EECS 391: Introduction to AI

Soumya Ray
Website: http://engr.case.edu/ray_soumya/eecs391_sp16/
Email: sray@case.edu
Office: Olin 516
Office hours: M 12:30-2pm
Announcements

• Read Chapter 13 (Basic Probability)
Today

• Automated Planning (Ch 10)
The Planning Problem

• Given:
  – An initial state of the world, described as a set of logical facts
  – A set of goal states, described as a set of logical facts
  – A set of actions, also described in logic

• Find a sequence of actions that will move the world from the initial state to the final state
  – This sequence is called a plan
  – Often also try to optimize some criteria
Blocks World

Task: Starting with initial configuration of blocks, produce a desired goal configuration by moving block around.
Representing States in STRIPS

- States in STRIPS are conjunctions of unnegated, ground, function-free literals
  - All conditions that hold in that state
  - $Block(A)$, $Block(B)$, $On(A,B)$, $On(B, Table)$, $GripperEmpty$
  - The “Closed World Assumption” is used
Representing Goals in STRIPS

• Goals are *conjunctons* of unnegated, ground, function-free literals

• Goals may not fully determine a state of the world
  – In this case, the goal is any state where these literals hold

• Example: $On(A,E), On(B,D)$
Representing Actions in STRIPS

• Want to represent an action of picking up a block from the table

\[ \text{Pickup\_from\_Table}(x) \]

**Preconditions:** Block\((x)\), GripperEmpty, Clear\((x)\), On\((x,\text{Table})\)

**Add List:** Holding\((x)\)

**Delete List:** GrippeEmpty, On\((x,\text{Table})\)

“Applicability”: action can be used at a state iff its preconditions are satisfied
Representing Actions in STRIPS

• An “action schema” represents a non-ground action using three parts:
  – The action name and parameter (variable) list
  – The *preconditions*: a list of unnegated function-free (non-ground) literals. Any variables in this list must be parameters to the action.
  – The *effects*: a conjunction of function-free literals describing how the state changes.
Add and Delete Lists

• Often, the unnegated literals in the action effects are collected into an “ADD” list, and the negated literals are collected into a “DELETE” list

  – Idea: Starting with initial state, to get result of applying action, add the literals in ADD list and delete the literals in the DELETE list
The STRIPS assumption

• Every possible effect of actions are listed

• If a literal does not appear in the effects list, it is *unchanged* in the resulting state
  – Solves the “frame problem” in situation calculus
Restrictions in STRIPS

- States are described by unnegated ground function-free literals
- CWA
- Ground conjunctive goals
- Conjunctive effects
Example: Blocks World

Table

B  C  A
Example

• Init($\text{On}(A,\text{Table}) \land \text{On}(B,\text{Table}) \land \text{On}(C,\text{Table})$ $\land \text{Block}(A) \land \text{Block}(B) \land \text{Block}(C) \land \text{Clear}(A) \land \text{Clear}(B) \land \text{Clear}(C)$)

• Goal($\text{On}(A,B)$)

• Action($\text{MoveToTable}(b,x)$),
  
  – Preconditions($\text{On}(b,x) \land \text{Clear}(b) \land \text{Block}(b) \land \text{Block}(x) \land \text{GripperEmpty}$)
  
  – AddEffects($\text{On}(b,\text{Table}) \land \text{Clear}(x)$)
  
  – DelEffects($\text{On}(b,x)$)
Planning Algorithms

• Given a STRIPS representation of a classical planning problem, how do we solve it?
  – Since the world is static, deterministic, fully observable, we could use search
  – Remember that in this case, the search algorithm is actually performing logical inference
Kinds of Search for Planning

• Search algorithms for classical planning fall into two categories
  – “State space planners”: States of the search problem are states of the world; search operators are actions of the world
  – “Plan space planners”: States of the search problem are partial plans; search operators are modifications to the current partial plan
Forward State-Space Search

• “Progression” planning

• Setup:
  – States=world states (in STRIPS)
  – Initial state=given
  – Operators=applicable actions (in STRIPS)
  – Goal test=given (in STRIPS)
  – Operator costs=unit (minimize number of actions)
Makespan

• Typically, in classical planning, we are interested in minimizing the duration of the plan
  – Equivalent to the number of actions in the current setup
  – This is called the makespan

• Nonclassical planning allows arbitrary plan metrics to be minimized
Forward State-Space Search

• We could apply any search algorithm, e.g. A*

• The key differences are:
  – Only applicable actions need to be explored at a state
  – Getting the next state is done through the STRIPS specification of states and actions
  – Heuristics are based on planning ideas
Search Heuristics

• From any state, want to estimate the number of actions to search termination admissibly

• Two possibilities:
  – Relax the planning problem
  – Consider subproblems
Relaxed Plans

• There are different ways to arrive at a less constrained planning problem

• One way is to remove all DELETE effects from STRIPS actions
  – This is admissible (why?)
  – To estimate this cost, need to run an internal planning loop; but this is usually very fast
Subproblems

• The goal is a conjunction of literals

• We can generate subproblems by just considering a single literal at a time
  – “Subgoal Independence” (admissible)

• Combine with max, as usual
Total Order Plans

• In a Total Order plan, every pair of actions $A_1$ and $A_2$ has a *temporal ordering constraint*
  
  – Either $A_1$ is done first, or $A_2$
  
  – Forward state space planners produce plans like this
Partial Order Plans

• In many situations, actions do not have to be done in order
  – Might be trying to achieve unrelated things

• This creates a *partial order* plan: a plan with some actions that have no temporal ordering constraints between them
  – i.e. there is some $A_1, A_2$ so that $A_1$ does not have to be completed before $A_2$ and $A_2$ does not have to be completed before $A_1$ for the plan to succeed
Blocks World

Goal: \( On(A,B), On(C,D) \)

A dummy action with no preconditions and effect==initial state

Start

End

Move(A,Table,B)  Move(C,Table,D)

A dummy action with no effects and preconditions==goal
Partial Order Plans

• Represented as a set of actions and ordering constraints \((A < B)\)

• Partial Order plans have two advantages over total order plans
  – Flexibility when executing the plan
  – Action Parallelism
Finding a POP

• To find POPs, we will perform a plan-space search

• The states of the search space will be incomplete POPs, augmented with some bookkeeping information

• Starting with an empty POP (only “Start” and “End”), we will add actions and ordering constraints until we have a complete POP (or fail)
States of the POP Search Space

• A state will have
  – The incomplete POP (list of actions and ordering constraints)
    • Ordering constraints can’t introduce cycles
  – A list of open conditions (initially the goal)
    • Open condition: A literal that the plan needs to make true that is not currently the effect of some action in the plan
    • The termination states of the search are those where this list is empty
  – A list of causal links
Causal Links

• Suppose an action $B$ in the incomplete POP has a precondition $p$
• Suppose the planner adds an action $A$ to the POP that has $p$ as an effect
• The *cause* for adding $A$ is so it fulfils $p$ for $B$
• This is indicated in the POP by adding a *causal link*, denoted $A \xrightarrow{p} B$
Conflict and Threats

• Suppose the planner has just added $A$ and a causal link $A \rightarrow B$

• Suppose there is an action $C$ in the plan that has $\neg p$ as an effect and could come after $A$ and before $B$

• We say that $C$ conflicts with, or threatens, the causal link $A \rightarrow B$
Consistent Plans

• A consistent plan is a POP where the ordering constraints have no cycles and there are no conflicts with causal links

• So a solution is a consistent plan with no open conditions
POP Search Algorithm

• Initial State: Plan=$(Start, End, Start< End)$, open conditions=preconditions of $End$ (goal), no causal links

• Search operators: Pick an open condition, and an action to satisfy it; generate next state (POP)

• Goal test: empty list of open conditions
Generating the Next State

• Suppose we pick condition $p$ on action $B$ to satisfy, using action $A$
  – $A$ might be an action already in the plan
• To generate the next state:
  – Remove $p$ from list of open conditions
  – Add $A$ to list of actions in POP
  – Add ordering constraints $Start < A, A < B, A < End$
  – Add $A$’s preconditions to list of open conditions
  – Add a causal link $A \rightarrow B$
  – Resolve conflicts/threats if any
Conflict Resolution

• Conflicts can arise between
  – The new causal link and existing actions
  – The new action and existing causal links

• To resolve a conflict, add ordering constraints
  – Suppose $C$ conflicts with $A \rightarrow B$
  – Then add either $C < A$ or $B < C$ (assuming no cycles)
  – This is a branch point in the search
Example

- **Init:** \(\text{At}(\text{Flat,Axle}), \text{At}(\text{Spare,Trunk})\)
- **Goal:** \(\text{At}(\text{Spare,Axle})\)
- **Remove(\text{Spare,Trunk})**
  - PRE: \(\text{At}(\text{Spare,Trunk})\)
  - EFF: \(\neg \text{At}(\text{Spare,Trunk}), \text{At}(\text{Spare,Ground})\)
- **Remove(\text{Flat,Axle})**
  - PRE: \(\text{At}(\text{Flat,Axle})\)
  - EFF: \(\neg \text{At}(\text{Flat,Axle}), \text{At}(\text{Flat,Ground})\)
- **PutOn(\text{Spare,Axle})**
  - PRE: \(\text{At}(\text{Spare,Ground}), \neg \text{At}(\text{Flat,Axle})\)
  - EFF: \(\neg \text{At}(\text{Spare,Ground}), \text{At}(\text{Spare,Axle})\)
- **LeaveOvernight**
  - PRE:
  - EFF: \(\neg \text{At}(\text{Spare,Trunk}), \neg \text{At}(\text{Spare,Ground}), \neg \text{At}(\text{Spare,Axle}), \neg \text{At}(\text{Flat,Axle}), \neg \text{At}(\text{Flat,Ground})\)
Example

Start
At(Flat,Axle), At(Spare,Trunk)

At(Spare, Trunk)
Remove(Spare,Trunk)
¬At(Spare, Trunk), At(Spare, Ground)

At(Spare, Ground), ¬At(Flat, Axle)
PutOn(Spare, Axle)
¬At(Spare, Ground), At(Spare, Axle)

At(Spare, Axle)
End

LeaveOvernight
¬At(Flat, Axle), ¬At(Spare, Ground), ¬At(Spare, Trunk)…
Example

Start

\[ \text{At(Flat, Axle), At(Spare, Trunk)} \]

\[ \text{At(Spare, Trunk)} \]
Remove(Spare,Trunk)
\[ \neg \text{At(Spare, Trunk), At(Spare, Ground)} \]

\[ \text{At(Flat, Axle)} \]
Remove(Flat,Axle)
\[ \neg \text{At(Flat, Axle), At(Flat, Ground)} \]

\[ \text{At(Spare, Ground), } \neg \text{At(Flat, Axle)} \]
PutOn(Spare,Axle)
\[ \neg \text{At(Spare, Ground), At(Spare, Axle)} \]

\[ \text{At(Spare, Axle)} \]
End