today’s topics:

- partial-order planning
- decision-theoretic planning
Partial Order Planning

- The answer to the problem we ended the last lecture with is to use partial order planning.
- Basically this gives us a way of checking before adding an action to the plan that it doesn’t mess up the rest of the plan.
- The problem is that in this recursive process, we don’t know what the rest of the plan is.
- Need a new representation *partially ordered plans*. 
Representation

Start

LeftShoeOn, RightShoeOn

Finish

Start

Left Sock

LeftSockOn

Left Shoe

LeftShoeOn, RightShoeOn

Finish

Right Sock

RightSockOn

Right Shoe
Partially ordered plans

- *Partially ordered* collection of steps with
  - *Start* step has the initial state description as its effect
  - *Finish* step has the goal description as its precondition
  - *causal links* from outcome of one step to precondition of another
  - *temporal ordering* between pairs of steps

- *Open condition* = precondition of a step not yet causally linked

- A plan is *complete* iff every precondition is achieved

- A precondition is *achieved* iff it is the effect of an earlier step and no possibly *intervening* step undoes it
Plan construction

Start

At(Home)  Sells(HWS,Drill)  Sells(SM,Milk)  Sells(SM,Ban.)

Have(Milk)  At(Home)  Have(Ban.)  Have(Drill)

Finish
Plan construction (2)

- **Start**
  - At(Home)
  - Sells(HWS, Drill)
  - Sells(SM, Milk)
  - Sells(SM, Ban.)

- **After**
  - At(HWS)
  - Sells(HWS, Drill)
  - Buy(Drill)

- **After**
  - At(x)
  - Go(SM)

- **After**
  - At(SM)
  - Sells(SM, Milk)
  - Buy(Milk)

- **After**
  - Have(Milk)
  - At(Home)
  - Have(Ban.)
  - Have(Drill)

- **Finish**
Plan construction (3)

Start

At(Home) Go(HWS)

At(HWS) Sells(HWS,Drill) Buy(Drill)

At(HWS) Go(SM)

At(SM) Sells(SM,Milk) At(SM) Sells(SM,Ban.)

Buy(Milk) Buy(Ban.)

At(SM) Go(Home)

Have(Milk) At(Home) Have(Ban.) Have(Drill)

Finish
Planning process

- Operators on partial plans:
  - *add a link* from an existing action to an open condition
  - *add a step* to fulfill an open condition
  - *order* one step wrt another to remove possible conflicts
- Gradually move from incomplete/vague plans to complete, correct plans
- Backtrack if an open condition is unachievable or if a conflict is unresolvable
function POP (initial, goal, operators) returns plan
    plan ← MAKE-MINIMAL-PLAN(initial, goal)
    loop do
        if SOLUTION?(plan) then return plan
        $S_{\text{need}}, c$ ← SELECT-SUBGOAL(plan)
        CHOOSE-OPERATOR(plan, operators, $S_{\text{need}}, c$)
        RESOLVE-THREATS(plan)
    end loop
end function

function SELECT-SUBGOAL(plan) returns $S_{\text{need}}, c$
    pick a plan step $S_{\text{need}}$ from STEPS(plan)
    with a precondition $c$ that has not been achieved
    return $S_{\text{need}}, c$
end function
POP algorithm, continued

procedure CHOOSE-OPERATOR( \( plan, \) operators, \( S_{\text{need}}, c \) )
    choose a step \( S_{\text{add}} \) from operators or STEPS(\( plan \)) that has \( c \) as an effect
    if there is no such step then fail
    add the causal link \( S_{\text{add}} \leftarrow c S_{\text{need}} \) to LINKS(\( plan \))
    add the ordering constraint \( S_{\text{add}} \prec S_{\text{need}} \) to ORDERINGS(\( plan \))
    if \( S_{\text{add}} \) is a newly added step from operators then
        add \( S_{\text{add}} \) to STEPS(\( plan \))
        add \( \text{Start} \prec S_{\text{add}} \prec \text{Finish} \) to ORDERINGS(\( plan \))
    end if
end procedure
procedure RESOLVE-THREATS( plan )
    for each $S_{\text{threat}}$ that threatens a link $S_i \leftarrow c S_j$ in LINKS(plan) do
        choose either
        \begin{itemize}
        \item Demotion: Add $S_{\text{threat}} \prec S_i$ to ORDERINGS(plan)
        \item Promotion: Add $S_j \prec S_{\text{threat}}$ to ORDERINGS(plan)
        \end{itemize}
        if not CONSISTENT(plan) then fail
    end for each
end procedure
Clobbering

• A *clobberer* is a potentially intervening step that destroys the condition achieved by a causal link. E.g., \(\text{Go(Home)}\) clobbers \(\text{At(Supermarket)}\):

- **Demotion**: put before \(\text{Go(Supermarket)}\)
- **Promotion**: put after \(\text{Buy(Milk)}\)
Properties of POP

• Nondeterministic algorithm: backtracks at *choice* points on failure:
  – choice of $S_{add}$ to achieve $S_{need}$
  – choice of demotion or promotion for clobberer
  – selection of $S_{need}$ is irrevocable

• POP is sound, complete, and *systematic* (no repetition)

• Extensions for disjunction, universals, negation, conditionals

• Can be made efficient with good heuristics derived from problem description

• Particularly good for problems with many loosely related subgoals
Example

"Sussman anomaly" problem

Start State

Clear(x) On(x,z) Clear(y)

PutOn(x,y)

~On(x,z) ~Clear(y)

Clear(z) On(x,y)

Goal State

Clear(x) On(x,z)

PutOnTable(x)

~On(x,z) Clear(z) On(x,Table)

+ several inequality constraints
Example (2)

On(A, B)  On(A, Table)  Cl(B)  On(B, Table)  Cl(C)

On(A, B)  On(B, C)

START

FINISH
Example (3)

On(C,A) On(A,Table) Cl(B) On(B,Table) Cl(C)

On(A,B) On(B,C)

PutOn(B,C)

FINISH
Example (4)

\[ \text{Start} \]

\[ \text{On}(C,A) \quad \text{On}(A,\text{Table}) \quad \text{Cl}(B) \quad \text{On}(B,\text{Table}) \quad \text{Cl}(C) \]

\[ \text{PutOn}(A,B) \]

\[ \text{PutOn}(B,C) \]

\[ \text{On}(A,z) \quad \text{Cl}(B) \]

\[ \text{Cl}(A) \quad \text{On}(A,z) \quad \text{Cl}(B) \quad \text{On}(B,z) \quad \text{Cl}(C) \]

\[ \text{Cl}(B) \]

\[ \text{PutOn}(B,C) \]

\[ \text{On}(B,z) \quad \text{Cl}(C) \quad \text{Cl}(B) \]

\[ \text{FINISH} \]

\text{PutOn}(A,B) \text{ clobbers Cl}(B) \Rightarrow \text{order after PutOn}(B,C) \]
Example (5)

On(A,B)  On(B,C)

START

On(C,A)  On(A,Table)  Cl(B)  On(B,Table)  Cl(C)

On(C,z)  Cl(C)

PutOnTable(C)

Cl(A)  On(A,z)  Cl(B)

PutOn(A,B)

Cl(B)  On(B,z)  Cl(C)

PutOn(B,C)

On(A,B)  On(B,C)

FINISH

PutOn(A,B) clobbers Cl(B) => order after PutOn(B,C)

PutOn(B,C) clobbers Cl(C) => order after PutOnTable(C)
Decision-theoretic planning

- Closed loop planning
- The central question in designing an agent is building it so that it can figure out what to do next.
- That is finding a set of actions which will lead to a goal.
- Previously we studied a traditional approach to planning from AI.
- This was the use of means-ends analysis along with the STRIPS representation.
• STRIPS:
  – add condition;
  – delete condition; and
  – precondition.

• Algorithms use:
  – Use precondition to decompose goals;
  – Use add condition to select actions; and
  – Use delete condition to constrain order on actions.
• The main limitations of this approach are:
  – Efficiency (doesn’t scale)
  – Robustness
• The second of these is what interests us here.
• The problem is:
  – Plan is linear
  – Planning is separated from acting
  – Actions are non-deterministic
• Though partial-order planning is an improvement on simple means-ends analysis, it still can’t cope with non-determinism.
• One way of thinking about this is in terms of closed loop planning.
• Classical planning has:

\[ \text{World} \rightarrow \text{Agent} \rightarrow \text{Plan} \]

- While close loop planning has actions which are dependent on what is observed in the world:

\[ \text{World} \rightarrow \text{Agent} \rightarrow \text{Plan} \rightarrow \text{World} \]

• Clearly this is the kind of planning that better fits agents.
Conditional planning is one approach to closed-loop planning. Conditional plans are allowed to have branches and loops where control choices depend upon observations.

For example:

1. pick up block $A$
2. while block $A$ not held
   pick up block $A$.
3. if block $C$ clear
   put block $A$ on block $C$.
4. else clear block $C$.

However, the situation gets more complex with unreliable sensors.
• To deal with unreliable sensors we need to bring in decision theory.
• (Just as we did to take account of dice rolls in game playing).
• A problem with using classical decision theory in the context of intelligent agents is that it is a one-shot process.
• The process only takes into account the current state and the one the decision will lead to.
• This is fine if the next state is the goal state.
• In contrast, what we are often interested in is determining a sequence of actions which take us through a series of states, especially when the choice of actions varies from state to state.
• We do this through the use of *decision theoretic planning* models.

• We will cover two closely related types of these models here:
  – Markov decision processes.
  – Partially observable Markov decision processes.

• Both are close in many ways to the kind of search models we studied earlier.
• The big change is that actions can have more than one outcome.
• So we start by considering planning as search.
Planning as search

• The earliest search models we looked at are a form of planning.
• In the sheep and dogs example, a solution was:
  – A sequence of actions;
  – Which led to a goal
• This is just a plan.
• Adding in a heuristic function gives us an idea of optimality:
• An optimal plan is:
  – A sequence of actions;
  – Which leads to a goal;
  – With minimum cost.
We can describe a state space search model as:

- a state space $S$;
- an initial state $s_0$;
- a set of actions, $A(s) \subseteq A$, applicable in each state $s \in S$;
- transition function $f(s, a)$ for $s \in S$ and $a \in A$;
- action costs $c(a, s) > 0$; and
- a set of goal states $G \subseteq S$
• This gives us a problem space that looks like:

![Diagram of a problem space showing a tree structure with nodes and arrows indicating paths.]

• A solution is a path through this space from initial state to a goal state.
There are lots of ways of searching this space.

One simple way is greedy search:

1. Evaluate each action \( a \) which can be performed in the current state:

\[
Q(a, s) = c(a, s) + h(s_a)
\]

where \( s_a \) is the next state.

2. Apply action \( a \) that minimises \( Q(a, s) \);

3. If \( s_a \) is the goal, exit
   else \( s := s_a \), goto 1.

This just picks the cheapest move at each point.
• This is a simple approach that uses little (and constant) memory.
• It can be easily adapted to give a closed-loop version:
  – Instead of $s_a$ being the state we expect to get, make it the one we observe.
• Like any depth first approach, it isn’t optimal.
• It might not even find solutions.
• (But we know how to use learning to ensure that it gets better over time).
Markov decision processes

- So far, there is nothing really new here.
- But it is only a small step to a much better representation.
- In a non-deterministic environment, we don’t have a simple transition function.
- Instead an action can lead to one of a number of states.
- When we can tell which state we are in, then we have a Markov decision process (MDP)
• An MDP has the following formal model:
  – a state space $S$;
  – a set of actions, $A(s) \subseteq A$, applicable in each state $s \in S$;
  – transition probabilities $\Pr_a(s' \mid s)$ for $s, s' \in S$ and $a \in A$;
  – action costs $c(a, s) > 0$; and
  – a set of goal states $G \subseteq S$
• Thus for each state we have a set of actions we can apply, and these take us to other states with some probability.
• We don’t know which state we will end up in, but we know which one we are in after the action (we have *full observability*).
• This gives us a problem space that looks like:

![Diagram](image)

• A solution is now choice of action in every possible state that the agent might end up in.
• We can think of this solution as a function $\pi$ which maps states into applicable actions, $\pi(s_i) = a_i$.
• This function is called a policy.
• What a policy allows us to compute is a probability distribution across all the trajectories from a given initial state.
• This is the product of all the transition probabilities, $\Pr_{a_i}(s_{i+1} \mid s_i)$, along the trajectory.
• Goal states are taken to have no cost, no effects, so that if $s \in G$:
  - $c(a, s) = 0$
  - $\Pr(s \mid s) = 1$
We can then calculate the expected cost of a policy starting in state $s_0$.

This is just the probability of the policy multiplied by the cost of traversing it:

$$\sum_{i=0}^{\infty} c(\pi(s_i), s_i)$$

An optimal policy is then a $\pi^*$ that has minimum expected cost for all states $s$.

As with the search version of the problem, we can solve this by searching, albeit through a much larger space.

Later we will look at ways to do this search.
Partially observable MDPs

- Full observability is a big assumption (it requires an accessible environment). Much more likely is *partial observability*.
- This means that we don’t know what state we are in, but instead we have some set of beliefs about which state we are in.
- We represent these beliefs by a probability distribution over the set of possible states.
- These probabilities are obtained by making observations.
- The effect of observations are modelled as probabilities $\Pr_a(o \mid s)$, where $o$ are observations.
Formally a POMDP is:

- a state space $S$;
- a set of actions, $A(s) \subseteq A$, applicable in each state $s \in S$;
- transition probabilities $Pr_a(s' \mid s)$ for $s, s' \in S$ and $a \in A$;
- action costs $c(a, s) > 0$;
- a set of goal states, $G$;
- an initial belief state $b_0$;
- a set of final belief states $b_F$;
- observations $o$ after action $a$ with probabilities $Pr_a(o \mid s)$
So we have a situation which looks like:

This is just an MDP over belief states.
• The goal states of an MDP are just replaced by, for example, states in which we are pretty sure we have reached a goal:

\[ \sum_{s \in G} b(s) > 1 - \epsilon \]

• We solve a POMDP by looking for a function which maps belief states into actions, where belief states \( b \) are probability distributions over the set of states \( S \).

• Given a belief state \( b \), the effect of carrying out action \( a \) is:

\[ b_a(s) = \sum_{s' \in S} \Pr_a(s' \mid s) b(s') \]
If we carry out $a$ in $b$ and then observe $o$, we get to state $b^o_a$:

$$b^o_a(s) = \frac{\Pr_a(o \mid s) b_a(s)}{\sum_{s' \in S} \Pr_a(o \mid s') b_a(s')}$$

The term on the bottom is the probability of observing $o$ after doing $a$ in $b$.

Thus actions map between belief states with probability:

$$b_a(o) = \sum_{s' \in S} \Pr(o \mid s') b_a(s')$$

and we want to find a trajectory from $b_0$ to $b_F$ at minimum cost.
Summary

• This lecture has looked at two more advanced approaches to planning:
  – partial order planning
  – decision theoretic planning
• partial order planning requires a new way of looking at the world, but the payoff is a more robust approach.
• we also looked at the POP algorithm, . . .
• . . . and saw how it could solve the Sussman anomaly.
• Starting from the notion of planning as search, we introduced the Markov decision process (MDP) representation.
• A solution to an MDP is a policy, a choice of what action to take in every state.