CSE537

AIMA CHAPTER 10.3: PLANNING GRAPHS

Resource: based on material & slide by Rob St. Amant (NCSU) and by Berthe Y. Choueiry (U of Nebraska)
**SEARCH AND PLANNING**

- **Planning**: generate seq. of actions to achieve one’s goals
- We have seen two examples of planning agents so far:
  - search-based problem-solving agent of Ch.3
    - can find sequences of actions that result in a goal state.
    - but deals with atomic states (needs good domain-specific heuristics)
  - hybrid logical agent of Chapter 7.
    - can find plans without domain-specific heuristics
      (uses domain-independent heuristics based on the logical structure of the problem)
    - but relies on ground (variable-free) propositional inference
      (it may be over worked when there are many actions and states.)
- We want representation for **planning problems**
  - that *scales up* to problems unable to be handled by earlier approaches.
The assumptions for classical planning problems

- Fully observable
  - we see everything that matters
- Deterministic
  - the effects of actions are known exactly
- Static
  - no changes to environment other than those caused by agent actions
- Discrete
  - changes in time and space occur in quantum amounts
- Single agent
  - no competition or cooperation to account for
FACTORED REPRESENTATION IN PLANNING LANGUAGE

- What is a good representation?
  + Expressive enough to describe a wide variety of problems
  + Restrictive enough for efficient algorithms to operate on it
  + Planning algorithm should be able to take advantage
    ✗ of the logical structure of the problem

- Historical AI planning languages
  + STRIPS was used in classical planners
    ✗ Stanford Research Institute Problem Solver
  + ADL addresses expressive limitations of STRIPS
    ✗ Action Description Language
    ✗ Adds features not in STRIPS
      ✗ negative literals, quantified variables, conditional effects, equality
  + We'll look at a simpler version of de facto standard language called PDDL
PDDL

- PDDL and most of the planning language use *factored* representation for states
  - Each state is represented as a collection of variables
- Planning Domain Definition Language
  - To see its expressive power, recall propositional agent in the Wumpus World, which requires $4Tn^2$ actions to describe a movement of 1 square
  - PDDL captures this with a single Action Schema
Each state is represented as a conjunction of fluents: grounded, functionless atoms.

- Ex> Poor ∧ Unknown might represent the state of a hapless agent,
- Ex> a state in a package delivery problem might be At(Truck1,Melbourne) ∧ At(Truck2,Sydney)

Database semantics is used

- the closed-world assumption: any fluents that are not mentioned are false,
- the unique names assumption: ex>Truck1 and Truck2 are distinct
- fluents not allowed: At(x, y) (because it is non-ground), ¬Poor (because it is a negation), and At (Father (Fred), Sydney ) (because it uses a function symbol).

This state representation allows alternative algorithms

- it can be manipulated either by logical inference techniques or by
- set operations (sets may be easier to deal with)
Actions are defined by a set of action schemas
- These implicitly define the ACTIONS(s) & RESULT(s, a) functions required to apply search techniques

Classical planning concentrates on problems where most actions leave most things unchanged.
- PDDL specify the result of an action in terms of what changes;
  everything that stays the same is left unmentioned.
Ground (variable-free) actions are represented by a single action schema - a *lifted* representation.

- Lifts from propositional logic to a restricted subset of First-order logic.

Consists of:
- The schema name,
- List of variables used,
  - Consider variables as universally quantified, choose any values we want to instantiate them,
- A precondition
  - PRECOND: defines states in which an action can be executed,
- An effect
  - EFFECT: defines the result of executing the action.
Each represents a set of variable-free actions

- Form: Action Schema = predicate + preconditions + effects
- Example action schema for flying a plane from one location to another:

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

- Action that results from substituting values for all the variables:

  Action(Fly(P1,SFO,JFK),
  \[\text{PRECOND: } \text{At}(P1,\text{SFO}) \land \text{Plane}(P1) \land \text{Airport}(\text{SFO}) \land \text{Airport}(\text{JFK})\]
  \[\text{EFFECT: } \lnot\text{At}(P1,\text{SFO}) \land \text{At}(P1,\text{JFK})\]
APPLYING ACTION SCHEMA

- Action \( a \) is **applicable** in state \( s \)
  - \( s \) entails the precondition of \( a \)
    - If \( a \)'s preconditions are satisfied in \( s \) ("\( a \) is applicable in \( s \)"")
      \[ a \in \text{ACTIONS}(s) \iff s |= \text{PRECOND}(a) \]
  - Given variables in \( a \), there can be multiple applicable instantiations
    - For \( v \) variables in a domain with \( k \) unique object names, worst case time to find applicable ground actions is \( O(v^k) \)

- Leads to one approach for solving PDDL planning problems
  - **Propositionalize** by replacing action schemas with sets of ground actions
    - then applying a propositional solver like SATPlan
  - Impractical for large \( v \) & \( k \)
Result of executing action $a$ in state $s$ is state $s'$

$$RESULT(s, a) = (s - DEL(a)) \cup ADD(a)$$

- Start with $s$
- Remove negative literal in the action’s effect
  (the *delete list*, DEL($a$))
- Add positive literals in action’s EFFECTs
  (the *add list*, ADD($a$))
- For example, with the action $Fly(P1,SFO,JFK)$,
  - we would remove $At(P1,SFO)$ and
  - add $At(P1,JFK)$.

Any variable in the effect must also appear in the precondition.

- When the precondition is matched against the state $s$, all the variables will be bound, and $RESULT(s,a)$ will therefore have only ground atoms.
PDDL: ACTION SCHEMAS

1. Variables & ground terms
   + Variables in effects must also be in precondition
     × so matching to state s yields results with all variables bound
       i.e. that contain only ground terms
     × Ground states are closed under the RESULT operation.

2. Handling of time
   + No explicit time terms
   + Instead time is implicitly represented in PDDL schemas
     × Preconditions always refer to time: t
     × Effects always refer to time: t + 1

3. A set of schemas defines a planning domain
   + A specific problem within the domain is defined with the addition of an initial state and a goal.
PDDL: Initial States, Goals, Solutions

- **Initial state**
  - Conjunction of ground terms

- **Goal**
  - Conjunction of positive and negative literals that contain variable.
    - Both ground terms & those containing variables
    - EX> \( \text{At}(p, \text{SFO}) \land \text{Plane}(p) \).
  - Variables are treated as existentially quantified
    - EX> so this goal is to have *any* plane at SFO

- **Solution**
  - A sequence of actions ending in \( s \) that entails the goal
    - EX> state \( \text{Rich} \land \text{Famous} \land \text{Miserable} \) entails the goal \( \text{Rich} \land \text{Famous}, \)
    - EX> state \( \text{Plane}(P1) \land \text{At}(P1, \text{SFO}) \) entails \( \text{At}(p, \text{SFO}) \land \text{Plane}(p) \)

- **We have defined planning as a search problem:**
  - have an initial state, an ACTIONS function, a RESULT function, and a goal test
WHY PLANNING GRAPHS

- All of the heuristics we have suggested can suffer from inaccuracies.
- A special data structure called a planning graph can be used to give better heuristic estimates.
- We can search for a solution over the space formed by the planning graph, using an algorithm called GRAPH PLAN.
  - These heuristics can be applied to any of the search techniques we have seen so far.
Graphplan was developed in 1995 by Avrim Blum and Merrick Furst, at CMU.

Constructs compact constraint encoding of state space from operators and initial state, which prunes many invalid plans.

A planning graph compactly encodes the space of consistent plans, while pruning . . .

- Partial states and actions at each time that are not reachable from the initial state.
- Pairs of actions and propositions that are mutually inconsistent at time i.
- Plans that cannot reach the goals.
PLANNING GRAPHS PROPERTIES

- A polynomial-size approximation to tree-based state space searching that can be constructed quickly.
- The plan graph does not eliminate all infeasible plans.
- Planning graph cannot answer definitely whether goal G is reachable from initial state S0, but it can estimate how many steps it takes to reach the goal.
  + Always correct when it reports the goal is not reachable.
  + Never overestimate the number of steps (admissible heuristic).

- Planning graphs
  + Provide a possible basis for better search heuristics.
  + Can be used directly, for extracting a solution to a planning problem, by applying the GRAPHPLAN algorithm.
Problem “Have cake and eat cake too”

PDDL Problem Description

Init(Have(Cake))
Goal(Have(Cake) \land Eaten(Cake))
Action(Eat(Cake))
  PRECOND: Have(Cake)
  EFFECT: \neg Have(Cake) \land Eaten(Cake))
Action(Bake(Cake))
  PRECOND: \neg Have(Cake)
  EFFECT: Have(Cake))

corresponding planning graph
Planning graph
+ Is a directed graph organized in time steps levels
+ Consist of alternating
  × $S_i$ level: contains all the literals that could result from any possible choice of action in $A_{i-1}$
  × $A_i$ level: contains all the actions that are applicable in $S_i$.
  × Precondition link
  × Effects link
  × Mutual exclusion (mutex) links: links joining nodes that cannot persist simultaneously
Start at level $S_0$, determine action level $A_0$ & next level $S_1$

- $A_0$: all actions whose preconditions are satisfied in the previous level (initial state)
  - Lines connect PRECONDS at $S_0$ to EFFECTs at $S_1$
  - Also, for each literal in $S_i$, there's a persistence action (square box) & line to it in the next level $S_{i+1}$

Level $A_0$ contains the actions that could occur

- Conflicts between actions are represented by arcs: mutual exclusion or mutex links
Level $S_1$ contains all the literals that could result
- From picking any subset of actions in $A_0$
- So $S_1$ is a belief state consisting of the set of all possible states
  - Each is a subset of literals with no mutex links between members
  - Conflicts between literals that cannot occur together are represented by the mutex links.

The level generation process is repeated

Termination condition (*leveling off*):
- When consecutive levels are identical
 Mutex relation holds between 2 **actions** at a level when

1. **Inconsistent effects**
   - One action negates the effect of another
   - Eat(Cake) and Have(Cake) have inconsistent effects because they disagree on the effect Have(Cake).

2. **Interference**
   - An effect of one action negates a precondition of the other;
   - Ex> Eat(Cake) interferes with the persistence of Have(Cake) by negating its precondition.

3. **Competing needs**
   - A precondition of one action is mutex with a precondition of the other
   - Ex> Bake(cake) & Eat(cake) <- compete on the value of Have(cake)
Mutex relation holds between 2 **literals** at a level when

1. One is the **negation** of the other
2. **Inconsistent support**
   - If each possible action pair that could achieve the literals is mutex
   - Ex> Have(Cake) & Eaten(Cake) at S₁
   - (the only way of achieving Have(Cake), the persistence action, is mutex with the only way of achieving Eaten(Cake), )
Construction has complexity polynomial in the size of the planning problem:

\[ O(n(a + l)^2) \]

- Given \( l \) literals and \( a \) actions,
- each \( S_i \) has no more than
  - \( l \) nodes and
  - \( l^2 \) mutex links, and
- each \( A_i \) has no more than
  - \( a + l \) nodes (including the no-ops),
  - \( (a + l)^2 \) mutex links, and
  - \( 2(al + l) \) precondition and effect links.
- entire graph with \( n \) levels has a size of \( O(n(a + l)^2) \)
PROPERTIES OF COMPLETED PLANNING GRAPH

- Provides information about the problem & candidate heuristics
- A goal literal g that does not appear in the final level cannot be achieved by any plan
- The level cost, the level at which a goal literal first appears, is useful as a cost estimate of achieving that goal literal
- Note that level cost is admissible, though possibly inaccurate since it counts levels, not actions
  - Planning graphs allow several actions at each level, whereas the heuristic counts just the level and not the number of actions.
  - We could find a better alternative level cost by using a serial planning graph variation, restricted to one action per level
    - Add mutex links between every pair of nonpersistence actions
Planning Graph provides

- Possible heuristics for the cost of a conjunction of goals

1. Max-level heuristic: highest level of any conjunct in the goal
   - Admissible, possibly not accurate

2. Level sum heuristic: the sum of level costs of conjuncts in the goal
   - Incorporates the subgoal independence assumption
     - So may be inadmissible to degree the assumption does not hold
     - Works well in practice for problems that are largely decomposable

3. Set-level heuristic: level where all goal conjuncts are present without mutex links
   - Admissible,
   - Dominates the max-level heuristic
   - Works well on tasks with good deal of interaction among subplans.
   - However, ignores interactions among three or more literals.
A Planning Graph is a relaxed version of the problem
- If a goal literal \( g \) does not appear, no plan can achieve it,
- If it does appear, is not guaranteed to be achievable
- Why?
  - The PG only captures pairwise conflicts & there could be higher order conflicts likely not worth the computational expense of checking for them
    - Similar to Constraint Satisfaction Problems where arc consistency was a valuable pruning tool
  - 3-consistency or even higher order consistency would have made finding solutions easier but was not worth the additional work
- Example where PG fails to detect unsolvable problem
  - Blocks world problem with goal of A on B, B on C, C on A
    - Any pair of subgoals are achievable, so no mutexes
    - Problem only fails at stage of searching the PG
**THE GRAPHPLAN ALGORITHM**

**GRAPHPLAN algorithm**

- Generates the Planning Graph & extracts a solution directly

```plaintext
function GRAPHPLAN(problem) return solution or failure
    graph ← INITIAL-PLANNING-GRAPH(problem)
    goals ← CONJUNCTS(problem. GOAL)
    nogoods ← an empty hash table
    for tl = 0 to ∞ do
        if goals all non-mutex in S_t of graph then
            solution ← EXTRACT-SOLUTION(graph, goals, NUMLEVELS(graph), nogoods)
            if solution ≠ failure then return solution
        if graph and nogoods have both leveled off then return failure
        graph ← EXPAND-GRAPH(graph, problem)

EXTRACT-SOLUTION: search for a plan that solves the problem.
EXPAND-GRAPH: adds a new level
```
EXAMPLE: SPARE TIRE PROBLEM

PDDL of spare tire problem (problem of changing a flat tire)

Init(At(Flat, Axle) ∧ At(Spare, Trunk))

Goal(At(Spare, Axle))

Action(Remove(Spare, Trunk)
  PRECOND: At(Spare, Trunk)
  EFFECT: ¬At(Spare, Trunk) ∧ At(Spare, Ground))

Action(Remove(Flat, Axle)
  PRECOND: At(Flat, Axle)
  EFFECT: ¬At(Flat, Axle) ∧ At(Flat, Ground))

Action(PutOn(Spare, Axle)
  PRECOND: At(Spare, Ground) ∧ ¬At(Flat, Axle)
  EFFECT: At(Spare, Axle) ∧ ¬At(Spare, Ground))

Action(LeaveOvernight
  PRECOND: 
  EFFECT: ¬At(Spare, Ground) ∧ ¬At(Spare, Axle) ∧ ¬At(Spare, Trunk) ∧ ¬At(Flat, Ground) ∧ ¬At(Flat, Axle))

Goal is to have a good spare tire properly mounted onto the car’s axle,
Initial state has a flat tire on the axle and a good spare tire in the trunk.
Notes:

- This figure shows the *complete* Planning Graph for the problem
- Arcs show mutex relations (arcs between literals are *omitted* to avoid clutter)
- Omits unchanging positive literals (for example, \text{Tire(Spare)})
- Omits irrelevant negative literals
- **Bold boxes & links** indicate the solution plan
S₀ is initialized to 5 literals
- from the problem initial state and the relevant negative literals
- no goal literal in S₀ so \textbf{EXPAND-GRA}PH add actions
  - those with preconditions satisfied in S₀
  - also adds persistence actions for literals in S₀
  - adds the effects at level S₁, analyzes & adds mutex relations
- repeat until the goal is in level Sᵢ or failure

\begin{align*}
\text{Init(At(Flat, Axle) \land At(Spare, Trunk))} \\
\text{Goal(At(Spare, Axle))} \\
\text{Action(Remove(Spare, Trunk)} \\
\quad \text{PRECOND: At(Spare, Trunk)} \\
\quad \text{EFFECT: \neg At(Spare, Trunk) \land At(Spare, Ground))} \\
\text{Action(Remove(Flat, Axle)} \\
\quad \text{PRECOND: At(Flat, Axle)} \\
\quad \text{EFFECT: \neg At(Flat, Axle) \land At(Flat, Ground))} \\
\text{Action(PutOn(Spare, Axle)} \\
\quad \text{PRECOND: At(Spare, Ground) \land \neg At(Flat, Axle)} \\
\quad \text{EFFECT: At(Spare, Axle) \land \neg At(Spare, Ground))} \\
\text{Action(LeaveOvernight)} \\
\quad \text{PRECOND:} \\
\quad \text{EFFECT: \neg At(Spare, Ground) \land \neg At(Spare, Axle) \land \neg At(Flat, Axle) \land \neg At(Spare, Trunk) \land \neg At(Flat, Ground) \land \neg At(Flat, Axle)} \)
\end{align*}
**GRAPHPLAN SPARE TIRE EXAMPLE**

**EXPAND-GRAPH adds constraints:** mutex relations
- inconsistent effects (action x vs action y)
  - Remove(Spare, Trunk) & LeaveOvernight:
    - At(Spare, Ground) & ¬At(Spare, Ground)
- interference (effect negates a precondition)
  - Remove(Flat, Axle) & LeaveOvernight:
    - At(Flat, Axle) as PRECOND & ¬At(Flat, Axle) as EFFECT
- competing needs (mutex preconditions)
  - PutOn(Spare, Axle) & Remove(Flat, Axle):
    - At(Flat, Axle) & ¬At(Flat, Axle)
- inconsistent support (actions to produce literals are mutex)
  - in S2, At(Spare, Axle) & At(Flat, Axle): only way to achieve At(Spare, Axle) is by PutOn(Spare, Axle) and that is mutex with the only action for obtaining At(Flat, Axle).

Init(At(Flat, Axle) ∧ At(Spare, Trunk))
Goal(At(Spare, Axle))
Action(Remove(Spare, Trunk)
  PRECOND: At(Spare, Trunk)
  EFFECT: ¬At(Spare, Trunk) ∧ At(Spare, Ground))
Action(Remove(Flat, Axle)
  PRECOND: At(Flat, Axle)
  EFFECT: ¬At(Flat, Axle) ∧ At(Flat, Ground))
Action(PutOn(Spare, Axle)
  PRECOND: At(Spare, Ground) ∧ ¬At(Flat, Axle)
  EFFECT: At(Spare, Axle) ∧ ¬At(Spare, Ground))
Action(LeaveOvernight
  PRECOND:
  EFFECT: ¬At(Spare, Ground) ∧ ¬At(Flat, Axle) ∧ ¬At(Spare, Trunk) ∧ ¬At(Flat, Ground) ∧ ¬At(Flat, Axle))
In S2, the goal literals exist, and they are not mutex with any other

- Just 1 goal literal so obviously not mutex with any other goal
- Since a solution may exist, EXTRACT-SOLUTION tries to find it

**EXTRACT-SOLUTION** as backward search problem (other methods possible)
- Initial state: last level of the PG, $S_n$, along with the goals from the planning problem
- Actions from $S_i$
  - Select *any conflict-free actions* in $A_{i-1}$ with effects covering the goals
  - Conflict free = no 2 actions are mutex & no pair of their preconditions are mutex
- Goal: Reach a state at level $S_0$ such that all goals are satisfied
- Cost: 1 for each action
If EXTRACT-SOLUTION fails

- At that point it records (level, goals) as a "no-good"
- Subsequent calls can fail immediately if they require the same goals at that level

Complexity

- We already know planning problems are computationally hard (PSPACE-complete)
  - Require good heuristics
- Heuristic for choosing an action at each level in backward search
  - Greedy search with level cost of literals
    - 1. Pick literal with highest level cost
    - 2. To achieve it, pick actions with easier preconditions
      - Choose action with smallest sum (or max) of level costs for its preconds
Alternative to backward search for a solution

EXTRACT-SOLUTION could formulate a Boolean CSP

- variables are actions at each level
- values are Boolean: an action is either *in* or *out* of the plan
- constraints are mutex relations & the need to satisfy each goal & precondition
GRAPHPLAN TERMINATION

GRAPHPLAN will in fact terminate and return failure when there is no solution.

- Recall that **level off** means consecutive PG levels are identical.
- Now note that a graph may level off before a solution can be found, on a problem for which there is a solution.
  - Ex. Air Cargo: 1 plane and n pieces of cargo at airport A, all of which have airport B as their destination. Where only one piece of cargo can fit in the plane at a time.
    - Graph levels off at level 4, from which full solution can’t be extracted (that would require 4n – 1 steps)
- We need to take account of the no-goods (goals that were not achievable) as well.
  - If it is possible that there might be fewer no-goods in the next level, then we should continue.
- Graph itself and the no-goods have both leveled off, with no solution found, we can terminate with failure.
Does GRAPHPLAN terminate?

Evidences that both graph and no-goods will level off

- Literals increase monotonically (and there are finite # of them)
  - Once a literal appears, its persistence action causes it to stay
- Actions increase monotonically (and there are finite # of them)
  - Once preconditions (literals) of an action appear at one level, they will appear at subsequent levels, and thus so will the action.
- Mutexes decrease monotonically
  - Of 2 actions are mutex at $A_i$, they are also mutex at all previous levels where they appear
    - The graph simplifying conventions may not show it
  - Same holds for 2 literals
- No-goods decrease monotonically
  - If a set of goals is not achievable at level $i$, they are not achievable at any previous level