Artificial Intelligence

Project1

信息与电子工程学院　信息工程 [redacted]

[redacted]

2023 年 4 月 1 日

1 N-Puzzle Promblem

1.1 Algorithm Description

First, the initial state and end state are passed in, and the open_list is defined to store the nodes to be accessed. The close_list is defined to store the nodes that have been accessed. The node accessed will be deleted from the open_list and added to the close_list. In the program we will take each node and determine whether the node is in the end state first. If so, the move_list from the initial state to the end state will be returned. If not, traver its child nodes which not in close_list, and update the cost of the child nodes. Then judege whether the node is in open_list, if so continue to next child node, if not, append it to open_list. Repeat the above process until the open_list is empty.

1.2 Key Function Design

1.2.1 Cost Design

Function update_cost(curr_state, dst_state, type).

In this function we will determine the heuristics function and cost function. For cost function, we just use $g = g + 1$ to calculate. For heuristics there are five different ways to calculate.

We first assume that the position of each point on a 4 * 4 chessboard are as shown in the following figure, so we can obtain the distance from each tile to its correct position as $x = |x_c - x_i|, y = |y_c - y_i|$, then we can calculate every heuristics.

(0,0)	(0,1)	(0,2)	(0,3)
(1,0)	(1,1)	(1,2)	(1,3)
(2,0)	(2,1)	(2,2)	(2,3)
(3,0)	(3,1)	(3,2)	(3,3)

Fig. 1: coordinate of each tile

First is hamming distance, this heuristics returns the number of tiles that are not in their final position. So we can traver all tiles, when $x \neq 0$ or $y \neq 0$, $h = h + 1$.

Second is euclidean distance, this heuristics is the sum of each number's euclidean distance to their correct position. So

$$h = \sum_{k=0}^{16} \sqrt{x_k^2 + y_k^2}$$

Third is chebyshev distance, this heuristics is the sum of each number's chebyshev distance to their correct position. Chebyshev distance is the maximun of x and y. So

$$h = \sum_{k=0}^{16} max(x_k, y_k)$$

Fourth is manhattan distance, this heuristics is the sum of each number's manhattan distance to their correct position. Manhattan distance is the sum of x and y. So

$$h = \sum_{k=0}^{16} x_k + y_k$$

The last is manhattan distance and linear conflict. Consider two tiles that are in the same row or column, and their goals are also in the same row or column, but they're in the wrong order. According to the Manhattan distance heuristic, both misplaced tiles are 1 tile away from their end position. However, it's impossible to swap the two tiles in just 2 moves. In fact, at least 2 additional moves are needed to move one out of the way for the other one to move into its place. So our heuristics is $manhattandistance + 2 * linearconflicts$. For example, in figure 2, there is linear conflict with 6 and 7, in the finnal heuristics we will calculate it as

$$h = h_{manhattan} + 1 * 2$$

.

Current State

X	X	X	X
X	7	6	X
X	X	X	X
X	X	X	X

Goal State

X	X	X	X
X	6	7	X
X	X	X	X
X	X	X	X

Fig. 2: Hamming Distance

1.2.2 Find Front Node

This function return the state of minimun $f(n)$. we just traver all state in open_list, compare their $f(n)$, then return the state of minimun $f(n)$.

1.2.3 State In List

This function return a flag and state, when state passed in is in list, we return ture and this state. Otherwise, return false and none.

1.2.4 Get Path

This function return the move_list for initiate state to current state. we can do a loop with condition of curr_state.pre_move != None. In this loop, we append the previous state's move in move_list, then go previous state. Finally, return the reverse of the move_list.

1.2.5 Expand State

This function return a list of current state's child nodes. We use function once_move to judge if child node is legal. If so, append it.

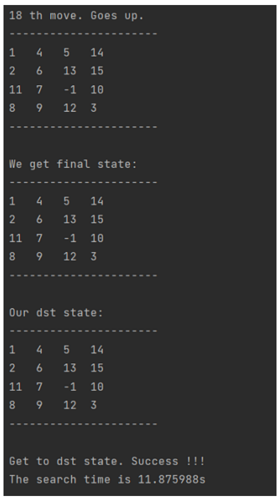
1.3 Test Result

1.3.1 Different Heuristics

Generate 120 moves and fix the inital state, different methods preformance as follow picture shows(other two methods take too much time). Sometimes the program can take a long time, but sometimes it can be quickly solved. But overall linear conflict with manhattan distance prforms best, and manhattan distance is better than euclidean and chebyshev which is better than hamming distance.

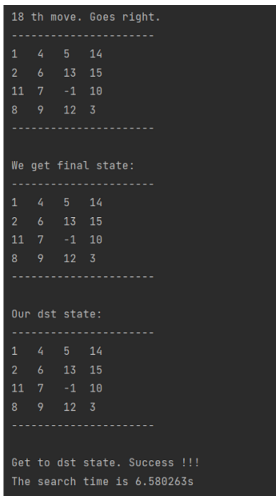
time(s) type / move	hamming	euclidean	chebyshev	manhattan	conflicts
20	0.042004	0.005996	0.025983	0.027014	0.003016
50	0.043990	0.117000	0.011003	0.052983	0.101919
80	22.651615	0.240972	1.235990	10.464123	0.199981
100	\	0.375999	13.310282	0.052976	1.241021
120	\	\	\	96.100062	1.327971
140	\	\	\	\	0.448739
170	\	\	\	\	73.362270

Table. 1: test results of five types



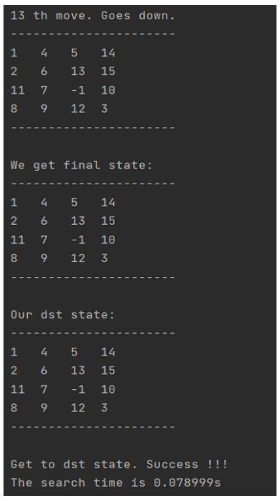
```
18 th move. Goes up.
---------------------
1   4   5   14
2   6   13  15
11  7   -1  10
8   9   12  3
---------------------

We get final state:
---------------------
1   4   5   14
2   6   13  15
11  7   -1  10
8   9   12  3
---------------------

Our dst state:
---------------------
1   4   5   14
2   6   13  15
11  7   -1  10
8   9   12  3
---------------------

Get to dst state. Success !!!
The search time is 11.875988s
```

(a) Chebyshev



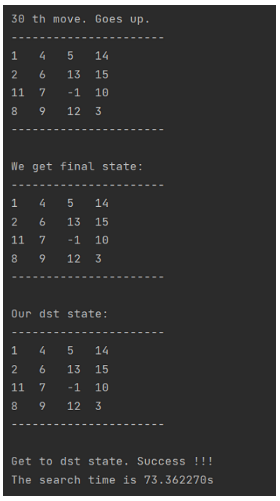
```
18 th move. Goes right.
---------------------
1   4   5   14
2   6   13  15
11  7   -1  10
8   9   12  3
---------------------

We get final state:
---------------------
1   4   5   14
2   6   13  15
11  7   -1  10
8   9   12  3
---------------------

Our dst state:
---------------------
1   4   5   14
2   6   13  15
11  7   -1  10
8   9   12  3
---------------------

Get to dst state. Success !!!
The search time is 6.580263s
```

(b) Manhattan



```
13 th move. Goes down.
---------------------
1   4   5   14
2   6   13  15
11  7   -1  10
8   9   12  3
---------------------

We get final state:
---------------------
1   4   5   14
2   6   13  15
11  7   -1  10
8   9   12  3
---------------------

Our dst state:
---------------------
1   4   5   14
2   6   13  15
11  7   -1  10
8   9   12  3
---------------------

Get to dst state. Success !!!
The search time is 0.078999s
```

(c) Conflict



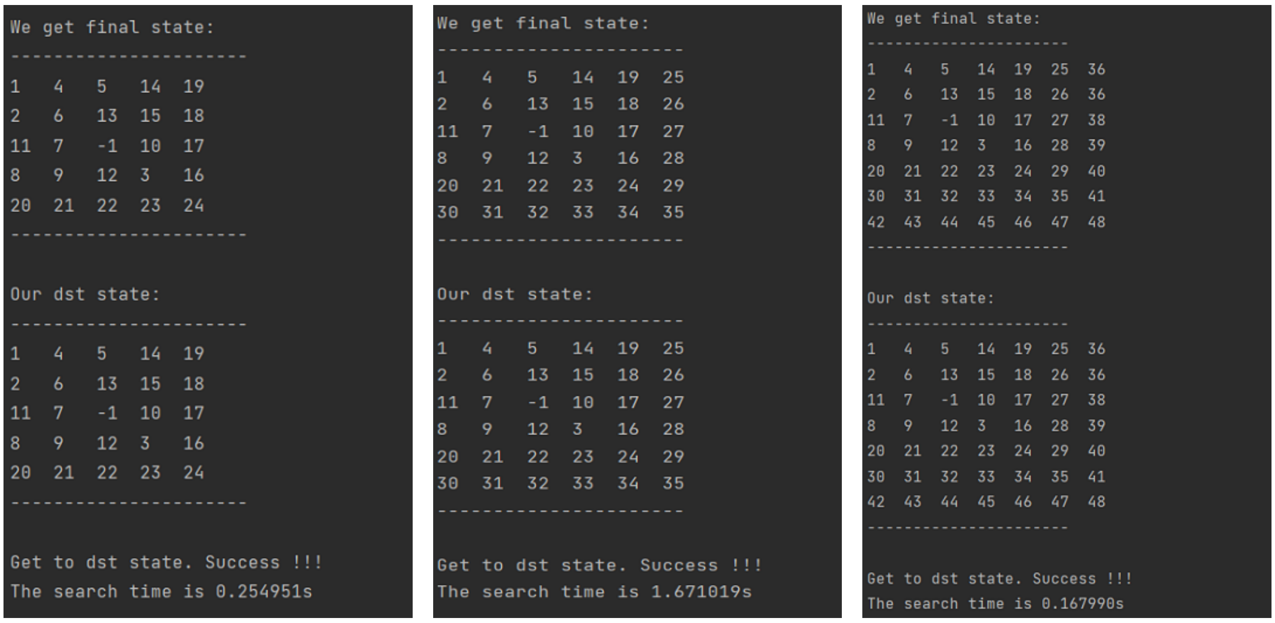
```
30 th move. Goes up.
---------------------
1   4   5   14
2   6   13  15
11  7   -1  10
8   9   12  3
---------------------

We get final state:
---------------------
1   4   5   14
2   6   13  15
11  7   -1  10
8   9   12  3
---------------------

Our dst state:
---------------------
1   4   5   14
2   6   13  15
11  7   -1  10
8   9   12  3
---------------------

Get to dst state. Success !!!
The search time is 73.362270s
```

(d) Conflict 170 moves

Fig. 3: Different Heuristics Preformance

1.3.2 Set Different Square Size

Generate 50 moves, set square size 5, 6, 7, we all can solve it.



```
We get final state:
---------------------
1   4   5   14  19
2   6   13  15  18
11  7   -1  10  17
8   9   12  3   16
20  21  22  23  24
---------------------

Our dst state:
---------------------
1   4   5   14  19
2   6   13  15  18
11  7   -1  10  17
8   9   12  3   16
20  21  22  23  24
---------------------

Get to dst state. Success !!!
The search time is 0.254951s
```

(a) Square Size 5

```
We get final state:
---------------------
1   4   5   14  19  25
2   6   13  15  18  26
11  7   -1  10  17  27
8   9   12  3   16  28
20  21  22  23  24  29
30  31  32  33  34  35
---------------------

Our dst state:
---------------------
1   4   5   14  19  25
2   6   13  15  18  26
11  7   -1  10  17  27
8   9   12  3   16  28
20  21  22  23  24  29
30  31  32  33  34  35
---------------------

Get to dst state. Success !!!
The search time is 1.671019s
```

(b) Square Size 6

```
We get final state:
--------------------
1   4   5   14  19  25  36
2   6   13  15  18  26  36
11  7   -1  10  17  27  38
8   9   12  3   16  28  39
20  21  22  23  24  29  40
30  31  32  33  34  35  41
42  43  44  45  46  47  48
--------------------

Our dst state:
--------------------
1   4   5   14  19  25  36
2   6   13  15  18  26  36
11  7   -1  10  17  27  38
8   9   12  3   16  28  39
20  21  22  23  24  29  40
30  31  32  33  34  35  41
42  43  44  45  46  47  48
--------------------

Get to dst state. Success !!!
The search time is 0.167990s
```

(c) Square Size 7

Fig. 4: Different Square Size

2 Tic Tac Toe

2.1 Algorithm Description

We use Minimax algorithm to obtain every move of computer players. In this process, we first give a current state and traverse all possible actions, then select the action with the largest Utility. Set the maximum depth of the search being 3. After 3 steps, return the minimax value. Then, select the maximum utility value as the choice of the computer player.

2.2 Key Function Design

2.2.1 Utility Design

We define X_n as the number of rows, columns, or diagonals with exactly nX and no O.Similarly, O_n is the number of rows, columns, or diagonals with nO. The utility function assigns +15 to any position with X3 = 1 and −10 to any position with $O_3 = 1$. All other terminal positions have utility 0. For nonterminal positions, we use a linear evaluation function defined as $Eval(s) = 3X_2(s) + X_1(s)−(3O_2(s) + O_1(s))$.

In program, First, count the number of O and X on each row, column, and diagonal to obtain the corresponding data array, such as o_counts_row, the three elements of the row array represent the number of first, second, and third row of O. Then determine if there is a row, column or diagonal where the number of O or X is 3. If so, return the Utility. Otherwise traverse each row, column, and diagonal, and count O_2 when the number of O is 2 and the number of X is 0. Similarly, count X_2, O_1, and X_1; Then, use the formula to calculate the Utility value and return.

2.2.2 Max And Min Value

There are tow funtions called min_value(current_state, depth) and max_value(current_state, depth), they return the current state max value or min value which calculated in depth.

In min value function, first define the variable $value = 100$, then judge if game is over which means $Utility = 15$ or $Utility = -15$, if so just return the utility. Then judge whether depth is 0, if so return utility of current state. Otherwise, traver all actions calulate value=min(value, max_value(result_state, depth-1)), then return value. That means we use a recursion and for each recursion, depth decrease 1. When depth is 0, means we get depth 3 of Utiltiy.

Max value function is same as min value function, we just change $value = -100$, and calulate value=max(value, min_value(result_state, depth-1)).

2.2.3 Get Available Actions

In this function, we travers all elements of state array and judge if it is not 0, if so we return this element's row and column, then append it to the list called $actions$.

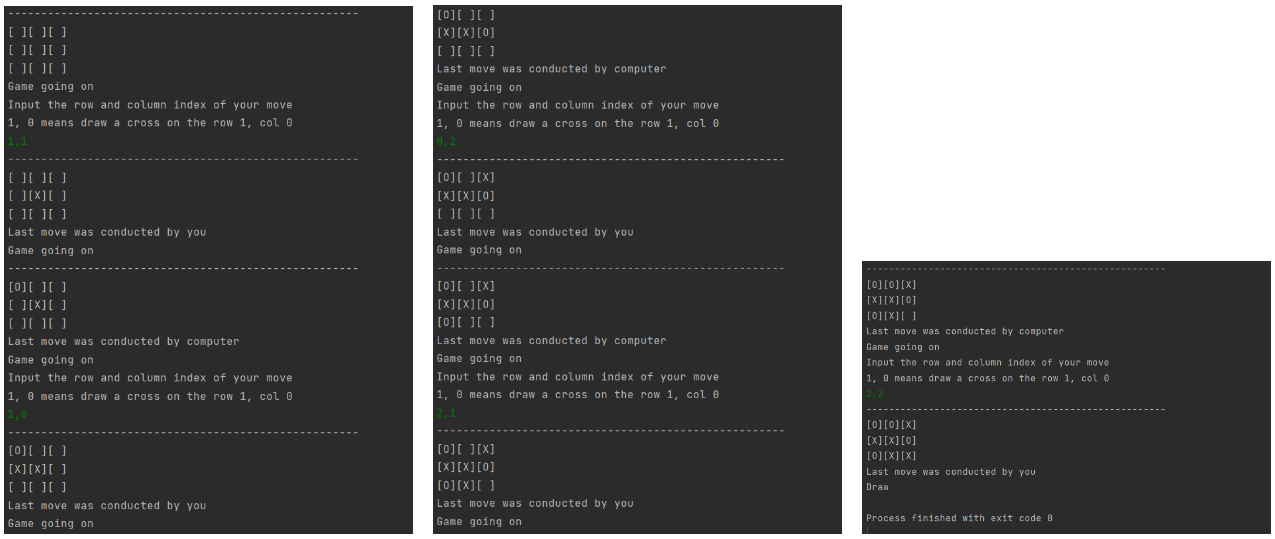
2.2.4 Action rResult

This function has three variables current_state, action, player, so we just change the action position number to player number, then return this state array.

2.3 Test Result

After testing, we can reach a draw every time.

Sample1:



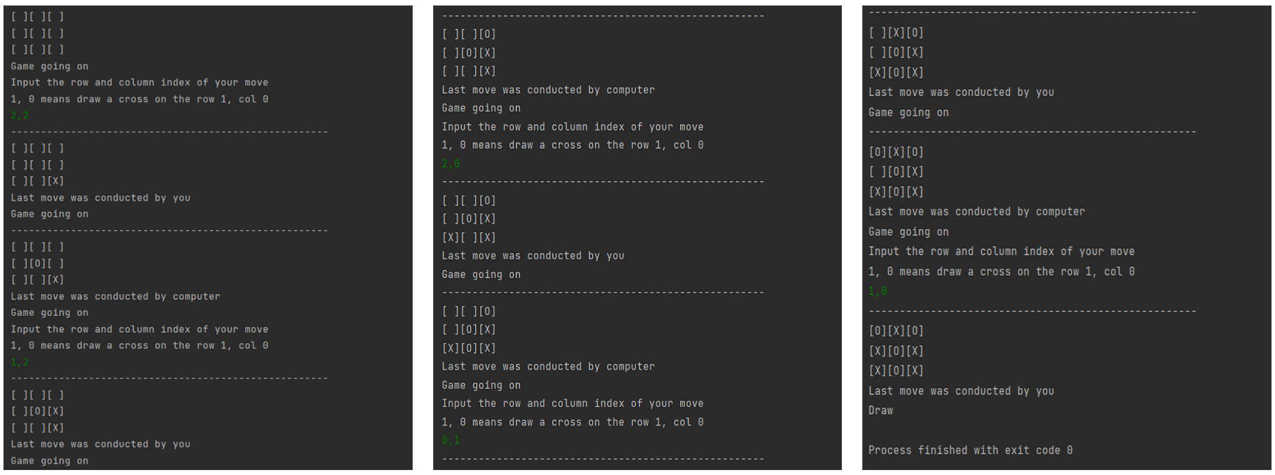
```
-----------------------------------------------------
[ ][ ][ ]
[ ][ ][ ]
[ ][ ][ ]
Game going on
Input the row and column index of your move
1, 0 means draw a cross on the row 1, col 0
1,1
-----------------------------------------------------
[ ][ ][ ]
[ ][X][ ]
[ ][ ][ ]
Last move was conducted by you
Game going on
-----------------------------------------------------
[O][ ][ ]
[ ][X][ ]
[ ][ ][ ]
Last move was conducted by computer
Game going on
Input the row and column index of your move
1, 0 means draw a cross on the row 1, col 0
1,0
-----------------------------------------------------
[O][ ][ ]
[X][X][ ]
[ ][ ][ ]
Last move was conducted by you
Game going on
```

```
[O][ ][ ]
[X][X][O]
[ ][ ][ ]
Last move was conducted by computer
Game going on
Input the row and column index of your move
1, 0 means draw a cross on the row 1, col 0
0,2
-----------------------------------------------------
[O][ ][X]
[X][X][O]
[ ][ ][ ]
Last move was conducted by you
Game going on
-----------------------------------------------------
[O][ ][X]
[X][X][O]
[O][ ][ ]
Last move was conducted by computer
Game going on
Input the row and column index of your move
1, 0 means draw a cross on the row 1, col 0
2,1
-----------------------------------------------------
[O][ ][X]
[X][X][O]
[O][X][ ]
Last move was conducted by you
Game going on
```

```
-----------------------------------------------------
[O][O][X]
[X][X][O]
[O][X][ ]
Last move was conducted by computer
Game going on
Input the row and column index of your move
1, 0 means draw a cross on the row 1, col 0
2,2
-----------------------------------------------------
[O][O][X]
[X][X][O]
[O][X][X]
Last move was conducted by you
Draw

Process finished with exit code 0
```

Fig. 5: sample1

Sample2:



```
[ ][ ][ ]
[ ][ ][ ]
[ ][ ][ ]
Game going on
Input the row and column index of your move
1, 0 means draw a cross on the row 1, col 0
2,2
-----------------------------------------------------
[ ][ ][ ]
[ ][ ][ ]
[ ][ ][X]
Last move was conducted by you
Game going on
-----------------------------------------------------
[ ][ ][ ]
[ ][O][ ]
[ ][ ][X]
Last move was conducted by computer
Game going on
Input the row and column index of your move
1, 0 means draw a cross on the row 1, col 0
1,2
-----------------------------------------------------
[ ][ ][ ]
[ ][O][X]
[ ][ ][X]
Last move was conducted by you
Game going on
```

```
-----------------------------------------------------
[ ][ ][O]
[ ][O][X]
[ ][ ][X]
Last move was conducted by computer
Game going on
Input the row and column index of your move
1, 0 means draw a cross on the row 1, col 0
2,0
-----------------------------------------------------
[ ][ ][O]
[ ][O][X]
[X][ ][X]
Last move was conducted by you
Game going on
-----------------------------------------------------
[ ][ ][O]
[ ][O][X]
[X][O][X]
Last move was conducted by computer
Game going on
Input the row and column index of your move
1, 0 means draw a cross on the row 1, col 0
0,1
-----------------------------------------------------
```

```
-----------------------------------------------------
[ ][X][O]
[ ][O][X]
[X][O][X]
Last move was conducted by you
Game going on
-----------------------------------------------------
[O][X][O]
[ ][O][X]
[X][O][X]
Last move was conducted by computer
Game going on
Input the row and column index of your move
1, 0 means draw a cross on the row 1, col 0
1,0
-----------------------------------------------------
[O][X][O]
[X][O][X]
[X][O][X]
Last move was conducted by you
Draw

Process finished with exit code 0
```

Fig. 6: sample1