Report

COSC276, Fall2021, PA2, Xuedan Zou

A. Description

In this project, we solved in total three main problems: the impletation of the general A*, the modeling of multi-robot coordination problem and the modeling of blind robot problem with Pacman physics. The later two problems were solved based on the first A* algorithm.

Firstly, A* is a heuristic algorithm which combines UCS and greedy algorithms. It always expands the paths that has the lowest expensive cost. Here the cost means the actual transition cost from the beginning node to the target node plus the heuristic value of the target node. To implement A* algorithm, we surely need a data structure to save our successor nodes, and hopefully, can order those nodes with their priority automatically. Here we choose **Heap** to achieve this requirement with an acceptable time complexity. However, we can not easily visit the elements in the heap but wehave to find if a specific node is exactly in the heap and replace that node with the lower cost new node. To solve this problem the following code is used:

```
if child_node < i:
                    # replace the old node with the new one:
                    i = pqueue[-1]
                    pqueue.pop()
                    heapify(pqueue)
                    heappush(pqueue, child_node)
```

We destroied the original heap's structure by using the last element of the heap to replace the node we plan to update, then we use *pqueue.pop()* to delete the last element (the node we plan to update) and *heapify(pqueue)* to reconstruct our heap, we finally push our new updated node to this heap. Since the time complextiy to construct a new heap is O(n), this method is acceptable. (other methods like marking the old node to be "unable to visit" and add the new node to the heap can surely work as well). We also use a **Dictionary** to save the states that we have explored before to avoid multiple exploration of the same state. Noticed that since *dictionary* can not save *list and set* to the hashnumber, we should translate our given state to "tuple". Besides, to better compare the priority of the states, we should rewrite the _ *lt* _ method by returning the nodes' priority comparition result. If we define constent heuristic value to our problem then A* algorithm is guaranteed to find the best solution.

Secondly, to solve the multi-robot coordination problem, we define each state as a list of position: *(n, a1, a2, b1, b2,....)* where n refers to the current state's moving robot, a1 and a2 refers to the first's robot's coordinates, followed by the other's. Each state there is only one robot that is plan to move. And for the moving robot it can choose to go NSWE four directions and also stay at the same place. So there are in total five potential actions for each state, we should check if there were any wall or robot for each possible moving action and if not that action could be done successfully. Following this idea we get our successor funtion to our problem. We then set different heuristic funtions to our model and use the previous A* algorithm to solve this problem.

Finally, we use a **set** of the all the possible robot's location to define the state in the blind robot problem. The location of the possible robot's location in each state can be expressed as *{(a1, b1), (a2, b2), (a3, b3),..}*. In the beginning of this problem we add all the floors' locations to the set since robot can be anywhere on the floor. Then there are NSWE four possible moving directions for each possible robot's location. We caculate the results of a moving direction for all previous possible locations and add them to the new set. Since set can help us merge those same possible robot's locations, the set is supposed to be a singleton in the end, meaning there is only one location for the current's robot and so that we guide our robot successfully. We use a constent heuristic funtion here which calculates the largest manhanttan distance of all the given robot's location by subtracting the lowest x positon value with the largest and subtracting the lowest y positon value with the largest. Finally A* algorithm is used to sovle the blind-robot problem with our given model and heuristic function.

B. Evaluation

Both of the multi-robot coordination problem and the blind robot problem can be successfully solved using our A* algorithm and given heuristic function. However, when we tested our method with large map, it can be quite time confusing (and even can not give out the result sometimes).

Below we choose some cool examples of the multi-robot coordination problem:

```
~
##.##
#...#
#.#.#
#...#
#BC.#
#A###
~
```

We ask the ending location of robots to be (1,0,1,1,2,1) just the same as the beginning and all robots do stay in the same place and do nothing.

```
~
A####
#.###
##.##
###.#
####.
~
```

Here we ask A to go to the right down side and obviously there is only a slash way to it so there is no result.

```
~
##.##
#..B#
#.#.#
#...#
#...#
```

```
#A###
```

~

Here we ask those two robots to change their locations:

~

```
##.##
#..B#
#.#.#
#...#
#...#
#A###
```

```
##.##
#...#
#.#B#
#...#
#...#
#A###
```

```
##.##
#...#
#.#B#
#...#
#A..#
#.###
```

```
##.##
#...#
#.#B#
#A..#
#...#
#.###
```

```
##.##
#...#
#.#.#
#A.B#
#...#
#.###
```

```
##.##
#...#
#.#.#
#A..#
#..B#
#.###
```

```
##.##
#...#
#.#.#
#..A#
#..B#
#.###

##.##
#...#
#.#A#
#...#
#..B#
#.###

##.##
#...#
#.#A#
#...#
#.B.#
#.###

##.##
#...#
#.#A#
#...#
#B..#
#.###

##.##
#..A#
#.#.#
#...#
#...#
#B###
~

~
ABC
DE.
FGH
~
```

Here we give another given state and ask the program to find the way. It is exactly the 9-puzzle question.

```
~
ABC
...
EFG
```

~

Here we ask the change of row between ABC and EFG.

We have also tested our program on some larger size of maps with interesting results (since the space is limited in the report I omit the last two solutions in this part).

For the blind robot problem, give this test for example:

~
```
#.....
####..
...#..
.#.#..
.#..#.
......
```
~

Our program gives out the result:

~
```
#RRRRR
####RR
RRR#RR
R#R#RR
R#RR#R
RRRRRR

#.RRRR
####.R
.RR#.R
R#R#.R
R#.R#R
.RRRRR

#.RRRR
####.R
RRR#.R
R#.#.R
.#RR#R
.R..R.

#.RRRR
####.R
RRR#.R
.#R#.R
.#.R#.
.R..R.
```

```
#..RRR
####.R
.RR#.R
.#R#.R
.#.R#.
..R..R

#...RR
####.R
..R#.R
.#R#.R
.#.R#.
...R.R

#.....
####RR
...#.R
.#R#.R
.#R.#R
...R.R

#.....
####..
...#RR
.#.#.R
.#R.#R
..RR.R

#.....
####..
...#..
.#.#RR
.#..#R
..RR.R

#.....
####..
...#..
.#.#.R
.#..#R
...RRR

#.....
####..
...#..
.#.#..
.#..#R
```

```
...RRR

#.....
####..
...#..
.#.#..
.#..#R
....RR

#.....
####..
...#..
.#.#..
.#..#R
.....R

#.....
####..
...#..
.#.#..
.#..#.
.....R
~
```

That is E,N,N,E,E,S,S,S,E,S,E,S,S.

C. Discussion

a. Multi-robot coordination problem

1. Since we already have the maze map in the beginning, for each moving robot, we only have to know the other robots' locations to decide its all potential following movements to the next state. Thus, as we disscussed in section A, we use *(i, a1, a2, b1, b2, ...,k1, k2)* to represent the state of a k robots system. Here i refers to the current moving robot, a1 and a2 refers to the first's robot's location, followed by the other's. For each state there is only one robot that is plan to move.
2. The upper bound on the number of states in the system can be *k*((n^2)!/(n^2-(k-2))!)*. Thinking about *(i, a1, a2, b1, b2,...,k1,k2)*, there are k possiblities for each i (in total k robots). For the first *(a1, a2)* we have n possiblities for both a1 and a2, so all the possiblities is *n*n*. For the second *(b1, b2)* we have the rest *n*n-1* possibilites for both b1 and b2. Following this idea, we can easily get the result: *k*n^2*(n^2-1)*(n^2-2)...*(n^2-(k-1))* and that is *k*((n^2)!/(n^2-(k-2))!)*.
3. Since there are w walls, we calculate those states without any colliisons: *k*(n^2-w)*(n^2-w-1)*...*(n^2-w-(k-1))*, that is *k*((n^2-w)!/(n^2-w-(k-2))!)*. Then we use the previous all possbile state minus this one and get all those which has collisons: *k*((n^2)!/(n^2-(k-2))!)-((n^2-w)!/(n^2-w-(k-2))!)*. (I am not really sure to be honest...)
4. Breadth-first search seems not to be a feasible solution to this kind of situation. Since the map is really large and with not many walls, if the start and goal pairs has a long distance then BFS will expand almost

all of the states and the upper bound on the number of states is almost 10*100*99*...89 (if there is no wall) and that is obviously not computaional feasible.

5. We can use euclidean distance here from the node to the current's robot's goal to define our heuristic function for this problem. Since in reality, the robot can only move NSWE and that uses manhattan distance. We know that two sides of a triangle is greater than the third, so does the manhattan distance(possible actual cost) is always greater than the euclidean distance here. Thus, our monotonic heuristic function is monotonic (constent).
6. Considering a 3*3 maze with 8 robots, our goal is to find the path to another state with given postion of each robot and that shows the 8-puzzle in the book is a special case of this problem. The heuristic function we choose here at least can still work in that 8-puzzel problem.
7. We can reset the start and final state in our program by letting each robot's positon in the beginning and the end to be mirroring. Then we can see that there will be no solution to this kind of situation and that proves the 8-puzzle is made of two disjoint sets. We can also run the problem by setting all the possible start and final states automatically, since the maze size is not so huge this should work.

b. Blind robot problem with Pacman physics

We have discussed the heuristic we used in this problem in the Description part. In short, it calculates the largest Manhanttan distance of all the given robot's location by subtracting the lowest x positon value with the largest and subtracting the lowest y positon value with the largest. Since all the possible locations of robot in the end should gather to be the only one and that can only moves with manhattan distance, we can prove that this heuristic function is optimistic. We can also calculate the Euclidean distance instead of the Manhanttan distance and that is also optimistic. But it will also be a lot more time confusing to do so (since we have to do square and sqrt).