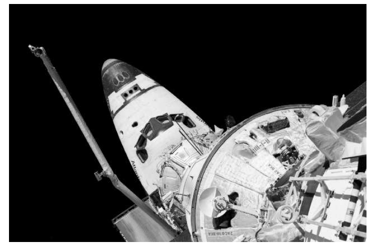
Planning I

CPS 170
Ron Parr

What Is Planning – An Example



Space shuttle arm is currently controlled by a highly trained human.

Planning Application

- Remove human from the control loop
- Specific goals for system:
  - Rearrange items in cargo bay
  - Connect space station pieces
- Assuming mechanical engineering issues can be resolved:
  - Arm could work while astronauts sleep
  - Complicated training could be eliminated

Characterizing Planning Problems

- Start state (group of states)
- Goal – almost always a group of states
- Actions
- Plan: A sequence of actions that is guaranteed to achieve the goal.
- So, how is this different from search?

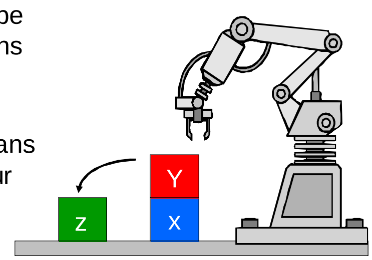
Like everything else, we can view planning as search.

What makes planning special?

- States typically specified by a set of relations or propositions:
  - On(solar_panels, cargo_floor)
  - arm_broken
- Typically we make a closed world assumption:
  - We only state that which is true
  - All else is assumed false
  - Why?

Planning With Logic

- Need to describe effects of actions with logic
- Test for the existence of plans that achieve our goals
- Difficulties
  - Consistency
  - Frame problem



Specifying Actions

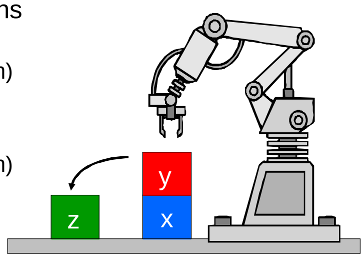
- Describing action effects is tricky
- Need a compact way of describing what changes and what does not change
  - The union of these is everything in the world
  - Can't afford to enumerate these for every action
- Standard approach: use STRIPS rules
  - Preconditions, add-list, delete-list

STRIPS

- Closed world assumption
- Preconditions specify when action is valid
- Think of the world as a database
  - Add list specifies what new things are true after taking the action (add to DB)
  - Delete list specifies what things are no longer true (delete from DB)

move(obj,from,to)

- Preconditions
  - clear(obj)
  - on(obj,from)
  - clear(to)
- Delete list
  - on(obj,from)
  - clear(to)
- Add list
  - on(obj,to)
  - clear(from)



move(y,x,z)

Limitations of STRIPS

- Strips assumes that a small number of things change with each action
  - Dominoes
  - Pulling out the bottom block from a stack
- Preconditions and effects are conjunctions
- No quantification

Planning Actions vs. Search Actions

- Plan actions are really action schemata
- Every strips rule specifies a huge number of ground-level actions
- Consider move(obj, from, to)
  - Assume n objects in the world
  - This action alone specifies $O(n^3)$ ground actions
  - Planning tends to have a very large action space
- Compare with CSPs

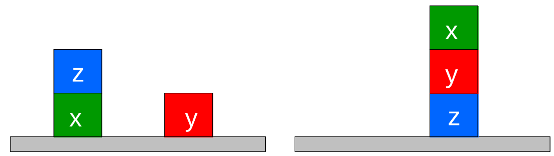
Planning vs. CSPs

- Both have large action spaces
- CSPs are atemporal
- We generally permit negations in CSPs, but try to avoid them in many planning formulations
- Effects of actions (assignments) are implicit
- The path matters: Knowing that solution exists isn't sufficient

Heuristics in planning

- In search, we assume that we can come up with reasonable heuristics
- Planning problems tend to defy natural efforts to develop good heuristics
- This is most evident in plans with conjunctive goals
- Making progress towards one conjunct can foil the other

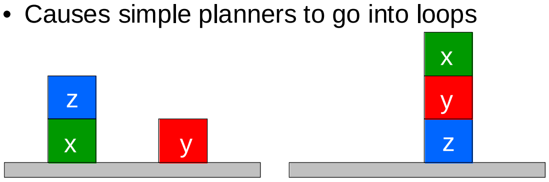
The Sussman Anomaly



Goal: on(x,y), on(y,z)

Problems with naïve subgoaling

- The number of conjuncts satisfied may not be a good heuristic
- Achieving individual conjuncts in isolation may actually may things harder
- Causes simple planners to go into loops



Summary: Planning Features

- State space is very large
- Goals usually defined over state sets
- Very large, implicitly defined action space
- Difficult to come up with good heuristics
- Path (plan) usually matters

How hard is planning?

- Planning is NP hard
- How can we prove this?
  - Reduce 3SAT to planning
  - Tricky if we don't permit negations
  - Make truth value a variable
  - val(x_i,true), val(x_i, false), val(x_i, undecided)

3SAT Reduction

- Given a 3SAT instance, what is our goal?
- Goal is a conjunction of all of the clauses
- Goal: satisfied(c_j) for all clauses c_j
- What are our actions?
- set_true(x_i, val), set_false(x_i, val), satisfy_c_j
- Start: unassigned(x_i) for all i

set_true(x_i)

- Preconditions
  - val(x_i, undecided)
- Delete list
  - val(x_i, undecided)
- Add list
  - val(x_i, true)
- set_false is similar

satisfy_c_j

- For each clause c_j = (xa, xb, xc) with truth values ta, tb, tc, we make three actions, one for each variable, e.g.,:
- Preconditions:
  - val(xa, ta)
- Delete list
- Add list
  - satisfied(c_j)

Why this works:

- Set actions force us to assign values to variables
- Once variables are set they can't be changed
- Clauses satisfied if any literals are satisfied
- We must satisfy all clauses to achieve the goal
- Goal is achievable iff formula is satisfiable

Is planning NP-complete?

- NO!
- Consider the towers of Hanoi:
  - http://www.mazeworks.com/hanoi/index.htm
  - Actions are exactly the same as the blocks moving actions
- Requires exponential number of moves
- Planning is actually PSPACE complete
- Planning with bounded plans is NP-complete

Should plan size worry us?

- What if you have a problem with an exponential length solution?
- Impractical to execute (or even write down) the solution, so maybe we shouldn't worry
- Sometimes this may just be an artifact of our action representation
  - Towers of Hanoi solution can be expressed as a simple recursive program
  - Nice if planner could find such programs

Advanced Planning Topics

- Research topic: automating abstraction
  - People solve towers of Hanoi by formulating high-level or abstract actions
  - Moving an entire subtower to another peg is formulated as an abstract action
- Research topic: Hierarchy
  - Decompose problem into subproblems
  - Combine subproblem solutions
- Using these methods is (relatively) easy
- Devising them automatically is quite hard

Planning Algorithms

- Extremely active and rapidly changing area
- Annual competitions pit different algorithms against each other on suites of challenge problems
- Algorithms compete in different categories
  - General
  - Domain specific
- Size of planning problems that can be solved has increased much faster than can be explained only by Moore's law in the past decade

Planning As Search

- Despite the special nature of planning problems, all planning algorithms can still be understood as variants of search
  - Forward search
    - Closest to classical search formulation
  - Backward search
    - Regression or means-ends analysis
  - Plan-space search
    - Closest to GSAT/walkSAT

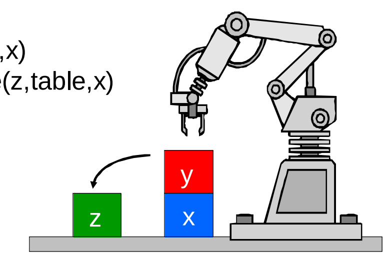
Goal Regression

- Goal regression is a form of backward search from goals (ends)
- Basic principle goes back to Aristotle
- Embodied in earliest AI systems
  - GPS: General Problem Solver by Newell & Simon
- Cognitively plausible
- Idea:
  - Pick actions that achieve (some of) your goal
  - Make preconditions of these actions your new goal
  - Repeat until the goal set is satisfied by start state

Goal Regression Example

Regress on(z,x)
through move(z,table,x)

New goal:
clear(x)



Goal: on(z,x)

Facts About Goal Regression

- Elegant solution to the problem of backward search from multiple goal states
  - In planning, goal state is usually a set of states
  - Goal regression does backward search at the level of state sets
- Goal regression is sound and complete
- Need to be careful to avoid endless loops on problems like Sussman anomaly

Plan Space Search

- Aim: Address subgoal interactions directly
- Start with a broken (often empty) plan
- Identify how the plan is broken
  - Unsatisfied preconditions or goals
  - Conflicting effects
- Modify plan to fix (some) problems
  - Rearrange actions
  - Add new actions
- This was a very popular view of planning until the mid 90s

Plan Space Search

- Plan space search tends to be messy
  - Plan modifications are complicated
  - Want to fix problems w/o creating new ones
  - Ensuring completeness and soundness is tricky
    - Planner must always find a plan if one exists
    - Plans actually should work
- Plan space search did well for many years because of the difficulty in coming up with good heuristics and the lack of fast, general methods for handling planning constraints

Issues

- Is forward search salvageable?
- Can we exploit structure in some way?
- What do the “modern” planners do?