Planning 2
approaches
Chapter 11.1-11.3
Some material adopted from notes by Andreas Geyer-Schulz and Chuck Dyer
Planning as State-Space Search

• Forward (progression) state-space search
  – Prone to exploring irrelevant actions
  – Uninformed forward-search in large state spaces is too inefficient to be practical
  – Need heuristics to make forward search feasible
Example: Air Cargo Problem

• 10 airports: each has 5 planes and 20 pieces of cargo
• Goal: Move all cargo at airport A to airport B
• Simple solution: Load 20 cargo onto plane \( p_1 \) at airport A, fly to airport B, unload cargo
• Average branching factor is huge:
  • Each of 50 planes can fly to 9 airports
  • 200 cargo can be unloaded/loaded onto any plane at airport
  • In any state min. 450 actions, max. 10,450 actions
• If we take average 2000 possible actions per state, search graph up to obvious solution has \( 2000^{41} \) nodes
Backward Relevant-States Search

• Start at the goal, apply actions backwards until reach initial state

• Only consider actions that are relevant to the goal (or current state), i.e.
  – Action must contribute to the goal
  – Must not have any effect which negates an element of the goal

• Consider a set of relevant states at each step, not just a single state (cf. belief state search)
Backward Relevant-States Search

• Must know how to regress from a state description to a predecessor state

• PDDL description makes it easy to regress actions:
  • Effects added by action need not have been true before
  • Preconditions must have been true before
  • Do not Del(a) as we don’t know whether or not fluents were true before

• Need to deal with partially uninstantiated actions and states, not just ground ones

• Backward search keeps branching factor lower than forward, but it’s harder to define good heuristics – so most current systems favor forward search
Heuristics for Planning

• Planning complex state representation, rather than ones, so we can define good domain-independent heuristics

• Admissible heuristics (i.e., not over-estimating) can be derived by defining a relaxed problem that’s easier to solve

  => Can use A* search to find optimal solutions

• Exact cost of a solution to easier relaxed problem becomes a heuristic for the original problem

• Heuristic examples: ignore preconditions, state abstraction, problem decomposition...
Planning as Boolean Satisfiability

• Reduces planning problem to classical propositional SAT problem
• SAT problem: is a propositional formula satisfiable? (i.e., is there an assignment that makes it true?)
• Making plans by logical inference
• To use SATPlan, PDDL planning problem description needs first to be translated to propositional logic
SATPlan

• SATPlan asks whether there exists any plan solving a given planning problem
  – SATPLAN is about satisficing (want any solution, not necessarily the cheapest or the shortest)

• Bounded SATPlan asks whether there exists a plan of length k or less
  – Can be used to ask for the optimal solution

• If we don’t allow functional symbols in the PDDL, both problems are decidable
**SATPlan Algorithm**

1. Construct a propositional sentence that includes
   a) Description of initial state
   b) Description of the planning domain (precondition axioms, successor state axioms, mutual exclusion of actions) up to some maximum time N
   c) Assertion that the goal is achieved at time N
2. Call SAT solver to return a model for this sentence
3. If a model exists, extract variables representing actions at each time from 0 to N and are assigned true, and present them in order of times as a plan
SOTA for Classical Planning?

• See the 2019 AAAI tutorial on the 2018 International Planning Competition for details

• A system using an approach inspired by SATPlan is good for finding an optimal plan

• The Fast Forward (FF) planner works well when satisficing is your goal
  – A forward chaining heuristic state space planner
  – It is the one used in Planning.Domains
  – Open source (written in c)
Fin