Assignment-02: The Informed Search

Zain Abbas, CS-131, 06/18/25

Abstract

To efficiently order disordered pancakes, we need to design an informed search algorithm such as A* that utilizes both cost function *g(s)* and heuristic function *h(s)* to optimize the search. We are provided with a special heuristic function called *gap heuristic function.* The *gap* heuristic function is admissible and consistent. It means that it never overestimates the cost to reach the goal. In addition, we also compared the efficiency of A* algorithm with UCS and Chebyshev heuristics and obtained feasible results to make use analysis.

Introduction

The pancake problem is a famous search problem that serves as benchmark for search algorithms. The objective is to sort a sequence of pancakes through a minimal number of flips. We are given a stack of 10 pancakes each of different sizes. The goal of the cook is to have them correct order, that is, the large on the bottom up to smallest on top. Successor states are obtained by flipping $k \in \{2, \ldots .10\}$ pancakes at the top of the stack i.e. by reversing the order of the first k sequence elements. We will use the A* algorithm to optimize the search and compare the results with UCS algorithm. In the extra experiment, we will compare the efficiency of the A* algorithm with the Chebyshev heuristic function by comparing the total time and nodes explored to order the pancakes. This classical problem provides a controlled environment for testing and analyzing search algorithms and heuristics. It connects real-world problems such as robot arm sorting parts.

Methodology

I. A algorithm*

To effectively order pancakes in descending order we will implement A* search algorithm that is based on a cost function *g(s)* and gap heuristic function *h(s)*.

$$f(n) = g(s) + h(s)$$

A* algorithm is the best first search algorithm that finds the shortest path from start node to finish in a search space. It combines features of both GBFS (Greedy Best First Search) and UCS (Uninform Cost Search), making it optimal (finds the least-cost path) to reach goal. The GBFS part focuses on forward search by using heuristic function that optimizes the distance of the state to the goal, whereas UCS focuses on backward search by using cost function of the total path.

II. Defining a Search algorithm

a. Initial state: Let s = $\{s_1 \ldots . s_{n+1}\}$ be an n-pancake state. We used Python *random* package to generate 10 uniquely sized pancakes in random order. Figure 1 shows pancake initial and goal state for 5 pancakes.

b. Actions: Choose a position k where 2 <= k <=10 and flip the top n pancakes.

c. Transition model: Given a current state, the result of applying action flip (k) is a new state, where the top k pancakes are reversed. The successor states are obtained by flipping $k \in \{2, \ldots .10\}$. An optimal solution is given by minimum flips. For example, figure 1 shows there are 8 gaps in the pancake stack, primarily below positions 2, 3, 4, 5, 6. There is a gap between 2nd and 3rd pancakes because their sizes differ greater than 1

d. Goal: Pancakes are sorted in descending order from top to bottom.

e. Path to Cost: Each flip costs 1, so that total cost is number of flips.

III. Backward cost

Backward cost is the total cost accumulated from the goal to initial state. Since each flip has unit = 1. The cost function is represented by
g(n), where g(n) = number of flips from goal to state n

IV. *Forward Cost*

Forward cost is the heuristic gap function. The heuristic value is the number of stack positions for which the pancake at that position is not of adjacent size to the pancake below:

$$h_{Gap}(s) = \ |\{i \mid i \in \{1, \ldots . n\}, \mid s_i - s_{i+1} \mid > 1\}\mid$$

In our scenario we have $h_{Gap}(s) = 8$ for the state. A heuristic value of 0 is achieved when there are no gaps at all.

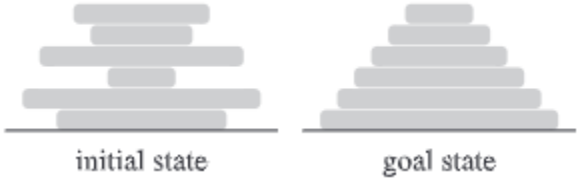

Figure 1: The above diagram shows an example of 5 pancakes in initial and goal state

Demo 1: A* algorithm

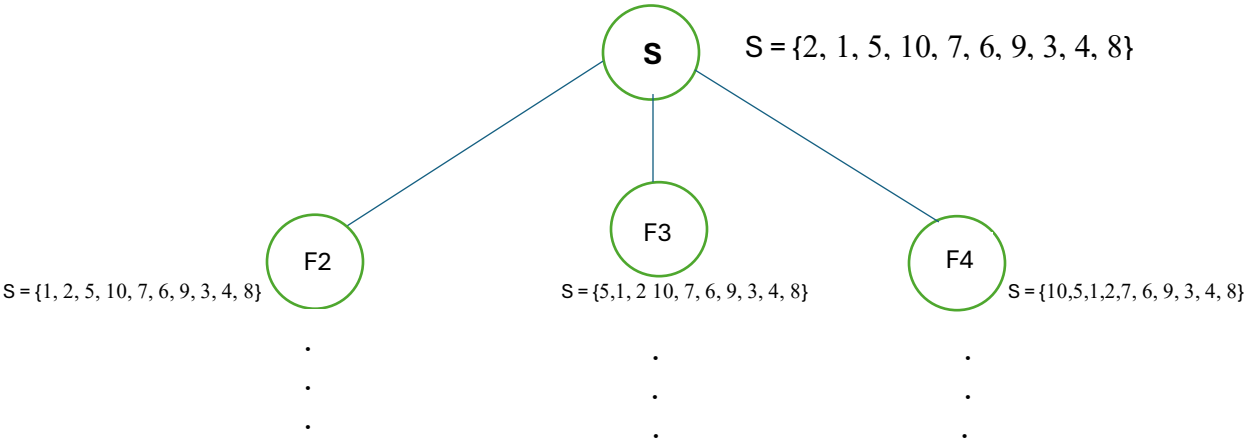

Figure 2: The above diagram shows starting node and parent nodes corresponding to k flips.

Initial state: s = {2, 1, 5, 10, 7, 6, 9, 3, 4, 8}
Backward cost g(s) = 0
Forward cost:
$h_1 = 0, h_2 = 1, h_3 = 1, h_4 = 1, h_5 = 0, h_6 = 1, h_7 = 1, h_8 = 1$
$h_{gap}(s) = h_1 + h_2 + h_3 + h_4 + h_5 + h_6 + h_7 + h_8 = 6$
path = [ ] # no flips made yet
$f_{initial} = g + h = 6$

loop from k = 2 to 10

k = 2
new_state = flip ([2, 1, 5, 10, 7, 6, 9, 3, 4, 8]; 2) → (1, 2, 5, 10, 7, 6, 9, 3, 4, 8)
new_path = path + [2] → [2] # number of pancakes flipped
$g_{new} = g + 1 = 1$
$h_{new} = h_{gap}$ (1, 2, 5, 10, 7, 6, 9, 3, 4, 8)
$h_{new} = h_1 = 0, h_2 = 1, h_3 = 1, h_4 = 1, h_5 = 0, h_6 = 1, h_7 = 0, h_8 = 1$
$h_{new} = 6$
$f_{new} = g + h = 7$

frontier.append({
state: [1, 2, 5, 10, 7, 6, 9, 3, 4, 8]
path: [2]
g(s) = 1
f = 7)

we can continue form k = 2….10, until we reach our goal. There is a check function that compares the current state to the goal state. If the current state matches with the goal state, the loop stops and prints the result.

UCS algorithm:

In UCS algorithm h = 0 and g = 1 for every flip.
So, the cost function is:

$$f = h + g = g \qquad \text{as } h = 0$$

UCs always expands the node with the lowest number of flips

Initial State: S = {2, 1, 5, 10, 7, 6, 9, 3, 4, 8, g = 1

K – flips	Flip(0; k)	g	f = g
2	[1, 2, 5, 10, 7, 6, 9, 3, 4, 8]	1	1
3	[5, 1, 2, 10, 7, 6, 9, 3, 4, 8]	1	1
4	[10, 5, 1, 2, 7, 6, 9, 3, 4, 8]	1	1
5	[7, 10, 5, 1, 2, 6, 9, 3, 4, 8]	1	1
6	[6, 7, 10, 5, 1, 2, 9, 3, 4, 8]	1	1
7	[9, 6, 7, 10, 5, 1, 2, 3, 4, 8]	1	1
8	[3, 9, 6, 7, 10, 5, 1, 2, 4, 8]	1	1
9	[4, 3, 9, 6, 7, 10, 5, 1, 2, 8]	1	1
10	[8, 4, 3, 9, 6, 7, 10, 5, 1, 2]	1	1

Table1: The above table shows node expansion for g = 1 only.

Each of these states would be added to the frontier with g = 1, h = 0 and f = 1. Since all the nodes have uniform cost, the UCS will pop one of the nodes arbitrarily, expand it and check if it's a goal state. UCS will eventually reach the goal state because it guarantees optimal solution, but it is slower than A* due to missing heuristic function.
After the completion of this loop, the UCS will explore g = 2 for flips k = 2….10

Results

To compare the performance of A* and UCS algorithm, we ensured that the random.seed(42) is fixed so that we can compare the same random sequence for both algorithms.

A algorithm* *UCS algorithm*

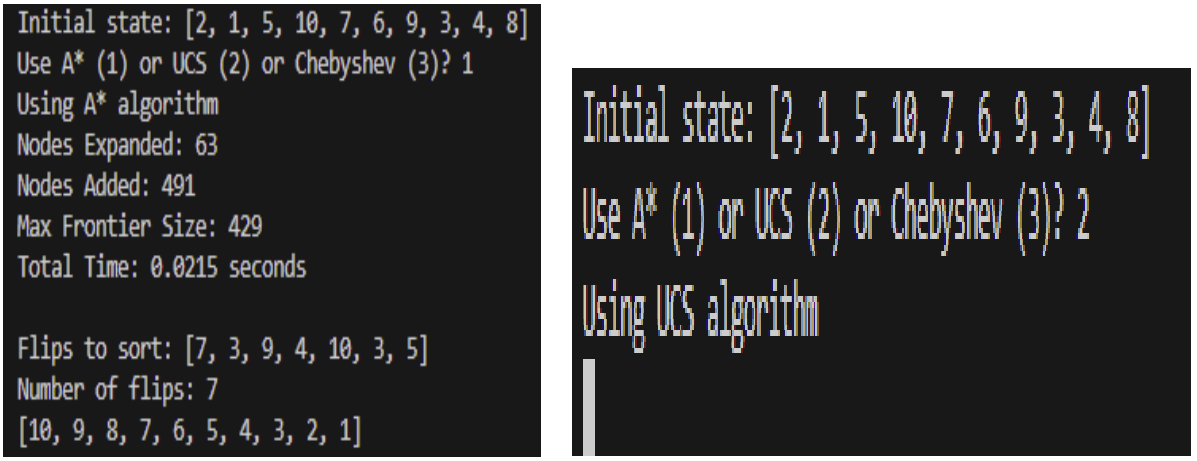

Figure 2: The above diagram compares A algorithm and UCS algorithm performance. The UCS algorithm stuck in an infinite loop whereas A* completes the search in an optimal way.*

A* algorithm converges very smoothly with a total time of 0.0215 seconds. We expanded to 63 nodes, added 491 nodes and the max frontier size was 429. Since the h_{gap} is admissible and consistent, the heuristic function guides the search toward more promising states. As a result, A* search algorithm expands fewer states and therefore effective. However, the UCS algorithm stuck in a loop forever. In UCS the search algorithm treats all unexplored nodes equally, expanding them in increasing order of path length. If the goal state is deep in search tree, UCS may explore hundreds of nodes before even getting close. Therefore, it opens node uniformly until it reaches the goal.

Extra experiment:

In the extra experiment, we updated the A* heuristic with Chebyshev algorithm.

It states:

$$h_{chebyshev}(state) = max_{0 \le i \le n} \left| i - pos_{goal[state[i]]} \right|$$

For each pancake in the current state **s**, it computes how far its current position **i** from its goal. For example, state = [2, 1, 5, 10, 7, 6, 9, 3, 4, 8], goal = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]. Pancake 2 is at index 0, but its goal is index 8, therefore absolute distance = $|0 - 8| = 8$. Thus, the Chebyshev heuristic seeks maximum distance for that state.

Chebyshev heuristic:

Pancake	Current Index	Goal Index	Distance
2	0	8	8
1	1	9	8 (**maximum)**
5	2	5	3
10	3	0	3
7	4	3	1
6	5	4	1
9	6	1	5
3	7	7	0
4	8	6	2
8	9	2	7

Updated A algorithm:*

Node	Description	k-flip	g(flips)	h (Chebyshev)	f = g + h
S	Initial State	2	0	8	8 **(explore first)**
A	After 1 flip	10	1	8	9
B	After 1 flip	3	1	3	4
C	After 1 flip	8	1	3	4
D	After 1flip	6	1	1	2
E	After 1 flip	5	1	1	2
F	After 1 flip	3	1	5	6
G	After 1 flip	5	1	0	**1 (goal)**

Python Output:

A* **(***Gap heuristic***)**

```
Initial state: [2, 1, 5, 10, 7, 6, 9, 3, 4, 8]
Use A* (1) or UCS (2) or Chebyshev (3)? 1
Using A* algorithm
Nodes Expanded: 63
Nodes Added: 491
Max Frontier Size: 429
Total Time: 0.0215 seconds

Flips to sort: [7, 3, 9, 4, 10, 3, 5]
Number of flips: 7
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

A* (*Chebyshev heuristic)*

```
Initial state: [2, 1, 5, 10, 7, 6, 9, 3, 4, 8]
Use A* (1) or UCS (2) or Chebyshev (3)? 3
Using Chebyshev algorithm
Nodes Expanded: 7767
Nodes Added: 52852
Max Frontier Size: 45086
Total Time: 116.7603 seconds

Flips to sort: [2, 10, 3, 8, 6, 5, 3, 5, 3]
Number of flips: 9
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Figure 3: The above diagram shows the performance of A (h_{GAP}) and A* ($h_{Chebyshev}$)*

The *Chebyshev* algorithm converges after expanding to 7767 nodes, adding 52852 nodes, with a max frontier size of 45086. It took total time about 116.7603 seconds. Whereas, gap heuristics expanded to 63 nodes, added 491 nodes, with max frontier size of 429 and total time about 0.0215 seconds

Based on the above results, it seems that Chebyshev focuses on the worst pancake (max distance), not on the overall order. Therefore, it might steer A* down a path that appears promising (based on one bad pancake) but isn't close to the goal. On the other hand, gap heuristics expands fewer nodes but generates more children because it aggressively pruned fewer options. It mainly focuses on the adjacent pairs, while Chebyshev overreacts to single pancake far from goal.

Conclusion

To optimize the search algorithm for sorting 10 disordered pancakes, we tested the A* algorithm using h_{gap} heuristics. This approach successfully ordered the pancakes in less time, demonstrating a strong balance between performance and accuracy. In addition, we compared the A* algorithm with the UCS results. We found that UCS gets stuck in an infinite loop or takes impractical amount of time, as it expands nodes blindly without any heuristic guidance. This makes UCS un-suitable for pancake sorting unless further optimized. Next, we experimented with the A* algorithm using Chebyshev heuristic, which attempts to sort the stack by focusing on the pancake furthest from its correct position. While this heuristic is more informed and theoretically accurate, it often causes algorithms to over-explore fewer promising paths due to its overemphasis on a single misplaced element. As a result, Chebyshev heuristic led to significantly more node expansions and a longer runtime compared to the gap heuristic.

In conclusion, A* with gap heuristic proved to be the most efficient and practical approach for pancake sorting. It provided, fastest results, expanded fewer nodes and avoided unnecessary computational overhead, making it the best choice among the tested strategies.