Some Actual Planning Applications

- Used to fulfill mission objectives in NASA’s Deep Space One (Remote Agent)
  - Particularly important for space operations due to latency
- Also used for Rovers
- Aircraft assembly schedules
- Logistics for the U.S. Navy
- Observation schedules for Hubble space telescope
- Scheduling of operations in an Australian beer factory
Scheduling

• Many “planning” problems are scheduling problems

• Scheduling can be viewed as a generalization of the planning problem to include resource constraints
  – Time & Space
  – Money & Energy

• Many principles from regular planning generalize, but some extensions (not discussed here) are used

Continuous Motion Planning

• Another variation on planning involves planning in continuous state spaces for, e.g., robots

• Main challenge is curse of dimensionality

• Can’t discretize high dimensional spaces by brute force

• Research focuses on sampling, more clever discretization approaches than brute force, exploiting hardware and domain features

• See: https://youtu.be/u4snHh_S_Ao
Characterizing Discrete Planning Problems

- Start state (group of states)
- Goal – almost always a group of states
- Actions

- Objective: Plan = A sequence of actions that is guaranteed to achieve the goal.

- Like everything else, can view planning as search...
- So, how is this different from generic search?

What makes planning special?

- States typically specified by a set of relations or propositions:
  - On(solar_panels, cargo_floor)
  - arm_broken
- Goal is almost always a set
  - Typically care about a small number of things:
    - at(Ron, airport),
    - parked_in(X, car_of(Ron))
    - airport_parking_stall(X)
  - Many things are irrelevant
    - parked_in(Y, car_of(Bill))
    - adjacent(X,Y)
- Branching factor is large
Planning Algorithms

• Active and rapidly changing area
• Regular competitions pit different algorithms against each other on suites of challenge problems
  
  http://www.icaps-conference.org/index.php/Main/Competitions

• Algorithms compete in different categories
  – Classical vs. probabilistic vs. temporal
  – Optimal vs. Satisficing vs. Bounded cost

• No clearly superior method has emerged

PDDL – A Language for Planning Problems

• Actions have a set of preconditions and effects
• Think of the world as a database
  – Preconditions specify what must be true in the database for the action to be applied
  – Effects specify which things will be changed in the database if the action is taken

• NB: PDDL supersedes an earlier, similar representation called STRIPS
move(obj,from,to)

- **Preconditions**
  - clear(obj)
  - on(obj,from)
  - clear(to)

- **Effects**
  - **Add**
    - on(obj,to)
    - clear(from)
  - **Delete**
    - on(obj,from)
    - clear(to)

*STRIPS had a separate delete category. PDDL implements deletions as negative effects, but the difference is primarily syntactic.

**Limitations of PDDL**

- 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

- Can support quantification (which can fix the domino problem) but not always implemented for efficiency reasons

- Typically (though not necessarily) implements a “closed world” assumption - We only assert that which is true; can’t assert that which is false
How hard is planning?

• Planning is NP hard

• We use a technique called **reduction** to show that planning is at least as hard (up to polynomial factor) as graph coloring

Graph Coloring Reduction

• Given a graph coloring problem, what is our goal?
• Goal is: colored($v_i$) for all nodes $v_i$
• Initial state is:
  – uncolored($v_i$) for all nodes $v_i$
  – color($V$,nil) for all nodes $v_i$
  – neighbor($v_i$, $v_j$) for all neighbors in the graph
  – Neighbors($v$,i) for all nodes $v$ with $i$ neighbors
• What are our actions?
  – color($V$,color)
Color\((v,c)\)

- **Preconditions**
  - uncolored\((v)\)
  - neighbors\((v,1)\)
  - neighbor\((v,u)\)
  - colored\((u,c')\)
  - \(c\neq c'\)

- **Effects**
  - Add
    - colored\((v)\)
    - color\((v,c)\)
  - Delete
    - uncolored\((v)\)

**Additional Actions**

- As described, we need different actions for different numbers of neighbors – why?

- How expensive is this?
What this Does

- Actions correspond to coloring graph nodes
- Only legal assignments are allowed
- Plan exists iff graph is colorable
- Result: Planning is at least as hard as graph coloring, i.e., NP-hard

What just happened?

- Example of a general technique: reduction
- Powerful technique to compare the difficulty of two problems
Is planning NP-complete?

• NO!
• Consider the towers of Hanoi:
  – PDDL actions are the block 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
Planning Using Search

- Forward Search:
  - Blind forward search is problematic because of the huge branching factor
  - Some success using this method with carefully chosen action pruning heuristics (not covered in class)

- Backward Search:
  - Tends to focus search on relevant terms
  - Called “Goal Regression” in the planning context

Goal Regression

- Goal regression is a form of backward search from goals
- 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(x,z) through move(z,table,x)

New goal: clear(x)

Goal: on(x,z)

Greed, decomposition in planning

• Does a greedy approach work for planning?

• Idea:
  – Pick actions that satisfy as many parts of the goal as possible
  – If no single action satisfies any part of the goal, break up the goal into pieces and plan to solve each of them individually

• Bad news: This works poorly in general
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 make things harder
- Causes simple planners to go into loops and/or take lots of wasted steps
Summary of Traditional Planners

- Backward search methods are more focused, with careful construction these could be sound and complete generic planners.

- Forward (traditional) search methods worked well when:
  - Search space was very narrow (only a small number of reasonable things to do, which prevented exponential growth in reachable search space)
  - Domain-specific knowledge could be used to narrow the search space

Modern Planners

- One family uses sophisticated heuristics (graphplan)
  - Uses various tricks to narrow search space
  - May use forward or backward planning

- Another family uses forward search with domain specific tricks to prune the search space

- Another family converts everything into a giant SAT(satisfiability) problem and runs a highly optimized SAT solver (SATPlan)
What’s Missing?

• As described, plans are “open loop”
• No provisions for:
  – Actions failing
  – Uncertainty about initial state
  – Observations

• Solutions:
  – Plan monitoring, replanning
  – Conformant/Sensorless planning
  – Contingency planning

Planning Under Uncertainty

• What if there is a probability distribution over possible outcomes?
  – Called: Planning under uncertainty, decision theoretic planning, Markov Decision Processes (MDPs)
  – Much more robust: Solution is a “universal plan”, i.e., a plan for all possible outcomes (monitoring and replanning are implicit)
  – Much more difficult computationally

• What if observations are unreliable?
  – Called: “Partial Observability”, Partially Observable MDPs (POMDPs)
  – Applications to medical diagnosis, defense, sensor planning
  – Way, way harder computationally