CS 534
Assignment #1: Search

Due: Feb 12 @ 11:59 p.m.

This assignment will require you to explore a variety of search techniques on two problems. Note: in addition to the programming component of developing the algorithms, there is a moderate amount of analysis/writeup required as well. If you finish your code just before the deadline, you will not finish the assignment. Start early.

Part 1. Heavy N-queens problem

The problem

For this problem, you will work on a variant of the N-queens problem we discussed in class. The queen pieces are very heavy, and therefore difficult to move. The cost to move a queen is 10 plus the square of the number of tiles you move it. So to move a queen upwards 4 squares in a column costs $10 + 4^2 = 26$. Otherwise, the rules are identical to the N-queens problem.

Approaches

You will use two approaches for this problem: A* and greedy hill climbing with restarts.

For A*, you need a heuristic function. The number of (directly and indirectly) attacking queens is a good place to start. Since moving a queen costs 10 points, you can modify the heuristic to be:

1. If no attacking queens: 0
2. If attacking queens: 10 + # of pairs of attacked queens

Note: it is not immediately clear if this heuristic is admissible. Moving one queen can reduce the cost dramatically (as we talked about in class). However, given the fixed cost of 10 to move a queen, perhaps no situation arises where the true cost is less than the heuristic value. For extra credit, prove either that the heuristic is admissible or produce a counterexample to demonstrate that it is not.

For hill climbing with restarts, you should use the same heuristic function as for A* and perform greedy hill climbing. If you have not found a solution, and fewer than 10 seconds have elapsed since the program started running, you should do another iteration of hill climbing with a random start state.

Program behavior

Your program should take as input the N value for the N-queens problem and the type of search to conduct (1 for A*, 2 for greedy hill climbing). For example, passing in a "30" results in a 30-queens puzzle. You can take input as command-line input, or read it from a file if command-line input is difficult for the language you are using. Just explain how in your writeup.

Your program should output:

- The start state (a simple text representation is fine)
- The number of nodes expanded. Consider an expansion as when you visit a node and compute the cost for all of the possible next moves. For hill climbing, remember to count this value across all of the restarts!
- Time to solve the puzzle
- The effective branching factor: consider how many nodes were expanded vs. the length of the solution path
- The cost to solve the puzzle. Note that A* computes cost automatically, but you will need to extend greedy search to keep track of this number. Note that cost does not include any failed runs with greedy search -- just compute the cost of the path it computes for the board it solves. (Note the comparison with A* is not quite fair since greedy search gets to discard its results from tougher boards)
- The sequence of moves needed to solve the puzzle, if any (hill climbing may fail).

Writeup

You should conduct experiments to determine:

1. How large of a puzzle can your program typically solve within 10 seconds using A*? Using greedy hill climbing?
2. What is the effective branching factor of each approach? For this computation, perform 10 runs of a puzzle half the maximum size you calculated for step #1.
3. Which approach comes up with cheaper solution paths? Why?
4. Which approach typically takes less time?

Part 2. Urban planning

The problem

For this problem, you will use hill climbing and genetic algorithms to determine the ideal location of industry, commerce, and residential sections a city. You will input a map with the following symbols:

- X: former toxic waste site. Industrial zones within 2 tiles take a penalty of -10. Commercial and residential zones within 2 tiles take a penalty of -20. You cannot build directly on a toxic waste site.
- S: scenic view. Residential zones within 2 tiles gain a bonus of 10 points. If you wish, you can build on a scenic site but it destroys the view.
- 0...9: how difficult it is to build on that square. You will receive a penalty of that many points to put anything on that square.

You will have to place industrial, residential, and commercial tiles on the terrain.

- Industrial tiles benefit from being near other industry. For each industrial tile within 2, there is a bonus of 3 points.
- Commercial sites benefit from being near residential tiles. For each residential tile within 3 squares, there is a bonus of 5 points. However, commercial sites do not like competition. For each commercial site with 2 squares, there is a penalty of 5 points.
- Residential sites do not like being near industrial sites. For each industrial site within 3 squares there is a penalty of 5 points. However, for each commercial site with 3 squares there is a bonus of 5 points.

Note that all distances uses the Manhattan distance approach. So distance is computed in terms of moves left/right and up/down, not diagonal movement.

Approaches

You will use hill climbing with restarts and genetic algorithms for this assignment. When your hill climbing restarts, it may **not** modify the map! It can only change where it initially places the different zones to begin the hill climbing process.

Your genetic algorithms will have to come up with a representation, selection and crossover methods, and must make use of culling, elitism, and mutation.

Program behavior

Your program will read in a file where the first 3 lines are the number of industrial, commercial, and residential locations (respectively). The remainder of the file is a rectangular map of the terrain to place the town. Your program should then run for approximately 10 seconds and output the following to a file:

- The score for this map
- At what time that score was first achieved.
- The map, with the various industrial, commercial, and residential sites marked.

Writeup

You should analyze your program's behavior to answer the following questions:

1. Explain how your genetic algorithm works. You must describe your selection, crossover, elitism, culling, and mutation approaches.
2. Create a graph of program performance vs. time. Run your program 10 times and plot how well hill climbing and genetic algorithms perform after 0.1, 0.25, 0.5, 1, 2, 3, 4, 5, 6, 7, 8, 9 and 10 seconds. If you only had 0.25 seconds to make a decision, which technique would you use? Does your answer change if you have 10 seconds?
3. How do elitism and culling affect performance?
4. How did you perform selection and crossover for your population? Find some known method for doing selection, other than the one described in class. Explain what you did.

What to hand in:

1. You should submit your assignment on Canvas.
2. Submit one file for part 1, and one file for part 2.
3. Include instructions for how to run your code. If the TA needs help from you or is confused, you will lose points.
4. The writeups from part 1 and 2.

Hints

1. There is a moderate amount of analysis and writeup for this assignment -- do not forget that component.
2. Your program should run for the specified length of time. Note the person grading your program may have a different hardware setup, so do not have a fixed number of iterations before the program terminates. Instead use calls to determine the amount of elapsed time. We are not worried about perfect precision -- if your program terminates after 10.2 seconds, that is ok. Anything in the range of 9.5 seconds to 10.5 seconds is fine.

Errata

1. Some points for this assignment are based on algorithm performance. For example, we will give you a test map for urban planning and see how well your program does. The scoring approach is there is a notion of “reasonable” performance, and failure to get a score that high will cost points.
2. For the first question, do iterative deepening A* to reduce memory usage. (Bigger deal than the admissibility of the heuristic)
3. EXPLAIN HOW TO SELECT WHICH METHOD FOR URBAN PLANNING PROBLEM: HILL CLIMBING VS. GENETIC ALGORITHM