Chapters 8: Planning

DIT410/TIN172 Artificial Intelligence

Peter Ljunglöf
modifed from slides by Poole & Mackworth

1 April, 2015

Outline

1. *Representing actions (8.1)*
   - State-space representation (8.1.1)
   - Feature-based representation (8.1.2)
   - STRIPS representation (8.1.3)

2. *Planning (8.2–8.4)*
   - Forward planning (8.2)
   - Regression planning (8.3)
   - Planning as a CSP (8.4)

What is planning?

- Planning is deciding what to do based on an agent's ability, its goals, and the state of the world.
- Initial assumptions:
  - The world is deterministic.
  - There are no external events outside the control of the robot that change the state of the world.
  - The agent knows what state it is in.
  - Time progresses discretely from one state to the next.
  - Goals are predicates of states that need to be achieved or maintained.
- The aim is to find a sequence of actions to solve a given goal.

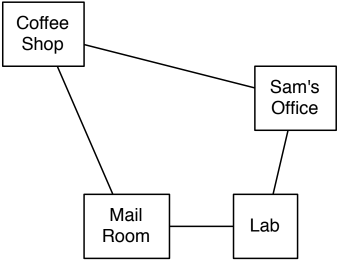
Outline

1. *Representing actions (8.1)*
   - State-space representation (8.1.1)
   - Feature-based representation (8.1.2)
   - STRIPS representation (8.1.3)
2. *Planning (8.2–8.4)*
   - Forward planning (8.2)
   - Regression planning (8.3)
   - Planning as a CSP (8.4)

Actions

- A deterministic action is a partial function from states to states.
- The preconditions of an action specify when the action can be carried out.
- The effect of an action specifies the resulting state.

Delivery robot example



Features:

$RLoc = \{lab, mr, off, cs\}$
– Rob's location

rhc – Rob has coffee

swc – Sam wants coffee

mw – Mail is waiting

rhm – Rob has mail

Actions:

mc – move clockwise

mcc – move counterclockwise

puc – pickup coffee

dc – deliver coffee

pum – pickup mail

dm – deliver mail

Outline

1. *Representing actions (8.1)*
   - State-space representation (8.1.1)
   - Feature-based representation (8.1.2)
   - STRIPS representation (8.1.3)

2. *Planning (8.2–8.4)*
   - Forward planning (8.2)
   - Regression planning (8.3)
   - Planning as a CSP (8.4)

Explicit state-space representation

State	Action	Resulting State
$\langle lab, \neg rhc, swc, \neg mw, rhm\rangle$	mc	$\langle mr, \neg rhc, swc, \neg mw, rhm\rangle$
$\langle lab, \neg rhc, swc, \neg mw, rhm\rangle$	mcc	$\langle off, \neg rhc, swc, \neg mw, rhm\rangle$
$\langle off, \neg rhc, swc, \neg mw, rhm\rangle$	dm	$\langle off, \neg rhc, \neg swc, \neg mw, \neg rhm\rangle$
$\langle off, \neg rhc, swc, \neg mw, rhm\rangle$	mcc	$\langle cs, \neg rhc, swc, \neg mw, rhm\rangle$
$\langle off, \neg rhc, swc, \neg mw, rhm\rangle$	mc	$\langle lab, \neg rhc, swc, \neg mw, rhm\rangle$
...	...	...

This table will have $\#states \times \#actions$

$$= (4 \cdot 2 \cdot 2 \cdot 2 \cdot 2) \times 6 = 384 \text{ rows.}$$

Outline

Feature-based representation of actions

Each action has a:

- **precondition** is a proposition that specifies when the action can be carried out.

For each feature there are:

- **causal rules** that specify when the feature gets a new value, and
- **frame rules** that specify when the feature keeps its value.

Example feature-based representation

Precondition of picking up coffee ($Act = puc$):

$$RLoc = cs \wedge \neg rhc$$

Rules for when the robot has coffee (rhc):

$$\begin{array}{lcll} rhc' & \leftarrow & Act = puc & \text{(causal rule)} \\ rhc' & \leftarrow & rhc \wedge Act \neq dc & \text{(frame rule)} \end{array}$$

Rules for when the robot is in the coffee shop ($RLoc = cs$):

$$\begin{array}{lcll} RLoc' = cs & \leftarrow & RLoc = mr \wedge Act = mc & \text{(causal rule)} \\ RLoc' = cs & \leftarrow & RLoc = off \wedge Act = mcc & \text{(causal rule)} \\ RLoc' = cs & \leftarrow & RLoc = cs \wedge Act \neq cc \wedge Act \neq mcc & \text{(frame rule)} \end{array}$$

Outline

1. *Representing actions (8.1)*
    - State-space representation (8.1.1)
    - Feature-based representation (8.1.2)
    - STRIPS representation (8.1.3)

2. *Planning (8.2–8.4)*
    - Forward planning (8.2)
    - Regression planning (8.3)
    - Planning as a CSP (8.4)

STRIPS representation

Divide the features into:

- primitive features
- derived features – there are rules specifying how they are derived from primitive features

Each action has:

- precondition that specifies when the action can be carried out.
- effect – a set of assignments of values to primitive features that are made true by this action.

The STRIPS assumption:

- every primitive feature not mentioned in the effects is unaffected by the action.

Example STRIPS representation

Pick-up coffee (puc):

precondition: $[RLoc = cs, \neg rhc]$ effect: $[rhc]$

Deliver coffee (dc):

precondition: $[RLoc = \mathit{off}, rhc]$ effect: $[\neg rhc, \neg swc]$

Move clockwise from mail room ($mc(mr)$):

precondition: $[RLoc = mr]$ effect: $[RLoc = cs]$

Move clockwise from office ($mc(\mathit{off})$):

precondition: $[RLoc = \mathit{off}]$ effect: $[RLoc = lab]$

$\vdots$

Outline

1. *Representing actions (8.1)*
   - State-space representation (8.1.1)
   - Feature-based representation (8.1.2)
   - STRIPS representation (8.1.3)

2. *Planning (8.2–8.4)*
   - Forward planning (8.2)
   - Regression planning (8.3)
   - Planning as a CSP (8.4)

Planning

Given:

- A description of the effects and preconditions of the actions
- A description of the initial state
- A goal to achieve

We want to find a sequence of actions that is possible and will result in a state satisfying the goal.

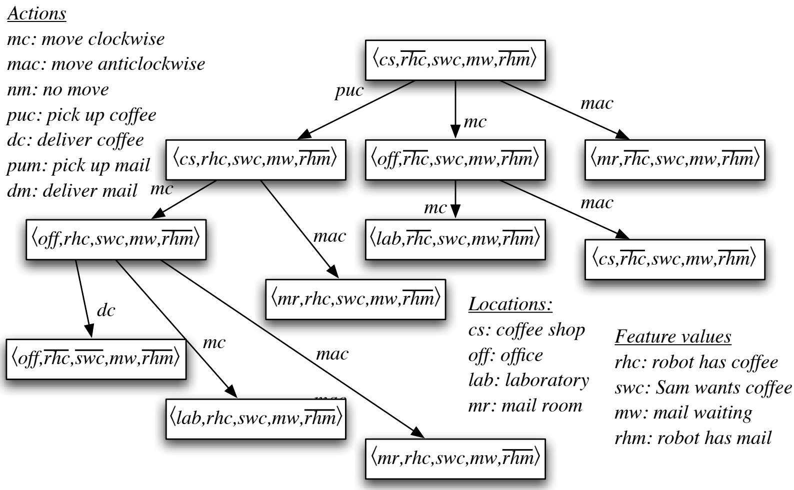
Outline

Forward Planning

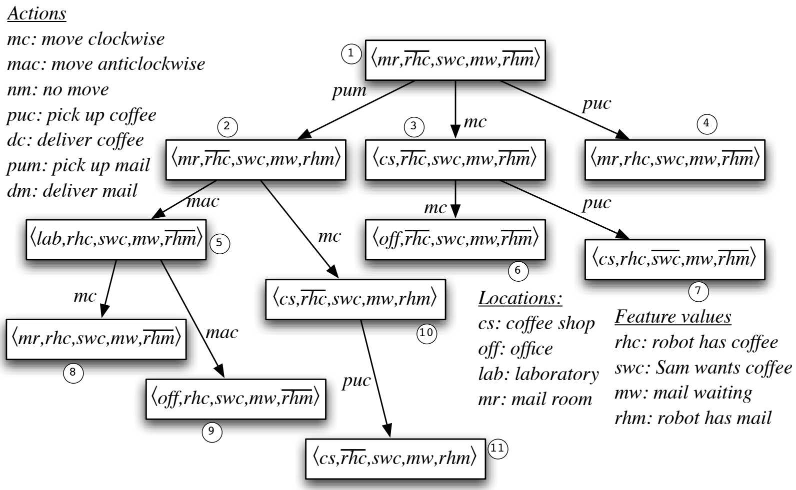
Idea: search in the state-space graph.

- The nodes represent the states
- The arcs (neighbors) correspond to the actions:
  - The arcs from a state s represent all of the actions that are legal in state s.
- A plan is a path from the state representing the initial state to a state that satisfies the goal.

Example state-space graph



What are the errors?



Forward planning representation

- The search graph can be constructed on demand:
  it only constructs reachable states.
- To do a cycle check or multiple path-pruning,
  the planner needs to be able to find repeated states.
- There are a number of ways to represent states:
  - As a specification of the value of every feature
  - As a path from the start state

Improving search efficiency

Forward search can use domain-specific knowledge specified as:

- a heuristic function that estimates the cost of achieving a goal
- domain-specific pruning of neighbors:
    - don't go to the coffee shop unless "Sam wants coffee" is part of the goal and Rob doesn't have coffee
    - don't pick-up coffee unless Sam wants coffee
    - unless the goal involves time constraints, don't do the "no move" action.

Outline

Regression/backward planning

Idea: search backwards from the goal description: nodes correspond to subgoals, and arcs to actions.

- Nodes are propositions: a formula made up of assignments of values to features
- Arcs correspond to actions that can achieve one of the goals
- Neighbors of a node N associated with arc A specify what must be true immediately before A so that N is true immediately after.
- The start node is the goal to be achieved.
- $goal(N)$ is true if N is a proposition that is true of the initial state.

Defining nodes and arcs

- A node N can be represented as a set of assignments of values to variables:

$$[X_1 = v_1, \ldots, X_n = v_n]$$

  - This is a set of assignments you want to hold.
  - *Note:* The assignment is on a *subset* of all variables.
- The last action is one that achieves one of the $X_i = v_i$, and does not achieve $X_j = v'_j$ (where v'_j is different to v_j).
- The neighbor of N along arc A must contain:
  - The prerequisites of action A
  - All of the elements of N that were not achieved by A

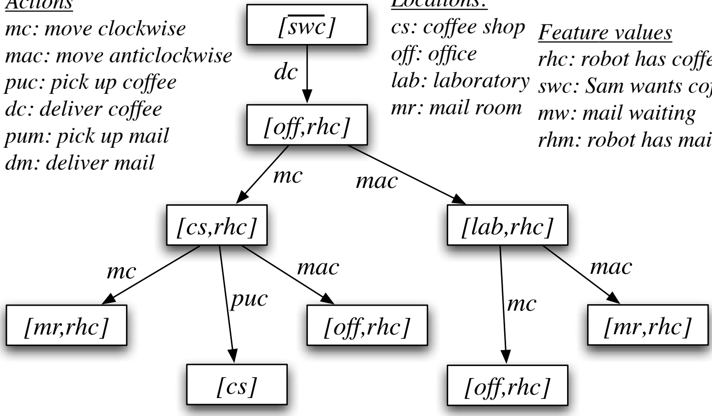
  N must be consistent.

Regression example

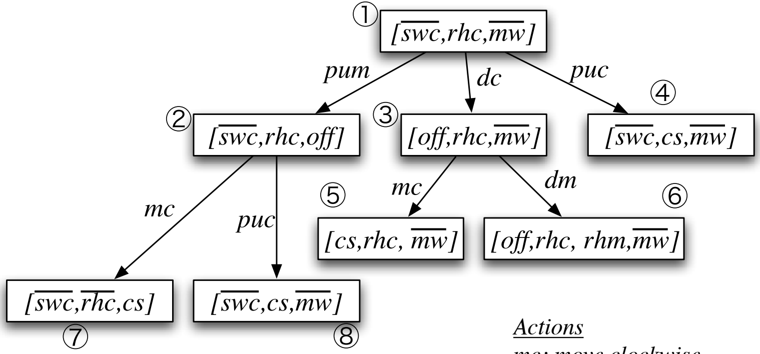
Actions
mc: move clockwise
mac: move anticlockwise
puc: pick up coffee
dc: deliver coffee
pum: pick up mail
dm: deliver mail

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

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



Find the errors



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

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

Actions
mc: move clockwise
mac: move anticlockwise
puc: pick up coffee
dc: deliver coffee
pum: pick up mail
dm: deliver mail

Formalizing arcs using STRIPS notation

Assume that G is $[X_1 = v_1, \ldots, X_n = v_n]$, then

$$\langle G, A, N \rangle$$

is an arc if:

- $X_i = v_i$ is on the effects list of action A (for some $1 \leq i \leq n$)
- $X_j = v'_j$ is not on the effects list for A (for all $1 \leq j \leq n$ and all $v'_j \neq v_j$)
- $N = \{X_k = v_k \mid 1 \leq k \leq n \ \wedge \ X_k = v_k \notin \mathit{effects}(A)\}$
  $\quad \cup \ \mathit{preconditions}(A)$
  and N is consistent
  (in that it does not assign conflicting values to any variable).

Loop detection and multiple-path pruning

- Goal G_1 is simpler than goal G_2 if G_1 is a subset of G_2.
  - It is easier to solve $[cs]$ than $[cs, rhc]$.
- If you have a path to node N have already found a path to a simpler goal, you can prune the path N.

Improving efficiency

- A search can use a heuristic function that estimates the cost of solving a goal from the initial state.
- You can use domain-specific knowledge to remove impossible goals.
    - E.g., it is often not obvious from an action description to conclude that an agent can only hold one item at any time.

Comparing forward and regression planners

- Which is more efficient depends on:
  - The branching factor
  - How good the heuristics are
- Forward planning is unconstrained by the goal (except as a source of heuristics).
- Regression planning is unconstrained by the initial state (except as a source of heuristics)

Outline

Planning as a CSP

- We search over *planning horizons*.
- For each planning horizon, we create a CSP that constrains possible actions and features.
- We also have to factor the actions into *action features*.

Example: Action features

- PUC: Boolean variable, the agent picks up coffee.
- $DelC$: Boolean variable, the agent delivers coffee.
- PUM: Boolean variable, the agent picks up mail.
- $DelM$: Boolean variable, the agent delivers mail.
- $Move$: variable with domain $\{mc, mcc, nm\}$ specifies whether the agent moves clockwise, counterclockwise or doesn't move

CSP variables

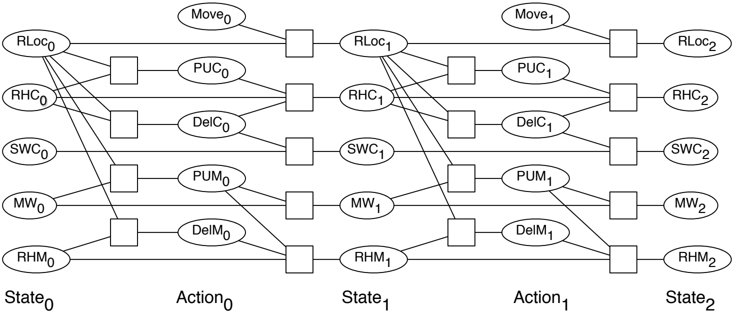
First we choose a planning horizon k:

- Create a variable for each state feature and each time from 0 to k.
- Create a variable for each action feature for each time in the range 0 to $k-1$.

Constraints

- state constraints are constraints between variables at the same time step.
- precondition constraints (between state variables at time t and action variables at time t), specify constraints on what actions are available from a state.
- effect constraints (between state variables at time t, action variables at time t and state variables at time $t+1$), encode the effects of a rule.
- action constraints specify which actions cannot co-occur. Sometimes they are called mutual exclusion or mutex constraints.
- initial state constraints are usually domain constraints on the initial state (at time 0).
- goal constraints constrains the final state to be a state that satisfies the goals that are to be achieved.

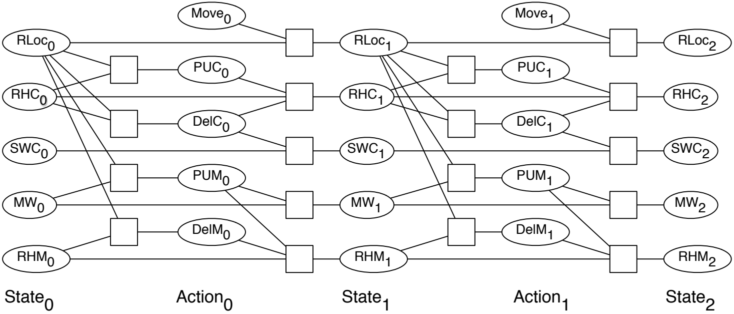
Example CSP for the delivery robot



$RLoc_i$ — Rob's location
RHC_i — Rob has coffee
SWC_i — Sam wants coffee
MW_i — Mail is waiting
RHM_i — Rob has mail

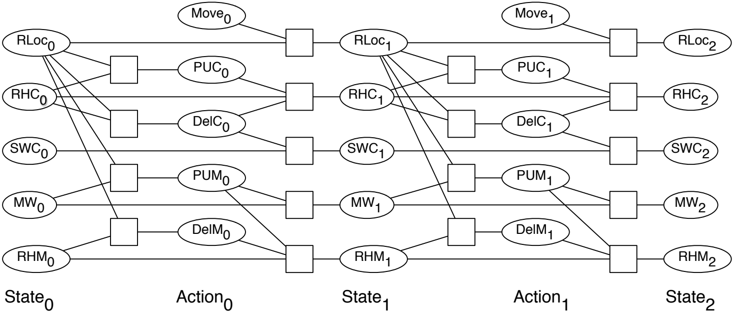
$Move_i$ — Rob's move action
PUC_i — Rob picks up coffee
$DelC$ — Rob delivers coffee
PUM_i — Rob picks up mail
$DelM_i$ — Rob delivers mail

Example precondition constraints



$$
\begin{aligned}
PUC_i &\rightarrow (RLoc_i = cs) \wedge \neg RHC_i \\
DelC_i &\rightarrow (RLoc_i = off) \wedge RHC_i \\
PUM_i &\rightarrow (RLoc_i = mr) \wedge MW_i \\
DelM_i &\rightarrow (RLoc_i = off) \wedge RHM_i
\end{aligned}
$$

Example effect constraints



$$
\begin{aligned}
RHC_{i+1} &\leftrightarrow PUC_i \vee (RHC_i \wedge \neg DelC_i) \\
SWC_{i+1} &\leftrightarrow SWC_i \wedge \neg DelC_i \\
MW_{i+1} &\leftrightarrow MW_i \wedge \neg PUM_i \\
RHM_{i+1} &\leftrightarrow PUM_i \vee (RHM_i \wedge \neg DelM_i)
\end{aligned}
$$