More Problem Solving Agents

CS156

Chris Pollett

Sep 3, 2014

Outline

- Problem Solving via Depth First Search
- Problem Solving via Breadth First Search
- Iterative Deepening
- Bi-directional Search
- $A^\star$-search

Introduction

- Remember next Monday there is a quiz!
- Last Wednesday, we began discussing problem solving agents.
- Problem solving agents were a particular kind of goal-based agent which use atomic representations of their environment, and which basically, cycle through different sequences of action until they find a sequence which achieves the desired goal.
- We said this cycling through different possible sequences is called **search**.
- We also described how to specify a problem, so that in principle we might get an agent to solve it.
- This involved specifying: (1) an initial state, (2) a set of possible actions, (3) a successor function, (4) a goal test, and (5) a cost function associated with each action.
- Today, we begin by looking at different ways for problem solving agents to carry out searches.

Searching for Solutions

- A **solution** to a search problem is an action sequence that takes the agent from the initial state to a goal state.
- We use the problem specification to define a **search tree**:
    - *The **root** of the tree is the initial state.*
    - *The **edges/branches** in our tree are labeled with actions.*
    - *The **nodes/vertices** in our tree correspond to states in the state space.*
- We typically imagine this tree as not given to us in toto. Instead, we start with the tree being just the root. We apply the goal test. If it satisfies the goal, we're done and we output the sequence of actions so far. If not, we **expand** the node we are working with. This involves applying each legal action to the current state and **generating** a set of new states. Then one picks from the tree a node (possible one of these **child nodes**) that has not been expanded to consider further.
- A node with no children in this tree is called a **leaf**.
- The set of all leaf nodes is called the **frontier** or

open list.

- A **search strategy** is a method for choosing a node on the frontier to consider next.

Loops

- Suppose we have a King in the bottom left hand corner (A1) of an otherwise unoccupied chess board.
- Our goal is to move the King to the top left hand corner (H8).
- Applying the goal test to A1 we see the square is not labeled (H8), so we expand the node.
- This gives us the search tree consisting of A1 with children/frontier (B1), (B2), (A2).
- If we pick (B1) to expand next we get the tree A1[B1[A1, A2, B2, C1, C2], B2, A2]
- Notice A1, A2, B2 are repeated. They are so-called **repeated states**.
- If we expand a **repeated state** we could end up on **loopy path**, and never end up at our goal of H8.
- Loopy paths are a special case of a more general problem which are two paths that go to the same state (**redundant paths**).
- To avoid this problem, any reasonable search algorithm, maintains in addition to a frontier list, a **closed list** also known as an **explored set** of

all states it has already seen.

Ways of Measuring our Search Strategy

We like to come up with means of distinguishing which is better amongst different possible search strategies. Some axes of comparison are:

- **Completeness**: Is the algorithm guaranteed to find a solution when there is one?
- **Optimality**: Does the strategy find the optimal solution? (For example, a shortest path in the case of the chess problem we just described.)
- **Time Complexity**: How long does it take to find a solution?
- **Space Complexity**: How much memory is needed to perform the search?
- Time and space complexity are often measured in terms of the **depth** d of the smallest goal state in the search tree as well as the maximum branching factor b of expanding a node.

Informed and Uninformed Search

- Search algorithms can be classified into two main categories: **informed** and **uninformed**.
- In **uninformed search**, the search strategy is not allowed to use any information about states beyond that provided in the problem definition. The algorithm is only allowed to generate successors and distinguish a goal state from a non-goal state.
- If an strategy uses some additional information, it is called an **informed search strategy**.
- We next look at some examples of uninformed search strategies.

Uninformed Search Strategies

- **Depth First Search (DFS)** -- always expand left most node that still can be expanded. (Charles Pierre Tr�maux, described by �douard Lucas 1882)
- **Breadth First Search (BFS)** -- expand root, then expand all children of root until reach goal. (Moore 1959 and Lee 1961)
- **Uniform Cost Search (UCS)** -- this is a generalization of BFS to the case where the costs of each action is its own rational number protentially different from 1. In this kind of search we expand the node on the frontier whose path cost from the root is the least.

Performance of DFS and BFS

- Let's consider the king on the chess board problem again, but let the chess board now be infinite in the up and right directions.
- Pick some fixed square (maybe H8, maybe some other square) as the goal state.
- In terms of our different means of measuring the performance of these algorithms, notice if the goal state was H8 and if depth first search was used, our agent may never find H8. It would could move A1, A2, A3, A4, ... So DFS is incomplete.
- How much memory does DFS use? Notice DFS only needs to keep track of the branch from the root it is on. So it takes O(d) space.
- Suppose we allowed the goal to be either the square H8 or the square A9. In this case, DFS finds a goal, but it does not find the shortest one, so it is not optimal.
- On the other hand, BFS is complete. It also will find H8 rather than A9 so it is optimal.

- Unfortunately, both its time and space complexity are $b + b^2 + ... + b^d = \Omega\left(b^d\right)$. So if the goal was A100, BFS would take $\Omega\left(8^{100}\right)$ time -- way too long.
- If ε is the least cost of an action and $C^\star$ is the least cost goal, one can show UCS has time and space complexity $O\left(b^{1+\frac{C^\star}{\varepsilon}}\right)$.
- Is there are way to get the best of both world the DFS and BFS worlds?

Iterative Deepening Search

- Let $DFS(n)$ denote running the depth first search algorithm on the search tree restricted to depth n.
- i.e., This algorithm first considers the leftmost branch of depth n, the next to leftmost branch of depth n, and so on.
- It stops if along any of these branches it finds a goal.
- This algorithm has space complexity $O(n)$ and time complexity $O(b^n)$.
- It is, of course, not complete.
- Now consider the algorithm which operates as follows. It checks if DFS(1) finds a solution, if not it runs DFS(2), if not it runs DFS(3) and so on...
- This is called **iterative deepening search** (**IDS**).
- Notice if there is a goal of depth less than n, then $DFS(n)$ finds it, and so does IDS.
- The space complexity IFS needs if the goal was

at depth n is $O(n)$.

- What is the time complexity?
  TIME(DFS(1)) + ... + TIME(DFS(n))
  $O(b) + + O(b^n) = O\left(b^{n+1}\right)$.
- IDS can be generalized to the rational action costs setting, by do DFS to cost ε, 2ε, ... This is called **iterative lengthening search**.

Bidirectional Search

- The idea of bidirectional search is that we run two simultaneous searches: one from the initial state and one from the goal state
- If we find a state that is common on the frontiers of these two searches then we stop.
- Notice if the depth of the goal is d, then this involves two searches of to depth $\frac{d}{2}$.
- If we use BFS to do these searches, then the time complexity will be $O\left(b^{\frac{d}{2}}\right)$ which is much smaller than $O\left(b^{d}\right)$.

Informed Search Strategies

- We now want to consider search strategies where one has problem specific knowledge.
- Here the node we select for expansion is chosen according to some evaluation function $f(n)$.
- In a **Best-First-Search** we assumes $f(n)$ is a cost and always expands node for which $f(n)$ is smallest.
- The question is how to choose f?
- A key component of f is usually some heuristic function, $h(n)$ where this is an estimate of the cost from a given node n to the goal.

Example Heuristic Functions

- In a maze game, $h(n)$ might be straight line distance (straight line mathematical distance from current position to the goal)
- Or it could be the Manhattan distance, (the distance to the goal using horizontal and vertical line segments of a given length).
- As an example, we could use the sum of the straight-line distance of each square from its ideal location as our heuristic with the 8-puzzle.
- **Greedy Best First Search (GBFS)** chooses for its next node to expand the node of smallest cost according to this heuristic.
- It is memory efficient in that it does not remember where it has been.

$A^\star$-search

- The problem with Greedy Best First Search is that it is not complete. You can end up going back and forth between two states.
- In 1968, Peter Hart, Nils Nilsson and Bertram Raphael of SRI modified came up with a different choice of f to solve this problem.
- Their algorithm is called$A^\star$ search.
- They define $f(n)$ to be $c(n) + h(n)$ where $c(n)$ is cost from the root node to the current node.
- They came up with this while devising a sequence of algorithms A1, A2, etc based-on Dijkstra's algorthm (1956) to help Shakey the Robot navigate a room filled with obstacles.