Project: 1

CS 205. Artificial Intelligence

ChenYang Yu
ID 862052273
cyu059@ucr.edu
15-February-2018

Instructor: Dr Eamonn Keogh

In completing this homework, I consulted…

- The slide from class
- Misplaced Tile heuristic https://heuristicswiki.wikispaces.com/Misplaced+Tiles

All the important code is original. Unimportant subroutines that are not completely original are…

- priority queue http://zh.cppreference.com/w/cpp/container/priority_queue
- implement hash table by unsorted map
http://www.cplusplus.com/reference/unordered_set/unordered_set/insert/
- time measurement http://man7.org/linux/man-pages/man2/gettimeofday.2.html

Project: 1 CS 205. Introduction to Artificial Intelligence
Due Date: Feb 13th

8-puzzle

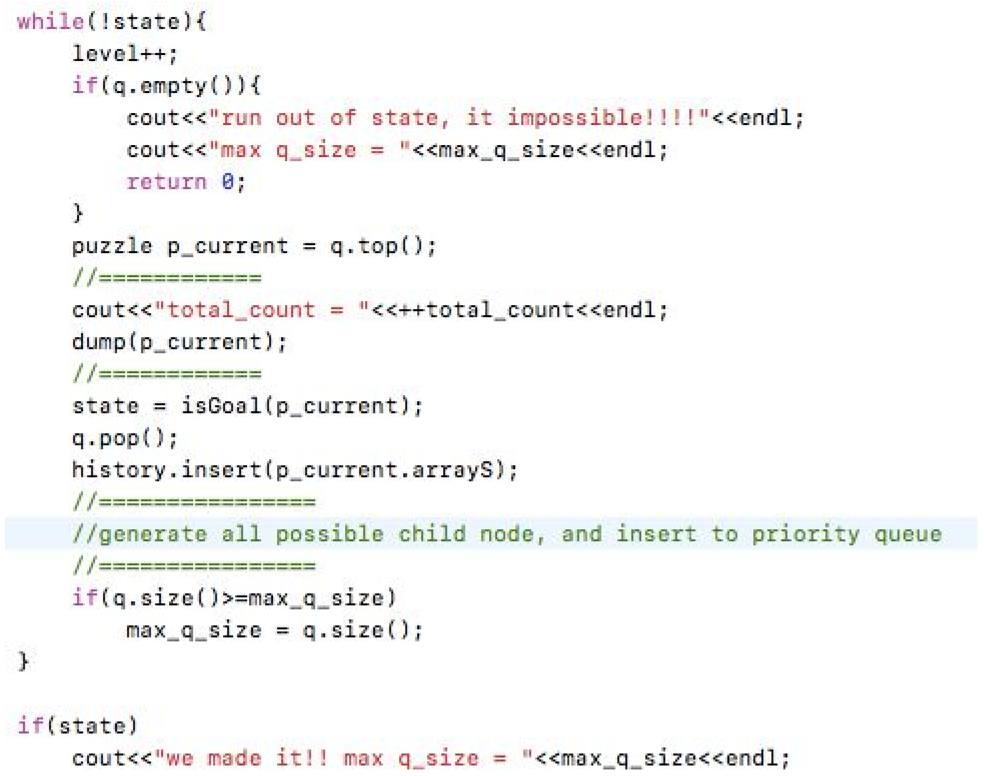
Target:

Implement the searching algorithm to solve n-puzzle problem: For a given initial n-puzzle, we can achieve the goal state by expanding a legal move. We should also be able to know if this puzzle is unsolvable. We choose three different algorithms to choose the next state and measure their performance under different level of puzzle.

Algorithm of searching

```
Push the initial puzzle into queue if it is not goal state (return success if it is the goal)
While( not goal state)
    If the queue is empty
        We run out of expandable state and fail
    Else
        Pop the puzzle p with the lowest distance from queue
        Expand all legal puzzle, calculate the distance and insert into queue
```

Implement as following

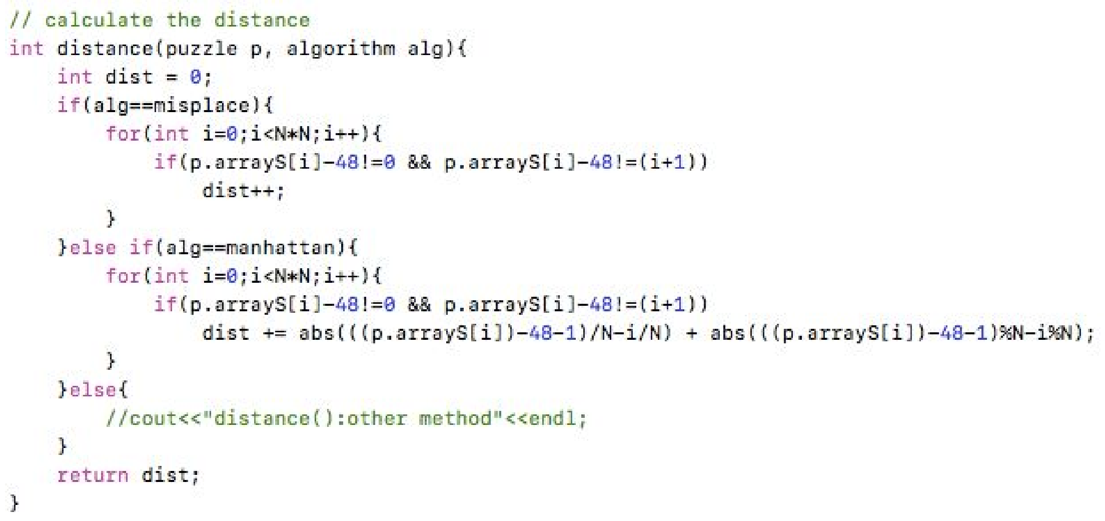


```
while(!state){
    level++;
    if(q.empty()){
        cout<<"run out of state, it impossible!!!!"<<endl;
        cout<<"max q_size = "<<max_q_size<<endl;
        return 0;
    }
    puzzle p_current = q.top();
    //============
    cout<<"total_count = "<<++total_count<<endl;
    dump(p_current);
    //============
    state = isGoal(p_current);
    q.pop();
    history.insert(p_current.arrayS);
    //================
    //generate all possible child node, and insert to priority queue
    //================
    if(q.size()>=max_q_size)
        max_q_size = q.size();
}

if(state)
    cout<<"we made it!! max q_size = "<<max_q_size<<endl;
```

Since the priority queue is based on distance, we expected we always get the lowest distance from the top of priority queue.

Calculation of distance

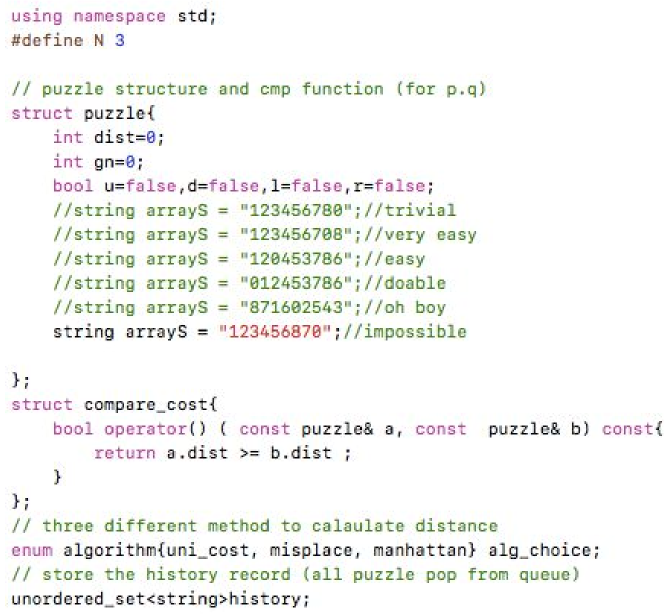


```
// calculate the distance
int distance(puzzle p, algorithm alg){
    int dist = 0;
    if(alg==misplace){
        for(int i=0;i<N*N;i++){
            if(p.arrayS[i]-48!=0 && p.arrayS[i]-48!=(i+1))
                dist++;
        }
    }else if(alg==manhattan){
        for(int i=0;i<N*N;i++){
            if(p.arrayS[i]-48!=0 && p.arrayS[i]-48!=(i+1))
                dist += abs(((p.arrayS[i])-48-1)/N-i/N) + abs(((p.arrayS[i])-48-1)%N-i%N);
        }
    }else{
        //cout<<"distance():other method"<<endl;
    }
    return dist;
}
```

1. Uniform Cost Search
   Take every step we expand require one cost. choose the lowest cost as best state. Return zero in the distance function.
2. A* with Misplace
   Use misplace heuristic function. For every element in puzzle, increase the counter if it it not the right digit. Return the counter in the end. Notice that we examine every element except zero since as long as we put other 8 puzzle in the right place, zero must be right. The sum of heuristic distance and cost is the A* misplace distance.
3. A* with Manhattan
   Use manhattan heuristic function. For every element in puzzle, calculate the distance of a digit between the current location and the location where it should be. Again, we examine every element except zero. The sum of heuristic distance and cost is the A* manhattan distance.

Function explanation

Basic data structure



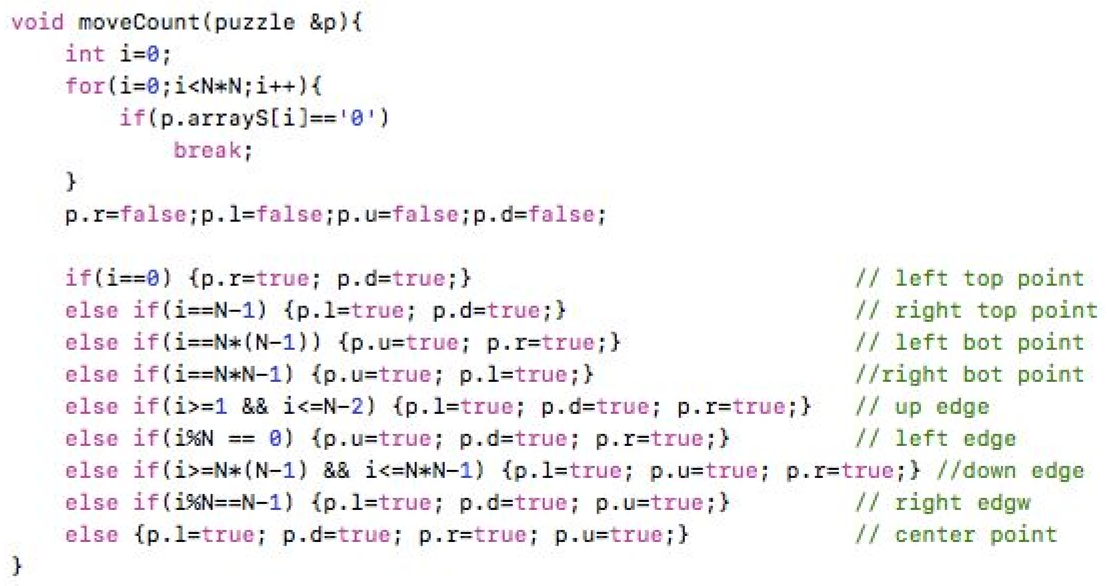
```
using namespace std;
#define N 3

// puzzle structure and cmp function (for p.q)
struct puzzle{
    int dist=0;
    int gn=0;
    bool u=false,d=false,l=false,r=false;
    //string arrayS = "123456780";//trivial
    //string arrayS = "123456708";//very easy
    //string arrayS = "120453786";//easy
    //string arrayS = "012453786";//doable
    //string arrayS = "871602543";//oh boy
    string arrayS = "123456870";//impossible

};
struct compare_cost{
    bool operator() ( const puzzle& a, const  puzzle& b) const{
        return a.dist >= b.dist ;
    }
};
// three different method to calaulate distance
enum algorithm{uni_cost, misplace, manhattan} alg_choice;
// store the history record (all puzzle pop from queue)
unordered_set<string>history;
```

Structure puzzle aim to store the puzzle and extra information for further usage. Initial distance is zero and the possible moving direction is all false (nowhere to go). All extra information will be assigned before we push them into queue.
The comparison function is used for queue, we sorting the queue according to the order of distance, in that case we can always choose the minimal distance from the top of the queue.

Check legal move

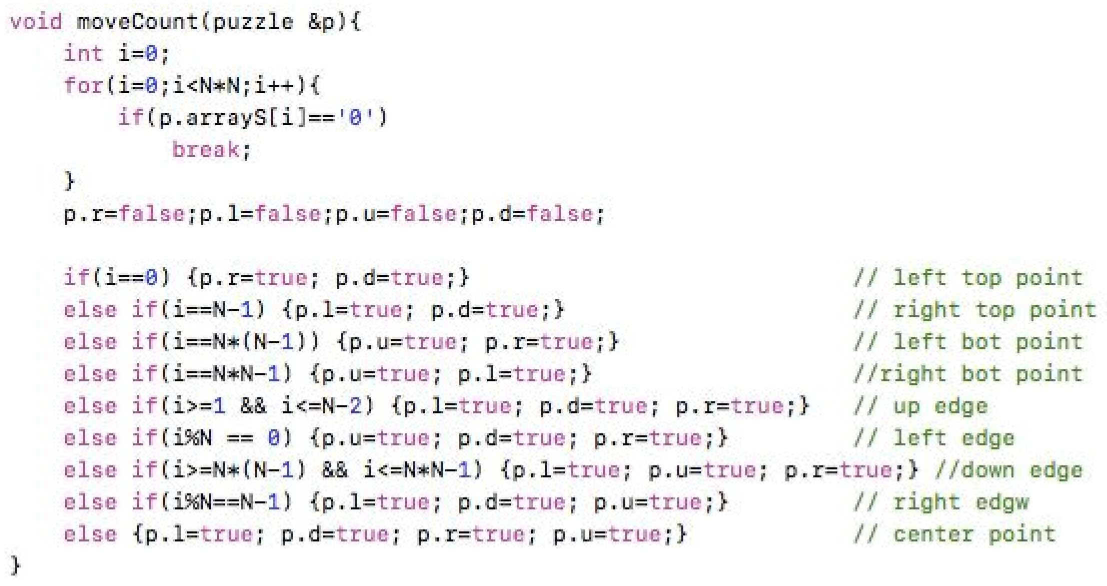


```
void moveCount(puzzle &p){
    int i=0;
    for(i=0;i<N*N;i++){
        if(p.arrayS[i]=='0')
            break;
    }
    p.r=false;p.l=false;p.u=false;p.d=false;

    if(i==0) {p.r=true; p.d=true;}                              // left top point
    else if(i==N-1) {p.l=true; p.d=true;}                       // right top point
    else if(i==N*(N-1)) {p.u=true; p.r=true;}                   // left bot point
    else if(i==N*N-1) {p.u=true; p.l=true;}                     //right bot point
    else if(i>=1 && i<=N-2) {p.l=true; p.d=true; p.r=true;}     // up edge
    else if(i%N == 0) {p.u=true; p.d=true; p.r=true;}           // left edge
    else if(i>=N*(N-1) && i<=N*N-1) {p.l=true; p.u=true; p.r=true;} //down edge
    else if(i%N==N-1) {p.l=true; p.d=true; p.u=true;}           // right edgw
    else {p.l=true; p.d=true; p.r=true; p.u=true;}              // center point
}
```

There are different situation when the blank is in 1. corner 2. edge 3. center. We assign different boolean value to indicate the legal movement.

Expand the legal node



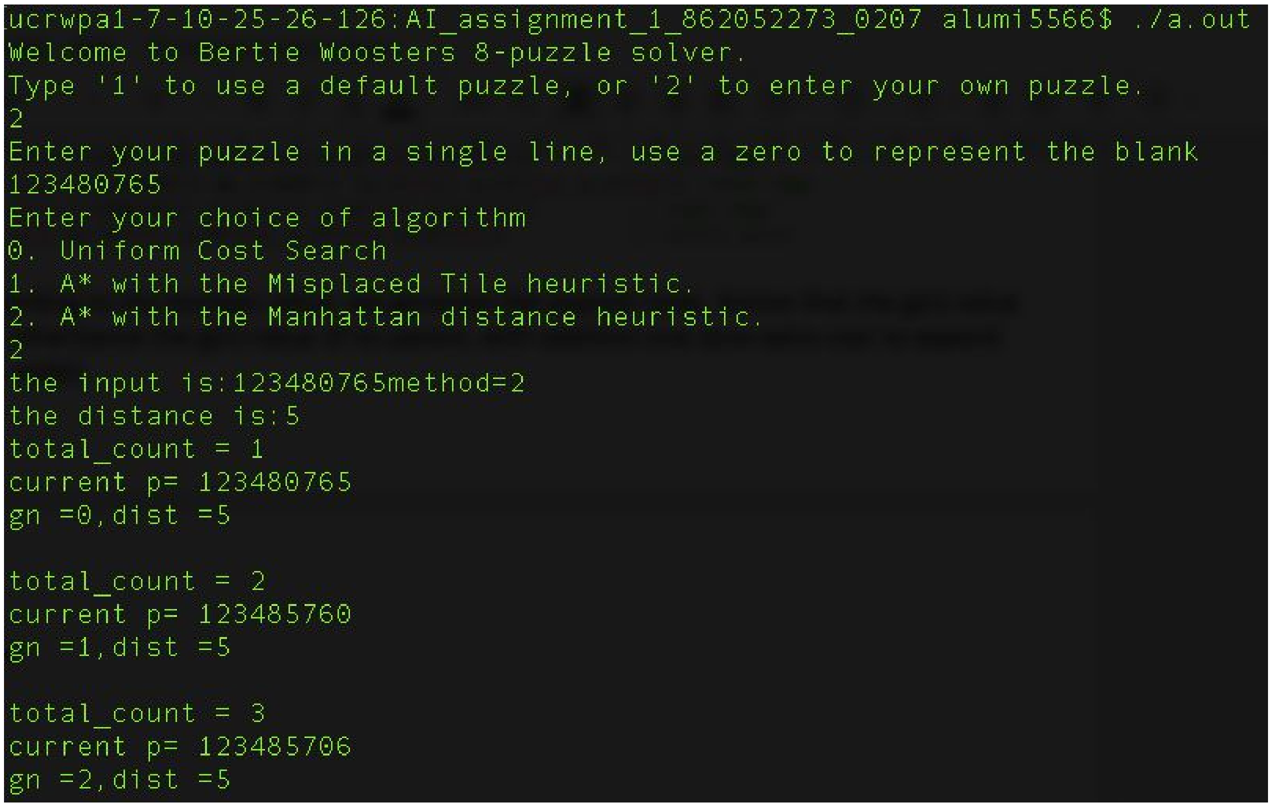
```
void moveCount(puzzle &p){
    int i=0;
    for(i=0;i<N*N;i++){
        if(p.arrayS[i]=='0')
            break;
    }
    p.r=false;p.l=false;p.u=false;p.d=false;

    if(i==0) {p.r=true; p.d=true;}                              // left top point
    else if(i==N-1) {p.l=true; p.d=true;}                       // right top point
    else if(i==N*(N-1)) {p.u=true; p.r=true;}                   // left bot point
    else if(i==N*N-1) {p.u=true; p.l=true;}                     //right bot point
    else if(i>=1 && i<=N-2) {p.l=true; p.d=true; p.r=true;}     // up edge
    else if(i%N == 0) {p.u=true; p.d=true; p.r=true;}           // left edge
    else if(i>=N*(N-1) && i<=N*N-1) {p.l=true; p.u=true; p.r=true;} //down edge
    else if(i%N==N-1) {p.l=true; p.d=true; p.u=true;}           // right edgw
    else {p.l=true; p.d=true; p.r=true; p.u=true;}              // center point
}
```

According to the boolean value, we generate the expand node. Notice that the g(n) value will inheritance the g(n) value of its parent, with addition one (one extra cost to expand this node)

Trace back

use {[1,2,3,
 4,8,0
 7,6,5]} as example and use A* manhattan as h(n)

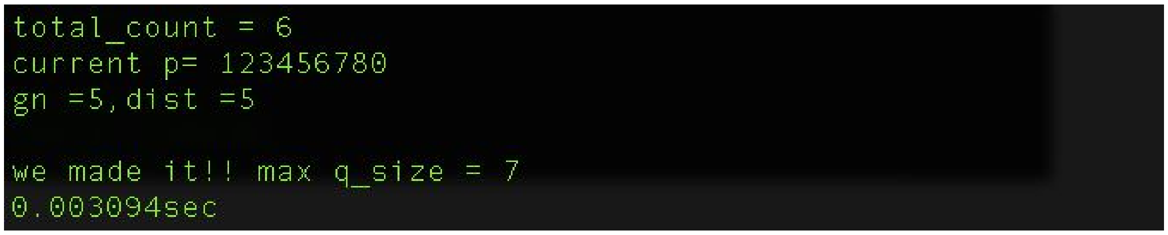


```
ucrwpa1-7-10-25-26-126:AI_assignment_1_862052273_0207 alumi5566$ ./a.out
Welcome to Bertie Woosters 8-puzzle solver.
Type '1' to use a default puzzle, or '2' to enter your own puzzle.
2
Enter your puzzle in a single line, use a zero to represent the blank
123480765
Enter your choice of algorithm
0. Uniform Cost Search
1. A* with the Misplaced Tile heuristic.
2. A* with the Manhattan distance heuristic.
2
the input is:123480765method=2
the distance is:5
total_count = 1
current p= 123480765
gn =0,dist =5

total_count = 2
current p= 123485760
gn =1,dist =5

total_count = 3
current p= 123485706
gn =2,dist =5
```

…



```
total_count = 6
current p= 123456780
gn =5,dist =5

we made it!! max q_size = 7
0.003094sec
```

Result:

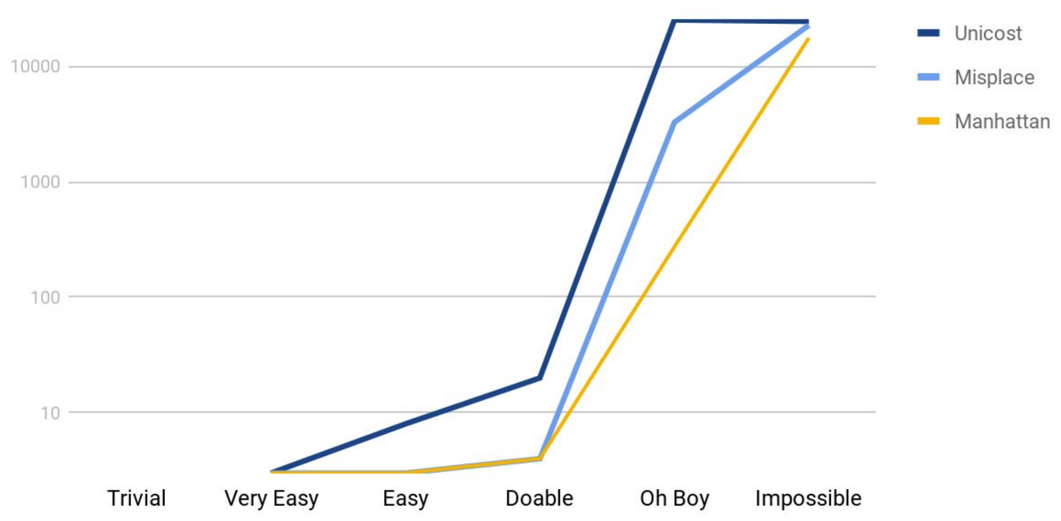


Fig.A Number of Nodes expands

	Trivial	Very Easy	Easy	Doable	Oh boy	Impossible
Unicost	0	2	7	25	92458	181440
Misplace	0	2	3	5	5579	181440
Manhattan	0	2	3	5	447	181440

Table.A Number of Nodes expands

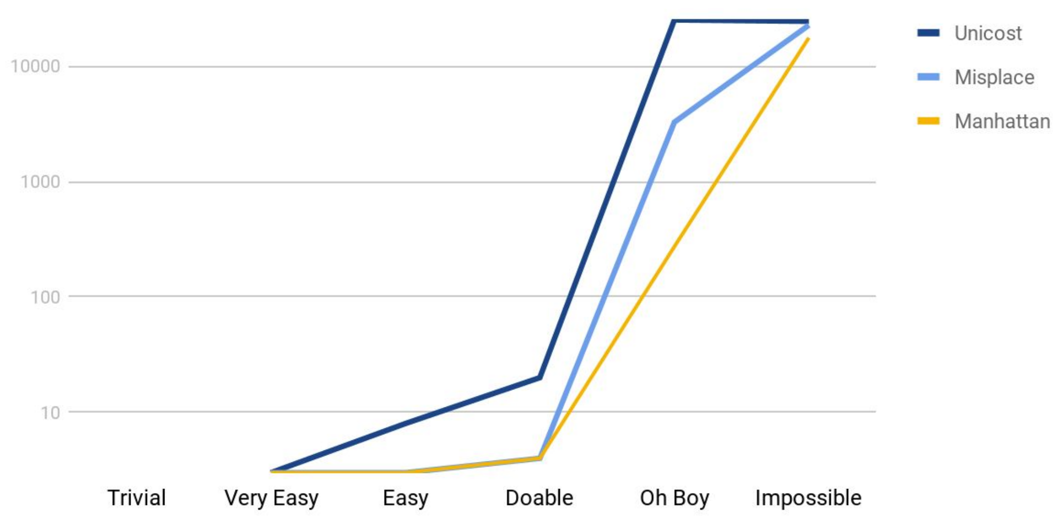


Fig.B Number of Nodes expands

	Trivial	Very Easy	Easy	Doable	Oh boy	Impossible
Unicost	0	3	8	20	25158	24470
Misplace	0	3	3	4	3270	22656
Manhattan	0	3	3	4	273	17618

Fig.B Maximal queue size

Testing environment

g++ 4.2.1 (Apple LLVM version 9.0.0) under OSX 10.13.3

Conclusion:

Table.A shows the states we need to check before we find the goal state(or decide there is no goal state), which indicates the time complexity. Table.B shows the maximal size of priority queue we use. Indicate the space complexity. On both criteria, the manhattan heuristic function is out performance the misplace heuristic function and unicost search. Overall the With the scale of problem increase, the difference between heuristic function becomes much more important.