CSCI1410 Fall 2017
Assignment 1: Search

Due Monday, September 18

1 Introduction

In this assignment, you will implement several search algorithms in their general forms. Then, you will use those search algorithms to solve tile game puzzles.

The tile game is similar to the one you encountered in class. There is still a 3-by-3 grid, and each cell in the grid is still occupied by a number. Also, the win condition is the same; arrange the numbers in order from lowest in the upper left, to highest in the lower right. However, the grid now has 9 numbers on it, instead of 8 numbers and an empty space. Additionally, movement is less restricted. In this game, you can swap any pair of adjacent numbers. The win condition and your goal during search is this board configuration:

\[
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9 \\
\end{bmatrix}
\]

2 Your Task

2.1 Coding

In `search.py`, you will implement the breadth-first (bfs), depth-first (dfs), iterative deepening (ida), bi-directional (bds), and A* (astar) search algorithms. Each search function takes in a `SearchProblem`, called `problem`, as an input, and outputs the whole path of the solution. The path should be represented as a list of states, where the first element is the start state and the last element is a goal state. In addition, you will fill in `tilegame_heuristic` with a heuristic function for tile games. Do not use any additional python libraries when implementing these searches.

2.1.1 Additional Specifications for Bi-Directional Search

- Find a path from the start state to the given goal state (if there are other goal states, ignore them).
• Run a breadth-first search from both the start state and the goal state, expanding one node at a time on each side.

2.2 Writeup

You should include in your submission a document, written in LaTeX, that describes your heuristic, proves that it is admissible*, and explains why you chose it as opposed to other possible heuristics.

* Note that if you cannot think of an admissible heuristic, you should, of course, not try to prove that your heuristic is admissible. Instead, prove that your heuristic is not admissible or, even better, that it is impossible to produce an admissible heuristic.

3 The Code Files

3.1 Files to Modify

• search.py - This is where you will implement your search algorithms.

3.2 Necessary Source Code

• searchproblem.py - This contains an abstract class, SearchProblem, for search problems. The SearchProblem class has three abstract methods that are shared among all search problems. Look at the function headers and their docstrings there before you begin.

• tilegameproblem.py - The TileGame class contained in this file extends the SearchProblem class. It contains implementations of the abstract methods from SearchProblem, internal helper functions, and utility functions that you may use in the code for your tile game heuristic and your testing. A state of the game is represented as a tuple of tuples, where each interior tuple is a row of the tile game. You will notice that the dimension of the Tile Game is adjustable. This is to ease your testing: you will find it easier to test your code on 2-by-2 games than on 3-by-3 games. Your heuristic is only required to work on 3-by-3 games.

• datastructures.py - This file includes implementations of a Queue, Stack, and Priority Queue. The Queue and Stack classes contains the methods push(item), pop(), and is_empty(). The PriorityQueue class contains the methods push_with_item(item, priority), pop(), and is_empty().

3.3 Testing Source Code

• dgraph.py - This is an implementation of a Directed Graph as a SearchProblem. You can use DGraph to create small test cases for your searches,
so if you would not like to use it for your testing, you can safely ignore
it. The implementation uses a matrix representation of a directed graph;
the states are each represented by a unique index in \( \{0, 1, ..., S\} \), where \( S \)
is the number of states, and the cost of moving directly from state \( i \) to
state \( j \) is the entry of a matrix at row \( i \), column \( j \). If it is impossible to
move directly from \( i \) to \( j \) (i.e., \( j \) is not a successor of \( i \)), then entry of the
matrix at row \( i \), column \( j \) should be \texttt{None} instead..

- \texttt{testsuite.py} - This contains a testing suite with one trivial test case
to help ensure that the input/output of each search function works prop-
erly. To run the test functions, choose one of the functions defined within
\texttt{testsuite.py} and pass in the appropriate search as the parameter for the
function. For example, to test breadth first search, use \texttt{bfs_test(bfs)}.

4 Grading

We will give you your score based on the rubric in \texttt{rubric.txt}. Here are some
details about the rubric:

- We will check each of your search algorithms to ensure that states are ex-
panded in a proper order. A state is expanded whenever \texttt{get_successors}
is called on it. For most search problems, there will be many correct orders
of expanding the states for each search algorithm. Our grading scripts will
give you full points if you expand each search problem in any of the proper
orders.

- For your Tile Game heuristic, you are graded based on “number of ex-
panded nodes when used with A*”. Your score for this will be determined
by the following formula:

\[
10 \cdot \frac{n_{\text{ours}}}{n_{\text{yours}}}
\]  

(1)

where \( n_{\text{ours}} \) and \( n_{\text{yours}} \) are the number of nodes expanded by our heuristic
and your heuristic, respectively, on a pre-selected suite of tile games. You
can score your heuristic on your own by using the following command:

\texttt{/course/cs1410/bin/test_heuristic/test_heuristic /path/to/your/search.py}

5 Install and Handin Instructions

To install, run \texttt{/course/cs1410/bin/cs1410_install Search} from within your
cs1410 directory.

To handin, run \texttt{/course/cs1410/bin/cs1410_handin Search} in the directory
with your \texttt{search.py} file.
In addition, please turn in both a signed collaboration policy and the written portion of the assignment to the CS1410 handin bin on the 2nd floor of the CIT. Since we cannot grade your work until you submit a signed collaboration policy, you should submit your signed collaboration policy by September 18 at 11:59pm, even if you plan to submit your writeup later.

In accordance with the course grading policy, your written homework should have just your banner ID on it and not your name or CS login. To further preserve anonymity, your signed collaboration policy and writeup should not be attached to each other in your handin.

Finally, please be sure to read the course grading policy before handing in.