this in time $O(n)$
- With additional domain-specific knowledge can do even better ...

23

Additional Domain-Specific Knowledge

A block x needs to be moved if any of the following is true:

- s contains $\text{ontable}(x)$ and g contains $\text{on}(x,y)$ - see a below
- s contains $\text{on}(x,y)$ and g contains $\text{ontable}(x)$ - see d below
- s contains $\text{on}(x,y)$ and g contains $\text{on}(x,z)$ for some $y \neq z$ - see c below
- s contains $\text{on}(x,y)$ and y needs to be moved - see e below



24

Domain-Specific Algorithm

loop

if there is a clear block x such that

x needs to be moved **and**

x can be moved to a place where it won't need to be

 moved

then move x to that place

else if there is a clear block x such that x needs to be moved

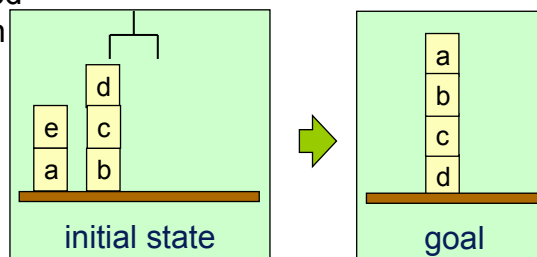
then move x to the table

else if the goal is satisfied

then return the plan

else return failure

repeat



25

Easily Solves the Sussman Anomaly

loop

if there is a clear block x such that
 x needs to be moved **and**
 x can be moved to a place where it won't need to be moved

then move x to that place

else if there is a clear block x such that
 x needs to be moved

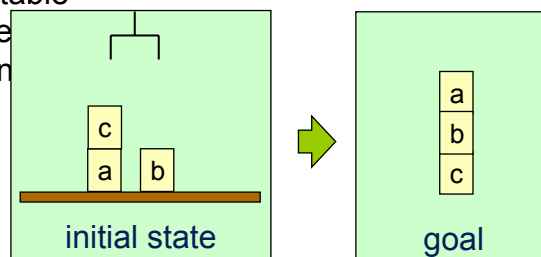
then move x to the table

else if the goal is satisfied

then return the plan

else return failure

repeat



26

Properties

The block-stacking algorithm:

- Sound, complete, guaranteed to terminate
- Runs in time $O(n^3)$
 - Can be modified to run in time $O(n)$
- Often finds optimal (shortest) solutions
- But sometimes only near-optimal (Exercise 4.22 in the book)
(Note: PLAN LENGTH for the blocks world is NP-complete)

27

REMINDER: Administrative Announcements

- There will be *no tutorial* on Friday.
- csc2542w09@cs email list created (though I prefer communicating via the course announcements web page)
- Suggested readings for next week:
 - Skim Chapter 3, Chapter 4
 - *Skip* Chapter 5
 - Read Chapter 6 & 7 (we'll cover this in class)