Cameron Jewell Worked with Ethan Smyth, Dillon Thompson, Naeem Ghossein

9/15/19

Artificial Intelligence

Homework 1

1. N-Queens problem for 25 queens

* Score function: f(s) = number of conflicting queens in state s (we want a low score)
* Neighborhood: set to the immediate vertical neighbors of a selected queen
* The best state that could be generated *SHOULD* contain a score of f(s) = 0 where there are no conflicts. In our case, the first acceptable solution we found was:

[25, 19, 10, 20, 15, 6, 16, 3, 24, 21, 8, 5, 17, 1, 7, 2, 22, 9, 18, 14, 11, 23, 12, 4, 13]

* The above solution resulted in a score f(s) = 0, or no conflicting queens

Our algorithm calculates the first solution achieved from a predesigned initial state. In our case, the initial state was queens along the diagonal, beginning with the bottom left corner. The solution in obtained by moving the first queen to all of its available neighbors and remaining at the location with the smallest f(s). This was repeated for all queens progressing from left to right across the board until the first score of 0 is achieved.

1. As this particular problem is not very complicated and can be solved by hand, the algorithm devised to solve this problem will be relatively simplified.

Each tile in this particular initial state is at most 2 moves away from its individual goal state. As such, each turn will require consideration of at most 4 moves. These possible moves will be defined as all tiles neighboring the empty tile. For instance, if the empty tile is along the outside of the board, you will have at most 3 neighbors and at minimum 2 neighbors. Below is pseudocode for a possible algorithm to solve the puzzle:

1. define state ({2,3,7,4,5,1,11,8,6,10,12,15,9,14,20,13,16,17,18,19})
2. score = scoreFunciton(state)

3 while score > 0

4 neighbors = getNeighbors(state)

5 for neighbor in neighbors

6 newState = makeMove(neighbor)

7 temp = scoreFunction(newState)

8 if temp <= score

9 state = newState

10 score = temp

A score function (f) can be defined as f(s) = sum of the total distance of every tile from their individual goal state (we want a low score). The neighborhood (n) is defined as all possible moves given a state (this is at most 4 possibilities and at least 2 possibilities).

Using the provided A\* algorithms, a total of 23 states were considered throughout the algorithm. The number of new unique states that were considered per step are: 4, 1, 2, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1, 2, 1. The number of steps required to solve this particular instance of the puzzle is 16. The **5th and 11th steps’ state** can be seen below.

A clock on the wall

Description automatically generated

1. We will prove A\* (given the heuristic function is admissible) is optimal using proof by contradiction. A\* is considered optimal if the heuristic is admissible (given) and the heuristic is monotonic. First, we assume the opposite, that A\* is not optimal.

For the heuristic to be monotonic, future states calculated by the heuristic function must show equal or better improvement (closer to the goal state). Given our assumption of the opposite, we may find heuristic states in which we are further from the goal state than before we started. This is in direct contradiction to the purpose of the algorithm to be ever approaching the goal state. Without a backtracking implementation, the algorithm has no use for increased estimates on the heuristics of a state.

Using the formula h(n) ≤ c(n, a, n’) + h(n’), we need to show h(n) !> c(n, a, n’) + h(n’). Since h(n) is admissible (given), for h(n) > c(n, a, n’) + h(n’) we must not have consistency AND the function must not be admissible. However, since we know the h(n) IS admissible, the above cannot be true and our assumption must be false. Thus proving the algorithm is optimal.

1. Uninformed cost search requires less space than Dijkstra’s algorithm as Dijkstra’s algorithm adds all nodes to a queue at the start before processing paths. Dijkstra’s algorithm is a variant on uninformed cost search since there is no uniformly defined goal state (rather an attempt at finding the least cost path for all nodes). Where Dijkstra’s algorithm finds paths between all nodes to narrow down the *best* solution, uniform cost search stops as soon as the finish is found.
2. A new heuristic function approach (similar to the previous definition) would be to gather the neighbor states as per usual and pick one available state at random and check for an improved score.

In our previous case the number of explored states per step were 4, 1, 2, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1, 2, 1. The improved heuristic function would have a worst case complexity of exactly the same as the previous heuristic. However, the best-case scenario would result in the number of explored states per step to be 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1. This is an improvement of 7 states over the previous heuristic for a total of 16 explored states. The number of steps would remain the same at 16. The **5th and 11th steps’ states** remain the same as well as per our definition of what states are considered.