CMPT417

Group Project
Use different algorithms to solve 15-puzzle

Instructor
Hang Ma

Students:
Zhi Feng #301368731
Shengyao Wang #301356790
Wenshan Deng #301360500

1. INTRODUCTION

15-puzzle was first proposed by a postman in New York in 1874. In 1914, a man named Sam Lloyd published a book called Cyclopedia of puzzles, which raised the question of 14-15, that is, the numbers 14 and 15 in the completed 15-puzzle were replaced, and the rest remained unchanged.

Its initial state is that a total of 15 numbers from 1 to 15 and a "blank" (hereinafter referred to as "#) are arranged on the 4*4 grid, in which the numbers are arbitrarily arranged on the first 15 grids, and" # "is on the last grid. Players can choose the number in the four directions of "empty" up, down, left and right, and let this number exchange positions with "empty".

The so-called restoration refers to when the 4*4 square is 1-15 from left to right, from top to bottom, plus the last "#", then the player wins.

5	1	3	4
2		7	8
9	6	10	12
13	14	11	15

Initial State

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

Final State

2. IMPLEMENTATION

Our project is to observe the performance of five algorithms, DFS, BFS, A*, PDB Heuristic and IDA*+PDB on the 15-puzzle problem.

2.1 DEPTH FIRST SEARCH

Depth first search is a kind of graph algorithm which is implemented as stack and FILO order. Its process is to start from the initial node, expand to the next node in a predetermined order, and then continue to expand new nodes from the next node, and continue to recursively execute this process until a node can no longer expand the next node. At this point, return to the previous node to find a new extension node. This search continues until the target node is found or all nodes are searched.

The method of depth first traversal of the graph is to start from a vertex v in the graph:

a) Access vertex v.

b) Starting from the unreached adjacency points of V, the depth first traversal of the graph is carried out. Until the vertex connected with the path of V in the graph is accessed.

c) If there are still vertices in the graph that have not been accessed at this time, start from a vertex that has not been accessed, and repeat the depth first traversal until all vertices in the graph have been accessed.

```
function Depth-First-Search(problem) returns solution
  nodes := Make-Queue(Make-Node(Initial-State(problem))
loop do
  if nodes is empty then return failure
  node := Remove-Front (nodes)
  if Goal-Test[problem] applied to State(node) succeeds
    then return node
  new-nodes := Expand (node, Operarors[problem]))
  nodes := Insert-At-Front-of-Queue(new-nodes)
end
```

Figure 1[1]: Pseudocode for Depth First Search

2.2 BREADTH FIRST SEARCH

Breadth first search is one of the simplest graph search algorithms, and this algorithm is also the prototype of many important graph algorithms. Dijkstra single source shortest path algorithm and prim minimum spanning tree algorithm both adopt an idea similar to width first search. Its alias is BFS, which belongs to a blind search method. Its purpose is to systematically expand and check all nodes in the graph to find the results. In other words, it does not consider the possible location of the result, and thoroughly searches the whole graph until it finds the result.

Breadth first search is realized by using queues. The whole process can also be seen as an inverted tree. Its process is:

(1) Put the root node at the end of the queue.

(2) Take an element from the head of the queue every time, check all the next level elements of this element, and put them at the end of the queue. And record this element as the precursor of its next level element.

(3) End the program when the element you are looking for is found.

(4) If the traversal of the whole tree has not been found, end the program.

Figure 2[1]: Pseudocode for Breadth First Search

```
function Breadth-First-Search(problem) returns solution
  nodes := Make-Queue(Make-Node(Initial-State(problem))
loop do
 if nodes is empty then return failure
  node := Remove-Front (nodes)
 if Goal-Test[problem] applied to State(node) succeeds
    then return node
  new-nodes := Expand (node, Operators[problem]))
  nodes := Insert-At-End-of-Queue(new-nodes)
end
```

2.3 A-STAR SEARCH

A-star algorithm is the most effective method to solve the shortest path in static road network. The formula is expressed as: F (n) =g (n) +h (n), where f (n) is the evaluation function of node n from the initial point to the target point, G (n) is the actual cost from the initial node to the N node in the state space, and H (n) is the estimated cost of the best path from n to the target node.

In order to find the optimal solution, the key lies in the selection of the evaluation function H (n): the evaluation value H (n) < = the actual value of the distance from n to the target node. In this case, the number of search points is large, the search range is large, and the efficiency is low. But the optimal solution can be obtained. If the estimated value > the actual value, the number of search points is small, the search range is small, and the efficiency is high, but the optimal solution cannot be guaranteed. The closer the estimated value is to the actual value, the better the appraisal function will be obtained.

The main search process of A-star algorithm:

(1) Create two tables. The open table saves all nodes that have been generated but not investigated, and the closed table records the nodes that have been visited.

(2) Traverse each node of the current node, put n nodes into close, and take the child node X of N nodes.

(3) Calculate the estimated value of X.

(4) Insert n nodes into the close table.

(5) Sort the nodes in the open table according to the evaluation value.

```
Input: A graph G(V,E) with source node start and goal node end.
Output: Least cost path from start to end.
Steps:
Initialise
        open_list = { start }                       /* Listof nodes to be traversed*/
        closed_list = { }                           /*List of already traversed nodes*/
        g(start) = 0                                /*Cost from source node  to a node*/
        h(start) = heuristic_function (start,end)   /*Estimated cost from node to goal node*/
        f(start) = g(start)+h(start)                /* Total cost from source to goal node*/

while open_list is not empty
        m = Node on top of open_list, with least f
        if m = = end
                return
        remove m from open _list
        add m to closed_list
        for each n in  child(m)
                if n in closed_list
                        continue
                cost= g(m)+ distance(m,n)
                if n in open_list and cost < g (n)
                        remove n from open_list as new path is better
                if n in closed_list and cost < g(n)
                        remove n from closed_list
                if n not in open_list and n not in closed_list
                        add n to open_list
                        g(n) = cost
                        h(n) = heuristic_function (n,end)
                        f(n) = g(n)+h(n)
return failure
```

Figure 3[4]: Pseudocode for A-Star Search

2.4 PATTERN DATABASE HEURISTICS

Unfortunately, A-star cannot solve random instances of the Fifteen Puzzle, because it stores every node generated, and exhausts the available memory in minutes on most problems. Pattern databases, originally applied to the 15-Puzzle, are one way to do this.

A pattern database (PDB) is a lookup table that stores solutions to all configurations of the sub-problem (patterns).This PDB is used as a heuristic during the search.

2.5 IDA-STAR+PDB

IDA-star algorithm is an A-Star algorithm based on iterative deepening. Iteration deepening has better effect only when the state grows exponentially, and A-Star is to prevent the state from growing exponentially. IDA-star algorithm actually uses iterative deepening and global optimality pruning at the same time, and expands in a relatively good order to ensure that the best solution can be found at the earliest time. The space required is relatively small, and sometimes it runs faster than A-Star.

The main search process of A-star algorithm: First, the H value of the initial state node is set as the threshold value max-H, and then the depth first search is performed. In the search process, all nodes with H value greater than max-H are ignored; If no solution is found, increase the threshold max-H and repeat the above search until a solution is found. Under the condition that the calculation of H value satisfies the requirements of A-star algorithm, it can be proved that the solution found must be the optimal solution. Here, an appropriate evaluation function is also used in the IDA-star algorithm to evaluate the distance from the target state.

In general problems, IDA-star algorithm is used in this way. When the evaluation function value of the current situation + the current search depth > the predefined maximum search depth, pruning is performed.

```
procedure IDA_STAR(StartState)
begin
   PathLimit := H( StartState ) – 1;
   Success := False;
   repeat
      inc(PathLimit);
      StartState.g:= 0;
      Push(OpenStack,StartState);
      repeat
         CurrentState:=Pop(OpenStack);
         If Solution(CurrentState) then
            Success = True
         Else if PathLimit >= CurrentState.g + H(CurrentState) then
            Foreach Child(CurrentState) do
               Push(OpenStack, Child(CurrentState));
      until Success or empty(OpenStack);
   until Success or ResourceLimitsReached;
end;
```

Pseudocode for IDA

IDA with pdb:
Build_target_pattern_database(StartState):

1. We have built a simple pattern with initial state.
2. Run Breadth First Search algorithm to find the solution of the given target pattern.
3. Dump the solution to local file or memory.

H(CurrentState):
We compare the current state with the generated pattern database.

a) If current state in database:
   i. Return solution
b) Return None

Then we can overwrite the "H()" function and use it with IDA algorithm to boost the procedure.

3. METHODOLOGY

In this experiment, we create a 4 x 4 matrix as our initial state. We set up some applicable data as our problem instance. Our goal is using Depth First Search, Breadth First Search, A* with Manhattan Distance, A* with Pattern Database Heuristic and IDA* with Pattern Database Heuristic to solve the matrix, which is to find the path from initial state to the final state.

In order to investigating concerning the performance of the algorithms on our problem instances, we compared the time complexity and space complexity in these five algorithms.How much time does the algorithms consume? What is the memory usage for each algorithm? We output the running time and memory usage of each algorithm as it completes.

4. EXPERIMENTAL SETUP

The experiments are implemented by Python 3.8 and edited using the text editor PyCharm. We test the program on Anaconda Jupyterlab/ win11/ i7-12700H up to 4.7GHz/ RTX3080ti/ 32GB memory and AMD Ryzen 7 5800H/ 3.2 GHz/ 16G memory

5. EXPERIMENTAL RESULTS

Final performance between our kinds of algorithm with initial state:
"1,2,3,4,5,6,7,8,0,10,11,12,9,13,14,15"

DFS:

- Duration:7ms
- Memory Usage: 27.26MB

BFS:

- Duration: 47ms

- Memory Usage: 27.26MB

A-Star:

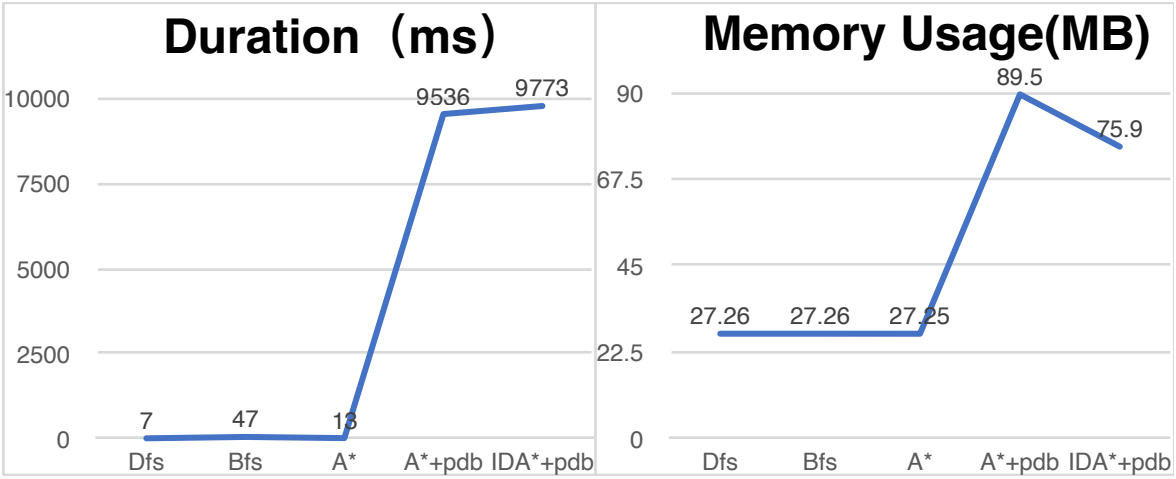
- Duration: 13ms
- Memory Usage: 27.25MB

A-Star with pattern database heuristic:

- Duration: 9536ms
- Memory Usage: 89.50MB

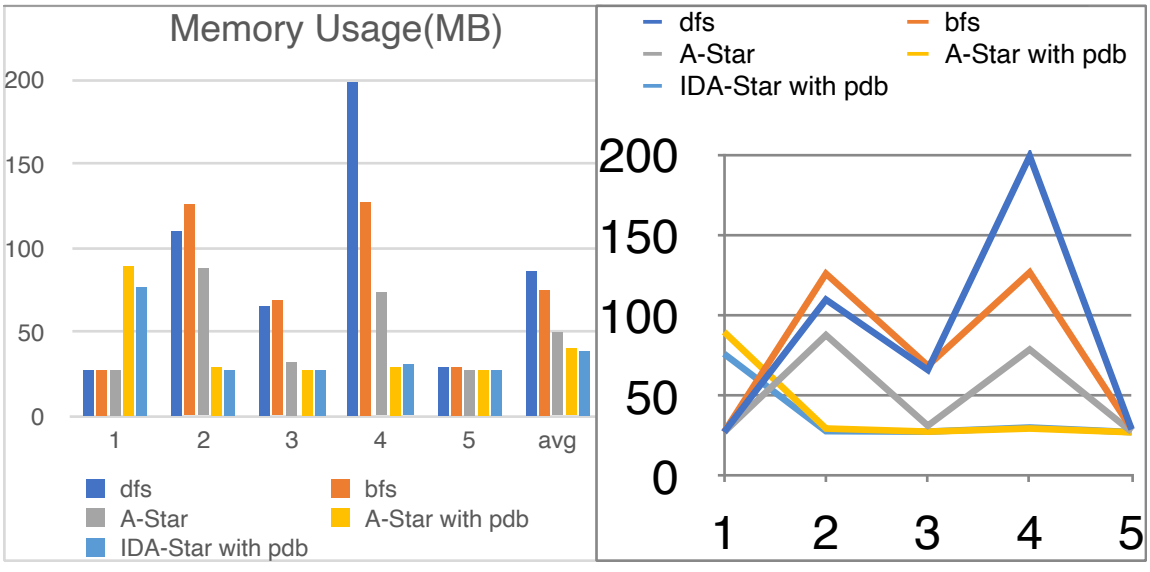
IDA-Star with pattern database heuristic:

- Duration: 9773ms
- Memory Usage: 75.90MB

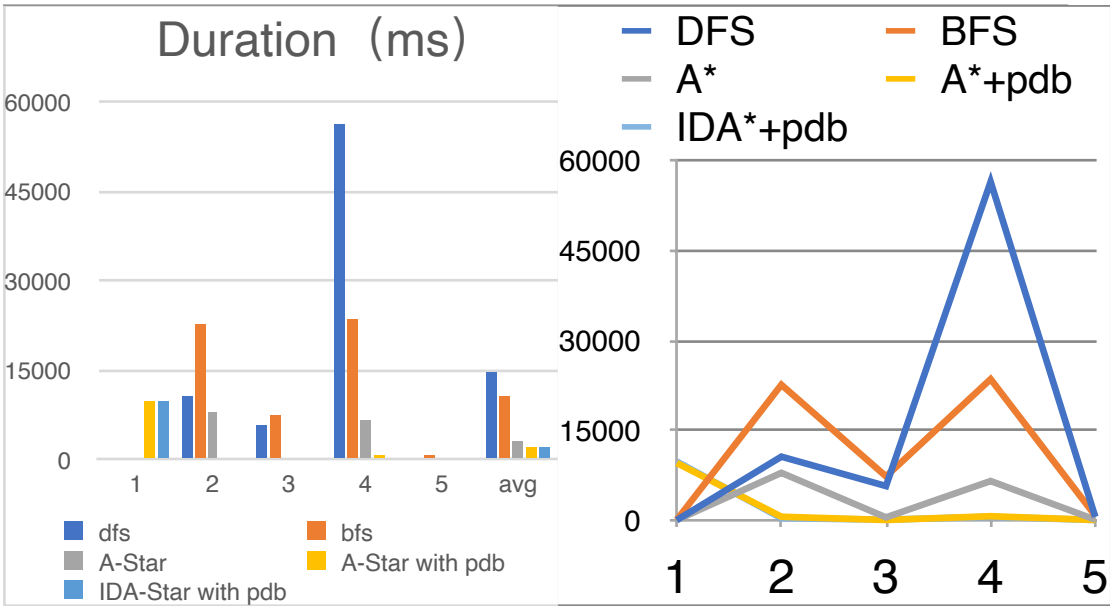


To ensure the accuracy of the data, we ran the program many times and got the following data:

Memory Usage	1	2	3	4	5	avg
dfs	27.26	109.56	65.92	198.68	28.94	86.072
bfs	27.26	125.69	68.51	126.52	29.39	75.474
A-Star	27.25	87.55	31.25	73.59	27.5	49.428
A-Star with pdb	89.5	29.52	27.59	29.51	27.19	40.662
IDA-Star with pdb	75.9	27.89	27.59	30.14	27.28	37.76



Duration	1	2	3	4	5	avg
DFS	7	10589	5689	56458	548	14658.2
BFS	47	22589	7298	23524	745	10840.6
A*	13	7894	458	6524	78	2993.4
A* with pdb	9536	568	25	652	29	2162
IDA* with pdb	9773	328	79	556	17	2150.6



6. CONCLUSION

It can be seen from the initial state that DFS has the fastest speed in solving problems, only taking 7ms, while BFS takes 47ms. The space used by these two algorithms is the same. The speed of A-star algorithm is 13ms, which is faster than BFS, but slower than DFS. The solving time of the two A-star and Ida star algorithms using PDB is 9536 and 9773ms respectively. It can be found that the A-star algorithm using PDB is slower than the ordinary A-star algorithm. However, according to the concept of PDB, the A-star algorithm using PDB should be better than the ordinary A-star algorithm. The reason for this situation is the selection of the initial state. For the current initial state, the ordinary A-star algorithm is better than the A-star algorithm adding PDB. Even DFS is faster than A-star, but when the initial state changes, the problem-solving time will also change. For more complex initial states, the A-star algorithm using PDB has shorter problem-solving time. This is because PDB saves many intermediate states and does not need to recalculate, which saves a lot of time and plays the role of a memo. After the experiment using the random initialization

state, we found that the average consumption time of IDA star algorithm using PDB is the least, so PDB can indeed help solve complex problems.

7. REFERENCE

[1] H. Ma, Lecture notes, Topic: " 4. Search, 5. Informed_Search and 13. Heuristic_Functions", CMPT 417, Simon Fraser University.
[2] D. Toby, "Solving Problem by Searching", Available: https://www2.cs.sfu.ca/CourseCentral/310/tjd/chp3_search.html.
[3] G. Aditya, "How to check if an instance of 15 puzzle is solvable?", Available: https://www.geeksforgeeks.org/check-instance-15-puzzle-solvable/.
[4] H. Sharma, A. Alekseychuk, P. Leskovsky, O. Hellwich, R.S. Anand, N. Zerbe, and P. Hufnagl, "Determining similarity in histological images using graph-theoretic description and matching methods for content-based image retrieval in medical diagnostics," Diagnostic Pathology, vol. 7, no. 1, pp. 20, 2012.
[5] N. Anders, "N-Puzzle Problem Solution using DFS, BFS, IDS, GBFS, UCS, A* Algorithms in Artificial Intelligence in C++", Available: https://myslns4u.com/archives/2457.
[6] Osmo, et al. "The 15 puzzle: how it drove the world crazy." Mathematical Intelligencer (2007).
[7] Ratner, D. , and M. K. Warmuth . "Finding a shortest solution for the N x N extension of the 15- Puzzle is intractable." AAAI Press(1986).