# Assignment – 1

Foundation of Artificial Intelligence – CS2701

Name: Khushboo Thaker

ID: 4147510

Email: kmt81@pitt.edu

## 1. What heuristic function did you use for each puzzle class?

**Water Jug Problem:**

For Water Jug Problem I used a heuristic which counts the steps to reach the Greatest Common Divisor (GCD) between two numbers. The intuition is, altleast required steps to reach goal are more than GCD. Below is the mathematical representation of the heuristic

$$if\ (jug1, jug2) == startnode, \qquad \boldsymbol{h_f} = \text{capacity}_{\text{j1}} + capacity_{j2}$$

$$if\ (jug1, jug2) == goalNode, \qquad h_f = 0$$

$$if\ jug1(jug2) == goalNode, \qquad h_f = 1 + I(jug2(jug1))$$

$$if\ jug1, jug2\ != 0\ and\ jug1, jug2\ != goal,$$

$$\boldsymbol{h_f} = count(euclideanstepgcd(jug1, jug2))$$

$$if\ jug1, jug2\ != goal\ \ and\ one\ of\ jug1\ or\ jug2\ is\ empty,$$

$$\boldsymbol{h_f} = count(euclideanstepgcd(capacity_{emptyjug}, value_{nonemptyjug}))$$

*Table 1 jugs.config goal (0,2) start(0,0) and max capacity (3,4)*

| | | | |
|---|---|---|---|
| Algorithm | Euclidean | Dot product | GCD step count |
| Greedy | 7 | 6 | 6 |
| Astar | **6** | **6** | **6** |
| IDAstar | 7 | 10 | 6 |

Thus it can be seen that GCD based heuristic gives the optimal output in all cases. Although the steps and Euclidean distance differs significantly it can be seen that Euclidean also gives a constant performance. But as the problem size increases Euclidean fails and GCD still remains good heuristic. Also if heuristic is not proper as in case of dot-product it can be seen a major drawback As for example dot-product gives path cost : 42 in against to GCD which gives path cost: 6 in (15,24) – (0,0) and (0,6) water jug problem. Also uniform search gives the optimal solution even though no heuristics applied

**Path Puzzle - refer** Table 2 – test_cities.config

For path puzzle I tried Euclidean, Dot-product and Manhattan distance. I found that the three heuristics perform almost same. The results are more dependent on algorithms you use. But again dot product heuristic fails here as well because it is not admissible and hence rather than picking up the optimum path of 17 (cut-off value 6) it picks up path with cost 49 (at cut-off value 32) for test_cities.config

**Pancake Puzzle**

Heuristics play a very important role in solving pancake problem as search space for this problem increases exponentially with increase in the n value (n = number of pancakes)

Heuristics used:

1. Distance of unaligned pancakes from their actual position
2. Gap Heuristic[1]
3. No of pancakes placed upside down
4. No of adjacent opposite sided pancakes + Gap Heuristic

From all the heuristics above Gap Heuristic performed the best. But still it works better for pancake sort problem but need a little modification for flip

Pancake Results: For problem – (-1, -3, -4, -5, -2)

| | - | | | Gap Heuristic + no of upside down pancakes (pancake) | | | Flips – no of upside down pancakes (flips) | | | Euclidean (euclidean) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **T(n)** | **S(n)** | **C** | **T(n)** | **S(n)** | **C** | **T(n)** | **S(n)** | **C** | **T(n)** | **S(n)** | **C** |
| Uniform | 3839 | 1376 | **8** | | | | | | | | | |
| Greedy | - | - | - | 3839 | 1336 | 15 | 3839 | 1500 | 16 | 3839 | 1279 | 14 |
| Astar | - | - | - | 3839 | 1329 | 9 | 3839 | 1483 | 9 | 3839 | 1336 | 11 |
| Idastar | - | - | - | **1221** | **1308** | 13 | 2670 | 1483 | **9** | **356** | **673** | **22** |

| | Gap Heuristic + adjacent opposite facing pancakes (pancakes) | | | Dot product (dotproduct) | | |
|---|---|---|---|---|---|---|
| | **T(n)** | **S(n)** | **C** | **T(n)** | **S(n)** | **C** |
| Greedy | 3839 | 1336 | 20 | 3839 | 1570 | 17 |
| Astar | 3839 | 1345 | 8 | 3839 | 1699 | **10** |
| Idastar | **128** | **47** | **11** | 24448 | 1674 | **63** |

Pancakes performance is very surprising. As can be seen gap heuristic is **good enough** (not admissible as greedy fails to get optimal solution) as it gets us to the optimal solution with less space and time complexity. Surprising fact about pancake problem is idastar is faster than all of the above problems with time complexity 128 and space complexity 47 (stack size) with reasonable path cost 11. The fast result obtained by Euclidean heuristics depends on the problem and order of child-nodes generated.

[1] Landmark Heuristics for Water Jug problem ( by Helmert)

IDAStar performs the best as it doesn't gets in to too deep of graph and finds the optimal solution fast. Also as discussed in few papers the worst case bound of this problem is about 15n/3 and 2n (not for burnt pancake version) so IDAStar is ideal for such problems

---

## 2. Did all the outcomes make sense (e.g., do the time/space complexities of different search strategies match your expectation based on our class discussions? What about optimality and completeness?)

Yes as per discussion we can see in the below tables that time and space complexity of algorithms matches to the required condition

*Table 2 – test_cities.config output*

| Algorithm | Space Complexity | | Time Complexity | | Optimal Path Found |
|---|---|---|---|---|---|
| | Frontier queue | Explored | | | |
| BFS | 30 | 24 | 24 | | |
| DFS | 53 | **Stores parents for recursive check = depth of node – which is much smaller than explored list | 22 | | |
| IDDFS | 27 | 24 | 54 | Depth iterated 6 | |
| Uniform | 10 | ** | 24 | | |
| Greedy | 24 | ** | 9 | not optimal | |
| Astar | 24 | ** | 11 | | |
| IDAstar | 25 | ** | 11 | Cutt off reached 5.65 | |

As can be shown in table below observation can be made as discussed also in class:

1. BFS uses a maximum storage space
2. Loop formation due to cycles in a graph is a major problem in DFS as it doesnot store visited list but the problem gets sorted in case of informed search if the heuristic is admissible
3. Greedy finds the solution in fastest time – but can get stuck to local optimum
4. Astar is good if the heuristic is admissible and solution lies in depth of the graph (water jug) while IDAstar is good when solution lies in upper graph (pancake)

Completeness:

Pancake problem is a good example to check the completeness of different algorithms.

1. BFS is slow but finds the solution for pancake problem – but not optimal when cost of different path is different as it will iterate to all the nodes

2. As n increases DFS becomes very slow and goes for searching depth whereas solution is above
3. IDFS just like BFS is complete but optimal if constant cost
4. Uniform cost always gives optimal solution and is complete
5. Greedy, Astar and IDAstar depend on heuristic functions to be admissible and optimistic, if these criteria are satisfied give the fast and optimal result

**Surprise:**

IDAstar algorithm which looks to be quiet resource full is very good for problems where upper bound (depth of solution) is known – pan cakes problem

After implementation only I understood the exponential growth in space requirement in case of BFS (visitedList)

Surprisingly Euclidean heuristic almost works out for every problem though not admissible and even optimistic

My pancake solution is very slow and that came as surprise to me. As it only works for n <= 9 other than that takes more than 1 hour of time. Though idastar is fastest amongst if heuristics are properly defined – works well with heuristic pancakes (gap + adjacent opposite sides of pan cakes)

*Table 3 Mapping problems to Algorithm*

| | |
|---|---|
| Water Jug | Astar |
| Path Finder | Greedy if local optimal is good enough, IDAStar |
| Pancakes | IDAstar |

References:

Class Notes

https://aakritty.wordpress.com/2014/02/10/solving-the-water-jug-problem/

Helmert, Malte. "Landmark heuristics for the pancake problem." Third Annual Symposium on Combinatorial Search. 2010

https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm

https://en.wikipedia.org/wiki/Pancake_sorting