Planning

CompSci 370
Ron Parr

Department of Computer Science
Duke University

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
  – Finally(!) used onboard on curiosity:

• 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

- Not the “hot” thing in AI now, but still active, important
- 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
  - Database stores true facts about the world – on(block, table)
  - 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. (Negative effects typically delete facts from the database, rather than asserting that things are false.)
Why Have Limitations?

• Planning languages are designed to allow fast search

• If preconditions were arbitrary logical statements, search might require proving theorems just to figure out if an action can be used

Planning Actions vs. Search Actions

• Plan actions are really action schemata
• Every PDDL 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
- CSP: Effects of actions (assignments) are implicit
- Planning: Path matters - Knowing that solution exists isn’t sufficient

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

- Assumptions about planning language:
  - No negations allowed
  - OK to test equality

- Given a graph coloring problem, what is our goal?
- Goal is: \( \text{colored}(v_i) \) for all nodes \( v_i \)
- Initial state is:
  - \( \text{uncolored}(v_i) \) for all nodes \( v_i \)
  - \( \text{color}(v_i, \text{nil}) \) for all nodes \( v_i \)
  - Available\((v_i,c_j)\) for all nodes and colors
- What are our actions?
  - \( \text{color}(V,\text{color}) \)

Coloring Actions color\((v_i,c)\)

- One action for each \( v_i \)
- Preconditions
  - \( \text{uncolored}(v_i) \)
  - \( \text{available}(v_i,c) \)
- Effects
  - Add
    - \( \text{colored}(v_i) \)
  - Delete
    - \( \text{uncolored}(v_i) \)
    - \( \text{available}(v_i,c) \)

Need to add one of these for each neighbor of \( v_i \)
This is why we have a separate action description
For each node.
What this Does

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

What just happened?

- Example of a general technique: *reduction*

  A instance → Poly-time xformation → B Solver

  poly time A solver if B is poly time

- Powerful technique to compare the difficulty of two problems
How to Think About This

• If planning can be solved in polynomial time, then graph coloring can be solved in poly time
  \[ O(\text{poly}(n)+\text{poly}(n)) = O(\text{poly}(n)) \]

• If graph coloring can’t be solved in poly time, then neither can planning

Planning Can be Harder than Graph Coloring

• Consider the towers of Hanoi:
  – [http://towersofhanoi.info/Animate.aspx](http://towersofhanoi.info/Animate.aspx)
  – PDDL actions are the disc moving actions

• Requires exponential number of moves

• Graph coloring can be verified in poly time

• Planning may require an **exponential size demonstration** that a plan is possible
Should plan size worry us?

• What if problem has exponential solution?
• In most cases, impractical to execute (or even write down) the solution, so why worry?

• May be artifact of representation
  – Towers of Hanoi solution can be expressed as a simple recursive program
  – Nice if planner could find such programs

• Common AI limitation: Discovering new representations

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 techniques (not covered in class)

• Backward Search:
  – Tends to focus search on relevant terms
  – Called “Goal Regression” in the planning context
Why Doesn’t A* help with Forward Search?

• Natural heuristics can be misleading

• Making progress towards achieving one part of a complex objective might make it harder to achieve another part

• Sussman anomaly is a classic example of this

The Sussman Anomaly

Goal: clear(x), on(x,y), on(y,z)
When Simple Heuristics Fail

- Goal `clear(x)`, `on(x,y)`, `on(y,z)`
- Does achieving one of these bring us closer to goal?
- What if we move `y` onto `z` first?
- What if we clear `x` by moving `z` onto `y`?

Backward Planning: 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)

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
  - Does backward search at the level of state sets
- Goal regression is sound and complete
- Can be more efficient than forward search unless forward search is guided by powerful heuristics
Summary of Traditional Planners

• Backward search methods are more focused gain efficiency by working with state sets

• Forward (traditional) search methods good 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 with powerful heuristics

Modern Planners (Oversimplified)

• One family of approaches uses search techniques combined with powerful domain independent (and/or domain specific) heuristics that take into account interactions between actions over time (e.g. certain sequences of actions are impossible or likely to be unhelpful)

• Another family converts everything into a giant logic problem (SAT) and uses a generic solver
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

• 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