Problem 1

Part 2

Breadth-First-Search

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Grid # | Total Nodes Generated | Max Nodes Stored At Once | Number of Iterations | Depth of Goal | Cost of path to goal | Length of path to goal |
| 1 | 71 | 7 | 25 | 7 | 17 | 8 |
| 2 | 34 | 2 | 15 | 10 | 30 | 11 |
| 3 | 234 | 7 | 83 | 31 | 77 | 32 |
| 4 | 331 | 8 | 100 | 27 | 60 | 28 |
| 5 | 189 | 5 | 79 | 44 | 104 | 45 |
| Total | 859 | 29 | 302 |  |  |  |

Depth-First-Search

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Grid # | Total Nodes Generated | Max Nodes Stored At Once | Number of Iterations | Depth of Goal | Cost of path to goal | Length of path to goal |
| 1 | 25 | 6 | 10 | 9 | 21 | 10 |
| 2 | 25 | 5 | 11 | 10 | 30 | 11 |
| 3 | 101 | 21 | 38 | 35 | 89 | 36 |
| 4 | 262 | 28 | 82 | 41 | 90 | 42 |
| 5 | 165 | 18 | 69 | 52 | 128 | 53 |
| Total | 578 | 78 | 210 |  |  |  |

Uniform-Cost-Search

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Grid # | Total Nodes Generated | Max Nodes Stored At Once | Number of Iterations | Depth of Goal | Cost of path to goal | Length of path to goal |
| 1 | 54 | 8 | 19 | 7 | 17 | 8 |
| 2 | 34 | 2 | 15 | 10 | 30 | 11 |
| 3 | 164 | 11 | 61 | 31 | 77 | 32 |
| 4 | 281 | 22 | 84 | 27 | 60 | 28 |
| 5 | 195 | 7 | 82 | 44 | 104 | 45 |
| Total | 728 | 50 | 261 |  |  |  |

A\*-Search: Manhattan Heuristic

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Grid # | Total Nodes Generated | Max Nodes Stored At Once | Number of Iterations | Depth of Goal | Cost of path to goal | Length of path to goal |
| 1 | 39 | 8 | 14 | 7 | 17 | 8 |
| 2 | 29 | 2 | 13 | 10 | 30 | 11 |
| 3 | 150 | 11 | 56 | 35 | 89 | 36 |
| 4 | 109 | 20 | 39 | 27 | 60 | 28 |
| 5 | 128 | 10 | 54 | 44 | 104 | 45 |
| Total | 455 | 51 | 176 |  |  |  |

A\*-Search: Euclidean Heuristic

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Grid # | Total Nodes Generated | Max Nodes Stored At Once | Number of Iterations | Depth of Goal | Cost of path to goal | Length of path to goal |
| 1 | 31 | 7 | 12 | 11 | 27 | 12 |
| 2 | 29 | 2 | 13 | 10 | 30 | 11 |
| 3 | 149 | 13 | 56 | 35 | 89 | 36 |
| 4 | 119 | 21 | 42 | 27 | 60 | 28 |
| 5 | 126 | 10 | 53 | 44 | 104 | 45 |
| Total | 454 | 53 | 176 |  |  |  |

Iterative-Deepening-Search

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Grid # | Total Nodes Generated | Max Nodes Stored At Once | Number of Iterations | Depth of Goal | Cost of path to goal | Length of path to goal |
| 1 | 25 | 6 | 10 | 9 | 21 | 10 |
| 2 | 25 | 5 | 11 | 10 | 30 | 11 |
| 3 | 101 | 21 | 38 | 35 | 89 | 36 |
| 4 | 262 | 28 | 82 | 41 | 90 | 42 |
| 5 | 165 | 18 | 69 | 52 | 128 | 53 |
| Total | 578 | 78 | 210 |  |  |  |

2. Which Search Algorithm

1. Fewest total nodes: A\* Euclidean Heuristic search, just beat A\* Manhattan Heuristic Search by 1 node.
2. Stored fewest nodes at one time: Breadth First Search
3. A\* Search Algorithms

3. The informed search algorithms did much better than the uninformed because they had a better understanding of their environments, and were able to more optimally choose successor nodes based on this knowledge. This let the algorithm choose optimal paths before choosing sub-optimal paths. Having no knowledge of an environment creates the necessity to check all possibilities for successor nodes, which can inefficient either with time or space complexity.

Part 4:

1. Local search does not do well in 8-puzzle, as only 10-15 puzzles were solved out of 25000 generated. 8-puzzle creates scenarios where it might be beneficial to make a move away from the goal in order to find a solution – otherwise, it can get stuck at any local maxima. 8-queens works well with local search because it can continue placing queens until it finds a goal state. In 8-puzzle, trying to improve the search space configuration is tricky and often plateaus without any major improvement.

2. Simulated annealing can be better than local-search because it can probabilistically infer successor moves based on the search space, which can use a function to provide more information than just the current configuration. Random Restart Hill Climbing would also work better because once it reaches a local maxima, unlike a regular hill-climbing algorithm, random restart allows for a new starting point, rather than staying stuck.

Problem 2

Part 1:

1.

1. A B C D E F
2. A D C F E B
3. A D C F E B (assuming d = 1, until algorithm stops at d = 3
4. A D B C E F
5. A C F E B D
6. A C E F B D

2.

a) Creating a 4x4x2 boolean array to mark states visited creates 32 representable states. Example: Visit(true, 1, 2) marks that 1 cannibal and 2 missionaries on the right (goal) side of the bank has occurred so far.

b) 32 representable states

c) Initial state: Visit(true, 0, 0), indicating that there are 0 cannibals and 0 missionaries on the left side of the bank at the start.

d) Visit(true, 3, 3), indicating the goal state of 3 cannibals and 3 missionaries on the right side of the bank.

e) Potential successors of an arbitrary state will include all possibilities of (false, cannibal+1, or missionary+1). A transition will be defined by marking one of these potential successor states as true.

f)

Visit(true, 0, 0)

Visit(true, 2, 0)

Visit(true, 1, 0)

Visit(true, 3, 0)

Visit(true, 2, 0)

Visit(true, 2, 2)

Visit(true, 1, 1)

Visit(true, 1, 3)

Visit(true, 0, 3)

Visit(true, 2, 3)

Visit(true, 3, 3)

3.

Let k(n) denote the least expensive path from a (starting node) to the goal node.

Prove: Every consistent heuristic is admissible, i.e. h(n) <= k(n).

Base Case: if starting point is the goal, then h(n) = 0 <= k(n).

Inductive Step: if the starting point to the goal consists of i steps, the best path from n to n’ (successor node) has i-1 steps. This gives us a proof by consistency: h(n’) <= k(n’).

h(n) <= c(n, a, n’) + h(n’) <= c(n, a, n’) + k(n’) = k(n).

4. Local Search in general

a) Two possible advantages of local search over graph / tree search are that it is very space efficient, only saving 1 node in memory at a time, and always improving, which allows it to often solve large, continuous problems (in addition to optimization problems). Two possible disadvantages of local search vs. graph / tree search are that when you hit a maxima or plateau, there’s no better successor. Additionally, there’s no clear answer on how often to restart or try to “repair” by choosing successor moves randomly (Lecture, Slide 13).

b) Hill-climbing search is a loop that continually move in direction of increasing value, and terminates when reaching a peak. Genetic algorithms start with k randomly generated states, represent each state as a string, and rate each state using an objective function (Lecture 6, Slide 21). Genetic algorithms are different than Hill-climbing algorithms, because they have an uphill tendency but not expectation, they explore randomly (mutations occur), and there is an exchange of information across parallel search threads.

c) Simulated annealing search picks a random move and accepts with a probability if it improves. Random restart searches independently, without passing information about probability.

5.

a)

if L:

pole = NudgeRight()

else:

pole = NudgeLeft()

# if initial state is V, do nothing.

b)

while(pole != vertical):

# Establishes a new state for the pole after being nudged.

randomNudge = randInt(0, 1)

if randomNudge == 0:

pole = NudgeLeft

else:

pole = NudgeRight

Problem 3: Adversarial Search

Minimax: minimize the possible loss for a worst-case scenario.

B(4)

C(3)

D(7)

A(7)

The leaf nodes are the same.

If we are using alpha-beta pruning left to right, we can skip child-nodes of D, which are: K, L, and M.

Apply minimax to the tree with chance nodes:

B(10)

C(20)

Chance node: 15

D(30)

E(20)

Chance node: 22

A(22)

Problem 1, Part 1: Appended code



















