Planning

Agenda

- Building on logic to form plans
- How do we choose from a number of possible actions:
  - to bring us closer to the goal?
  - in a more "intelligent“ manner than search?

Quick Primer on First-Order Logic

- Predicates:
  - assume that Spot and Fido are dogs
  - then the predicate, Dog(x)
    returns TRUE if x is Spot or Fido

Motivation

- previously considered decision-making problems from the perspective of search and game-playing
  - Given current state, enumerate all possible future states
  - Pick best action to execute from current state
- when does this make sense?

Grocery Shopping as Search

"from home, get milk, some bananas, and a cordless drill"

- assume we will use search technique with heuristic: minimize the number of items we have not yet acquired
- how to choose best operator? may be thousands!
- before agent can purchase anything, it has to get to the store, but how does a search technique know this?

A Better Way (sometimes)

- many actions are obviously useless or unproductive
- can be quite expensive to examine all of these
- what if we know outcome of actions?

planning = find a sequence of actions that achieves a goal when performed in a given state

Planning Domain Definition Language

- states: conjunction of positive, functionless atoms
  - e.g., *Poor* ∧ *Unknown* or
    At($Truck_1$, *Melbourne*)
- initial state: specifies everything that is true in the world (everything else assumed false)
- action schema: describes what changes, using a subset of first-order logic
- goals: represented by conjunctions of literals that may contain variables
  - e.g., *At(p,SFO)* ∧ *Plane(p)*

Comparison of Actions

in search

- described by state transitions
- must be considered in-order

in planning

- described by:
  - **preconditions**: what must be true before action can be performed
  - **effects**: what must be true (what changed) after action is executed
- can be considered in any order

Action Schema

Example: for flying a plane from one location to another:

Action (*Fly (p, from, to),*

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

EFFECT: $\neg$*At(p,from)* $\wedge$ *At(p,to))*

- every literal not mentioned in EFFECT remains unchanged
- effects divided into positive literals, ADD(a) & negative literals, DEL(a)
- result of performing action a in state s:
  RESULT(s,a) = (s – DEL(a)) $\cup$ ADD(a)

Forward State-Space Search (Progression) Planners

- search forward from initial state
- determine which operators apply using preconditions
- use effects lists to compute new state

Algorithm

procedure PROGRESSION-PLAN(*s, plan*)

  for some possible operator, α_ι

    u ← result of UNIFY(*s,* preconditions of α_i)

    if unification step succeeds then

      add α_i to *plan*

      apply substitution list of *u* to DeleteList(α_i) and AddList(α_i)

      t ← *s*

      for each d_i in DeleteList(α_i) delete d_i from *t*

      for each a_i in AddList(α_i) add a_i to *t*

      if GOAL-TEST (*t*) succeeds then return *plan*

  return PROGRESSION-PLAN (*t, plan*)

Blocks World Action Schema

move block b from x to y:

Action(Move(b,x,y)

PRECOND: *On(b,x)* $\wedge$ *Clear(b)* $\wedge$ *Clear(y),*

EFFECT: *On(b,y)* $\wedge$ *Clear(x)* $\wedge$ $\neg$*On(b,x)* $\wedge$ $\neg$*Clear(y))*

move block b from x to the table:

Action(MoveToTable(b,x)

PRECOND*: On(b,x)* $\wedge$ *Clear(b),*

EFFECT: *On(b,Table)* $\wedge$ *Clear(x)* $\wedge$ $\neg$*On(b,x))*

Practical Considerations

- main problem: often have huge search space because of branching factor
- not practical for real-world problems

Backward State-Space (Regression) Planners

- search backwards from goal state to initial state
- advantage: consider only relevant actions; i.e., those that achieve one of the conjuncts of the goal
- significantly decreases branching factor

Regression Planner

procedure REGRESSION-PLAN (*t*, *plan*)
  for some possible operator, α
    if current state contains ≥1 literal *L*, that unifies with a member a_i of AddList(α)
      add α to head of *plan*
      u ← result of UNIFY(*L*, a_i)
      p' ← apply substitution list of *u* to Preconditions(α)
      t' ← apply substitution list of *u* to terms of *t*
      regress each member *m* of *t'* through a as follows:
        if *m* is in AddList(α) then True
        else if *m* is in DeleteList(α) then False
        else leave *m* as is
      previous state *s* ←UNION (*p', t'*)
     if *s* = INITIAL-STATE then return *plan*
    return REGRESSION-PLAN (*s*, *plan*)

Partial Order Planning (POP)

start with initial plan

- consists only of Start and Finish states
- at each iteration, add one more step
- if inconsistent, backtrack

only adds steps that achieve unachieved preconditions

Homework

- Readings:
  - Ch 15.3
- Video:
  - Eisner lecture on Hidden Markov Models
- Assignment #1 submission
  - due September 30 on Moodle