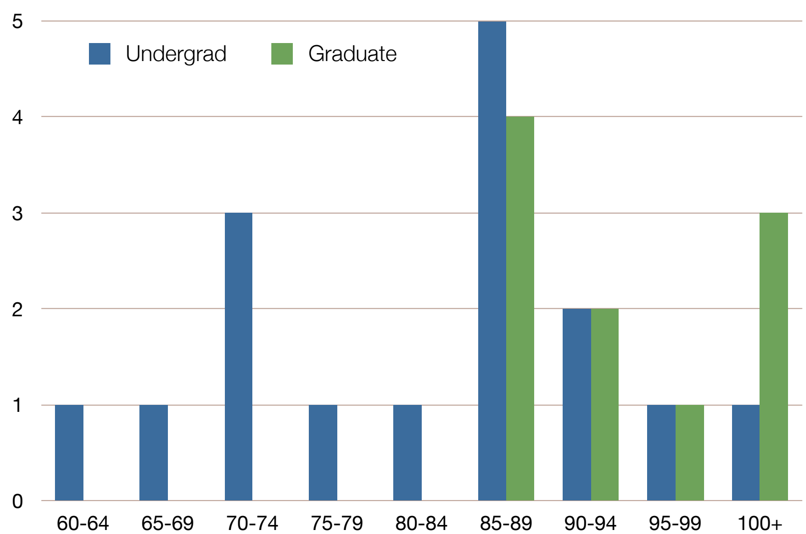
Introduction to Artificial Intelligence
COMP 3501 / COMP 4704-4
Lecture 10: Planning

Prof. Nathan Sturtevant
JGH 318

Today

- Exam review
- Classical Planning



Undergrad Average: 82 Stdev: 11.5

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

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):
  - At(*x*, *y*), ¬*Poor*, At(*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:
  - Action(Fly(P_1, SFO, JFK),

    Precond: At(P_1, SFO) $\wedge$ Plane(P_1) $\wedge$ Airport (SFO) $\wedge$ Airport(JFK)

    Effect:¬At(P_1, SFO) $\wedge$ At(P_1, JFK))
- Can generalize into an action scheme

Actions in PDDL

- Sample action schema:
  - Action(Fly(*p*, *from*, *to*),

    Precond: At(*p*, *from*) $\wedge$ Plane(*p*) $\wedge$ Airport (*from*) $\wedge$ Airport(*to*)

    Effect:¬At(*p*, *from*) $\wedge$ At(*p*, *to*))
- Variables are universally quantified

Applying Actions in PDDL

- We can apply an action if its preconditions are entailed by the KB
  - $a \in \text{ACTIONS}(s) \Leftrightarrow s \models \text{PRECOND}(a)$
- $\text{RESULT}(s, a) = (s - \text{DEL}(a)) \cup \text{ADD}(a)$
  - 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

- Init(At(C1, SFO) ∧ At(P1, SFO) ∧ At(C2, JFK) ∧ At(P2, JFK) ∧ Cargo(C1) ∧ Cargo(C2) ∧ Plane(P1) ∧ Plane(P2) ∧ Airport(JFK) ∧ Airport(SFO))
- Goal(At(C1, JFK) ∧ At(C2, SFO))
- Action(Load(c, p, a)

  Pre: At(c, a) ∧ At(p, a) ∧ Cargo(c) ∧ Plane(p) ∧ Airport(a)

  Effect: ¬At(c, a) ∧ 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) $\wedge$ Tile(t) $\wedge$ Blank(s2) $\wedge$ Adjacent(s1, s2)

    Effect: On(t, s2) $\wedge$ Blank(s1) $\wedge$ $\neg$On(t, s1) $\wedge$ $\neg$Blank(s1))

Heuristic: Ignore delete lists

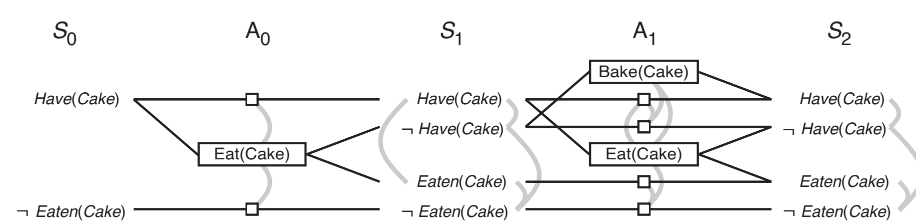
- Apply actions as normal
  - Do not delete items from the original state
  - Fluents in the state monotonically increase
  - May not have “legal” state representation
- STP?

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