Lecture 5: Informed Search

Dr John Levine

CS310 Foundations of Artificial Intelligence
February 4th 2016

Search: the story so far

- Lecture 3: introduction to state space search: how to represent a problem in terms of states and moves
- Lecture 4: uninformed search through states using an agenda: depth-first search and breadth-first search
- Lecture 5: making it smart: informed search using heuristics; how to use heuristic search without losing optimality – the A* algorithm

Searching Using an Agenda

- When we expand a node we get multiple new nodes to expand, but we can only expand one at a time
- We keep track of the nodes still to be expanded using a data structure called an agenda
- When it is time to expand a new node, we choose the first node from the agenda
- As new states are discovered, we add them to the agenda somehow
- We can get different styles of search by altering how the states get added

Depth-First Search

- To get depth-first search, add the new nodes onto the start of the agenda (treat the agenda as a stack):

```
let Agenda = [S0]
while Agenda ≠ [] do
    let Current = First(Agenda)
    let Agenda = Rest(Agenda)
    if Goal(Current) then return("Found it!")
    let Next = NextStates(Current)
    let Agenda = Next+Agenda
```

Properties of Depth-First Search

- Depth-first can often get to the answer quickly
- The agenda stays short: $O(d)$ for memory, where d is the depth of the tree
- The time taken to find the solution is $O(d)$ in the best case and $O(b^d)$ in the worst case (where b is the average branching factor)
- But if there are loops or infinite branches in the search space, it may not return a solution
- It isn't guaranteed give the shortest solution

Breadth-First Search

- To get breadth-first search, add the new nodes onto the end of the agenda (treat the agenda as a queue):

```
let Agenda = [S0]
while Agenda ≠ [] do
        let Current = First(Agenda)
        let Agenda = Rest(Agenda)
        if Goal(Current) then return("Found it!")
        let Next = NextStates(Current)
        let Agenda = Agenda + Next
```

Properties of Breadth-First Search

- Breadth-first can often take a long time to get to the answer
- The agenda can get very big: $O(b^d)$ for memory, giving exponential space consumption
- Also exponential time complexity: $O(b^d)$ nodes will be expanded
- But it isn't bothered by loops or infinite branches in the search space
- And it always gives the shortest solution, in terms of the number of steps in the plan

Heuristic Search

- DFS and BFS are both searching blind – they search all possibilities in an order dictated by NextStates(S_i)
- When people search, we look in the most promising places first – try { [a], [b], [c] } → { [a, b, c] }
- There are six possible moves, but somehow it seems like the best move is move(b,c) giving { [a], [b, c] }
- The most promising states are often those which are closest to the goal state, *G*
- But how can we know how close we are to the goal state?

Heuristic Search

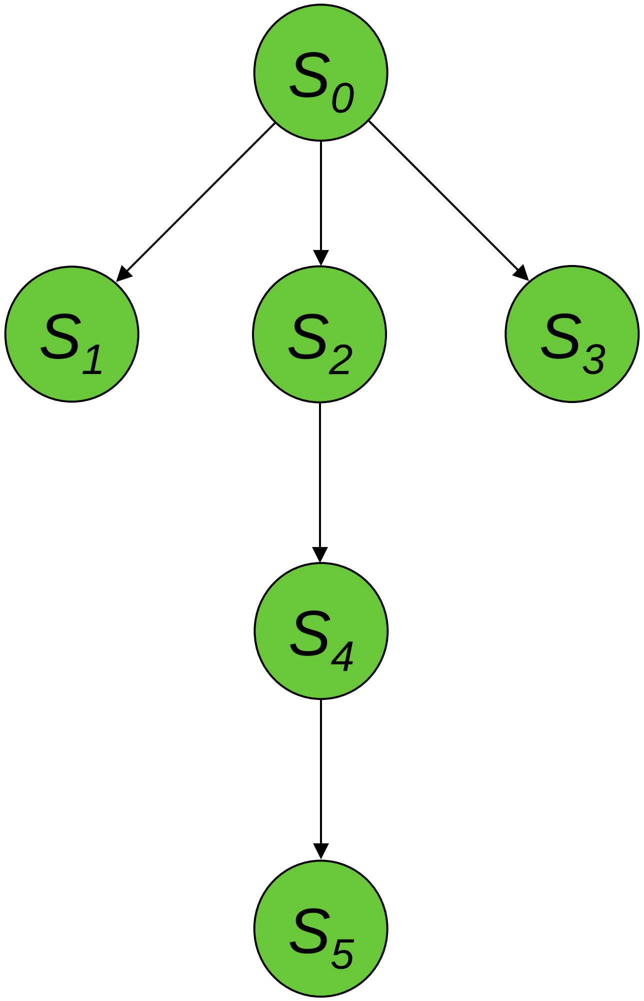
- We can often estimate the distance from S_i to G by using a heuristic function, $h(S_i,G)$
- The function efficiently compares the two states and tries to get an estimate of how many moves remain without doing any searching
- For example, in the blocks world, all blocks that are stacked up in the correct place never have to move again; all blocks that need to move that are on the table only need to move once; and all other blocks only need to move at most twice:

$$h(S_i,G) = 2*B_{bad} + 1*B_{table} + 0*B_{good}$$

Enforced Hill Climbing

- The easiest way to use a heuristic estimate to search is to require that every single move we make takes us closer to the goal
- The form of search doesn’t even require an agenda, since at each decision point, we take the action that looks best to us and repeat until we’re done
- Problems: dead ends, plateaus, solution quality (i.e. the number of steps can be very poor)
- Used to good effect in the FF planner (which reverts to best-first search if enforced hill climbing fails)

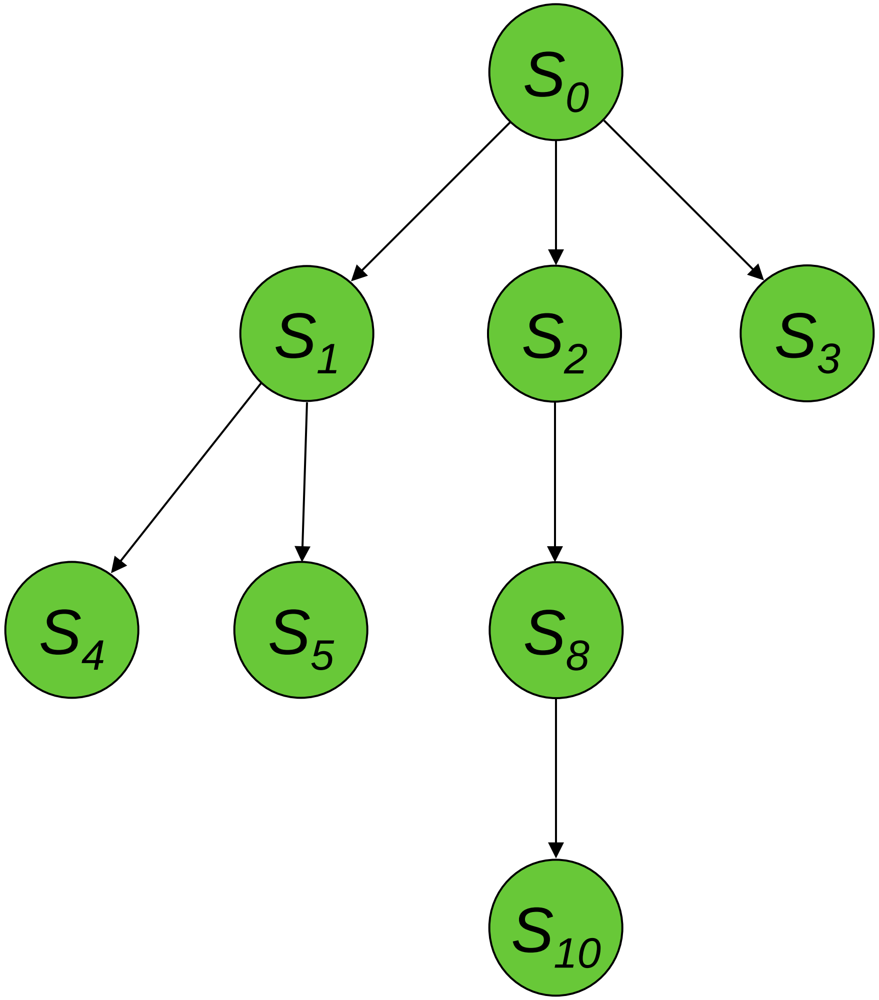
Enforced Hill Climbing



Best-First Search

- Enforced hill climbing is great when it works, but for some problems it's better to keep track of the nodes we haven't yet expanded, using the agenda
- We can then use the heuristic function to determine which node to expand next
- As new states are discovered, we add them to the agenda and record the value of the heuristic function
- When we pick the next node to explore, we choose the one which has the *lowest value* for the heuristic function (i.e. the one that looks nearest to the goal)

Best-First Search



Best-First Search

- To get best-first search, pick the best node on the agenda as the one to be explored next:

let Agenda = [S_0]

while Agenda ≠ [ ] do

    let Current = Best(Agenda)

    let Agenda = Rest(Agenda)

    if Goal(Current) then return("Found it!")

    let Next = NextStates(Current)

    let Agenda = Agenda+Next

Best-First Search and Algorithm A

- Best-first search can speed up the search by a very large factor, but can it isn't guaranteed to return the shortest solution
- When deciding to expand a node, we need to take account of how long the path is so far, and add that on to the heuristic value:

$$f(S_i,G) = g(S_0,S_i) + h(S_i,G)$$

- This will give a search which has elements of both breadth-first search and best-first search
- This type of search is called "Algorithm A"

Algorithm A*

- If *h*(S_i,*G*) never over-estimates the distance from Si to the goal, it is called an admissible heuristic
- If *h*(S_i,*G*) is admissible, then Algorithm A will always return the shortest path (like breadth-first search) but will omit much of the work if the heuristic function is informative
- The use of an admissible heuristic turns Algorithm A into Algorithm A*
- Uses: problem solving, route finding, path planning in robotics, computer games, etc.

Why is A* Optimal?

- Suppose a suboptimal goal node, S_k, appears in the agenda – we haven't selected it yet, so we don't yet know that it's a goal node
- Also on the agenda, there must be a node, S_i which is on the optimal path from S_0 to the goal state
- Since the heuristic function, $h(S_i,G)$, is admissible, this means:

$$g(S_0,S_k) + h(S_k,G) > g(S_0,S_i) + h(S_i,G)$$

so S_k will never be selected over S_i for expansion.

Heuristic Functions

- Consider the 8-puzzle:

7	2	4
5		6
8	3	1

→

1	2	3
4	5	6
7	8	

- Can we come up with a good admissible heuristic function for this problem?