

Lecture 12 Planning: Intro and Forward Planning,

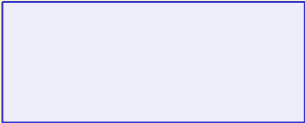
Lecture Overview

- ➔ Planning: Intro
 - STRIPS representation
 - Forward Planning
 - Heuristics for Forward Planning

Course Overview

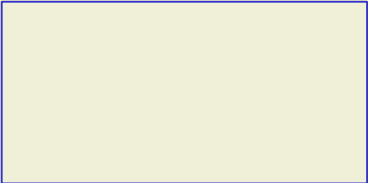
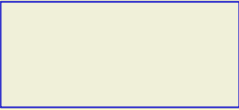
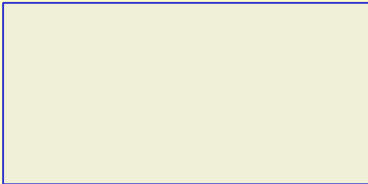
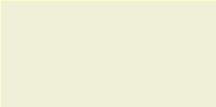
Problem Type

Arc Environment Stochastic



Representation Reasoning Technique

Deterministic



Consistency

Constraint Satisfaction *Vars + Constraints* Search

Static *Belief Nets*

Query

Logics

Variable

Search

Elimination

Sequential

STRIPS

Decision Nets Variable

Planning

Search

Elimination

First Part of
the Course

Markov Processes Value

Iteration

5

Course Overview

Problem Type

Arc Environment Stochastic

Representation Reasoning Technique
Deterministic

Consistency

Constraint Satisfaction *Vars + Constraints* Search

Static *Belief Nets*

Query

Logics

Variable

Search

Elimination

Sequential

STRIPS

Decision Nets Variable

Planning

Search

Elimination

We'll focus
on Planning

Markov Processes Value

Iteration

Planning Problem

- Goal
- Description of states of the world
- Description of available actions => when each action can be applied and what its effects are
- **Planning**: build a sequence of actions that, if executed, takes the agent from the current state to a state that

achieves the goal

But, haven't we seen this before?

Yes, in search, but we'll look at a new R&R suitable for planning

Standard Search vs. Specific R&R systems

- Constraint Satisfaction (Problems):
- **State:** assignments of values to a subset of the variables
- **Successor function:** assign values to a “free” variable
- **Goal test:** all variables assigned a value and all constraints satisfied?
- **Solution:** possible world that satisfies the constraints
- **Heuristic function:** none (all solutions at the same distance from start)

- Planning :
 - State
 - Successor function
 - Goal test
 - Solution
 - Heuristic function
- Inference
- State
- Successor function
- Goal test
- Solution
- Heuristic function

Standard Search vs. Specific R&R systems

CSP problems had some specific properties

- **States** are represented in terms of features (variables with a possible range of values)
- **Goal**: no longer a black box => expressed in terms of constraints (satisfaction of)
- But actions are limited to assignments of values to variables
- No notion of path to a solution: only final assignment matters

Key Idea of Planning

- “Open-up” the representation of states, goals and actions
 - Both states and goals as set of features
 - Actions as preconditions and effects defined on state features



- agent can reason more deliberately about which actions to consider to achieve its goals.

Key Idea of Planning

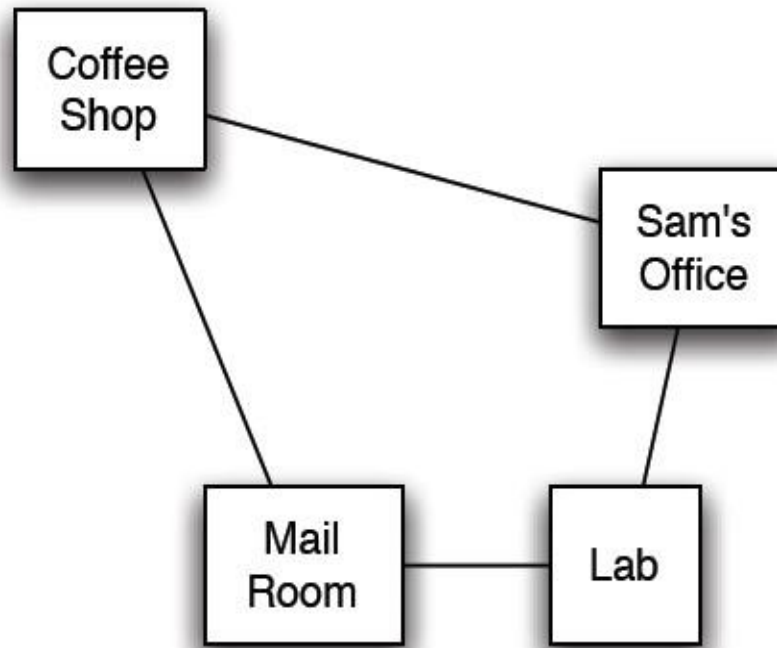
- This representation lends itself to solve planning problems either
 - As pure **search** problems
 - As **CSP** problems
- We will look at one technique for each approach

- this will only scratch the surface of planning techniques
- but will give you an idea of the general approaches in this important area of AI

Let's start by introducing a very simple planning problem, as our running example

Running Example: Delivery Robot (textbook)

- Consider a delivery robot named Rob, who must navigate the following environment, and can deliver coffee and mail to Sam, in his office

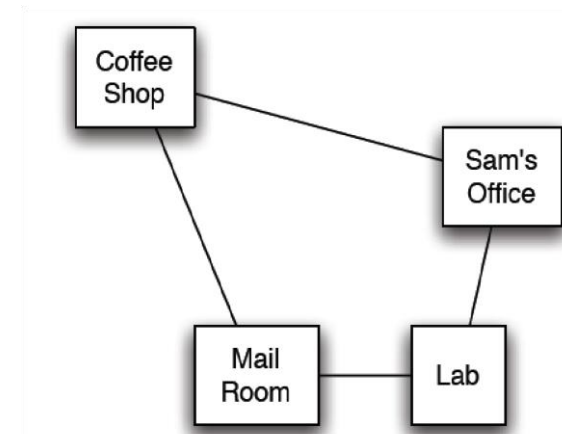


Slide 14

Delivery Robot Example: features

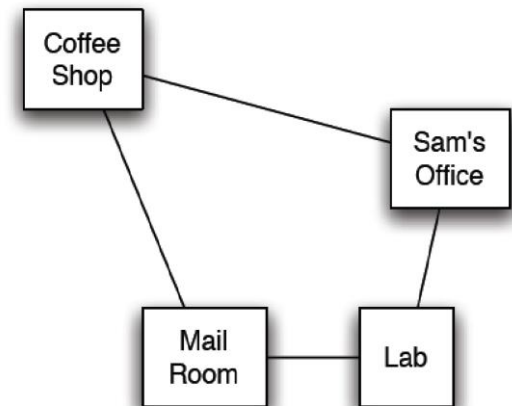
- **RLoc** - Rob's location

- Domain: {coffee shop, Sam's office, mail room, lab} **short** {cs, off, mr, lab}
- **RHC** - Rob has coffee
- Domain: {true, false}. Alternatively notation for **RHC = T/F**: rhc indicates that Rob has coffee, and *rhc* that Rob doesn't have coffee
- **SWC** - Sam wants coffee {true, false}
- **MW** - Mail is waiting {true, false}
- **RHM** - Rob has mail {true, false}
- An example state is



Delivery Robot Example: features

- **RLoc** - Rob's location
- Domain: {coffee shop, Sam's office, mail room, lab} **short** {cs, off, mr, lab}
- **RHC** - Rob has coffee
- Domain: {true, false}. Alternatively notation for **RHC** = T/F: **rhc** indicates that Rob has coffee, and **!rhc** indicates that Rob doesn't have coffee
- **SWC** - Sam wants coffee {true, false}
- **MW** - Mail is waiting {true, false}
- **RHM** - Rob has mail {true, false}



- An example state is $\langle lab, \overline{rhc}, swc, \overline{mw}, rhm \rangle$

Rob is in the lab, it does not have coffee, Sam wants coffee, there is no mail waiting and Rob has mail

Delivery Robot Example: Actions

The robot's **actions** are:

puc- Rob picks up coffee

- must be **at the coffee shop** and

not have coffee

delC- Rob delivers coffee

Preconditions for
action application


- must be **at the office**, and must **have coffee** **pump**- Rob picks up mail
- must be in the **mail room**, and **mail must be waiting** **delM**- Rob delivers mail
- must be **at the office** and **have mail** **move**- Rob's move actions - there are 8 of them
 - move clockwise (**mc-x**), move anti-clockwise (**mcc-x**) from location x (where x can be any of the 4 rooms) • must be in **location x**

Modeling actions for planning

- The key to sophisticated planning is **modeling actions**

- Leverage a feature-based representation:
- Model when **actions are possible**, in terms of the values of the features in the current state
- Model **state transitions** caused by actions in terms of changes in specific features

Lecture Overview

- Planning: Intro
-  STRIPS representation
- Forward Planning
- Heuristics for Forward Planning

STRIPS actions: Example

STRIPS representation

(STanford Research Institute Problem Solver)

STRIPS - the planner in Shakey, first AI robot

http://en.wikipedia.org/wiki/Shakey_the_robot In

STRIPS, an action has **two parts**:

1. **Preconditions**: a set of assignments to features that must be satisfied in order for the action to be legal/valid/applicable

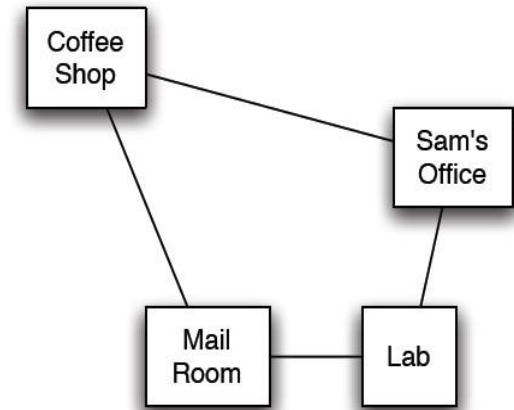


2. **Effects**: a set of assignments to features that are caused by the action

STRIPS representation of the action

picks up coffee, puc:

- preconditionsLoc= and RHC=
- effectsRHC= **cs = coffee shop**

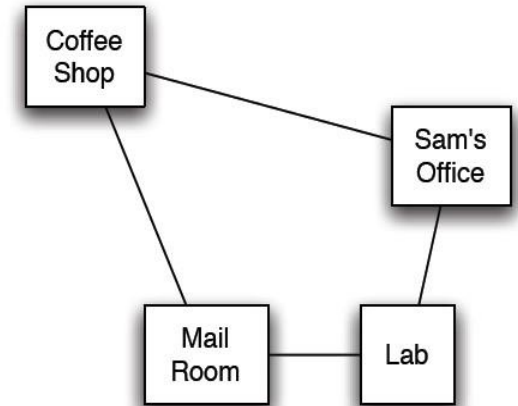


off = Sam's office
mr = mail rom

STRIPS actions: Example

STRIPS representation of the action **pick up coffee**, puc:

- preconditions Loc= cs and RHC= F
- effects RHC= T **cs = coffee shop**



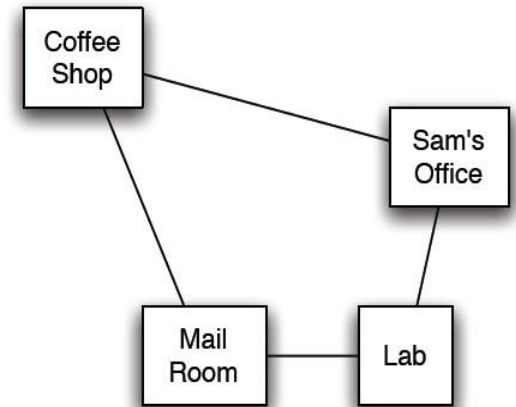
off = Sam's office
mr = mail rom

STRIPS representation of the action **deliver coffee**, Del :

- preconditions Loc= and RHC=
- effects RHC = and SWC =

STRIPS representation of the action **pick up coffee**, puc:

- preconditionsLoc= cs and RHC= F
- effectsRHC= T **cs = coffee shop**



off = Sam's office
mr = mail rom

STRIPS representation of the action **deliver coffee**, Del :

- preconditions Loc= off and RHC= T
- effects RHC = F and SWC = F

STRIPS actions: Example

Note in this domain Sam doesn't have to want coffee for Rob to deliver it; one way or another, Sam doesn't want coffee after delivery.

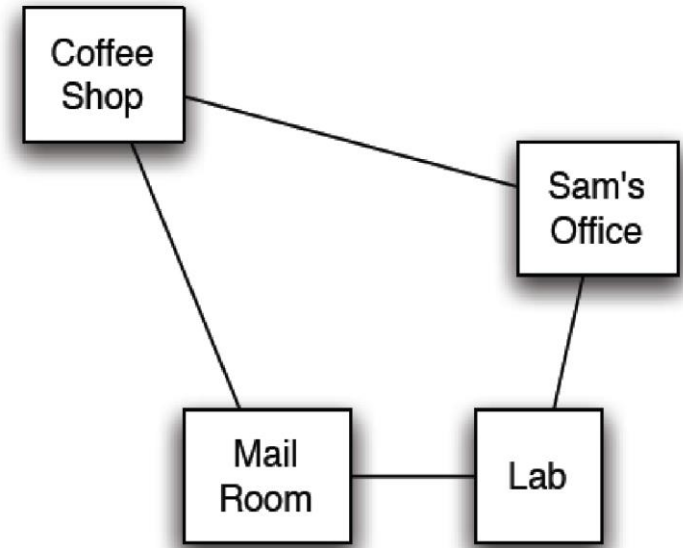
STRIPS actions: MC and MAC

STRIPS representation of the actions related to moving clockwise

- **mc-cs**
preconditions Loc= cs effects Loc= off

- **mc-off** preconditions $\text{Loc} = \text{off}$ **cs = coffee shop** effects $\text{Loc} = \text{labf}$
 $\text{off} = \text{Sam's office}$
- **mc-lab** **mr = mail rom**
- **mc-mc ...**

There are 4 more actions for Move Counterclockwise (mcc-cs, mcc-off, etc.)



The STRIPS Representation

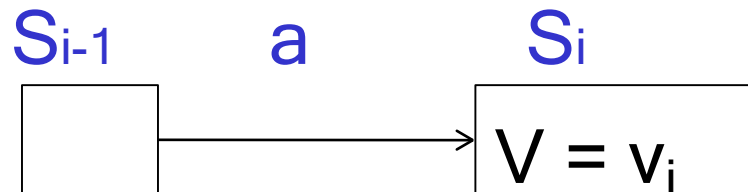
- For reference:
The book also discusses a **feature-centric** representation (not required for this course)
- for every feature, where does its value come from?
- **causal rule**: expresses ways in which a feature's value can be changed by taking an action.
- **frame rule**: requires that a feature's value is unchanged if none of the relevant actions changes it.
- STRIPS is an **action-centric** representation:
- for every action, what does it do?
- This leaves us with no way to state **frame rules**.

STRIPS Actions (cont')

- The STRIPS assumption:
- all features not explicitly changed by an action stay unchanged

The STRIPS assumption: all features not explicitly changed by an action stay unchanged

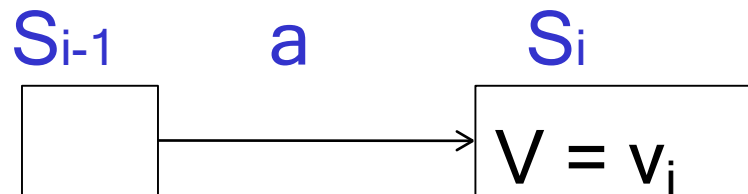
- So if the feature V has value v_i in state S_i , after action a has been performed,
- what can we conclude about a and/or the state of the world S_{i-1} immediately preceding the execution of a ?



STRIPS Actions (cont')

The STRIPS assumption: all features not explicitly changed by an action stay unchanged

- So if the feature V has value v_i in state S_i , after action a has been performed,
- what can we conclude about a and/or the state of the world S_{i-1} immediately preceding the execution of a ?



STRIPS Actions (cont')

A. $V = v_i$ was TRUE in S_{i-1}

C. At least one of A and B

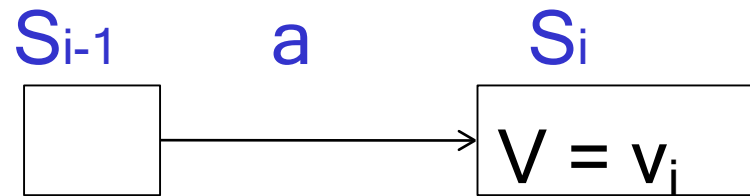
B. One of the effects of a is to set $V = v_i$

D None of the above

The STRIPS assumption: all features not explicitly changed by an action stay unchanged

- So if the **feature V** has value v_i in state S_i , after action a has been performed,
- what can we conclude about a and/or the **state of the world S_{i-1}** immediately preceding the execution of a ?

STRIPS Actions (cont')



C. At least one of A and B

Solving planning problems


- STRIPS lends itself to solve planning problems either
 - As pure search problems
 - As CSP problems
- We will look at one technique for each approach

Forward planning

- To find a plan, a solution : search in the state-space graph

Slide 29

Lecture Overview

- Planning: Intro
- STRIPS representation
-  • Forward Planning
- Heuristics for Forward Planning

- The **states** are the **possible worlds**
 - ✓ full assignments of values to features
- The **arcs** from a state s represent all the **actions** that are **possible** in state s
- A **plan** is a path from the state representing the initial state to a state that satisfies the goal

Which actions **a** are possible in a state **s**?

Forward planning

- To find a plan, a solution : search in the state-space graph
 - The **states** are the **possible worlds**
 - ✓ full assignments of values to features
 - The **arcs** from a state s represent all the **actions** that are **possible** in state s
 - A **plan** is a path from the state representing the initial state to a state that satisfies the goal

Which actions a are possible in a state s ?

A. Those where a 's effects are satisfied in s

B. Those where a 's preconditions are satisfied in s

C. Those where the state s' reached via a is on the way to the goal

-

C. Both A and B

- The **states** are the **possible worlds**
 - ✓ full assignments of values to features
- The **arcs** from a state s represent all the **actions** that are **possible** in state s
- A **plan** is a path from the state representing the initial state to a state that satisfies the goal

Which actions a are possible in a state s ?

B. Those where a 's preconditions are satisfied in s

Example

Suppose that we are in a state where

Forward planning

- To find a plan, a solution : search in the state-space graph
 - Rob is in the coffee shop and does not have coffee;
 - Sam wants coffee
 - Mail is waiting
 - Rob does not have mail
- $\langle cs, \overline{rhc}, swc, mw, \overline{rhm} \rangle$
- And the goal is that Sam does not want coffee anymore swc

Goal: swc

```
graph TD; CS[Coffee Shop] --- SO[Sam's Office]; CS --- MR[Mail Room]; SO --- Lab; MR --- Lab;
```

mc: move clockwise
 mac: move anticlockwise
 mcc: move counterclockwise
 nm: no move
 puc: pick up coffee
 dc: deliver coffee
 pum: pick up mail
 dm: deliver mail



cs: coffee shop

off: office

lab: laboratory

mr: mail room

Feature values

rhc: robot has coffee

swc: Sam wants coffee

mw: mail waiting

*rh**m*: robot has mail

43

Goal: swc

```
graph TD; CS[Coffee Shop] --- SO[Sam's Office]; CS --- MR[Mail Room]; SO --- Lab; MR --- Lab;
```

mc: move clockwise
 mac: move anticlockwise
 mcc: move counterclockwise
 nm: no move
 puc: pick up coffee
 dc: deliver coffee
 pum: pick up mail
 dm: deliver mail

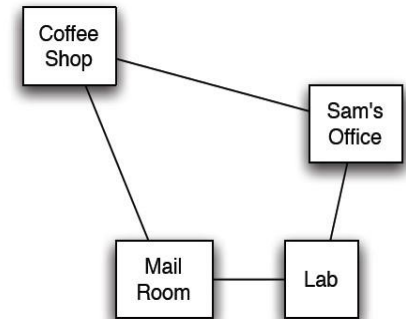
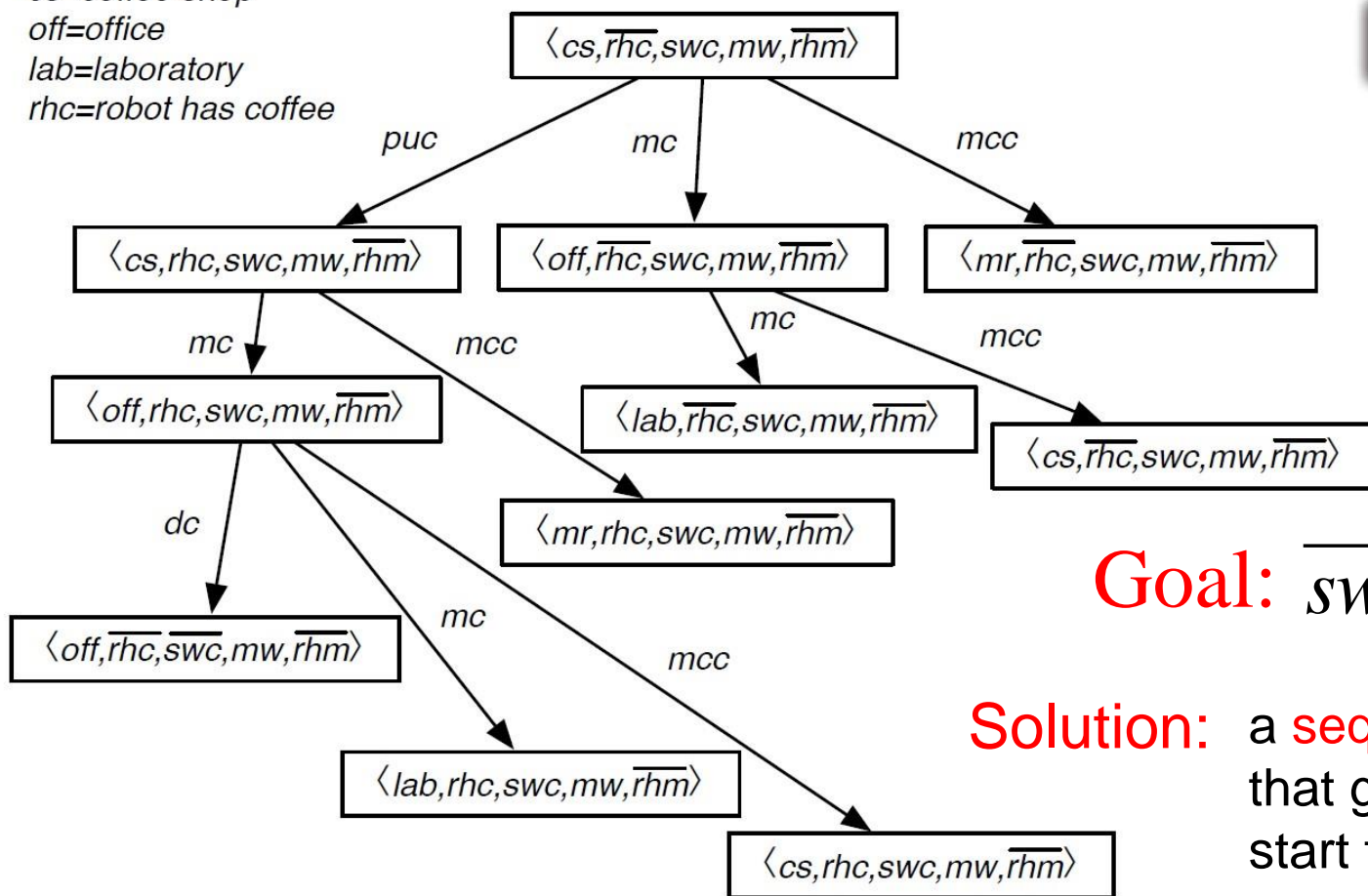


cs: coffee shop
off: office
lab: laboratory
mr: mail room

rhc: robot has coffee
swc: Sam wants coffee
mw: mail waiting
rhm: robot has mail

45

cs=coffee shop
off=office
lab=laboratory
rhc=robot has coffee

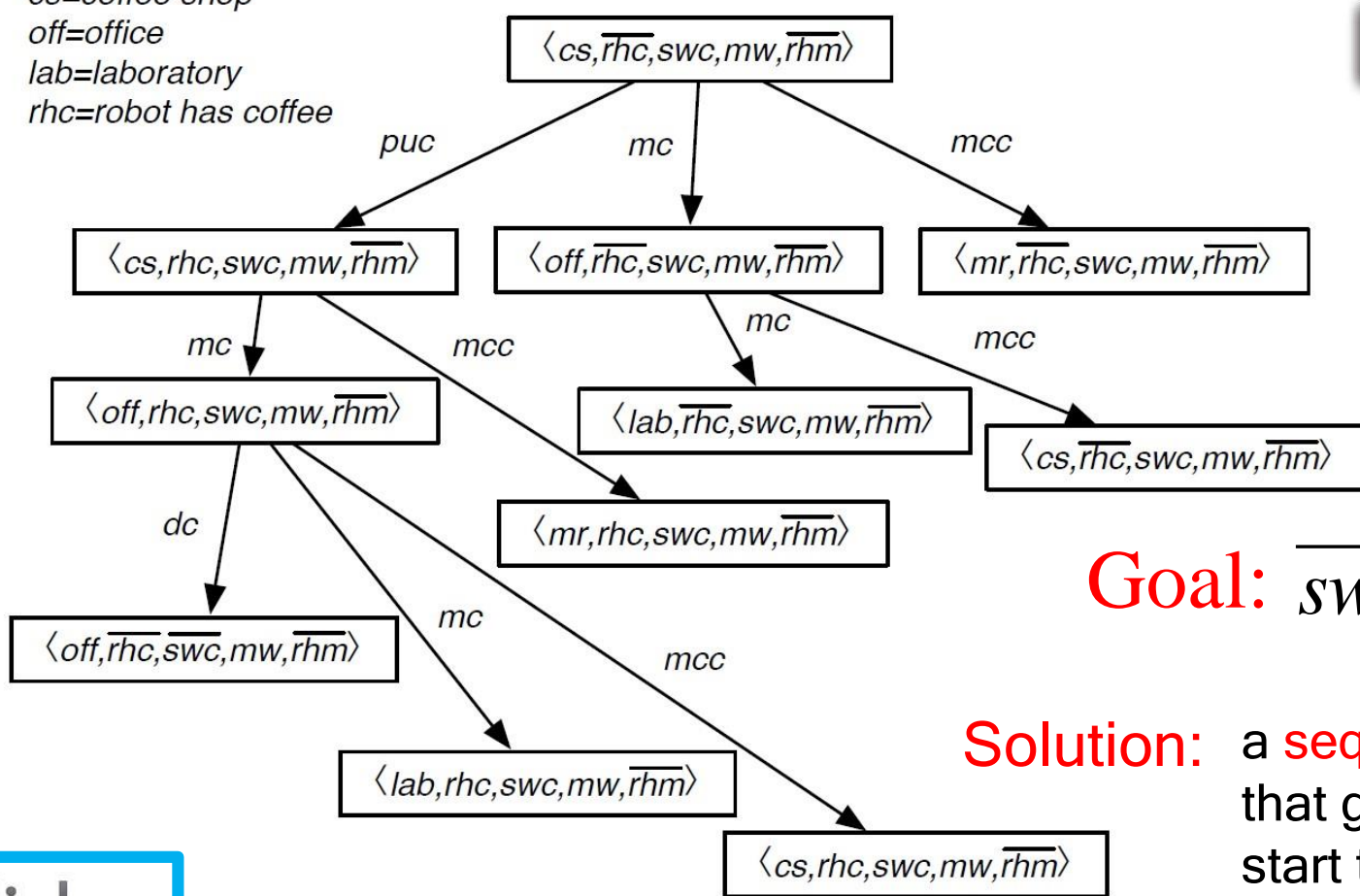


Goal: \overline{swc}

Solution: a sequence of actions that gets us from the start to a goal

Example for state space graph

cs=coffee shop
off=office
lab=laboratory
rhc=robot has coffee



Goal: \overline{swc}

Solution: a sequence of actions that gets us from the start to a goal



What is a solution to this planning problem?

A (puc, mc)

B (puc, mc, mc)

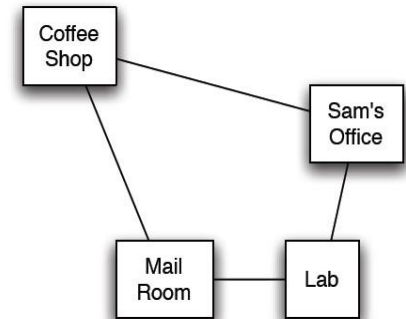
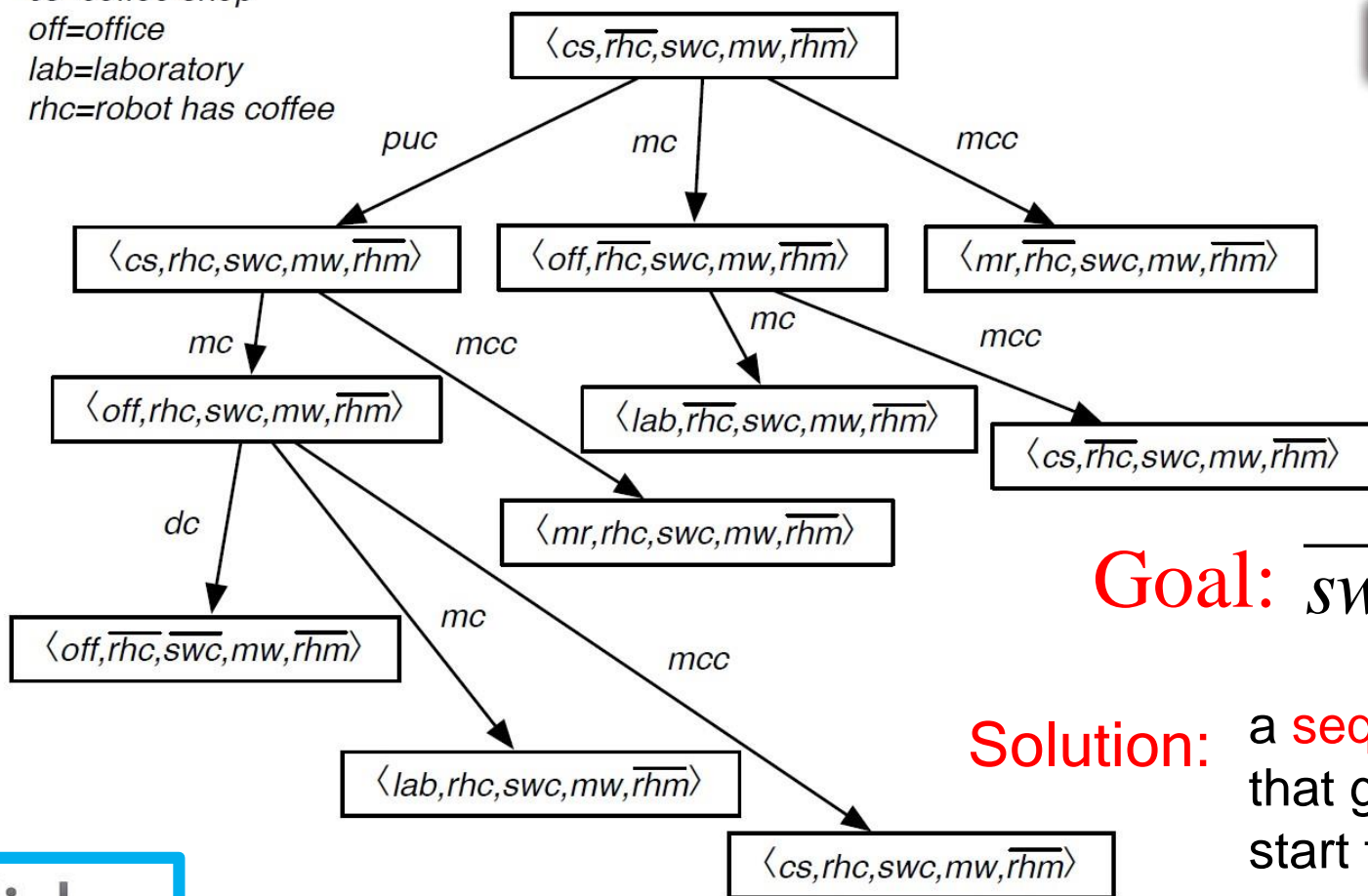
Example for state space graph

C (puc, dc)

D (puc, mc, dc)

What is a solution to this planning problem?

cs=coffee shop
off=office
lab=laboratory
rhc=robot has coffee



Goal: \overline{swc}

Solution: a sequence of actions that gets us from the start to a goal



D (puc, mc, dc)

Example for state space graph

Standard Search vs. Specific R&R systems

Constraint Satisfaction (Problems):

- **State:** assignments of values to a subset of the variables
- **Successor function:** assign values to a “free” variable
- **Goal test:** set of constraints
- **Solution:** possible world that satisfies the constraints
- **Heuristic function:** *none (all solutions at the same distance from start)* Planning :
- **State:** full assignment of values to features
- **Successor function:** states reachable by applying actions with preconditions satisfied in the current state

- **Goal test:** partial assignment of values to features
- **Solution:** a sequence of actions
- **Heuristic function**

Inference

- **State**
- **Successor function**
- **Goal test**
- **Solution**
- **Heuristic function**

Forward Planning

- Any of the search algorithms we have seen can be used in Forward Planning
- Problem?
- Complexity is defined by the **branching factor**, which is

A. Number of actions defined in the planning problem

B. Number of actions applicable in a state



C. Average number of preconditions in the actions applicable in a state

D. Average number of effects in the actions applicable in a state

Forward Planning

- Any of the search algorithms we have seen can be used in Forward Planning
- Problem?
- Complexity is defined by the branching factor, which is

Number of applicable actions to a state

- Can be very large • Solution?

44

Standard Search vs. Specific R&R systems

Constraint Satisfaction (Problems):

- **State:** assignments of values to a subset of the variables
- **Successor function:** assign values to a “free” variable
- **Goal test:** set of constraints
- **Solution:** possible world that satisfies the constraints


46

- **Heuristic function:** *none (all solutions at the same distance from start)* Planning :
- **State:** full assignment of values to features
- **Successor function:** states reachable by applying actions with preconditions satisfied in the current state
- **Goal test:** partial assignment of values to features
- **Solution:** a sequence of actions
- **Heuristic function**

Inference

- State
- Successor function
- Goal test
- Solution
- Heuristic function

Lecture Overview

- Planning: Intro
- STRIPS representation
- Forward Planning
-  Heuristics for Forward Planning

Heuristics for Forward Planning

Not in textbook, but you can see details in Russel&Norvig,
10.3.2

- Heuristic function: estimate of the distance from a state to the goal
- In planning this distance is the.....

Not in textbook, but you can see details in Russel&Norvig,
10.3.2

- Heuristic function: estimate of the distance from a state to the goal

A. # of goal features not true in s

Heuristics for Forward Planning

- In planning this distance is the..... B. # of actions needed to get from s to the goal

C. # of legal actions in s

Not in textbook, but you can see details in Russel&Norvig,
10.3.2

- Heuristic function: estimate of the **distance** from a state to the goal
- In planning this distance is the..... B. # of actions needed to get from s to the goal

Heuristics for Forward Planning

- Finding a good heuristic is what makes forward planning feasible in practice
- **Factored representation** of states and actions allows for definition of **domain-independent heuristics**
- We will look at one example of such domain-independent heuristic that has proven to be quite successful in practice

Heuristics for Forward Planning:

- We make two simplifications in the STRIPS representation
 - ✓ All features are binary: T / F
 - ✓ Goals and preconditions can only be assignments to T
e.g. positive assertions
- Definition: a **sub goal** is the specific assignment for one of the features in the goal
- e.g., if the goal is $\langle A=T, B=T, C=T \rangle$ then....

S1

$A = T$ $A = T$

Goal

$B = F \quad B = T$

$C = F \quad C = T$

Heuristics for Forward Planning:

- We make two simplifications in the STRIPS representation
 - ✓ All features are binary: T / F
 - ✓ Goals and preconditions can only be assignments to T
e.g. positive assertions
- Definition: a **subgoal** is the specific assignment for one of the features in the goal

Heuristics for Forward Planning: ignore delete-list

- e.g., if the goal is $\langle A=T, B=T, C=T \rangle$ then....

S1

Goal

A = T — A = T

B = F B = T

C = F C = T

- One strategy to find a non-trivial admissible heuristics is

- to relax the original problem

Heuristics for Forward Planning: ignore delete-list

Heuristics for Forward Planning: ignore delete-list

A. To set all $h(n)$ values to 0 to relax the original problem

- One strategy to find a non-trivial admissible

heuristics is

-

B. To relax some constraints on the actions in the original problem

C. To simplify the goal in the original problem

D. To run an uniformed search strategy (e.g. DFS or BFS) in the original problem

- One strategy to find an admissible heuristics is

B. To relax some constraints on the actions in the original problem

Heuristics for Forward Planning: ignore delete-list

56

- One strategy to find an admissible heuristic is
- to relax the original problem
- One way: remove all the effects that make a variable = F.
Action aeffects (B=F, C=T)
- Name of this heuristic derives from complete STRIPS representation
- Action effects are divided into those that add elements to the new state (add list) and those that remove elements (delete list)

- If we find the path from the initial state to the goal using this relaxed version of the actions:
- the length of the solution is an underestimate of the actual solution length. **Why?**

Heuristics for Forward Planning: ignore delete-list

- One strategy to find an admissible heuristic is
- to relax the original problem
- One way: remove all the effects that make a variable = F.
Action aeffects (B=F, C=T) ✗
- If we find the path from the initial state to the goal using this relaxed version of the actions:
- the length of the solution is an underestimate of the actual solution length
- Why?

S_0

$B = F$

$B = T$

$C = F$

$C = T$

Heuristics for Forward Planning: ignore delete-list

$A = T$

$A = T$

Goal

- One strategy to find an admissible heuristics is
- to relax the original problem
- One way : remove all the effects that make a variable = F. ~~Action affects (B=F, C=T)~~
- If we find the path from the initial state to the goal using this relaxed version of the actions:
- the length of the solution is an underestimate of the actual solution length

S_0

$B = F$

$C = F$

$B = T$

$C = T$

Heuristics for Forward Planning: ignore delete-list

- Why? In the original problem, one action (e.g. aabove) might undo an already achieved goal (e.g. by a1 below)

$A = T$

$A = T$ Goal

- One strategy to find an admissible heuristic is
- to relax the original problem
- One way : remove all the effects that make a variable = F. Action aeffects (B=F, C=T)
- If we find the path from the initial state to the goal using this relaxed version of the actions:

S_0

$B = F$

$B = T$

$C = F$

$C = T$

Heuristics for Forward Planning: ignore delete-list

- the length of the solution is an underestimate of the actual solution length
- Why? In the original problem, one action (e.g. aabove) might undo an already achieved goal (e.g. by a1 below). It would have to be achieved

again

A = T

a1 \rightarrow B = T

A = T **Goal**

\rightarrow B = F

\rightarrow C = T \rightarrow

a

S₀

B = F

C = F

B = T

C = T

Example for ignore-delete-list

- Let's stay in the robot domain • But say our robot has to bring coffee to Bob, Sue, and Steve:
- $G = \{\text{bob_has_coffee}, \text{sue_has_coffee}, \text{steve_has_coffee}\}$
- They all sit in different offices
- Original actions
 - ✓ “pick-up coffee” achieves $\text{rhc} = \text{T}$
 - ✓ “deliver coffee” achieves $\text{rhc} = \text{F}$
- “Ignore delete lists” \Leftrightarrow remove $\text{rhc} = \text{F}$ from “deliver coffee”

- ✓ once you have coffee you keep it
- ✓ Problem gets easier: only need to pick up coffee once, navigate to the right locations, and deliver

Heuristics for Forward Planning: ignore delete-list

But how do we compute the actual heuristics values
for **ignore delete-list** ?

Heuristics for Forward Planning: ignore delete-list

But how do we compute the actual heuristics values for **ignore delete-list**?

- To compute $h(s_i)$, run forward planner with
- s_i as start state
- Same goal as original problem
- Actions without “delete list”
- Often fast enough to be worthwhile

- ✓ Planning is PSPACE-hard (that's really hard, includes NP-hard)
- ✓ Without delete lists: often very fast

Example Planner

- FF or Fast Forward
- Jörg Hoffmann: Where 'Ignoring Delete Lists' Works: Local Search Topology in Planning Benchmarks. J. Artif. Intell. Res. (JAIR) 24: 685-758 (2005)
- Winner of the 2001 AIPS Planning Competition
- Estimates the heuristics by solving the relaxed planning problem with a planning graph method (next class) •

Uses Best First search with this heuristic to find a solution

Final Comment

- You should view Forward Planning as one of the basic planning techniques
- • By itself, it cannot go far, but it can work very well in combination with other techniques, for specific domains
- • See, for instance, descriptions of competing planners in the presentation of results for the 2002 and 2008 planning competition (posted in the class schedule)

Learning Goals for Planning so Far

- Included in midterm
- Represent a planning problem with the STRIPS representation
- Explain the STRIPS assumption
- Solve a planning problem by search (forward planning).
Specify states, successor function, goal test and solution.
- Construct and justify a heuristic function for forward planning