CS 1571: Project 1

Instructed by *Prof. Diane Litman*

Grader: *Ahmed Magooda*

Due on Oct. 2, 2017

Zac Yu (LDAP: zhy46@)

October 2, 2017

Problem 1

1 Wireless Sensor Monitoring Problem (`monitor`)

1.1 Problem Formulation

Intuitively, we will start with a empty state where no targets are monitored by sensors. For each step, we can assign a sensor to a target. Since the order doesn't matter, we can start from sensor 1 to sensor n, as long as we allow a sensor to not monitor any target or to monitor an monitored target if and only if there are more unassigned sensors than unmonitored targets.

A tricky part to begin with was to express the problem of maximizing the profit (duration when all targets are monitored) in terms of a problem of minimizing the cost. I ended up setting the cost to be the negative profit, and thus when the cost (i.e. negative profit) is minimized, the profit is maximized. Naturally, the step cost is chosen to be the negative duration some sensor-target pair can last.

Another issue I encountered when formulating this problem early on was that the path cost from one state to the next is not measured by the previous path cost + the step cost, but rather the maximum (or minimum of the absolute values) between the path cost and the step cost. I eventually made the path cost calculation a problem dependent function to allow such customization.

1.2 Heuristic Function

I chose the heuristic function to be the minimized potential cost when all unmonitored targets are monitored. Specifically, for each unmonitored target, we find the longest time it can last when monitored any of the unassigned sensors. Notice that we allowed various targets being monitored by the same sensor when calculating this maximized monitoring duration, since we don't want to overestimate the cost. Finally, we compare this maximized duration to the current path cost, if return the positive difference if the current step cost (current negative profit) is smaller than the negative maximized duration, which is the minimized potential cost, and therefore the heuristic function is admissible. The evaluation function for A^* is also consistent since the profit is non-increasing (and hence the cost is nondecreasing).

1.3 Performance

`test_monitor1.config`	Breadth First	Iterative Deepening	Uniform Cost	Greedy	A^*
Time	11	19	7	12	7
Frontier	6	N/A	4	5	4
Visited	5	N/A	3	7	3
Cost	88.0	88.0	88.0	35.36	88.0

Problem 1 continued on next page...

For the first input file, since the size is small, all searching algorithms finished fairly quickly and all except for the greedy algorithm yielded the optimal solution. We notice that the uniform-cost search and A^* took the same amount of time to complete.

`test_monitor2.config`	Breadth First	Iterative Deepening	Uniform Cost	Greedy	A^*
Time	8236	10715	1589	94	219
Frontier	6	N/A	1138	65	162
Visited	5	N/A	451	29	57
Cost	11.42	11.42	13.65	10.11	13.65

For the second input file, which is moderately larger in size, the breadth-first search and iterative-deepening search took considerable amount of time. The greedy algorithm finished very quickly but yielded a suboptimal result. However, the heuristic function is proven to be effective when comparing the time and space cost between those of the uniform-cost search and of the A^* search. While both were successful at finding the optimal solution, the A^* search, thanks to the heuristic function, only takes about $\frac{1}{8}$ of the space and time. We also observed that the space and time complexities generally matches our expectation based on our class discussions.

Problem 2

2 Data Aggregation Problem (`aggregation`)

2.1 Problem Formulation

This formulation of the data aggregation problem is more straightforward. We start with an initial state of empty path, and then pick any of the node to go next with a delay of 0. What's next is just standard graph search of traveling among nodes.

Unlike the traveling salesman problem, the graph is not guaranteed to be complete and we might need to visit a node multiple times before we finish visiting all nodes. During my initial experiments, I actually discovered that for input file `test_aggregation2.config`, when having the rule that no node shall be visited for more than one time, it is impossible to visit all nodes.

Another thing worth mentioning is that we are provided with coordinates of the location of each node as part of the input. This geometric distance between nodes, however, does not serve as a candidate reflection of the delay to go from one to the other. Nor is this distance used in any way to calculate the cost. After some experiments, I decided to disregard this piece of data.

2.2 Heuristic Function

I used the weight of the minimum spanning tree of all unvisited nodes (calculated with Kruskal's algorithm in $O(E\log(E+V))$ time) as the heuristic function to ensure that it is consistant in the sense that the evaluation function of path cost + heuristic (i.e. that for A^*) along any path are nondecreasing and that it does not overestimate the cost. Therefore, the heuristic function is admissible and consistent and should always lead to the most optimal solution with A^*.

2.3 Performance

`test_aggregation1.config`	Breadth First	Iterative Deepening	Uniform Cost	Greedy	A^*
Time	12	20	20	12	13
Frontier	6	N/A	10	6	7
Visited	5	N/A	10	6	6
Cost	15	15	9	25	9

Problem 2 continued on next page...

The size of the first input file for the aggregation problem is even smaller. All searching algorithms finished fairly quickly and both uniform-cost search and A^* search yielded the optimal solution as expected.

`test_aggregation2.config`	Breadth First	Iterative Deepening	Uniform Cost	Greedy	A^*
Time	70593+	149231182+	103619+	518	102272+
Frontier	59238+	N/A	85916+	411	82987+
Visited	11351+	N/A	17699+	107	19282+
Cost	timed out	timed out	timed out	148	timed out

The search space for the second input file, with a total of 45 nodes, is gigantic. Out of all search algorithms, only the greedy algorithm was able to finish before the 30-minute deadline, with a likely suboptimal cost of 148. However, we can fairly confident that given longer time, A^* will be able to find the most optimal solution a lot faster than the uniform-cost search, as indicated by the performance data on the first input file. We can potentially derive a better heuristic function for this problem that takes into account that each node can be visited for multiple times since the MST weight heuristic is more suitable for a problem with the visit-each-node-only-once rule. I would prefer using A^* search for this problem.

`test_aggregation3.config`	Breadth First	Iterative Deepening	Uniform Cost	Greedy	A^*
Time	25229	35685	30996	207	5716
Frontier	17715	N/A	21416	130	3918
Visited	7513	N/A	9580	77	1798
Cost	25	25	22	54	22

The result for the third input file again confirms that the heuristic function is consistent and optimal, yielding the optimal solution with an A^* search.

Problem 3

3 The Burnt Pancake Problem (`pancakes`)

3.1 Problem Formulation

This formulation of the burnt pancake problem is straightforward. For each state, we can simply use a tuple to represent the order, similar to the input format. Actions are to flip $1, 2, \ldots, n$ pancakes from the top as long as the resulting tuple has not been seen before.

3.2 Heuristic Function

Since we want to turn the into pancake into increasing order as much as possible and that connected pancakes can be easily moved with two flip operations, we can count the breakpoint, defined as an occurrence of two adjacent pancakes in the order tuple not differ by 1. Note that both $-4, -3, -2$ and $2, 3, 4$ are favored instances and both have adjacent element all differ by 1 (ordered, right element − left element). This method is inspired by paper [1]. Note that the heuristic function is not guaranteed to be both consistent and admissible since as mentioned above, flipping negative continues adjacent pancakes won't increase breakpoints. The know that A^* will then always lead to the optimal solution.
Note that the actual heuristic function used for this problem is the number of breaking points − path cost, to ensure the optimality of A^*.

Problem 3 continued on next page...

3.3 Performance

`test_pancakes1.config`	Breadth First	Iterative Deepening	Uniform Cost	Greedy	A^*
Time	61523+	231167229+	92293+	314	62576+
Frontier	55363+	N/A	83056+	283	61252+
Visited	6151+	N/A	9228+	31	1276+
Cost	timed out	timed out	timed out	22	timed out

The size of the first input file for the aggregation problem is still relatively small, consists of 11 elements. But we observe that the search space is already to large enough for any uninformed search to complete. I am also disappointed that the A^* algorithm couldn't finish. The result from the greedy algorithm, however, is promising.

`test_pancakes2.config`	Breadth First	Iterative Deepening	Uniform Cost	Greedy	A^*
Time	62576+	202338475+	101874+	3678	132499+
Frontier	61242+	N/A	99748+	3603	129748+
Visited	1276+	N/A	2078+	75	2703+
Cost	timed out	timed out	timed out	68	timed out

As expected, the greedy search found a feasible solution really quickly. Unfortunately, the search space of this problem is simply too large for A^* to find the optimal solution within the deadline of 30 minutes. At the same time, we observe that the significant growth of time and space cost when a comparatively small increase of the size of input.

`test_pancakes3.config`	Breadth First	Iterative Deepening	Uniform Cost	Greedy	A^*
Time	78238+	1815758	121828+	120	6563
Frontier	62588+	N/A	97460+	97	5252
Visited	15646+	N/A	24364+	23	1311
Cost	timed out	9	timed out	14	9

I am very happy to see that the A^* search work with a larger size of input. Since the step cost of this problem is 1, the iterative deepening search was also successful at finding the optimal solution. However, we notice that the A^* search found the optimal solution much with much shorter time compared to the iterative deepening search.

References

[1] Bruno Bouzy. An experimental investigation on the pancake problem. In *Workshop on Computer Games*, pages 30–43. Springer, 2015.