Analysis of Iterative Deepening A* Algorithm

Wenrong Jiang1*

1School of Computer and Information, Shanghai Polytechnic University, Shanghai, China
*Corresponding author email: wrjiang@sspu.edu.cn

Abstract. The A* algorithm is a direct search method. It can effectively solve the shortest path problem in static routing networks. It is also an effective method to deal with a large number of search problems. The closer the artificially estimated distance value in the algorithm is to the actual value, the faster the final search speed will be. This article focuses on the heuristic A* algorithm and discusses the problem of out-of-order Rubik's Cube searching the shortest feasible solution. It discusses the implementation ideas, advantages and disadvantages, and practical application value of the A*(IDA*) algorithm based on iterative deepening. The development of heuristic algorithms presents some personal insights.

1. Introduction
In the modern game industry, graph search technology has become a basic core design. Regardless of the type of game, the search technology of the road map plays a decisive role. Today's popular RPG, FPS, and other types of games rely heavily on path-finding methods to enable AI gamer to freely control the movement of characters and things to achieve defense or offensive tactics during tactical deployment. Similarly, it can also be applied to games based on maze or map traversal as the main carrier of 2, 2.5, and 3D games. In addition, in the field of artificial intelligence and other technologies, machines with path-finding features, such as unmanned driving and obstacle avoidance, have achieved good results to a certain extent.

Path-class problems can usually be mapped onto a tree structure diagram. The starting point is the root node of the tree, and then the remaining sub-nodes are extended, corresponding to the next step in the search. With a little expansion of the nodes, the choice of paths will become increasingly larger. We use such a problem to find a feasible solution with the shortest search path through appropriate algorithms.

2. Domestic and Foreign Research

2.1. Heuristic Algorithm
The heuristic algorithm is not only an intuitive algorithm, but also an algorithm based on experience. Under a guaranteed computational cost (time and space), a feasible solution is provided for each instance of the optimization problem that needs to be solved. Different from the traditional optimization method, it does not consider the degree of difference between the solution (different valuation functions produce different feasible solutions) and the optimal solution. It has the advantages of short time, less space occupation and easy implementation.

2.2. A* Algorithm and IDA* Algorithm
A* algorithm: In the BFS algorithm, if you set the valuation function f(n)=g(n)+h(n) for each condition n, and select each node from the Open list, When expanding, nodes with the lowest f-value
are selected. Among them, \( g(n) \) refers to the actual cost from the initial state to the current state \( n \), and \( h(n) \) refers to the estimated cost from the current state \( n \) to the target state.

\[ f(n) = g(n) + h(n), \]

satisfies the following limitations:

- \( g(n) \) is the actual number of steps from \( s_0 \) to \( n \) (not necessarily optimal), so:
  - \( g(n) > 0 \) and \( g(n) \geq g^*(n) \), \( h(n) \) is the predicted number of steps from \( n \) to the target. The estimation is always too optimistic, that is, \( h(n) \leq h^*(n) \), \( h(n) \) is compatible, then the A algorithm is transformed into the A* algorithm.

\[ h(n) \] compatibility:

If the \( h \) function also satisfies any state \( s_1 \) and \( s_2 \):

\[ h(s_1) \leq h(s_2) + c(s_1, s_2) \]

- \( c(s_1, s_2) \) is the number of steps that \( s_1 \) transfers to \( s_2 \), then \( h \) is said to be compatible.

The \( h \)-compliance ensures that as a step forward, \( f \) increases, so that A* can find the optimal solution more efficiently.

\[ h \text{ compatible} \]

\[ \Rightarrow g(s_1) + h(s_1) \]

\[ \Rightarrow g(s_1) + h(s_2) + c(s_1, s_2) = g(s_2) + h(S_2) \]

\[ \Rightarrow f(s_1) \leq f(s_2) \]

That is, \( f \) is incremented.

**Figure 1.** Path Sorting of A* Algorithm

IDA* is an iterative deepened A* algorithm, which can effectively solve the space growth problem caused by the A* algorithm. Set the maximum depth to reach each time, deepen the maximum depth if it does not reach the target state. Using the evaluation function, cut the path where \( f(n) \) is greater than depth.

### 2.3. Dijkstra Algorithm

Based on the greedy strategy and BFS to solve the shortest path problem, the Dijkstra's algorithm mainly extends from the starting point to the outer layer until it reaches the end point.

**Figure 2.** A example of Dijkstra algorithm
Table 1. Dijkstra algorithm solves the example problem process

| Source | Destination | Optimal Path | Measurement value |
|--------|-------------|--------------|-------------------|
| V1     | V2          | /            | $\infty$         |
| /      | V3          | \{V1, V3\}   | 10                |
| /      | V4          | \{V1, V5, V4\} | 50               |
| /      | V5          | \{V1, V5\}   | 30                |
| /      | V6          | \{V1, V5, V4, V6\} | 60           |

2.4. Comparison of A* Algorithm and IDA* Algorithm

Table 2. Comparison of A* Algorithm and IDA* Algorithm

| Advantage | A* Algorithm | IDA* Algorithm |
|-----------|--------------|----------------|
|           | Compared with the breadth-first search strategy and the depth-first search strategy, the A* algorithm is not a blind search but a prompted search. | Using backtracking methods saves space by not saving intermediate states. |
| Disadvantage | The algorithm generally uses a lot of space for storing the searched intermediate states and prevents duplicate searches. | Repeated search: Every time depth is increased in the trace-back process, it must be searched again. |
| Application | Starting from the initial state => After a series of intermediate states => Eventually reach the target state | Starting from the initial state => After a series of intermediate states => Eventually reach the target state |

3. Discussion of the Problem

3.1. Topic Background
The rotation game uses a # shaped board, which can hold 24 pieces of square blocks (see Fig.1). The blocks are marked with symbols 1, 2 and 3, with exactly 8 pieces of each kind. Initially, the blocks are placed on the board randomly. Your task is to move the blocks so that the eight blocks placed in the center square have the same symbol marked. There is only one type of valid move, which is to rotate one of the four lines, each consisting of seven blocks. That is, six blocks in the line are moved towards the head by one block and the head block is moved to the end of the line. The eight possible moves are marked with capital letters A to H. Figure 3 illustrates two consecutive moves, move A and move C from some initial configuration.

![Figure 3. Two consecutive moves](image)

3.2. The Input of Problem
The input consists of no more than 30 test cases. Each test case has only one line that contains 24 numbers, which are the symbols of the blocks in the initial configuration. The rows of blocks are listed from top to bottom. For each row the blocks are listed from left to right. The numbers are separated by
spaces. For example, the first test case in the sample input corresponds to the initial configuration. There are no blank lines between cases. There is a line containing a single '0' after the last test case that ends the input.

```
1 1 1 3 2 3 2 3 1 3 2 2 3 1 2 3 1 2 1 3 3
1 1 1 1 1 1 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3
0
```

3.3. The Output of Problem

For each test case, you must output two lines. The first line contains all the moves needed to reach the final configuration. Each move is a letter, ranging from 'A' to 'H', and there should not be any spaces between the letters in the line. If no moves are needed, output 'No moves needed' instead. In the second line, you must output the symbol of the blocks in the center square after these moves. If there are several possible solutions, you must output the one that uses the least number of moves. If there is still more than one possible solution, you must output the solution that is smallest in dictionary order for the letters of the moves. There is no need to output blank lines between cases.

```
AC
2
DDHH
2
```

3.4. Topic Analysis and Algorithm Ideas

We already know the starting state. We only need to search and save the state information of each step to ensure that we find the optimal solution. According to the topic requirements, the total number of grids is 24, then the corresponding number of states is 6. The search algorithm will inevitably lead to timeout issues. So we can not but introduce a new search method which is IDA* algorithm.

In the IDA* algorithm, the main function generally uses an incrementally larger search depth to obtain an optimal solution, and if a dfs search strategy is used, the entire search space needs to be searched, and the optimality of all legal solutions is determined in the search function. Solution (by comparing the depth of the legal solution, choose the smallest depth, in fact, in the bfs search, the depth corresponds to the optimal solution required in the problem).

In IDA*'s search core algorithm, the first half is pruning and returns false. This is the same as dfs. In the regular search, when the optimal solution is found, it returns true, and when the dfs in the following if statement recursively is true, it returns true, so exit the entire loop, otherwise it will die. To increase efficiency, you can copy the current state to a temporary state, and then use the temporary state to perform a recursive search. This eliminates the need to restore the entire state when the recursive function returns.

4. Related Data Analysis

4.1. Run Result

```
1 1 1 3 2 3 2 3 1 3 2 2 3 1 2 2 3 1 2 1 3 3
1 1 1 1 1 1 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3
0
```

Figure 4. Run result
4.2. **IDA* Algorithm Complexity Analysis**

| Algorithm name | IDA* |
|----------------|------|
| Time complexity | $\theta(O(1 \sim 2N))$ |
| Occupied memory | $\theta(O(N))$ |
| Instruction manual | \// |
| Use scene | Way-finding problem |

The space complexity of the A* algorithm is still quite high. When it is actually used, each operation of the priority queue is a single operation (n represents the total number of nodes stored in the priority queue). Therefore, when the search is performed later, it is preferred for maintenance. In the queue, the expenses are relatively large, and the time complexity will gradually increase.

IDA* is formally based on DFID. The idea is heuristically searched. It is a very clever use of the deepening feature of DFID iterative depth. It uses depth deepening operations instead of depth cost function. The maximum search depth is the heuristic value of the initial state, and then deepens the maximum search depth in turn, so that heuristic search in the form of DFID is implemented. The space complexity of IDA* algorithm is quite low, and it is not necessary to construct a hash table. At the same time, it is also relatively simple to implement the programming. In addition, the promotion problem of the 8 puzzle, such as the 15 puzzle, also has an efficient solution (this is impossible for the A* algorithm). Because the 15 puzzle is difficult to construct a hash table in reality, because at least a space of 15! is needed. The only drawback is that there is a certain degree of repeated search, but since the search size increases exponentially during the search, IDA* repeats The state quantity of the search is the quantity of the search state in the deepening process of the last iteration, so the state quantity of the repeated search compared with the current search quantity is still relatively small.

5. **Key Issues and Solutions**

In IDA*'s search core algorithm, the first half is pruning and returns false. This is the same as dfs. In the conventional search, when the optimal solution is found, it returns true. In the IDA* algorithm, the main function generally uses a step-up search depth to obtain an optimal solution, and if the dfs search strategy is used, it needs to be applied to the entire system. The search space is searched, and the best solution in all legal solutions is determined in the search function (by comparing the depth of the legal solution, the minimum depth is selected. In fact, in the bfs search, the depth corresponds to the optimal solution required in the title. Amount), and when dfs in the following if statement is recursively true, return true, so exit the entire loop, otherwise it will die. To increase efficiency, you can copy the current state to a temporary state, and then use the temporary state to perform a recursive search. This eliminates the need to restore the entire state when the recursive function returns.

For the selected sample questions, the biggest problem to be solved is running time considerations. A lot of speed data makes the program run out during the commit code phase. We need to further optimize the algorithm.

The specific part of the code is as follows:

```c
if(depth-now<8-count(p))         //Valuation function pruning operation
    return 0;
```

Compare the current state with the estimated state, if it is less than the number of the maximum value in the central area, you can skip this dfs, directly return to the function, enter the next decision. Here is the program code and explanation.

//IDA*algorithm*
#include<stdio>
#include<string>
#include<algorithm>
#include<iostream>
using namespace std;
int a[25],depth;
bool ok(int * p) // Judging whether it has reached the target state
{
    int k=p[7];
    return (k==p[8])&&(k==p[9])&&(k==p[12])&&(k==p[13])&&(k==p[16])&&(k==p[17])&&(k==p[18]);
}
int c[5],ans;
int va[]={7,8,9,12,13,16,17,18};
int pos[]={1,2,3,4,6,5,8,7};
char out[100];
int count(int *p) //Statistics current central area contains the same number of maximum
{
    c[1]=c[2]=c[3]=0;
    for(int i=0;i<8;i++)
        c[p[va[i]]]++;
    return max(c[1],max(c[2],c[3]));
}
void rotate(int * p,int a,int b,int c,int d,int e,int f,int g)
{
    int tp=p[a];
    p[a]=p[b],p[b]=p[c],p[c]=p[d],p[d]=p[e],p[e]=p[f],p[f]=p[g],p[g]=tp;
}
int dfs(int * p,int now,int pre)
{
    if(depth-now<8-count(p)) //Valuation function pruning operation
        return 0;
    if(now>=depth) return 0;
    int tmp[25];
    for(int w=0;w<8;w++)
    {
        int i=pos[w];
        for(int i=1;i<=24;i++)
            tmp[i]=p[i];
        if(pre-i==4||i-pre==4)continue;
        switch(i)
        {
            case 1:rotate(tmp,1,3,7,12,16,21,23);out[now]=\'A\';break;
            case 2:rotate(tmp,2,4,9,13,18,22,24);out[now]=\'B\';break;
            case 3:rotate(tmp,11,10,9,8,7,6,5);out[now]=\'C\';break;
            case 4:rotate(tmp,20,19,18,17,16,15,14);out[now]=\'D\';break;
            case 5:rotate(tmp,23,21,16,12,7,3,1);out[now]=\'F\';break;
            case 6:rotate(tmp,24,22,18,13,9,4,2);out[now]=\'E\';break;
            case 7:rotate(tmp,5,6,7,8,9,10,11);out[now]=\'H\';break;
            case 8:rotate(tmp,14,15,16,17,18,19,20);out[now]=\'G\';break;
        }
        if(ok(tmp))
        {
            out[now+1]=\'0\';
            ans=tmp[7];
            return 1;
        }
    }
}
if(dfs(tmp,now+1,i))
    return 1;
}
return 0;

int main()
{
    while(scanf("%d",a+1)&&a[1])
    {
        for(int i=2;i<=24;i++)
            scanf("%d",a+i);
        if(ok(a))
        {
            printf("No moves needed\n");
            printf("%d\n",a[7]);
        }
        else
        {
            depth=1;
            while(1)
            {
                if(dfs(a,0,-100))
                    break;
                depth++;
            }
            printf("%s\n",out);
            printf("%d\n",ans);
        }
    }
    return 0;
}

6. Summary
In summary, there are many applications in our lives for the heuristic algorithm using the A* algorithm. In the process of solving the POJ2286, it is not difficult for us to see that the time complexity of IDA* is not very high. In the space storage, the memory occupied is not large. Learning and mastering the A* algorithm plays a decisive role in solving the problem of the shortest path. Similarly, the path finding problem in artificial intelligence can also find a new breakthrough through the A* algorithm. The emergence of the IDA* algorithm not only symbolizes the development of the computer field, but also reflects the crystallization of human wisdom. Looking ahead, the A* algorithm will give us more surprises and apply it to more fields and situations, such as unmanned driving, route finding and other issues.

7. Acknowledgments
This work is supported by the Key Disciplines of Computer Science and Technology of Shanghai Polytechnic University under Grant No. XKKZD1604.

8. References
[1] Xie Peiyang. Detailed Explanation of A-Star Algorithm [EB/OL]. https://zhuanlan.zhihu.com/p/20860404.
[2] Wenrong Jiang. Research on Existential Problems Based on Drawer Principle[J]. 2019 International Conference on Mathematics, Big Data Analysis and Simulation and Modeling (MBDASM2019), Changsha, China, August 30-31, 2019. 4-7.
[3] Wenrong Jiang. Analysis and Discussion of Radar Construction Problems with Greedy Algorithm[J]. 3rd International Conference on Computer, Communication and Computational Sciences (IC4S2018), Bangkok, Thailand, Springer. October 20, 2018. 303-312.

[4] Wenrong Jiang, Jian Chen, Wei Tao. The Development of Automotive Interior Sales Website[J]. Communications in Computer and Information Science (Volumes 268), 2011(12):342-348. EI:20122015019844

[5] Wenrong Jiang, Shijie He. The research of small and medium-sized wedding photography enterprise’s product types and consumer participation behavior[C]. Proceedings of the 2016 International Conference on Social Science, Humanities and Modern Education(SSHME 2016), Guangzhou, China. Atlantis Press. May 30, 2016, 103-107. WOS:000390400800020.