Overview

- Last time
  - Resolution in first-order logic; relating Prolog, FO logic and resolution

- Today
  - Overview of classical planning
  - Representing planning problems
    - Planning Domain Definition Language (PDDL)
  - State space linear planning

- Learning outcomes covered today:
  Identify or describe approaches used to solve planning problems in AI and apply these to simple examples

What is planning?

- “Devising a plan of action to achieve one’s goals”
  Planning = How do I get from here to there?

Automated Planning

- We will look at two popular and effective current approaches to automated classical planning:
  - Forward state-space search with heuristics
  - Translating to a Boolean satisfiability problem

- There are also other approaches
  - e.g. planning graphs: data structures to give better heuristic estimates than other methods, and also used to search for a solution over the space formed by the planning graph
Representing Planning Problems

• Recall search based problem-solving agents
  – Find sequences of actions that result in a goal state
  BUT deal with atomic states so need good domain-specific heuristics to perform well
• Planning represented by factored representation
  – Represent a state by a collection of variables
• Planning Domain Definition Language (PDDL)
  – Allows expression of all actions with one schema
  – Inspired by earlier STRIPS planning language

PDDL – Representing States (I)

• A state is represented by a conjunction of fluents
• These are ground, functionless atoms
  – Example: At(Truck1,Manchester) \( \land \) At(Truck2,Warrington)
• Closed world assumption (no facts = false)
• Unique names assumption (Truck1 distinct from Truck2)

Defining a Search Problem

• Define a search problem through:
  1. Initial state
  2. Actions available in a state
  3. Result of action
  4. Goal test

PDDL – Representing States (II)

• Not allowed:
  At(x,y) non-ground (i.e. variables alone)
  ¬ Poor negation
  At(Father(Fred),Liverpool) uses function

• A state is treated as either
  – conjunction of fluents, manipulated by logical inference
  – set of fluents, manipulated with set operations
PDDL – Representing Actions

• Actions described by a set of action schemas that implicitly define \( \text{Actions}(s) \) and \( \text{Result}(s, a) \) functions

• Classical planning: most actions leave most states unchanged
  – Relates to the Frame Problem: issue of what changes and what stays the same as a result of actions

• PDDL specifies the result of an action in terms of what changes – don’t need to mention everything that stays the same

Action Schema (I)

• Represents a set of ground actions
• Contains action name, list of variables used, precondition and effect
• Example: action schema for flying a plane from one location to another

\[
\text{Action(Fly(p,from,to),} \\
\text{PRECOND: At(p,from) \land Plane(p) \land} \\
\text{Airport(from) \land Airport(to) } \\
\text{EFFECT: \neg At(p,from) \land At(p,to))}
\]

Action Schema (II)

• Free to choose whatever values we want to instantiate variables
• Precondition and effect of an action are each conjunctions of literals (positive or negated atomic sentences)
  – Precondition defines states in which action can be executed
  – Effect defines result of action
• Sometimes we want to propositionalise a PDDL problem (replace each action schema with a set of ground actions) and use a propositional solver (e.g. SATPLAN) to find a solution
  – More on this later...

Action Schema (III)

• Action \( a \) can be executed in state \( s \) if \( s \) entails the precondition of \( a \) (\( a \in \text{Actions}(s) \) ) ⇔ \( s \models \text{Precond}(a) \)

where any variables in \( a \) are universally quantified

• Example:

\[
\forall p, from, to \ (\text{Fly}(p, from, to) \in \text{Actions}(s)) \iff s \models (\text{At}(p, from) \land \text{Plane}(p) \land \text{Airport(from)} \land \text{Airport(to)})
\]

• We say that \( a \) is applicable in \( s \) if the preconditions are satisfied by \( s \)
Action Schema (IV)

• Result of executing action \( a \) in state \( s \) (\( s' \))
  \[
  \text{Result}(s,a) = (s-Del(a)) \cup \text{Add}(a)
  \]

• **Delete list** \( \text{Del}(a) \): fluents that appear as negative literals in action’s effect

• **Add list** \( \text{Add}(a) \): fluents that appear as positive literals in action’s effect

• Note that time is implicit: preconditions have time \( t \), effects have \( t+1 \)

Planning Domain

• A set of action schemas defines a planning domain

• A specific problem within a domain is defined by adding initial state and goal
  
  − Initial state: conjunction of ground atoms
  − Goal: conjunction of literals (positive or negative) that may contain variables
    
    • e.g. \( \text{At}(p,LPL) \land \text{Plane}(p) \)

• Problem solved when we find sequence of actions that end in a state that entails the goal
  
  − e.g. \( \text{Plane}(\text{Plane}_1) \land \text{At}(\text{Plane}_1,LPL) \) entails the goal \( \text{At}(p,LPL) \land \text{Plane}(p) \)

Example: Air Cargo Transport

\[
\begin{align*}
\text{Init} & : \text{At}(C_1,SFO) \land \text{At}(C_2,JFK) \land \text{At}(P_1,SFO) \land \text{At}(P_2,JFK) \land \\
& \quad \text{Cargo}(C_1) \land \text{Cargo}(C_2) \land \text{Plane}(P_1) \land \text{Plane}(P_2) \land \\
& \quad \text{Airport}(JFK) \land \text{Airport}(SFO) \\
\text{Goal} & : \text{At}(C_1,JFK) \land \text{At}(C_2,SFO)
\end{align*}
\]

Example from Chapter 10 of AIAMA
Example: Air Cargo Transport

Init(At(C₁,SFO) ∧ At(C₂,JFK) ∧ At(P₁,SFO) ∧ At(P₂,JFK) ∧ Cargo(C₁) ∧ Cargo(C₂) ∧ Plane(P₁) ∧ Plane(P₂) ∧ Airport(JFK) ∧ Airport(SFO))

Goal(At(C₁,JFK) ∧ At(C₂,SFO))

Action(Load(c,p,a),
  PRECOND: At(c,a) ∧ At(p,a) ∧ Cargo(c) ∧ Plane(p) ∧ Airport(a)
  EFFECT: ¬At(c,a) ∧ In(c,p))

Action(Unload(c,p,a),
  PRECOND: In(c,p) ∧ At(p,a) ∧ Cargo(c) ∧ Plane(p) ∧ Airport(a)
  EFFECT: At(c,a) ∧ ¬In(c,p))

Action(Fly(p,from,to),
  PRECOND: At(p,from) ∧ Plane(p) ∧ Airport(from) ∧ Airport(to)
  EFFECT: ¬At(p,from) ∧ At(p,to))

Example from Chapter 10 of AIAMA

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

• Problem defined with 3 actions
• Actions affect 2 predicates
• When a plane flies from one airport to another, all cargo inside goes too
  – in PDDL we have no explicit universal quantifier to say this as part of the Fly action
  – so instead we use the load/unload actions:
    • cargo seizes to be At the old airport when it is loaded
    • and only becomes At the new airport when it is unloaded

• A solution plan:
  [Load(C₁,P₁,SFO),Fly(P₁,SFO,JFK),Unload(C₁,P₁,JFK),
   Load(C₂,P₂,JFK),Fly(P₂,JFK,SFO),Unload(C₂,P₂,SFO)].

• Problem – spurious actions like Fly(P₁,JFK,JFK) have contradictory effects
  – Add inequality preconditions ∧ (from ≠ to)
Example: Air Cargo Problem

- Consider this 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 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 its 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 (Regression) Relevant-States Search (I)

- 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 (Regression) Relevant-States Search (II)

- We 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 consider $\text{Del}(a)$ 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 search BUT using state sets means it’s harder to define good heuristics – so most current systems favour forward search

Exercise

- Consider the following air cargo problem

  - Goal: deliver a specific piece of cargo to SFO $\text{At}(C_2, SFO)$

- Which action does this suggest that will lead to this goal?
Exercise

• Consider the following air cargo problem

• Goal: deliver a specific piece of cargo to SFO $\text{At}(C_2, SFO)$
• Suggests the action

$\text{Action}(\text{Unload}(C_2, p', SFO),$
$\text{PRECOND}: \text{In}(C_2, p') \land \text{At}(p', SFO) \land$
$\text{Cargo}(C_2) \land \text{Plane}(p') \land \text{Airport}(SFO))$
$\text{EFFECT}: \text{At}(C_2, SFO) \land \lnot \text{In}(C_2, p'))$

unloading from an unspecified plane $p'$ at SFO

• What is the regressed state description?

Heuristics for Planning

• As planning uses factored representation of states (rather than atomic states), it is possible to define good domain-independent heuristics
• An admissible heuristic (i.e. does not overestimate distance to goal) can be derived by defining a relaxed problem that is easier to solve
  – Can then make use of A$^*$ search to find optimal solutions
• The exact cost of a solution to this easier problem becomes a heuristic for the original problem
• Examples of heuristics: ignore preconditions, state abstraction, problem decomposition...

Planning as Boolean Satisfiability

• Reduces planning problem to classical propositional SAT problem
• SAT problem: is this propositional formula satisfiable? (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 is the question of whether there exists any plan that solves a given planning problem
  – SATPLAN is about satisficing (want any solution, not necessarily the cheapest or the shortest)
• Bounded SATPLAN is the question of whether there exists a plan of length $k$ or less
  – Bounded SATPLAN can be used to ask for the optimal solution
• If in the PDDL language we do not allow functional symbols, both problems are decidable

SATPlan Algorithm

1. Construct a propositional sentence that includes
   (a) description of the initial state
   (b) description of the planning domain (precondition axioms, successor state axioms, mutual exclusion of actions) up to some maximum time $t_n$
   (c) the assertion that the goal is achieved at time $t_n$
2. Call SAT solver to return a model for the sentence from 1.
3. If a model exists, extract the variables that represent actions at each time from $t_0$ to $t_n$ and are assigned true, and present them in order of times as a plan

Summary

• Planning systems are problem-solving algorithms that operate on explicit propositional or relational representations of states and actions
  – PDDL describes
    • initial and goal states as conjunctions of literals
    • actions in terms of preconditions and effects
• State-space search in forward or backward direction
• Can get effective heuristics by relaxing the planning problem
• Can make plans by logical inference
  – Boolean satisfiability and SATPLAN

• Next time
  – Planning in complex environments