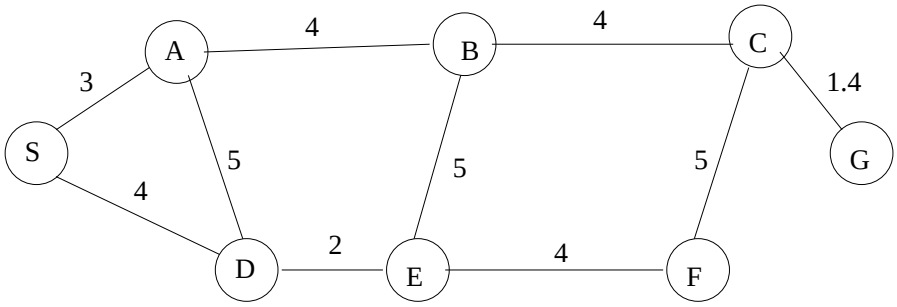
University of Pittsburgh
CS 2710 Foundations of Artificial Intelligence — Handout 3
Professor Milos Hauskrecht — September 11, 2019

Problem assignment 2

Due: Wednesday, September 18, 2019

Problem 1

Consider the following graph that represents road connections between different cities. The weights on links represent driving distanes between connected cities. Let S be the initial city and G the destination.



Part a. Show how the uniform search tree works by giving **the order in which nodes are expanded**. Is the path found by the algorithm optimal?

Part b. Assume the following set of the straight line distances between G and other cities.

S	A	B	C	D	E	F
10	10	6	1.4	9	7	2

Show how the greedy search algorithm with the straight-line distance heuritic works. Is the path the algorithm finds optimal?

Part c. Show how the A* with the straight-line distance heuristic works. Is the path found optimal?

Problem 2

The idea of the bidirectional search is to search both *forward* from the initial state and *backward* from the goal, and to stop when the node expanded in one direction has already been expanded in the other direction. The solution path is created by merging both paths to that node. Typically, bidirectional search is breadth-first,which is guaranteed to be complete, and is also optimal (assuming all edge costs are the same).

Suppose we try to enhance bidirectional search with an additional heuristic and apply A* to solve it. That is, from each direction in the search procedure we will use A* search with an admissible heuristic function to decide which node to expand first. We will have two admissible heuristic functions, one for each direction – one that under-estimates the cost from a node to the goal, and one that under-estimates the cost from a node to the initial state. We assume we can compute both the predecessors and successors of a node. For the sake of simplicity assume that all edges have the same cost.

Part a. Is bidirectional A* complete? If so, provide a proof of completeness. If not, explain why not or give a counterexample.

Part b. Is bidirectional A* optimal? If so, provide a proof of optimality. If not, explain why not or give a counterexample.

Problem 3. Search for the 8-puzzle problem.

In this problem we continue our exploration of search algorithms for the 8-puzzle problem. We will use the evaluation-function driven search procedure to incorporate various exploration strategies. The procedure searches the space by expanding the nodes with the minimum value first. You are given two files:

- *Puzzle*8.*py* which gives the definition of the Puzzle 8 problem, and TreeNode, HashTable, and Priority queue structures implemented as classes. Please note this file is slightly different from *Puzzle*8.*py* file you were given for HW-1 !!!
- $f\_driven_search.py$ which implements an evaluation function driven search algorithm. Briefly the procedure searches the space by expanding the nodes in the exploration fringe with the minimum f_value. These nodes are kept in the priority queue.
- heuristic.py that calculates the h function for the uniform cost search.

Part a. Uniform cost search

The *f_driven_search.py* code we gave you allows you to modify/update the evaluation function driven search as well as use your own heuristic function by importing a new definition of the $h\_function$. This function together with the g-value for the node (automatically calculated) define the f-value of the node. The file currently implements the uniform cost search where $h(n) = 0$ and hence $f(n) = g(n)$.

Remark: The uniform search algorithm for the puzzle-8 problem in fact implements the breadth-first search since all operator costs are one. The difference is that we simulate the breadth-first search through a more flexible evaluation-function representation and priority queue operations.

The *f_driven_search.py* currently does not calculate any search statistics similarly to the initial code you were initially given in HW-1. Please define a new version of the $eval\_function\_driven\_search(problem)$ such that it calculates the following stats:

- the total number of nodes generated
- the total number of nodes expanded
- the maximum length of the queue
- the length of the solution

Include the new function in file *main3a.py*. Run it on at least first three initial game configurations and report statistics.

Part b. Uniform cost search with elimination of state repeats

Modify the function $eval\_function\_driven\_search(problem)$ in the *main3a.py* file to include the check and elimination of all state repeats. Call the new function:
$eval\_function\_driven\_search\_repeats(problem)$ and include it in file *main3b.py*. Your program should be able to solve all 5 example configurations.

Part c. A* algorithm with the misplaced tile heuristic

Our next step is to implements the $A*$ search procedure with the misplaced tiles heuristic. In order to do so you will need to write a new $h\_function$ definition and import it to the *Puzzle8.py* file. Please write *heuristic1.py* file that implements $h_function$ using

the misplaced tile heuristic. Run *main*3*b.py* with *Puzzle*8*.py* importing the *h_function* from *heuristic*1*.py* instead of the current *heuristic.py*

The program should run on all five test examples and collect the same set of statistics as above.

Part d. A* algorithm with Manhattan distance heuristic

Similarly to Part c, write *heuristic*2*.py* that implements the Manhattan distance heuristic. Run *main*3*b.py* with *Puzzle*8*.py* importing the *h_function* from *heuristic*2*.py*.

Part e. Analysis of results

Analyze the performance of all methods (parts a through d) in terms of the collected statistics and include the analysis in the report. You should:

- Summarize the results of the methods in different tables, one table for every configuration tested: Uniform cost search, Uniform cost search with elimination of repeats, A^* with misplaced tile heuristic, $A*$ with Manhattan distance heuristics.
- Which method is the best in terms of the respective statistics? Explain why.
- State which heuristic would you suggest to use and explain why.

In addition, answer the following questions.

- Would A* work without state repeats elimination? Why or why not?
- Assume we create a heuristic function h_3 such that it averages the values of the misplaced tile heuristic (h_1) and the Manhattan distance heuristic (h_2):

  $$h_3(n) = \frac{1}{2}\left[h_1(n) + h_2(n)\right].$$

  Is h_3 an admissible heuristic?

Code to be submitted for Problem 3: Files *main*3*a.py*, *main*3*b.py*, *heuristic*1*.py*, *heuristic*2*.py* as specified above. Please note the TA for the course will run your code to check if the code is consistent with the reported results.