CS 2710/ISSP 2160 Fall 2013

Assignment 1: Problem-Solving Search

This must be your own individual work. If you base your heuristics on any papers, books, web resources, etc., you should cite them.

1 Introduction

After starting with a couple short-answer questions, this assignment will give you experience with heuristic search algorithms by comparing alternative search strategies and experimenting with heuristic evaluation functions.

It will also give you experience writing a report logically arguing for the relative strengths and weaknesses of algorithms for particular problems.

Your work will be graded for correctness, completeness, clarity, and the logic of your presentation.

Please submit one zipfile containing: your report, your code, any required input files, and a README file explaining how to run your program.

You will submit your solutions via courseweb; detailed instructions will be added here soon.

Please use the following naming convention:

lastnameFirstname(Version).zip examples:
akkayaCem.zip
akkayaCemv2.zip

Please submit by 11:59pm on the due date to avoid a late penalty.

2 Short-Answer Questions

Begin your report with answers to the following questions.

Q1 The wrap-up for slides Chapter3Part1 includes this note: The book writes separate code for breadth-first search, i.e., they don't call treesearch as in the class code. Their

version applies the goal test to a node when it is first generated, before it is added to the fringe. This can save time and space: In the worst case, when the goal is on the right frontier of the search tree, my version of breadth-first search generates, and adds to the fringe, an extra level of nodes than their version does. In figure 3.21 (time and space), breadth-first search is $O(b^d)$, and uniform-cost search is $O(b^{(d+1)})$ if all edge-costs are equal.

Q1.a (7%) Argue that the text's implementation maintains the completeness and optimality properties of breadth-first search.

Q1.b (3%) Give an example showing that optimality is lost if we were to change treesearch as follows: it applies the goal test to a node when the node is first generated, before it is added to the fringe. Give the state space, name the search algorithm, and show a trace of the algorithm finding a suboptimal solution. Give a **small** example. You can define the problem in the style used in searchP*.py

Q2 (5%). In class, we said that breadth-first search is optimal if the edgecosts are all equal. On page 82 in R&N just below figure 3.11, we learn that this isn't strictly true - it is optimal if the path cost is a nondecreasing function of the depth of the node. Explain why this is so.

3 Problem Implementation and Analysis

The description below assumes you will use code from searchP*.py and utils.py on the course web page. If you prefer to reimplement this code yourself in another language, that would be fine. Please be sure that your code enables you to make the comparisons and produce the results required in the report, and that you exactly replicate the algorithms.

The features of Python used in searchP*.py should be sufficient for your project. If you don't know Python, go through the tutorial (the first for Python 2* and the second for Python 3*):

`http://docs.python.org/2/tutorial/`
`http://docs.python.org/3/tutorial/`

Then, try to understand simplesearchP*.py, then searchP*.py. If you have questions, please see the TA or me in office hours.

In search.py, feel free to remove the main code and any printing code you don't want to use.

The Problem 6 dictionaries in the file implement the Romanian route-finding problem from the book. The heuristic is "straight-line distance": h(state) is the straight-line distance from state to the goal.

3.1 Problem Implementation

8-Puzzle:

Q3 (8%) Write versions of goalp, successor, and edgecost to implement the 8-puzzle.

Also, define 3 heuristic evaluation functions for the 8-puzzle:

Q4 (4%) Number of tiles out of place

Q5 (4%) Manhattan distance: This is the sum of the (horizontal and vertical) distances of each tile from its goal position.

Q6 (6%) Another heuristic. Simply taking the MAX of the two above will give you a grade of B for this specific part of the assignment. Something more interesting/original will give you an A for this part. Something really interesting, such as a pattern-database approach, will give you extra credit.

Romanian route-finding problem:

Q7 (1%) Write versions of goalp, successor, edgecost, and h to implement the Romanian route-finding problem. These are just accesses to the hashtables/dictionaries of Problem 6 in search.py.

3.2 Statistics Gathering

Add global variables to the code, *maxQlen* and *numGenerated* that record the maximum fringe size and the number of nodes generated during a search process. Also, add appropriate print statements, calls to search algorithms, and statistics-gathering code, as needed, to develop the report described below.

3.3 Report

The report should do the following.

For the 8-puzzle only:

Q8 (6%): Show that your implementation of the 8-puzzle works correctly. That is, demonstrate that your successor function, goalp function, edgecost function, and h functions work correctly by showing pieces of a script of your code running in a verbose mode (i.e., with extra print statements). Please include just enough to convince us your code works, and please be sure this is readable (add comments to the file to tell us what we are looking at, if it isn't obvious).

For both the 8-puzzle and the Romanian route-finding problems:

- Run treesearch (not graphsearch) versions of depth-first-search, breadth-first-search, uniform-cost-search, iterative-deepening-search, best-first-search, Astar search, and IDAstar search. Run them on multiple start states. Compute the statistics mentioned above (e.g., maxQlen). Run the heuristic searches with all of the heuristics you defined for that problem.
- Your report should compare the various search algorithms and heuristics, supporting your arguments with statistics and observations from your runs. The statistics include (1) maxQlen and numGenerated mentioned above, (2) information about whether a goal was found, and (3) information about the quality of that goal (is it optimal?). In your report, you can report raw counts, averages, as well as averages of maxQlen and numExpanded as a function of optimal solution length.
- You should decide how to best organize your report. Specific issues to address are:

  Q9 (11%): What are the properties of the problem? That is, how big is the search space? what is the average branching factor?

  Q10 (15%): What are the strengths and weaknesses of each algorithm for this problem, and why.

  Q11 (15%): For each of your heuristics, is it admissible? Please explain why it is or isn't. Please provide evidence from your experiments that the heuristic is or is not admissible.

  Q12 (15%): For the 8-puzzle, consider for each pair of heuristics whether one is more informed than the other. Please explain your answer. If one is more informed than the other, show the effect of this (i.e., give the relevant statistic(s) showing the effect of one being more informed than another).

Please use common sense in managing your computational processes. We don't want the class to drag the computers down too much.

Not all initial states for the 8-puzzle are solvable; that is, there are initial states such that no path exists from it to a goal state.

Here are some sample solvable initial states for the 8-puzzle:

```
Goal:        Easy:        Medium:      Hard:        Worst:

1 2 3        1 3 4        2 8 1        2 8 1        5 6 7
8   4        8 6 2          4 3        4 6 3        4   8
7 6 5        7   5        7 6 5          7 5        3 2 1
```