**James Hahn**

**CS1571 – Artificial Intelligence**

**Dr. Diane Litman**

**M/W 11am-12:15pm**

**Homework #1**

**Heuristic functions**

Data Aggregation –

* *unicost -* choose the edge with the smallest time latency next
* *greedy –* continue on the path with the smallest most-recently-added edge
* *A\* -* continue on the path with least sum of [path cost by sum of edge latencies] and [most-recently-added edge weight]

Monitoring –

* *unicost* – go to the state with the “highest” cost. In other words, go to the state that currently monitors its targets for the longest amount of time
* *greedy* - go to the state with the “highest” cost. In other words, go to the state that currently monitors its targets for the longest amount of time
* *A\** - go to the state with the “highest” cost. In other words, go to the state that currently monitors its targets for the longest amount of time

Pancakes –

* *unicost* – since we have no information about the state and we just move one step along the path at a time and nothing is weighted, uniform-cost search on this problem is exactly the same as BFS
* *greedy* – continue with the state that matches in similarity to the goal state as closely as possible (refer to **note 1**)
* *A\** - go to the state with the lowest sum of [differences between it and the goal state] and [number of steps, “flips”, taken so far] (refer to **note 1**)

**Note 1 –** In the pancakes problem, we use the “similarity to the goal state” as the heuristic. This solution is unique to this problem and essentially what it does is scan through the current state of the pancakes, including its negatives, and does two things. The difference (from it and the goal state) starts as an integer at value 0. If the current pancake is negative, we add 1 to the difference. If the current pancake is not in the current index in the list, we add 1 to the difference. Because of this, any given pancake can add up to 2 points to the total difference variable. This means if all the pancakes were burnt-side up (negative) and none were in their correct order, the “difference” value would be 2N where N is the number of pancakes; this indicates that it is nowhere close to the goal state. Conversely, if the “difference” value was 0, we have reached the goal state.

**Did outcomes make sense?**

Yes. As far as each algorithm goes, *bfs* creates the most nodes by far, as expected, because it is an exhaustive search essentially. Depending on the problem, *iddfs* creates a much smaller frontier than *bfs* and the others because we’re doing a DFS down the tree; if the branching factor at each level is small, even if the solution path is long, the frontier size will be very minimal. Another point is *unicost* tends to produce optimal solutions more often than both *bfs* and *iddfs*, which is great news. Also, depending on the problem we’re searching, *greedy* may return the optimal solution or may not; it definitely performs poorly with test\_pancakes3.config, returning a result just as bad as *iddfs*. The *greedy* algorithm performs quicker than all the other algorithms when utilizing test\_pancakes3.config; this makes sense because we’re just trying to make it through the search space to the goal state as quickly as possible, attempting to travel along the “quickest” route there. Unfortunately, it performs poorly in terms of achieving the optimal solution.

In the Pancakes problem, *unicost* returns the same results as *bfs*. This makes complete sense because a “flip” has no weight to it; every flip performed essentially has weight 1. If you refer to the textbook and slides, *unicost* performs the same as *bfs* if this very fact is true. So because of this, this result makes perfect sense.

Also for the Pancakes problem, when ran on test\_pancakes3.config with *A\**, it does not produce the optimal solution. This is fine because our textbook states when forming a heuristic that should be optimal: “The next step is to prove that whenever A\* selects a node n for expansion, the optimal path to that node has been found.” This piece of information directly indicates that *A\** cannot be optimal for the Pancakes problem because given any configuration of pancakes in a stack, although we can measure similarity to the goal state, we will never truly know exactly how many flips we are from the goal state. Think back to our classroom examples with the cities and distances to the goal state; we had that information, which helped ensure that our informed search, *A\**, would always move closer to the goal state. That was an easy example because we knew our precise distance from the goal. However, in the Pancakes problem, we can measure similarity to the goal state, but we cannot tell the exact number of flips, or distance in this case, from the goal state. Because of this, it makes sense that *A\** does not produce optimal results for the Pancakes problem for some configuration files.

One thing to note is that the *A\** algorithm on the Pancakes problem takes quite a bit longer than I would’ve liked. I think this can be explained by the fact that even if we believe we’re moving closer to the goal state and reducing the differences between our current state pancake stack and the goal state pancake stack, that flip may in turn only allow us to make worse flips from there, which actually move us further away from the goal state. Take for example the goal state (1, 2, 3, 4, 5) and the current states **t** (1, 2, 3, 4, -5) and **u** (-2, -1, 3, 4, 5). With the similarity metric I provided earlier, **t** has a similarity of 1, indicating it’s close the goal state. Meanwhile, **u** has a similarity of 4. However, **u** requires one flip (between the -1 and 3) to reach the goal state and **t** requires many more than that because we need to flip the -5, or bottom pancake. As we can see, in this scenario, *A\** would choose to go to **t**, but we’d actually be moving further away from the goal state.

All of the algorithms are complete, as expected. The only algorithm I wasn’t expecting to necessarily be complete was *greedy*, simply because of the Pancakes problem. You can technically just go in a circular pattern with what seems to be infinite flips. With *greedy* just clinging onto what seems to be the path closest to the goal state, we can actually end up going further away from the goal state in the long run as mentioned above. It’s complete on the test\_pancakes3.config, but it takes too long to measure completeness even on input with fifteen pancakes.

\*\*\*DISCUSS SPACE COMPLEXITIES HERE\*\*\*

**Anything surprising?**

Nothing was extraordinarily surprising. One thing that was a bummer in this project is the branching factor of any given problem for the search space. If the Pancakes problem has input of twenty pancakes, there are twenty different positions that you can flip the pancakes for any given configuration. This means the branching factor is already size twenty and this doesn’t become smaller as we scan the search space. So, in reality, the worst-case runtime for the Pancakes problem with twenty pancakes is 20^n where n is the level of any given path.

Overall, the runtime was the worst thing to deal with in this project and I hope we’re not judged too harshly on how our project performs in terms of runtime because right now mine can only realistically run on input of smaller sizes because the branching factor of the search space becomes ridiculous and searching through every node is unfeasible otherwise.

\*\*\*DISCUSS DIFFICULTY TO FIND HEURISTIC FUNCTIONS HERE\*\*\*