CSCI - B 551: Assignment #1

Due on Sunday, September 25, 2016

Prof. Crandall

bansalro/vpatani/zehzhang

Contents

Problem 1

Solution to Question 1 will be in the folder *problem1*
Abstraction

Problem 2

15 puzzle Problem Abstraction:

- **Initial State:** A random/user input state with 15 tiles on the board, each occupying exactly one place on the board. No duplicates are allowed. The board may often times be unsolvable.
- **Goal State:** Tiles arranged in the correct order from 1 to 15 and the blank space at the end after the 15th tile, wherein one tile exactly occupies one place.

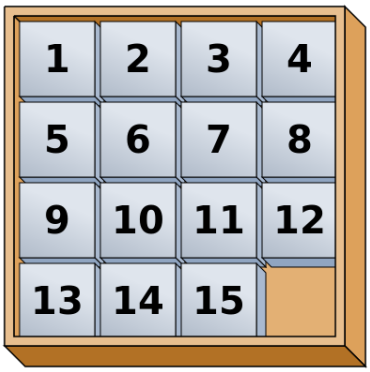

Figure 1: A Solved Board

- **Valid State:** Any state generated by the successor() is valid, where each tiles occupies exactly one place.
- **State Space:** State Space is the collection of all the states that have been discovered but not visited, they are stored in a fringe, which constantly pops the state with the lowest cost (i.e. the sum of g(s) + f(s)).
- **Successor Functions:** Through each state there are 4 possible states, wherein the tile can slide D, U, R, L into the empty place. Technically only 3 would be good to use depending upon the previous state, otherwise you would be replicating the parent state by moving back in the same position. Eg: If your parent is L, then moving in the Right direction would lead you back to the same state, hence we can eliminate that.
- **Cost Functions:** Cost Function is the cost of travelling from one state to another which is 1 for each movement. Also we use a heuristic to estimate the length of the goal which helps us get to the solution quicker than the brute force way.

Implementation Details: Try to make your solver as fast as possible. Which heuristic functions did you try, and which works best?

- We tried three different heuristics:
  - **Hamming Distance:** This heuristic calculates the number of misplaced tiles. In other words, all it does is that it checks whether if a tile is in its location, if not adds 1 to the cost. This is **not** a particularly good estimate of how far are we from the solution. The idea of a heuristic is to estimate the distance from the goal which in turn allows us to select a good probable path. Even though this is admissible since it does not overestimate the cost, it duly underestimates it most of the times.

Problem 2 continued on next page...

- **Permutation Inversion:** This heuristic particularly checks for inversion, which means it checks how many pairs are out of order. This is often times **not admissible** and hence can be discarded, since heuristic functions do not allow us to overestimate the distance to the goal.
- **Manhattan Distance:** This heuristic calculates the steps for each tile to reach its correct position when no other tiles is on the table. This is the **best of the three**, since it does not over estimate but also gives us a fair amount of idea as to how far are we from the solution. Also, the cases for edge movements have been handled for the flip tile moves otherwise it would have been over estimated because we can travel 3 moves in the x direction, but with the flip tile movement if at all you have to travel 3 steps, you could always do it one step and the same goes for y.

- We implemented A search algorithm to find the optimal path to the solution, code would be present in the *problem2* folder. We have used tuples to enhance efficiency and also because it can be easily hashed which in turn reduces the access times to check visited states and storing the elements in the fringe set.

- **Algorithm:**

```
if (current_state is goal_state) return initial state
fringe.add (initial_state)
Repeat until fringe is empty:
     fringe.pop() <- current
     visited.add (current)
     if current == goal: return path
     for each_successor in successor_list(s):
          if successor in visited: ignore
          calc_heuristic()
          if successor in fringe: old_successor = fringe.pop()
               compare successors and insert smaller one
          if not in fringe, insert.
```

- Tests carried for professors input, we obtain a path of 24 steps (LLUURULULDRDRDRRULDRUUUL) within about 2 seconds (0:00:02.189130) on our local machines. There are various test cases we tried, some take a while and some happen quickly, it fairly depends upon the complexity of the input.

Problem 3

The Marriage Problem Abstraction:

- **Monte Carlo Descent**
  - **State Space:** Represented by an M-element vector, where the index of each element corresponds to a certain guest, and each element has a non-negative integer as its value corresponding to the index of the table where the guest are seated. M is the total number of the guests.
  - **Initial State:** No guests have been seated in any table, represented by an m-element vector where each element is set to 0, which means that the corresponding guest has not been seated in any table yet.
  - **Goal State:** Every guest has been seated in a table and the number of the tables is as least as possible, represented by an m-element vector where no elements are 0 and the maximum element is as least as possible.
  - **Cost Function:** It equals to the number of the tables.
  - **Successor function:** Assign a table to a guest who has not been seated yet, given the condition that
    1. This new state is never visited before
    2. After assigning, the number of guests in the assigned table will not exceed a given number N, and
    3. Guests in that table do not know each other before. item It is represented by assigning a positive integer P to an element which is currently 0, and making sure that after assigning
       * The new state is not in our visited state records.
       * The number of P in the vector will not exceed N.
       * Guests with the index that has an element of P do not know each other.
  - **Edge Weights:** It will equal 0 or 1. If after assigning, there will be one more table, then edge weight equals to 1. Otherwise, if the number of tables do not change, the edge weight equals to 0.
  - **Heuristic Function:** It equals to the number of the tables,. Since we directly take the cost function as the heuristic function, it must be admissible.

$$h(s) = numberoftables$$

  - **Algorithm:**
    * Repeat L times (in my code, L = 10 * the number of the guests):
    * s = initial state
    * Repeat K times (in my code, K = 10 * the number of the guests):
      · If s is our goal, then return s
      · Pick s' from the successors of s at random
      · If h(s') $\leq$ h(s) (we want to find the s' that makes h(s') as least as possible!), then s = s'
      · Else with probability of exp(-(h(s')-h(s))/T), s = s' In my code, because h(s') is either equal to h(s) or 1 $\geq$ h(s), so in this case I can directly replace $h(s') - h(s)$ by 1. And $T = MAX_TEMPERATURE * (1thetimeswerepeat2)/K)$, which makes T decrease as the times we repeat 2) increases.
      · After repeating Step 1 and 2 L times, L results will be got. Then the algorithm compares the results with each other based on the number of the tables and chooses the result with the least number of tables as the final result.

Problem 3 continued on next page...

    - The problem I met is though Monte Carlo is much faster, it struggles to get the best result sometimes when just running it once. Therefore I repeat it several times and choose the best result as the final result.

- **A Search** The Abstract is similar to Monte Carlo
  - **State Space:** Represented by an M-element vector, where the index of each element corresponds to a certain guest, and each element has a non-negative integer as its value corresponding to the index of the table where the guest are seated. M is the total number of the guests.
  - **Initial State:** No guests have been seated in any table, represented by an m-element vector where each element is set to 0, which means that the corresponding guest has not been seated in any table yet.
  - **Goal State:** Every guest has been seated in a table and the number of the tables is as least as possible, represented by an m-element vector where no elements are 0 and the maximum element is as least as possible.
  - **Cost Function :** It equals to the number of the tables.
  - **Successor Function:** Assign a table to a guest who has not been seated yet, given the condition that 1) this new state is never visited before, 2) after assigning, the number of guests in the assigned table will not exceed a given number N, and 3) guests in that table do not know each other before. It is represented by assigning a positive integer P to an element which is currently 0, and making sure that after assigning, 1) the new state is not in our visited state records, 2) the number of P in the vector will not exceed N, and 3) guests with the index that has an element of P do not know each other.
  - **Edge Weights:** It will equal 0 or 1. If after assigning, there will be one more table, then edge weight equals to 1. Otherwise, if the number of tables do not change, the edge weight equals to 0.
  - **Heuristic Function:** t equals to the number of the tables,. Since we directly take the cost function as the heuristic function, it must be admissible.

$$h(s) = numberoftables$$

  - **Algorithm:**
    - Add initial state to the fringe. The fringe is implemented by using a priority queue (in Python, its called heapq). The priority is generated first based on the number of the tables in the state. If there are two states with the same total number of the tables, then they will be compared based on the order they are added to the fringe.
    - Repeated while fringe is not empty:
      - Pop the head h of the fringe
      - If h is our goal, return h
      - Push the successors of h into the fringe
    - Return False

References

1. 15 Puzzle solved image
2. Wolfram
3. Puzzle Generator
4. Solvability Test
5. Solvability
6. Stack Exchange
7. Stack Exchange

References continued on next page...