Today

- Exam review
- Classical Planning

Background

- Logical planning did not include a heuristic function
- Even moving forward one step in the world is relatively expensive
- Compare to custom sliding-tile puzzle:
  - Just swap two elements in an array
- Perhaps a simpler representation is required

Undergrad Average: 82 Stdev: 11.5
PDDL - states

- Factored representation of the world
- World is collection of variables
- Each variable is true or false
- All names are unique
- No explicit negation
- Anything not mentioned is false
- Illegal (variables, negation, functions):
  - \text{At}(x, y), \neg \text{Poor}, \text{At}(\text{Father}(Fred), Sydney)

The frame problem

- Any logic language must represent what changes when actions take place
- Some languages assume everything changes
- Anything that doesn’t change must be re-derived
- PDDL assumes that most things stay the same

Actions in PDDL

- Sample action:
  - \text{Action}(\text{Fly}(P_1, \text{SFO}, \text{JFK}),
    \text{Precond: } \text{At}(P_1, \text{SFO}) \land \text{Plane}(P_1) \land \text{Airport (SFO)} \land \text{Airport(JFK)}
    \text{Effect: } \neg \text{At}(P_1, \text{SFO}) \land \text{At}(P_1, \text{JFK}))

- Can generalize into an action scheme

Actions in PDDL

- Sample action schema:
  - \text{Action}(\text{Fly}(p, \text{from}, \text{to}),
    \text{Precond: } \text{At}(p, \text{from}) \land \text{Plane}(p) \land \text{Airport (from)} \land \text{Airport(to)}
    \text{Effect: } \neg \text{At}(p, \text{from}) \land \text{At}(p, \text{to}))

- Variables are universally quantified
Applying Actions in PDDL

• We can apply an action if its preconditions are entailed by the KB
  • $a \in \text{ACTIONS}(s) \iff s \models \text{PRECOND}(a)$

• $\text{RESULT}(s, a) = (s - \text{DEL}(a)) \cup \text{ADD}(a)$
  • Simply remove the fluents in the delete list
  • Add the fluents from the add list
  • All fluents in a state must be grounded (ie no variables)

Cost of applying actions

• Assume an action has $v$ variables
• Assume there are $k$ ground objects in the world
• $O(v^k)$ possible actions can be applied

Goals

• The goal is just a list of fluents
  • When they are true, the goal is reached

Example Problem

• $\text{Init(At(C1, SFO) \land At(P1, SFO) \land At(C2, JFK) \land At(P2, JFK) \land Cargo(C1) \land Cargo(C2) \land Plane(P1) \land Plane(P2) \land Airport(JFK) \land Airport(SFO))}$
• $\text{Goal(At(C1, JFK) \land At(C2, SFO))}$

• $\text{Action(Load(c, p, a)}$
  • $\text{Pre: At(c, a) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)}$
  • $\text{Effect: \neg At(c, a) \land In(c, p)}$
• Action unload?
• Action fly?
Class Problem

- “Implement” Represent 3-peg towers of hanoi
- 4 disks
- Disks must be ordered largest to smallest

Search in Planning

- Forward Search
  - Look for actions that can be applied to each state
  - Continue forward until goal is reached
- Backwards Search
  - Look for actions that achieve one of the goal fluents
  - Action must not delete one of the goal fluents
    - eg if goal is to have money and to own something, cannot *Buy* as last action if it takes away money

Heuristics for Planning

- Without a heuristic, finding a goal is too expensive
  - \(2^n\) or \(3^n\) states with \(n\) fluents
- Where do heuristics come from?
- How can we relax the planning problem?

Heuristic: Ignore Preconditions

- Can apply any move at any time
  - Might have to ignore delete effects
  - May not have “legal” state representation
- Test on STP
  - Action(Slide(t, s1, s2)
    - Precond: On(t, s1) \& Tile(t) \& Blank(s2) \& Adjacent(s1, s2)
    - Effect: On(t, s2) \& Blank(s1) \& ¬On(t, s1) \& ¬Blank(s1))
Heuristic: Ignore delete lists

- Apply actions as normal
- Do not delete items from the original state
- Fluents in the state monotonically increase
- May not have “legal” state representation
- STP?

Pattern Databases

- Previous two approaches change the number of actions, not the number of states
- Pattern databases can also be used in planning
  - Special case of other approaches
  - Trick is to choose the right abstraction to get good heuristic values

Planning Graph

- Compute possible fluents at each depth
- Compute actions that *might* be able to be applied
- Mutual exclusions represent what we know cannot occur at the same time at this level of the graph
- Can be used as a heuristic for search
Homework: 10.2