Fundamental of AI Coursework

15-Puzzle-Problem

Lei Zhang

ID: 27678385

Email：lz3g15@soton.ac.uk

***Abstract***

This paper provides four methods for solving 15 puzzle problem, I programming using Matlab.

The aim is to compare the differences between those four searching methods. The four searching methods I applied are DFS (deep first search), BFS (breadth first search), IDS (iterative depth search) and greedy search. The paper will gives the performances of all four methods in solving the problems.

***Introduction***

Brief introduction about the game, the game is an Non-competitive game, it played on a 4\*4 board, as shown below, there is 3 characters and 1 block with a smile face inside, the face is a blank block, it can move to neighbors blocks and switch the position with it, and the player keep moving the face block to meet the goal state from the start state. In order to comment the performance comprehensively, I chose several different start states, with different Manhattan distance, at first I thought this should be a great and reasonable way to measure the difficulty, however it turns out sometime is not.

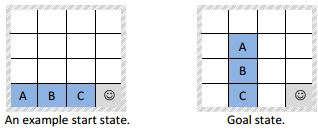


Figure-1 Game Example

***Four searching methods***

***DFS-Deep First Search***

The first search I use is Deep First Search, this method will make the face block moves randomly in the board, then compare the current state with the goal state after each moves, if the state is the goal state, stop and return, otherwise keep moving until find the right state. This method is easy to apply but it will takes much cost to find the way, like moves millions steps to meet the goal. After I finish this part code, I find a problem, the direction of the face moves is settled before it start to search, the behavior of the block is the same as the DFS, just keep moving until find the right state, however the principle is a little bit different, so I improved it in IDS search. another problem is about the differences of tree search and graph search, tree search allowed to move back to visited block, which may make the block do nothing but move between two same blocks forever, so in order to prevent this situation, I use graph search for all four searching method.

***BFS-Breadth First Search***

Breadth first search is a better method of search, and the first goal it finds will be the optimal solution. BFS will find all available states to go, and moves to one of them first, then check is it the goal state, it true return and stop, if it false, jump to the other state just opened and check it out, if all states are not the goal state, moves to next depth and do it again until find the true state. however due to my poor programming skills, i didn't forbid the move the same state, in this situation, BFS will find the same state from two different states, and both store it and expand it, this will waste times and node for searching goal state.

***IDS-Iterative Depth Search***

Iterative Depth Search is almost the DFS, but one thing different, it limits the depth, so the DFS will not expand the node over the limits; it will check the current state, if it is the goal state, stop and return, otherwise, trackback to the previous state and expands another one. IDS can make the search more efficient, and prevent the block from stuck into circulation, and most important, it meet the goal fast. However, just like DFS, IDS may not give the best way towards the goal, in another words, it's not the shortest path.

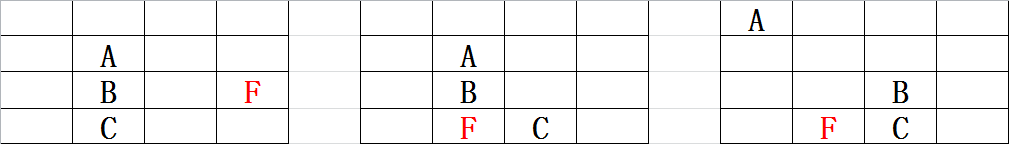
***Greedy search***

The three searching methods mentioned above all blind search, and now I applied another searching method, Greedy search. Unlike the three searching methods above, Greedy search is a kind of heuristic search. The difference between heuristic search and blind search is heuristic search pick directions. According to a function called greedy function, the face block will pick the node which is more close to the goal, and expand it. The greedy function is the Manhattan distance between the current state and goal state, if the distance is short, means this state is more probability to be right state towards the goal state.

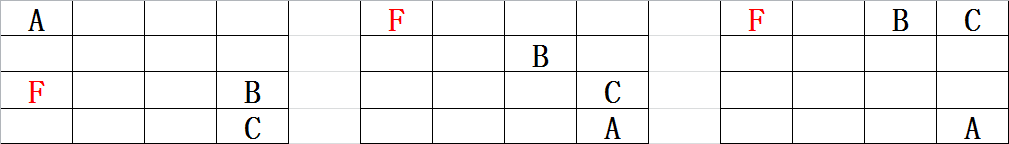
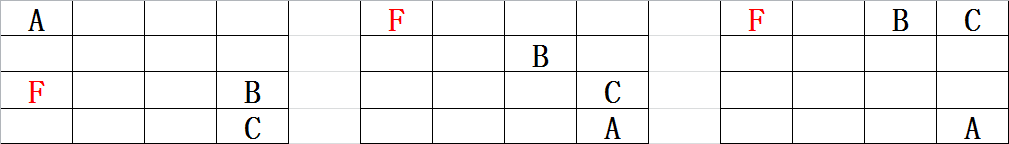
***Result and Analysis***

This section is to present the result and analysis them, I will compare the performance of these four searching methods using 6 different initial state with different difficulties. The difficulties are defined by the Manhattan distance. The performance will be represented by the number of steps and how much states they searched. The aim is to find out the difference between those four methods.

According to the Manhattan distance, I set 5 different initial states, their difficulty are 1,3,6,10 and 18. F represents the moving block. Every initial state I run 3 times to show the difference.



Difficulty-1 Difficulty-3 Difficulty-6

Difficulty-10 Difficulty-18

We can see from figure-2&3, even in low difficulty position, DFS need to search lots of state to find the right state and in the same initial state, the solution can be very different. That is because all moves in DFS are random; it’s easy to stick in circulation or moves towards different directions. So whatever the difficulty is, it always searches many states.

However I find that if the character all distributed in the corner or the distance between them is far away like difficulty 10, DFS can get the answer in thousands steps however, IDS and BFS will spend tons of times to search states, the number of state can over million. That is because they need to expand several search paths, which takes a lot of time. This really surprises me. DFS can provide a path, however in most case, it won’t be optimal, and however I couldn’t ignore the performance when it deals the high level difficulties.

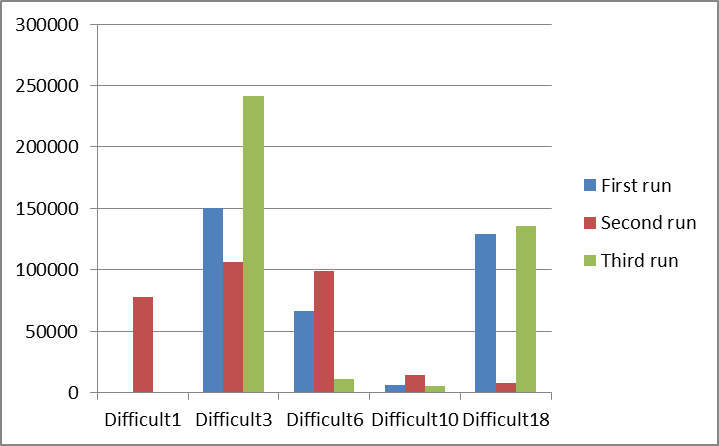
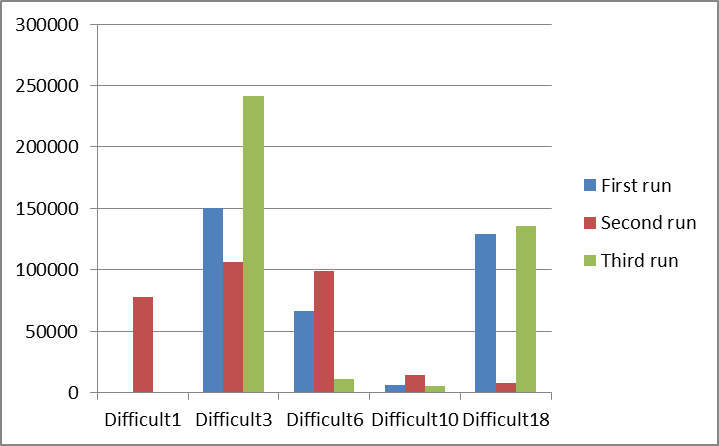


Figure-2 DFS Step Performance Figure-3 DFS Visited Nodes Performance

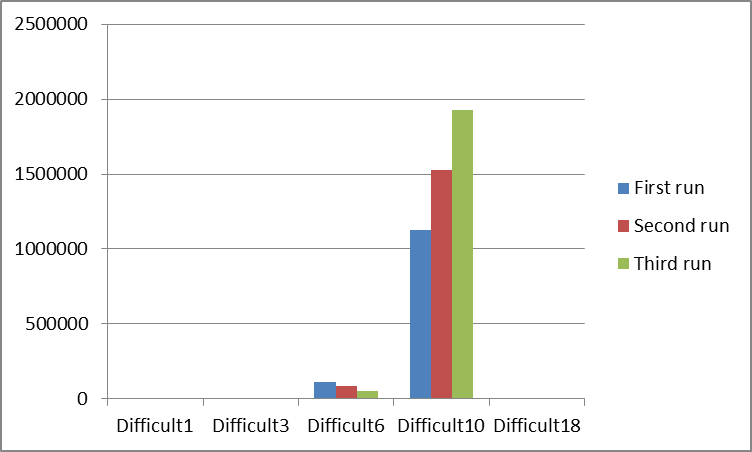
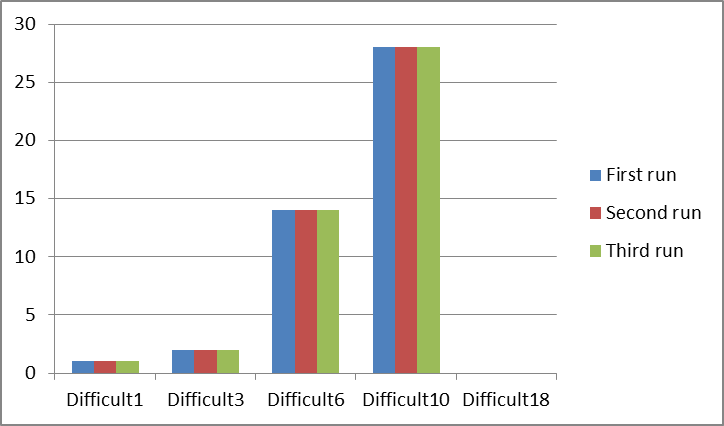


Figure-4 BFS Step Performance Figure-5 BFS Visited Nodes Performance

From Figure-4&5, I find that BFS will always get the optimal solution, however when dealing with high level puzzles, it takes really long times in searching. For the reason that it need to check all states in the same depth first, so it will hard for BFS to search the big steps puzzle. As I tested, in difficulty 10, BFS search over 1 million states but still can’t find the solution. However if the steps is small, BFS can find the optimal solution.

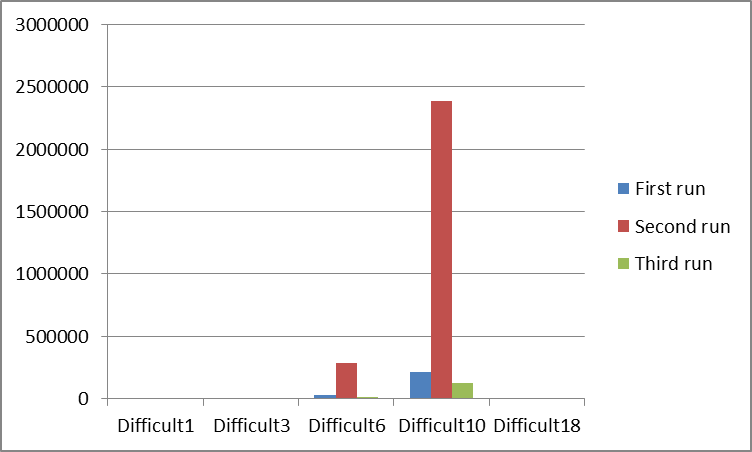
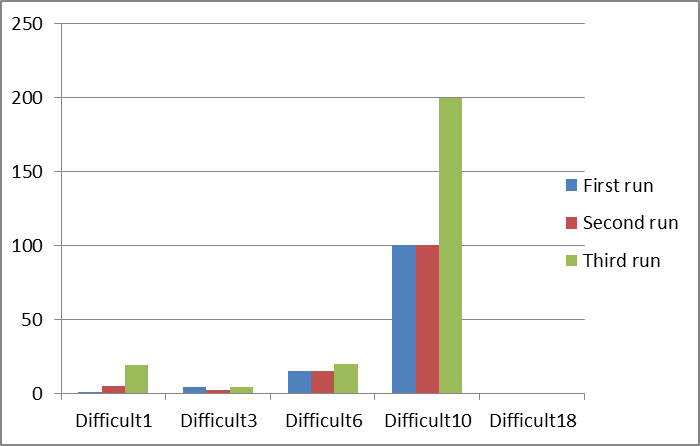


Figure-6 IDS Step Performance Figure-7 IDS Visited Nodes Performance

IDS’s performance shows in Figure-6&7, unlike other three search methods, IDS’s performance also limited by the depth-limit, if the limit is smaller than the smallest depth, it can’t get the solution, so if IDS finish all states but haven’t find the solution, it should deepen the depth, and do the search again. The searching method is a combination of BFS and DFS, for the reason that, when dealing puzzle like difficulty 10, it will search lots of states to find the solution.

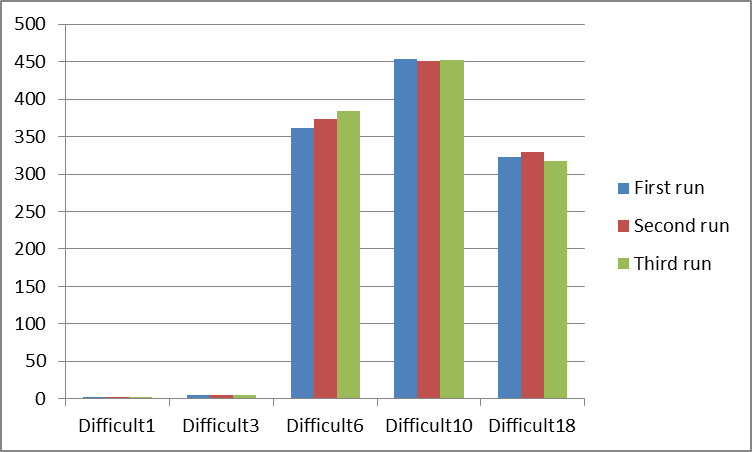
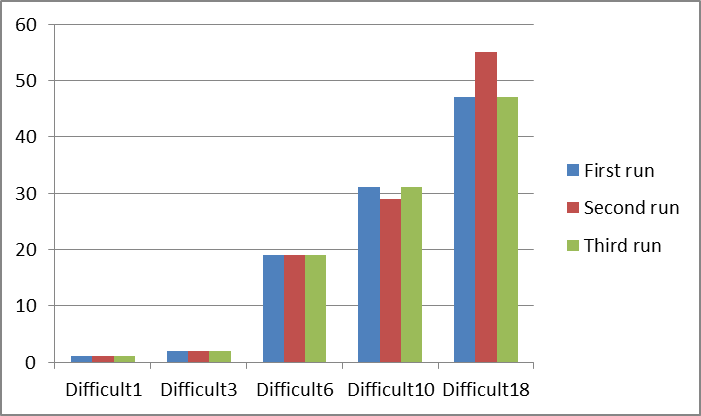


Figure-8 Greedy Step Performance Figure-9 Greedy Visited Nodes Performance

The last one is the Greedy search, from the figure we can see it searches much less than the other three methods, even when dealing with difficulty 18 and difficulty 10. The greedy function did lead the face block towards the goal. However, the solution it gets still may not be the optimal solution.

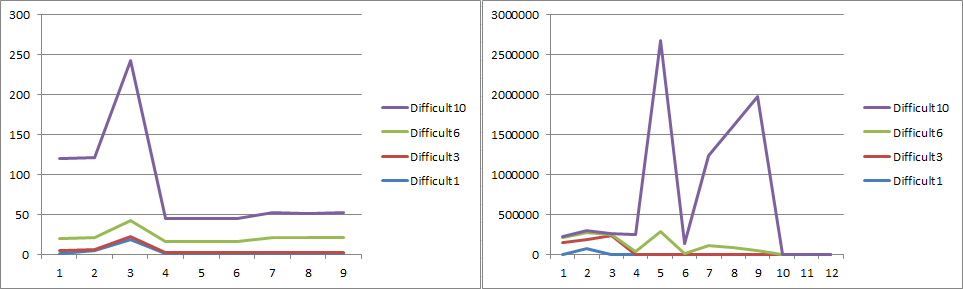


Figure-10 Overall steps comparison (without DFS) Figure-11 Overall visited nodes comparison

***Conclusion***

The four methods have their own advantages; DFS can give solutions when dealing with big-step-puzzle, but waste too much steps; BFS can find the optimal solution in small step puzzle, however it not good in doing long step puzzle; IDS is an average way to gives a solution but may not be optimal and search too much nodes; Greedy search searches far less states and give the solution, but not really optimal.

Fundamental of AI Coursework 2

Improved Method for 15-puzzle-problem

Lei Zhang

ID: 27678385

Email：lz3g15@soton.ac.uk

***Abstract***

This part I will pick task3 and introduce the Idea I have to improve the performance of searching on part 1, then gives the result and analysis.

***Introduction***

In this coursework I choose to try to improve the searching method used in coursework 1, the aim is to get the optimal solution with the least searching nodes. The general idea is to try some other Heuristic searching methods and improve it. So what I chose first is A-star method, then from A-start method, I applied a bidirectional A-star. After that, I got the idea how to use a learning method to do the searching, however I couldn’t realize it.

***Heuristic searching***

When it comes to improve the previous method, there are two ways appears in mind.

First one, improve the evaluation function. Greedy and A-star are using Manhattan distance in their evaluation function, but Manhattan distance is not accurate in this situation, the steps the face block use to move all characters is not the same as the distance between the goal and initial state. So if I want to prove the performance, I may need to calculate the real distance to goal state.

Second ideal is to change the way to use the evaluation function. The biggest problem of heuristic searching is the solution may not be optimal, so if I can apply the evaluation twice or searching several solutions then compare them and get the optimal one.

According to these two ideas I start my improvement.

***A-star search***

A-star searching generally is a improve method based on greedy search. Greedy search only calculate the distance between current state and goal state, A-star will add the cost from the initial state to current state, the reason to do so, is to make the solution more optimal. But this will cost more time to search states, I will show the result soon after.

***Bidirectional A-star search***

Bidirectional A-star is to start the search from both the initial state and the goal state (like Figure-1 shows), the aim is to make the two search tree meet in same state, and then we can get an entire route from the initial state to the goal state.

The general flow is, I expand a node, then chose the shortest node towards the current goal, then this node will became the goal of the bidirectional search and so on, when compare the distance, not only compare the current depth but all unexpanded states, and the distance should be recomputed towards the current goal. About re-calculate the distance, I found sometimes, it much possible to reach the optimal solution without re-calculate, this maybe because the wrong pick avoid the local maximum solution, but I can’t prove it, so it’s just my opinion.

The key point in this method is how to set the goals for both direction searches. When picking the state from depth 1, the Manhattan distance is compute between current states to the last goal state. Then the goal for the bidirectional search is the state chosen from depth 1, which just been chose to be expand, and the next goal for state in depth 2 is the new goal just been expand. And finally they will meet some where in around the middle.

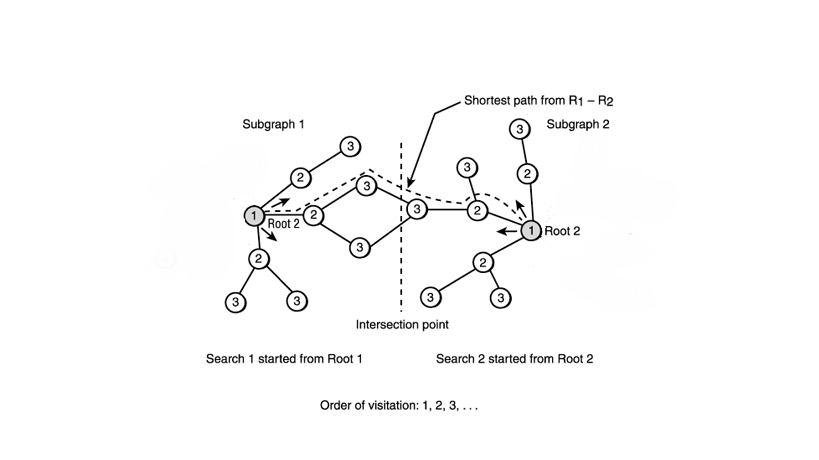


Figure-1 Bidirectional Search Structure

***Bidirectional A star with local search***

This method is designed to find the optimal solution. Local search is an improvement when one solution has been found, it start the search again from another state, this state can be the neighbor state to the current solution or just pick a random state, trying to find another solution, and compare these two solutions, if the new solution is better, then start the search from this state, until it can’t find any better solution, that means, I have got the best one.

So to my situation, I need to make the search keep going, and search for another several times, I believe this could be a solution to find an optimal solution, however it means I need to search much more nodes than current state, which is not I prefer, so finally this method been abandon.

The last method I come to thought is using the learning method to search the puzzle. The principle is to using the Neural Networks to learn the true distance between the current states to the goal state, and then the face block can pick the shortest path to the goal.

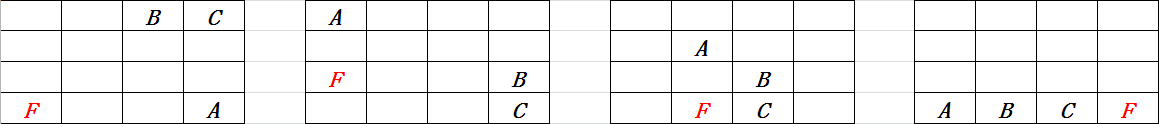
The first problem is how to treat the neural network. This requires lots of data, I need to know what the true distance is, how much steps it needs to take from this state to the goal. One way is to using the BFS to do once, and then I can know the true steps. So a 4\*4 board have 43680 potential combinations of states. Using BFS to compute the true steps from each state to goal state, this will became the training data of the neural network, then apply the trained neural network to the evaluation function, and done, the function will lead to the shortest path.

***Result and Analysis***

In this section, I will show the result of A-star and Bidirectional A-star in solving the 15-puzzle-problem and analysis the performance of these two searches. Greedy search will set as the contrast.

The performance is represented by the steps, how much steps it need to take towards the goal, and visited states, how much states it need to search. The best situation is to search the minimal states and get the shortest path.

I prepared 4 initial states this time (shown in figure-2), including the example state. Given the Manhattan distance can’t measure the difficulty well, so this time the difficulty is measured by how much steps it need to take from the start to the end, the steps is measured by A-star (running several times with the same initial and get the shortest path, I want to BFS, but my BFS take incredible long time when dealing long steps problem).here I don’t compare the time, the time also limited by the code structure, our aim Is just compare the searching methods.



True step-31 True step-12 True step-24 True step-16

Figure-2 Difficulty Chart

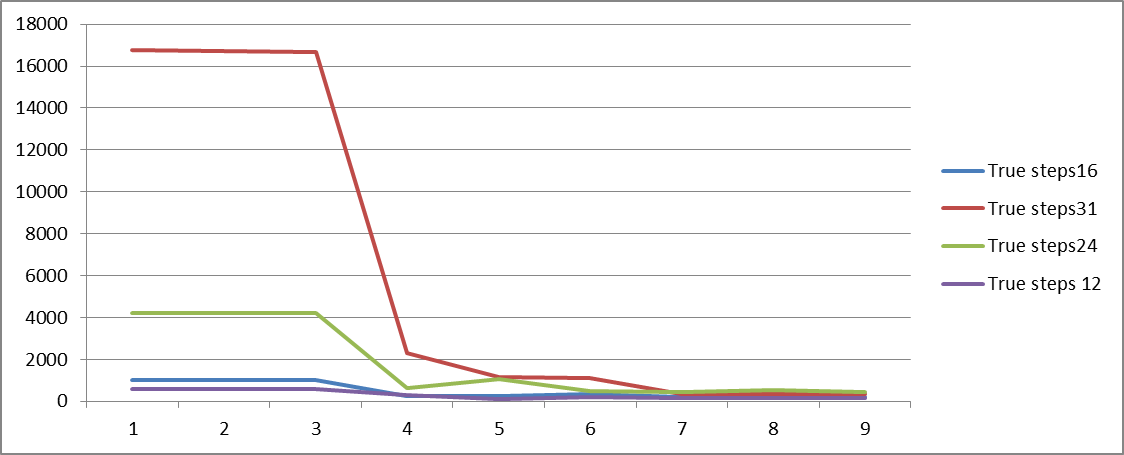


Figure-3 Overall Visited Nodes

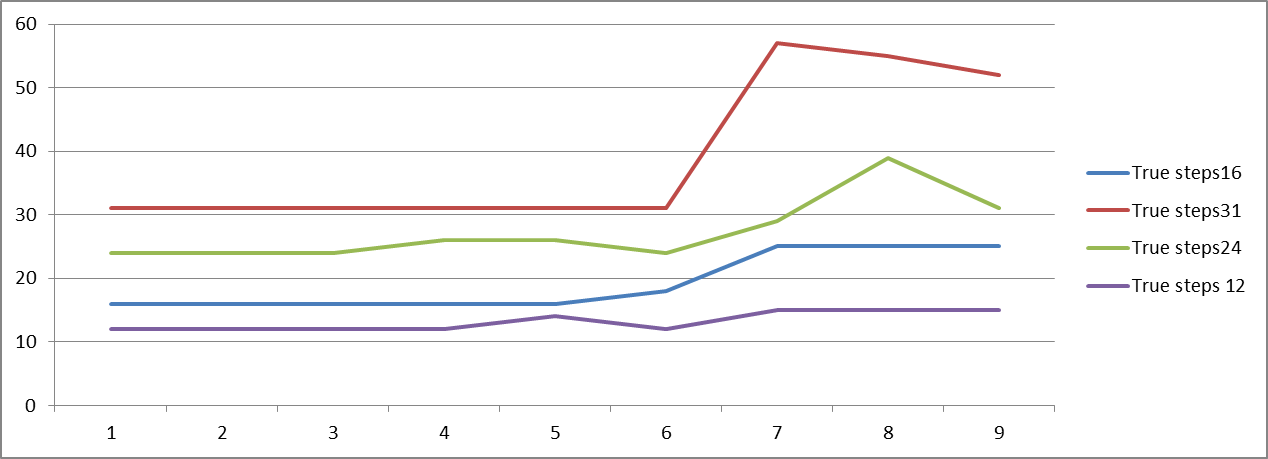


Figure-4 Overall Steps

Figure-3 shows the number of visited nodes each method from different initial state. Figure-4 shows the steps each method gives.

1-3 represents the three run of A-star, 4-6 represents the three run of Bidirectional A-star and the last three is Greedy method.

From Figure-3 we can see, A-star tends to search more states than the other two, especially in long steps search, like 31 steps. However A-star can provide a stable and optimal solution, which can be observed from 1-3 in Figure-5.

Bidirectional A-star searches much less states than A-star, however it not always gives the optimal solution, from Figure-4 4-6 we can see the steps Bidirectional A-star provides is unstable, but the differences in steps is not big compared with A-star, the value is around 1-2 steps.

Greedy method always searches least nodes, but provides more steps than the other two, which is unacceptable.

***Conclusion***

Bidirectional A-star searches less than A-star, and provides a similar solution, compared with A-star, so I believe Bidirectional A-star is actually better than A-star. But we still can’t ignore the disadvantage of it; it can’t always provide an optimal solution. However if I try several times like I do in the test, I can still find an optimal solution. Last about the Neural network, I had a consider about that, but since I don’t have so much time left, so I can’t show the result about that.