More Planning

CS156

Chris Pollett

Nov 5, 2014

Outline

- Heuristics for Planning
- Planning Graphs

Introduction

- On Monday, we introduced the planning logic PDDL.
- We showed how problems that could in theory be solved by problem solving agents could be described in it.
- We then considered the complexity of planning as well as some of the issues that might be involved in forward and backward search for solutions
- We start today by considering some heuristics that might be helpful in performing this search for solutions

Heuristics for Planning

- One way to try to make planning problems easier is to try to come up with plans for relaxed problems.
- We can think of a planning search problem as a graph where the nodes are states and the edges are actions.
- We want to find a path from an initial state to a goal state.
- We can make this problem easier by adding more edges to the graph, or by grouping multiple nodes together and viewing them as one node (reducing the number of states).
- We next look at these approaches in more detail.

Ignore Preconditions Heuristic

- One way to try to implement the first strategy is to use the **ignore preconditions heuristic**. This heuristic drops all the preconditions from actions.
- It can still be the case that the number of steps to solve the resulting problem is more than the number of unsatisfied goals because some actions might undo the the effects of others.
- So to make this heuristic even more guaranteed, we also delete all effects from our actions except those that are literals in the goal.
- We then count the minimum number of actions required such that the union of those actions' effect satisfies the goal.
- An exact count is an NP-hard problem, but in p-time using a greedy algorithm one can approximate the true minimum to within a $\log n$ factor where n is the number of literals in the goal. (This isn't an admissible heuristic because we might overestimate the cost of a solution).
- Still, by using cost to a given state + this heuristic, we can search for solutions in an $A^\star$-star like fashion.

Ignore Delete Lists

- Another heuristic we can use in searching for solutions to planning problems is to remove all the delete lists from the effects of all actions.
- This means as one searches toward the states monotonically increase in the number of true literals until eventually (hopefully) we find the goal.
- It turns out this way of relaxing planning problems still leaves one with an NP-hard search problem.
- Nevertheless, an approximate solution to this problem can be found in polynomial time using hill climbing.

State Abstraction

- The relaxed problems made by these two ignore heuristics leave us with a simplified but still expensive planning problem just to compute the value of the heuristic.
- This is because these methods do not reduce the total number of states of the problem, which for things like the cargo problem can easily have 10^{155} states for a 10 airport, 50 plane, 200 pieces of cargo problem.
- So we want to consider relaxations that decrease the number of state by performing **state abstraction** -- a many-to-one mapping from states in the ground representation of the problem to the abstract representation.
- For instance, one thing we could do is assume that all of the packages are at one of 5 airports and all packages at a given airport have the same destination -- say a hub.

Problem Decomposition

- An important idea in defining heuristics is **decomposition**; dividing a problem into parts, solving each part independently, and then combining the parts.
- The **subgoal independence assumption** is the assumption that the cost of solving a conjunction of subgoals is approximated by the sum of the costs of solving each subgoal independently.

Planning Graphs 1

- A **planning graph** is a special data structure which can be used to give better heuristic estimates for the cost of a plan.
- It is a polynomial size approximation to the tree one would get by starting at the initial state expanding all possible next actions, then next actions of those actions, etc.
- A planning graph is organized into **levels**: first a level S_0 for the initial state; then level A_0 consisting of nodes for each ground action that might be applicable in S_0. Then alternating levels S_i followed by A_i; until we reach a termination condition.
- Roughly, S_i contains all the literals that could hold at time i. If it is possible that either P or $\neg P$ could hold, then both will be represented in S_i.
- Roughly, A_i contains all the actions that could have their preconditions satisfied at time i.
- Planning graphs only work for propositional planning problems.

Planning Graphs 2

Init(*Have*(*Cake*))
Goal(*Have*(*Cake*) ∧ *Eaten*(*Cake*))
Action(*Eat*(*Cake*)
PRECOND: *Have*(*Cake*)
EFFECT: ¬ *Have*(*Cake*) ∧ *Eaten*(*Cake*))
Action(*Bake*(*Cake*)
PRECOND: ¬ *Have*(*Cake*)
EFFECT: *Have*(*Cake*))

Figure 10.7 The "have cake and eat cake too" problem.

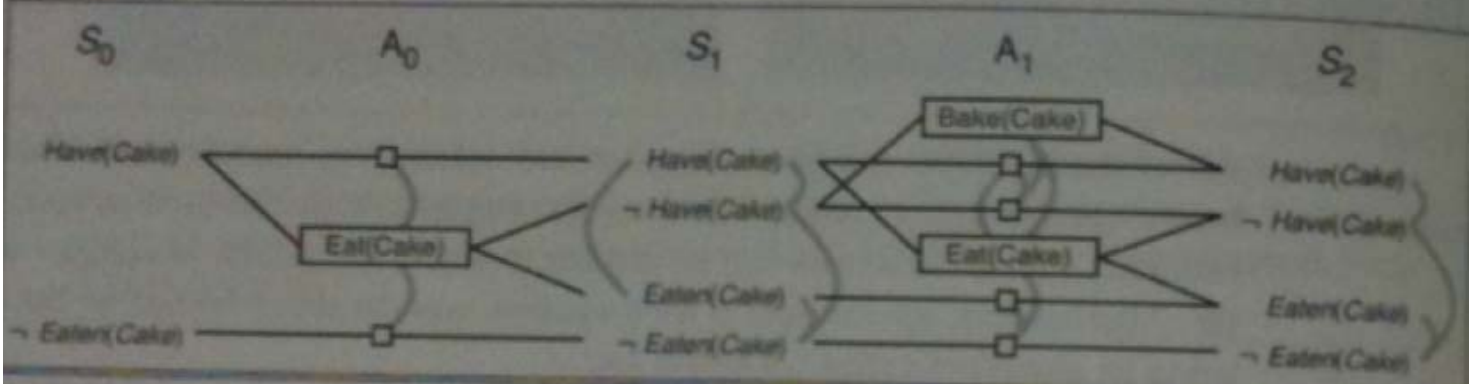

Figure 10.8 The planning graph for the "have cake and eat cake too" problem up to level S_2. Rectangles indicate actions (small squares indicate persistence actions), and straight lines indicate preconditions and effects. Mutex links are shown as curved gray lines. Not all mutex links are shown, because the graph would be too cluttered. In general, if two literals are mutex at S_i, then the persistence actions for those literals will be mutex at A_i and we need not draw that mutex link.

- The above picture is an example planning graph. Each action at level A_i is connected to its preconditions at S_i and its effects at S_{i+1}.
- We also want to say that a literal can persist if no action negates it. To do this we have **persistence actions** (no-ops). For every literal C, we add to the problem a persistence action with precondition C and effect C. These are drawn with small square boxes in the picture above.
- We also indicate things that cannot simultaneously happen using **mutex** (mutual exclusion) links.

- We continue building the graph from S_0 to A_0 to S_1, ... until we get to two consecutive levels which are identical. We then say the graph has **leveled off**. In the above, this happens at S_2.
- For a planning graph with 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 no-ops), $(a + l)^2$ mutex links, and $2(al + l)$ precondition and effect links. So a graph of n levels has size $O\left(n(a + l)^2\right)$, a polynomial. The construction time is the same.

Using Planning Graphs for Heuristic Estimation

- If any goal fails to appear in the final level of the planning graph of the problem, we know the problem is unsolvable.
- We can estimate the cost of achieving any goal g_i from state s as the level at which g_i first appears in the planning graph constructed with starting state s. This is called the **level cost of** g_i.
- For example $Have(Cake)$ has level cost 0 and $Eaten(Cake)$ has level cost 1 on the previous slide.
- To estimate the conjunction of goals, there are three common approaches: **max-level**, the largest level-cost amongst the goals; **level-sum**, the sum of the level costs, and **set-level**, the first level in graph in which all the literals in the conjunctive goal appear in the planning graph without any pair of them being mutually exclusive.