

Assignment 1 Web Board Summary

Revised January 21, 2004, by Kristin Branson

The CSE 150 discussion board has received a lot of traffic over the past couple days. Here is a summary of the posts about Assignment 1. Please see the discussion board posts for more detail.

1 The A* Algorithm

In Assignment 1, you are to implement the A* algorithm and run it on the eight-puzzle and robot navigation problems. The general A* code and the code specific to these two problems should be your own. However, you *are* allowed to reuse old code for standard data structures, for instance for the priority queue in this assignment. If you use old code, it must be from a standard library, it must be an appropriate choice, and the reuse must be carefully acknowledged and explained in comments.

For the eight-puzzle problem and also for the robot navigation problem, you must provide a main program that will input a problem instance and then execute A*. We will use this command-line interface to check the correctness of your software on problem instances that we invent. The output of this main program should be the solution path, its cost, its depth, and the number of nodes generated.

Your A* code should be general and reusable for different problems. Each problem has its own state representation, goal testing function, heuristic function, and node expansion function. Your code should be reasonably efficient and fast. For the eight-puzzle, generating one thousand nodes per second is reasonable; it is possible to do much better. Remember that the ACS computers will likely be overloaded when the assignment deadline approaches. Do not wait until the last hours to generate your eight-puzzle table.

Total-cost-so-far $g(n)$ is problem-specific so in general it must be supplied by the user. Usually the function that expands nodes provides $g(n)$ for each node n that it creates. Remember that total-cost-so-far $g(n)$ is usually different from the depth of n . Depth increases by 1 for each expansion, but cost increases by the cost of the action, which can be different for different actions. In the robot navigation problem, the cost of an action (one straight line segment movement) is its Euclidean length. For the eight-puzzle, the cost of each action is always one, so total-cost-so-far $g(n)$ and depth happen to be the same.

2 The Eight-Puzzle Problem

For the eight-puzzle problem, you must replicate the table in Figure 4.8 of your text book, and present your results in your report. To replicate this table, you need to generate 1200 random problems with solution lengths from 2 to 24 (100 for each even number).

How do you generate 1200 problems with these properties? Note that the goal state can only be reached from half of the $9!$ possible states. One way to generate random states is to start from the goal configuration and to “shuffle” the tiles randomly many times. Since A* finds the shortest solution, you can choose 100 of the problems which A* finds to have solutions of length 2, 100 of the problems which A* finds to have solutions of length 4, and so on.

It is possible that your random problem generator will generate the same problem multiple times. In particular, there are only four different problems with solution length 2. If you have difficulty generating and solving 100 problems of each length, explain the issues in your report (briefly!).

The text is vague about the details of how Figure 4.8 was generated. Where the text is ambiguous, you can use any method you like. The most important thing is that you *document exactly what you do*

in your report. Here are some of the ambiguities discussed on the discussion board.

- Two columns of the table show “search cost.” It is unclear what search cost means. The text says it is the number of nodes expanded, i.e. the number of nodes the “expand” function is called on. The numbers in the table suggest that it is actually nodes generated, i.e. the number of nodes expanded plus the number that are in the fringe when the A* function finishes. Use whichever value you like, but be sure to document your choice.
- Nodes in the fringe are sorted by f -value. The text does not detail how ties should be broken. If you have duplicate f -values, you need a tie-breaker. If you can think of an intelligent tie-breaking policy, that is good; explain it in your report. If not, a randomized policy is best. A systematic policy (such as “always in front” or “always behind”) is dangerous because it may be systematically bad.

When A* terminates with success, it is guaranteed to have found *one of* the shortest paths. The tie-breaker policy can affect *which* shortest path is found. It can also affect how many nodes with the same f -value as that of the shortest path are expanded.

- Some students have suggested that the low “search cost” numbers in the table might have been achieved by not allowing multiple nodes with the same state in the fringe at a time. Duplicate nodes should be replaced by the duplicate node with the lowest f -value. This is a good observation.

Nowhere in the text of the chapter describing how Figure 4.8 was generated, or the A* algorithm description, does it mention that they did not allow duplicate states to be inserted into the fringe. However, this is a reasonable improvement, as it does not require extra memory, though it does require more time. The savings in nodes generated may indeed be worth this factor.

It will make a big difference what data structure (i.e. what indexing algorithm) you use to find duplicates. Linear search is a bad idea! You can either allow duplicates or not allow duplicates. Again, the most important thing is that you document your choice.

If the choices you make result in your numbers not matching the numbers in Figure 4.8, or you are not able to solve 100 problems of each length, that is acceptable. We will not penalize you for this if your code is correct and efficient.

3 The Robot Navigation Problem

For the robot navigation problem, you should run your code on the four test cases that are published at <http://www.cs.ucsd.edu/~elkan/150/robot.html>. You should report the results of these tests in a table similar to Figure 4.8. Your table should have a row for each of the four problems. You should report solution cost, solution depth, search cost, and effective branching factor. You do *not* need to generate random navigation problem instances.

You may assume that the goal is always a single point in the plane. You may also assume that obstacles do not intersect, and that all obstacles are convex.

Generally speaking, the navigation problem instances are easier than the eight-puzzle problem instances, because the optimal path is only a few segments long. But the branching factor is high.

For more advice on the robot navigation problem, be sure to read the section notes from January 16, at <http://www-cse.ucsd.edu/~elkan/150/jan16.pdf>.