15-381 Problem Set I: SEARCH

Handed out: 25 January 2000
Due back: 10 February 2000

This is the first problem set for 15-381, incorporating programming and non-programming parts. If you obtain concrete help from anyone (e.g. TA's, fellow students, friends, etc.), you ***must*** document the nature of the help you receive, which we'll take into account in the grading. Note that it is much better to seek help and turn in a assignment complete and on time than to not seek help and fail to complete it, either on time or at all. In general, we will not deduct much if anything for seeking help, so long as that help is indirect (e.g. help with debugging your program, rather than someone telling you exactly how to solve one of the problems).

Implementing the Search Engine

You are to implement a universal forward-search engine that is given as input:

- A 2D rectangular grid, which implicitly defines the set of possible states, to be read in as a simple ascii file, and represented internally in any way you wish that permits you do implement the search engine. The external ascii representation is as follows:

```
O O X X X O
O O O O O O
X X O O O O
X X O X O O
O O O O X O
```

  The O's indicate legal states, and the X's blocked-off (illegal) ones. The coordinates are (1,1) as the upper left corner and (6,5) as the bottom right corner in the above example. In general the rectangular grid may be of any size and the blocked squares may be anywhere. The coordinates are always (x,y) pairs.

- The initial state (assume it is unique), will be represented as an (x,y) coordinate in the grid above. For instance (1,1) would be a legal S_0. But, so would any other non-blocked state.
- The goal state (also assume uniqueness), also represented in the same way. For instance (3,4) would be a legal S_G.
- The start and goal points will be read in from the keyboard in the form (x,y). The kind of search, as will be explained later, should also be read in from the keyboard.
- The successor function which is defined as:
  SUCC(x,y) = legal({(x+1, y), (x, y+1), (x-1, y), (x, y-1)})
  where "legal" is a filter that throws out points outside the grid boundary, such as (2,6) or (0,3) in the example above, and also throws out any X'ed point.
- A keyword parameter that can be set to BFS (breadth-first search), DFS (depth-first search), or BestFS (Best-first search).
- When you do Best-First search, use the Manhattan distance between S_{curr} and S_G.

The output of the program should be a sequence of states that starts with the initial state and ends with the goal state. For instance, given S_0 = (1,1) and S-(G) = (3,4), an acceptable solution would be: [(1,1)

(1,2) (2,2) (3,1) (3,2) (3,3) (3,4)], as it is anchored at S_0 and S_G, and visits only legal states.

You should use one function to do all three searchers, not three different functions disguised as one. (Hint: using a next-state queue of generated but as yet unvisited states as your primary data structure, BFS implies a FIFO discipline, and DFS implies a LIFO or "stack" discipline, and BestFS implies sorting the queue according to the heuristic evaluation function.)

Note that you will need a means of checking whether a state has already been visited in order to avoid loops. For simplicity you may assume that all problems given are solvable, and all grids are finite (in fact, they will be 10 by 10 or smaller).

You should write your program in C or C++, and you must hand in formatted, documented code plus an executable compiled on linux. The place for handins will be announced later.

Extra Credit Opportunities

A more efficient search method also checks for equivalences. In other words, a state may be reached via different paths from the initial state. For extra 5 points credit, ensure that only the shortest path remains alive in the search when two paths from the initial state merge.

The grid-search problem is of course amenable to bi-directional breath-first search. For 10 points extra credit, implement bi-directional breadth-first search (with BiBFS as a fourth parameter value to your universal search program). Make sure you do interwoven forward and backward fringe expansion and check for intersections.

The grid-search problem is equally amenable to island-driven search. For 10 more points extra credit, allow a fifth value to the search-mode parameter Island-BFS for island-driven BFS. In this case, instead of an initial state you should give the system a list of states, the first of which is the initial state and each subsequent one is an island that must be visited on the way to the final state. Note that there are no guarantees of the islands being equally spaced.

Testing your Search Program

Your universal search (including any extra-credit variants) should run correctly on any grid that conforms to the specifications mentioned above. You can find some grids at cyprus.res.cmu.edu.

Grading

Grading will be based on mainly three things. 1. Does the program find a correct (and occasionally shortest) path? 2. Does the program conform to standards mentioned above? 3. Does the program implement correct techniques for each different kind of search? Note that you may not use the distance function for DFS and BFS and BiBFS.

Short Answers

Please provide short answers to the following when you turn in the final homework:

- Why do you think that BestFS does well, in terms of visiting fewer states than the other searches most of the time? Create a pathological case (grid + S_0 + S_G) where BestFS with Manhattan distance would do worse that BFS.
- How would you implement A^* on the gridworld? Give f, g and h estimation functions and say why f is admissible.
- What is the average branching factor on a 10-by-10 grid where 20% of the points are blocked off?
- How many states will be visited (at most) for island-driven biBFS search with 2 evenly-spaced islands, if the average-branching factor is 3 and and distance to nearest goal is 18?