15-Puzzle and Pattern Databases

Angus Fan

angusf@sfu.ca

Richard Dong

rcdong@sfu.ca

Introduction

In this assignment, a program was implemented to optimally solve random instances of the 15-puzzle. The 15-puzzle is a sliding puzzle that consists of 4x4 tiles. The object of the puzzle is to move the empty space around until reaching the goal state. An image of the goal state is provided in figure 1. This project was coded in C++ using modules from the standard library such as iostream, vector, algorithm,cmath, queue, unordered\_map, string, fstream,sstream, time.h. A class for the 15-puzzle was created containing the functions FifteenPuzzle, getInitial, find\_blank\_square, actions\_possible, result, goal\_test and solvable. The Node class contains the functions Node, getDepth, getState, compare\_states. Additional functions to help create the solver for the 15-puzzle was search, ida\_star, display\_state, bfs\_to\_database, copy\_array, toString, get\_actions, get\_results, P\_Hub.

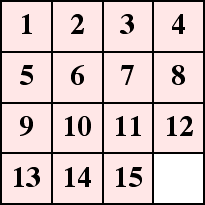


Figure 1

Functions similar to the ones in EightPuzzle were used. Using AI techniques such as ida\_star, admissible heuristics, search trees and databases enabled the solver to work on the 15-puzzle. To efficiently solve the puzzle, a memory-efficient A\* algorithm IDA was implemented. To compliment the IDA\* search, a pattern database heuristic is used. The 5-5-5 partition was chosen which splits the 15-puzzle into three blocks containing 5 tiles because the time it takes to construct the database is shorter (16!/(16-5)! \* 3 entries) compared to other partitions such as 6-6-3 (16!/(16-6)!\*2 + 16!/(16-3)! entries) or 7-8. Figure 2 shows a visualization of the three different partitioning techniques for the pattern database.

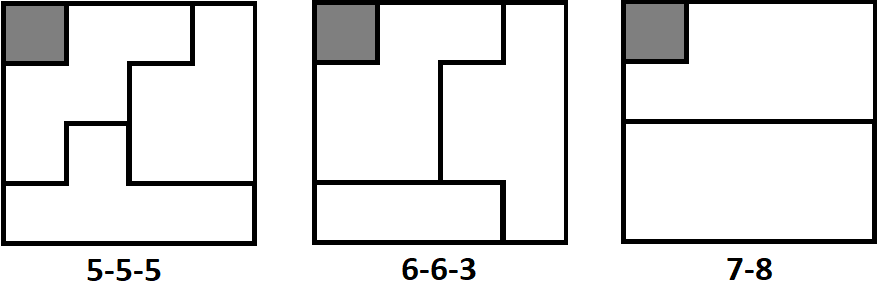


Figure 2

Then, BFS was used on the partitions with the blank tile for each group or pattern. The database is in the form of a table containing both the pattern value and the respective BFS cost. Combining the IDA with the heuristic, at each cycle of this algorithm when creating nodes in the graph, the cost for the node at the puzzle state is calculated. This is done by discovering the cost of the groups using our pattern database thus giving us the heuristic value. We could then use these AI techniques to solve the 15-puzzle in a reasonable amount of time.

Reasoning for C++

As stated in the introduction, this assignment was coded in C++ and not python. The reason for this is that C++ is an efficient programming language which allows for the 15-puzzle to be optimally solved. C++ executes faster and creates more time-efficient programs. Compared to python where the programs are interpreted, C++ is compiled. Another benefit is that there will be no signal type errors on runtime due to the initialization of variable types.

Explanation of IDA\*

As previously mentioned, in this assignment Iterative deepening A\* was used. IDA\*, a graph traversal and path search algorithm identifies the smallest path between the goal nodes and a specified start node. Similar to iterative deepening DFS, IDA\* uses a heuristic function to determine the cost to reach the goal from A\* search. IDA\* uses the function f(n) = g(n) + h(n), where g(n) is the traversal cost to node n from the root and h(n) is a heuristic estimate of the traversal cost to the goal from n. At every iteration a DFS is executed. When the total cost of f(n) = g(n) + h(n) is larger than the initial cost estimate, a branch is cut off. Two important properties of IDA\* are as followed:

1. IDA\* finds the shortest path from the initial node to any goal node when the heuristic function h is admissible: h(n) < h\*(n)
2. IDA\* does not maintain the nodes in memory except the current path which means the memory is linear in the length of the solution: h(n) < cost(n,n’) + h(n’)

Pseudocode:

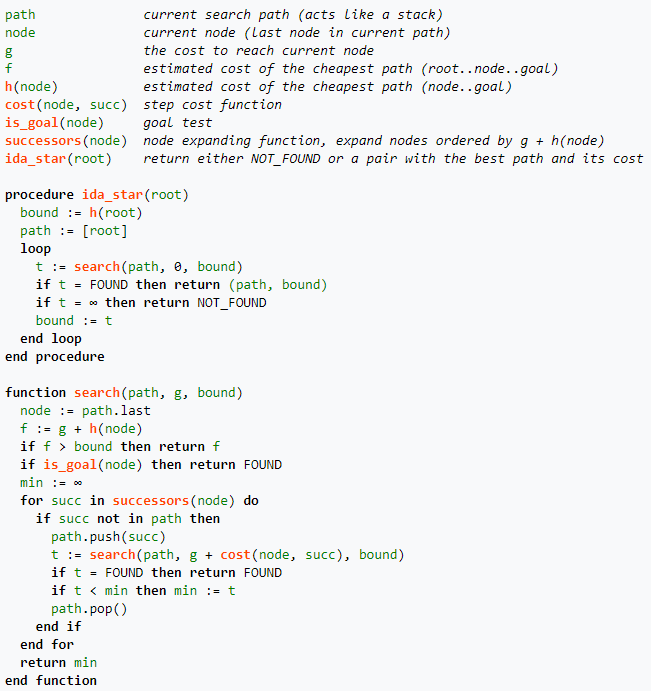


Figure 3

Explanation of Database

The pattern database is a heuristic function which is saved in a look-up table containing the cost of the subproblems. To create the database, dj\_pattern.cpp contains a function called bfs\_to\_database which utilizes breadth first search, an algorithm which traverses a tree/graph. BFS continuously explores the branch until backtracking to expand other nodes with time complexity O(|V| + |E|). The pseudocode of BFS is given in figure 4, and the pseudocode bfs\_to\_database is shown in figure 5. The patterns are split up in 3 different formats shown in figure 6. These patterns were stored as csv files as pattern1.csv, pattern2.csv and pattern3.csv respectively. These files are then read in the main function as a heuristic function for our IDA\* search algorithm.

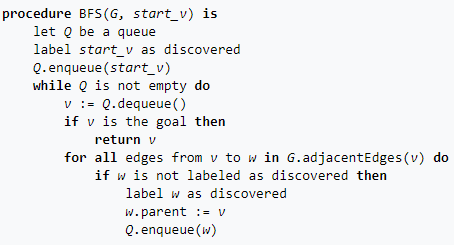


Figure 4

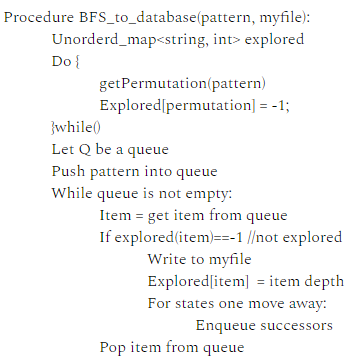


Figure 5

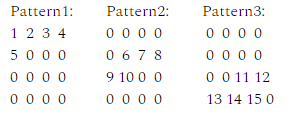
****

Figure 6

Instructions to run

Enter in the terminal the following command:  
  
g++ --std=c++11 dj\_pattern.cpp a5.cpp -o a5

Followed by the command:

./a5

The current a5.cpp will the 15-puzzle solver with the state:

7 5 2 4

1 6 3 8

9 14 10 11

13 0 15 12

The time to run IDA\* star and the number of nodes will be shown in the terminal.

Results

Using a memory efficient A\* algorithm IDA combined with a pattern database to make use of heuristics and search trees, the 15-puzzle was solved in an efficient manner. The 5-5-5 pattern database was chosen for this assignment. The table and chart below provide trials for 25 different 15-puzzles. Most of the states were solved in less than a second with a few outliers which skewed the average. The average then was still only 3.768 seconds for the 25 trials. The node length travelled on average was 18.16. Using AI techniques learned in this course, the 15-puzzle solver was efficiently implemented.

Trials

|  |  |  |
| --- | --- | --- |
| 15 puzzle | Length (nodes) | Running Time (Seconds) |
| Test 1 (1,2,3,4,5,6,7,8,9,10,11,12,0,13,14,15) | 3 | 0.017 |
| Test 2 (1,2,3,0,5,6,7,4,9,10,11,8,13,14,15,12) | 13 | 0.0193 |
| Test 3 (1,2,3,4,5,6,7,8,9,10,0,11,13,14,15,12) | 2 | 0.0113 |
| Test 4 (1,2,3,4,5,6,7,8,9,10,11,12,13,14,0,15) | 1 | 0.0112 |
| Test 5 (1,2,3,4,5,6,8,0,9,10,7,11,13,14,15,12) | 4 | 0.0228 |
| Test 6 (1,0,2,3,5,6,8,4,9,10,7,11,13,14,15,12) | 7 | 0.0282 |
| Test 7 (5,1,2,3,9,6,8,4,0,10,7,11,13,14,15,12) | 10 | 0.0347 |
| Test 8 (5,1,2,0,9,6,4,3,13,7,8,15,14,10,12,11) | 21 | 0.203 |
| Test 9 (5,1,2,4,9,7,3,8,0,6,10,11,13,14,15,12) | 10 | 0.0376 |
| Test 10 (7,5,2,4,1,6,3,8,9,11,12,15,13,10,0,14) | 17 | 0.281 |
| Test 11 (0,2,3,4,5,7,1,8,9,6,12,15,13,11,10,14) | 22 | 3.437 |
| Test 12 (2,7,3,4,9,5,1,8,6,11,12,15,13,0,10,14) | 32 | 12.312 |
| Test 13 (0,2,7,4,9,5,3,8,6,11,1,15,13,10,12,14) | 27 | 5.218 |
| Test 14 (9,2,7,4,0,5,3,8,6,11,1,15,13,10,12,14) | 39 | 20.578 |
| Test 15 (6,0,3,4,2,1,7,8,5,10,11,12,9,13,14,15) | 11 | 0.0312 |
| Test 16 (6,1,4,8,2,7,3,0,5,10,11,12,9,13,14,15) | 14 | 0.0937 |
| Test 17 (6,7,1,8,0,2,4,3,5,10,11,12,9,13,14,15) | 19 | 43.234 |
| Test 18 (9,5,2,4,1,6,3,8,0,10,7,11,13,14,15,12) | 24 | 0.140 |
| Test 19 (9,5,2,4,0,1,3,8,10,6,7,11,13,14,15,12) | 21 | 0.0156 |
| Test 20 (3,0,6,4,5,9,2,8,1,10,7,11,13,14,15,12) | 29 | 0.828 |
| Test 21 (3,9,6,4,5,2,0,8,1,10,7,11,13,14,15,12) | 29 | 0.765 |
| Test 22 (3,9,6,4,5,2,8,11,1,10,0,7,13,14,15,12) | 26 | 1.312 |
| Test 23 (4,0,2,11,3,5,9,6,1,10,8,7,13,14,15,12) | 29 | 1.656 |
| Test 24 (4,5,2,11,3,0,9,6,1,10,8,7,13,14,15,12) | 30 | 4.031 |
| Test 25 (7,5,2,4,1,6,3,8,9,14,10,11,13,0,15,12) | 14 | 0.0156 |
| Average | 18.16 | 3.768 |

