University of Pittsburgh
CS 2710 Foundations of Artificial Intelligence — Handout 2
Instructor: Dr. Patrick Skeba — January 27, 2025

Problem assignment 1

Due: Wednesday, February 5, 2025

Problem 1. Search Method Review

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

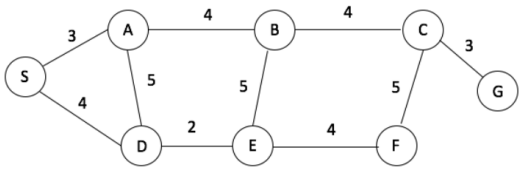

Figure 1:

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

Expanded	Priority Queue	Path Cost	Path
S	A 3, D 4	0	S
A	D 4, B 7	3	SA
D	E 6, B 7	4	SD
E	B 7, F 10	6	SDE
B	F 10, C 11	7	SAB
F	C 11	10	SDEF
C	G 14	11	SABC
G		14	SABCG

The path returned by the uniform cost search is the optimal distance for the path.

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

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

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

Current Node	Neighboring Cities	Path	Path Cost
S	A 10, D 9	S	0
D	A 10 E 6	SD	4
E	B 7 F 2	SDE	6
F	C 2	SDEF	10
C	G 0 B 7	SDEFC	15
G		SDEFCG	18

The Greedy Search Algorithm did not find the optimal path cost, its result was 4 longer than the uniform cost search.

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

Current Node	Neighboring Cities	Path	Path Cost
S	A 13, D 13	S	0
(tie, but choose A by decreasing h(n) priority) A	D 13, B 11	SA	3
B	D 13, E 11, C 6	SAB	7
C	D 13, E 11, G3, F 7	SABC	11
G		SABCG	14

A* found the optimal distance

Problem 2. Search for the Puzzle 8 problem.

In this problem we continue our exploration of search algorithms for the Puzzle 8 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 evaluation function value first. You are given three files:

- `Puzzle8.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 `Puzzle8.py` file you were given for homework assignment 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) defines the f-value of the node. The files given to you implement the uniform cost search where $h(n) = 0$ and hence $f(n) = g(n)$.

Remark: The uniform cost 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 homework assignment 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 `main2a.py`. Run it on at least first three initial game configurations and report statistics.

Problem	1	2	3
Nodes Generated	81	1382	5622
Nodes Expanded	30	501	1963
Maximum Length of Queue	53	883	3661
Solution Length	4	7	9

Part b. Uniform cost search with elimination of state repeats.

Modify the function `eval_function_driven_search(problem)` in the `main2a.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 `main2b.py.` Your program should be able to solve all 5 example configurations.

Problem	1	2	3	4	5
Nodes Generated	29	149	312	1170	47820
Nodes Expanded	17	88	174	724	31592
Maximum Length of Queue	14	66	140	448	16230
Solution Length	4	7	9	11	19

Part c. A algorithm with the misplaced tile heuristic*

Our next step is to implement 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 the `h_function` using the misplaced tile heuristic. Run `main2b.py.` with `Puzzle8.py` importing the `h_function` from `heuristic1.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.

Problem	1	2	3	4	5
Nodes Generated	9	10	16	47	2398
Nodes Expanded	5	7	10	28	1494
Maximum Length of Queue	6	5	8	21	906
Solution Length	4	7	9	11	19

Part d. A algorithm with the Manhattan distance heuristic*

Similarly to Part c, write heuristic2.py that implements the Manhattan distance heuristic. Run main2b.py with `Puzzle8.py` importing the h function from heuristic2.py.

Problem	1	2	3	4	5
Nodes Generated	9	16	19	32	600
Nodes Expanded	5	10	12	19	367
Maximum Length of Queue	6	8	9	15	235
Solution Length	4	7	9	11	19

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.

  Tables provided above.

- Which method is the best in terms of the respective statistics? Explain why.

  A* with the manhattan distance heuristic preformed the best overall, with slightly more nodes generated in problems 2 and 3. It did markedly better in the larger solution length problems, however, which suggests the heuristic is more efficient.

- State which heuristic you would suggest to use and explain why.

  I would choose to use A* with Manhattan search in situations where it is not too costly to calculate distance, if the puzzle got infinitely larger, each tile would need

to be searched for and that would take much longer than comparing immediate tile states.

In addition, answer the following questions.

- Would A* work without state repeats elimination? Why or why not?

  A* would work without state repeats because A* is complete even without state repeats. It is complete because it uses a heuristic to determine what the best path to a node is, so looping will not occur because a loop will never be the best path to a node.

- 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? You must demonstrate why or why not.

In order for a h3 statistic to be admissible, it must be less than the true h value. since h3 as calculated is always going to be in between two admissible statistics (h2 and h1), it will be less than one of them, and since that one is already admissible, and therefore less h3 will also be less and therefore admissible.

Code to be submitted for Problem 2:

1. `main2a.py`
2. `main2b.py`
3. `heuristic1.py`
4. `heuristic2.py`

Please note the TA for the course will run your code to check if the code is consistent with the reported results.