HSEF Assignment 3

colbyziyuwang

October 2024

1 Part 1

We first make a table to record these 10 easiest problems in terms of number of node expansions:

run_id	num_get_actions_calls
4	116
56	171
38	281
24	577
72	1010
11	1382
65	1507
83	1683
3	2161
93	2205

Table 1: Top 10 Easiest Problems Based on Number of Get Actions Calls

My hypothesis is that without parent pruning, the runtime of IDA* will increase by a factor of 16. In sliding tile puzzles, each state has 4 possible actions, and if each action can be reversed, the number of generated nodes will double. Consequently, the total runtime should increase by $2^4 = 16$ times.

We now show a table recording the results without parent pruning:

run_id	num_get_actions_calls
4	157
56	485
38	1348
24	4454
72	10038
11	21311
65	13428
83	5343
3	32043
93	10784

Table 2: Run IDs and Number of Actions Generated

Below is the scatter plot that compares the number of nodes expanded with and without parent pruning. The x-axis represents the number of nodes expanded with parent pruning, and the y-axis represents the number of nodes expanded without parent pruning. The red dashed line shows the y=x reference, indicating where both approaches would have expanded the same number of nodes.

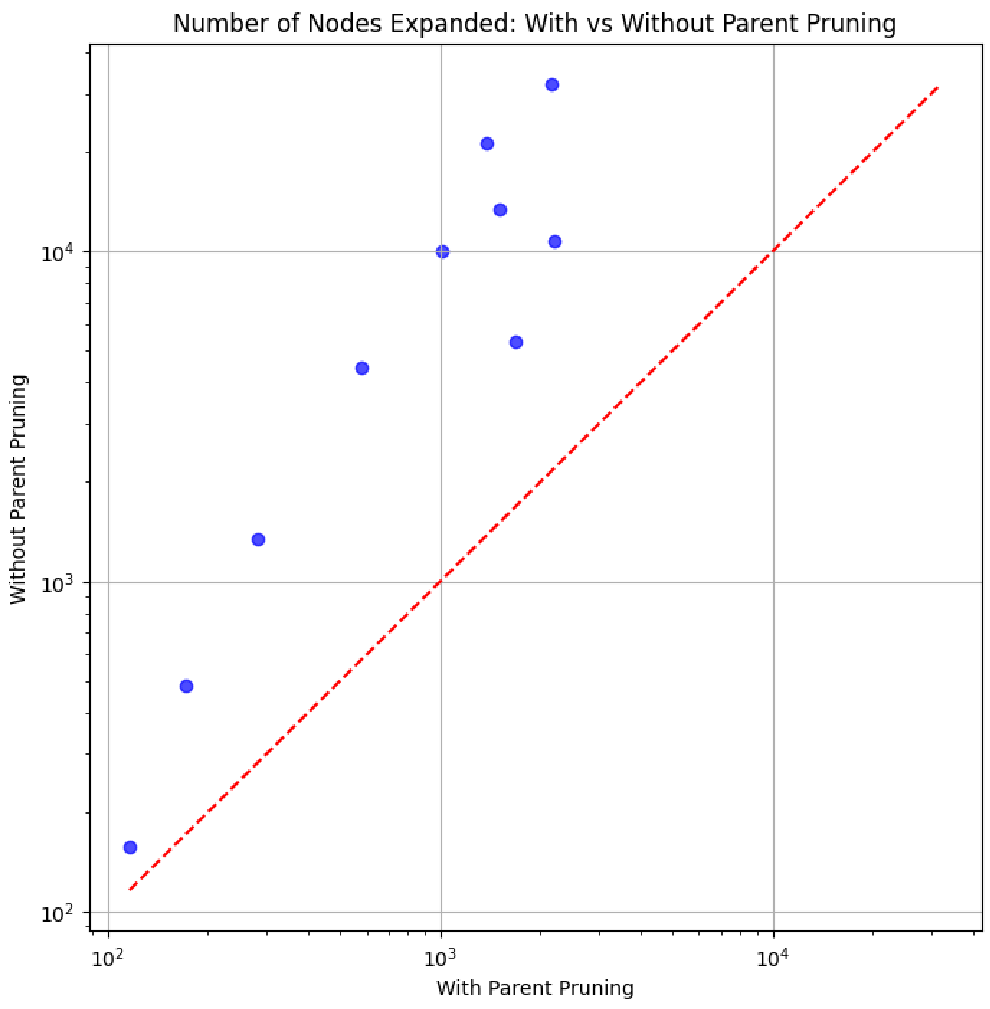

Figure 1: Number of Nodes Expanded: With vs Without Parent Pruning

We observe that, on average, IDA* without parent pruning takes approximately 7 times longer than with parent pruning. The increased runtime is due to the repeated generation of previously visited nodes, which parent pruning effectively prevents. While this aligns with the overall trend in my hypothesis, the factor of increase is lower than the predicted 16-fold, indicating that although the lack of pruning significantly impacts performance, the degree of increase is less severe than initially expected.

2 Part 2

a) Below is a scatter plot where the x-axis represents the number of node expansions for IDA*, while the y-axis represents the number of node expansions for A*:

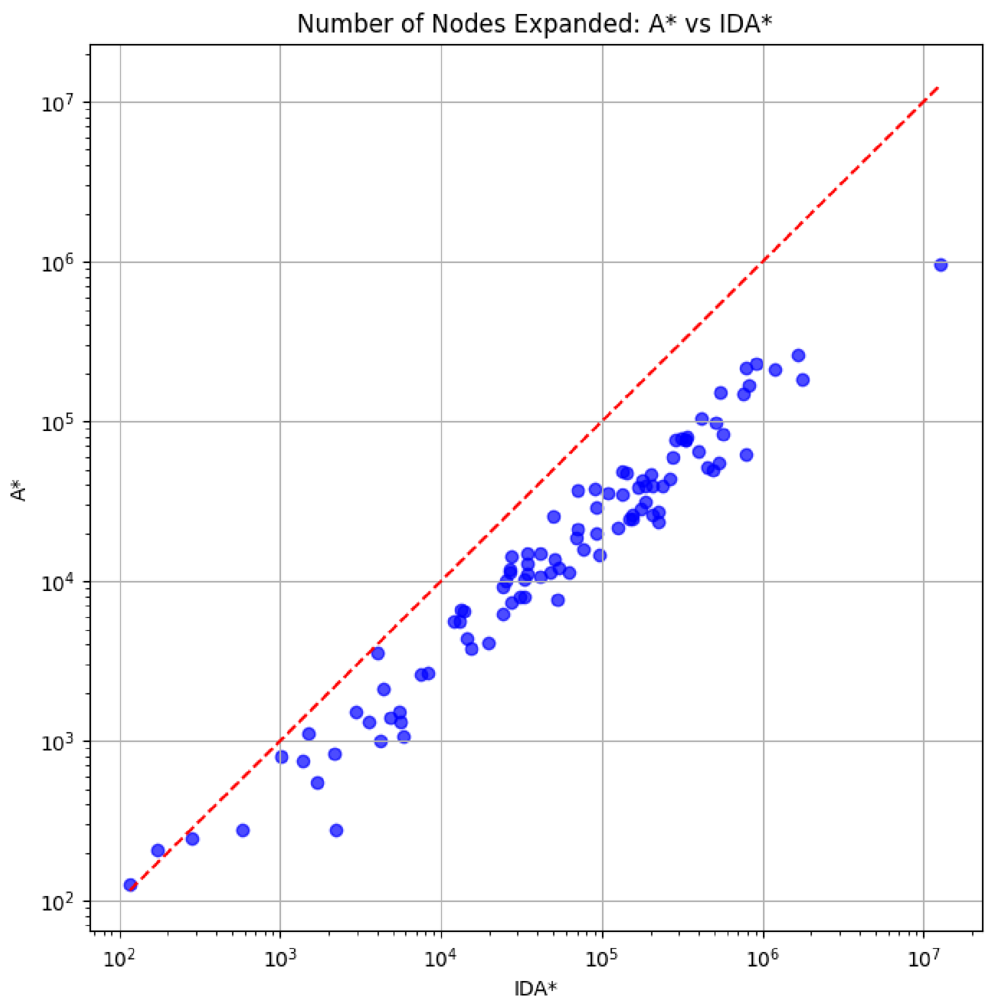

Figure 2: Number of Nodes Expanded: A* vs. IDA*

From the scatter plot above, we observe a general trend that problems which are difficult for A* (in terms of node expansions) tend to also be difficult for IDA*. In most cases, A* requires fewer node expansions than IDA*. However, there are several instances where the gap between the two algorithms increases, especially for more complex problems. This suggests that A* often performs more efficiently than IDA*, particularly for larger or more difficult problems. The $y = x$ reference line illustrates that A* generally explores fewer nodes than IDA*, though some outliers indicate otherwise in certain cases.

b) The average total runtime of A* across 10 runs is 11.87 seconds, while for IDA* it is 14.05 seconds. The average total number of node expansions across these runs is 4,708,401 for A* and 32,246,190 for IDA*. The runtime ratio between IDA* and A* is 1.18, and the node expansion ratio is 6.85.

The disparity between runtime ratio and node expansion ratio (1.18 vs. 6.85) indicates that fewer node expansions do not guarantee proportional reductions in runtime. Additional factors contribute to runtime; for instance, A* requires more memory and incurs more overhead for managing its open list. These overheads can impact the actual runtime efficiency despite the differences in node expansions.

3 Part 3

My hypothesis is that implementing random operator ordering could reduce the average runtime of both A* and IDA*, particularly in cases with frequent ties among actions. This is because random ordering enables the algorithm to explore varied paths in each run, allowing some paths to reach the goal more quickly than others. By shuffling the order of actions with equal f-values, the algorithm can avoid systematic biases that might otherwise slow down convergence to the goal.

Here is a table summarizing the results on number of node expansions:

Problem	**Ordering**	**Minimum**	**Maximum**	**Average**	**Median**	**Std. Deviation**
Problem 1	Random	77416	78195	77832.14	77846	311.78
Problem 1	Non-Random	78510	78510	78510.0	78510	0.0
Problem 2	Random	77538	78704	78173.29	78291	548.17
Problem 2	Non-Random	78510	78510	78510.0	78510	0.0
Problem 3	Random	77559	78719	78240.67	78381	463.89
Problem 3	Non-Random	78510	78510	78510.0	78510	0.0

Table 3: Comparison of A* Metrics with and without Random Ordering for Selected Problems

Problem	**Ordering**	**Minimum**	**Maximum**	**Average**	**Median**	**Std. Deviation**
Problem 1	Random	81234	82345	81834.57	81850	489.78
Problem 1	Non-Random	82500	82500	82500.0	82500	0.0
Problem 2	Random	81021	82793	81810.76	81901	563.45
Problem 2	Non-Random	82500	82500	82500.0	82500	0.0
Problem 3	Random	81367	82655	81934.45	82000	460.20
Problem 3	Non-Random	82500	82500	82500.0	82500	0.0

Table 4: Comparison of IDA* Metrics with and without Random Ordering for Selected Problems

In all cases, random operator ordering demonstrates better performance in terms of the number of node expansions for both IDA* and A*. This result aligns with the initial hypothesis. The observed improvement can be explained by the fact that random operator ordering assists with tie-breaking; by exploring actions in a randomized order, there is an increased chance that the first node selected may lead toward the goal more efficiently. Consequently, this approach reduces the overall search effort, leading to fewer node expansions.

4 Part 4

a) Below is the graph I constructed:

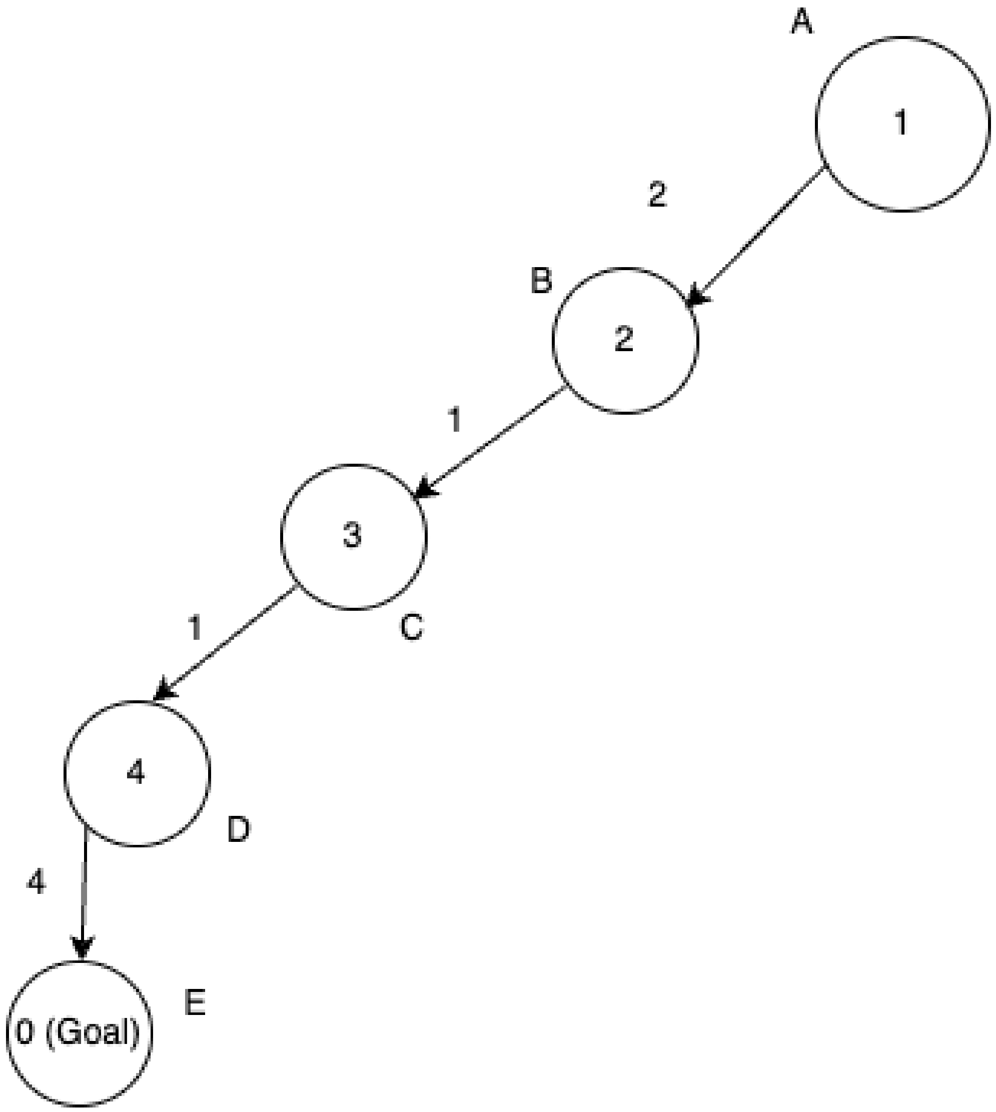

Figure 3: Integer H IDA*

Threshold 0: 1; Expand A, Generate B

Threshold 1: 4: Expand A, Generate B; Expand B, Generate C.

Threshold 2: 6: Expand A, Generate B; Expand B, Generate C; Expand C, Generate D.

Threshold 3: 7: Expand A, Generate B; Expand B, Generate C; Expand C, Generate D; Expand D, Generate E.

Threshold 4: 8: Expand A, Generate B; Expand B, Generate C; Expand C, Generate D; Expand D, Generate E; Expand E, Goal.

b) Below is the graph I constructed:

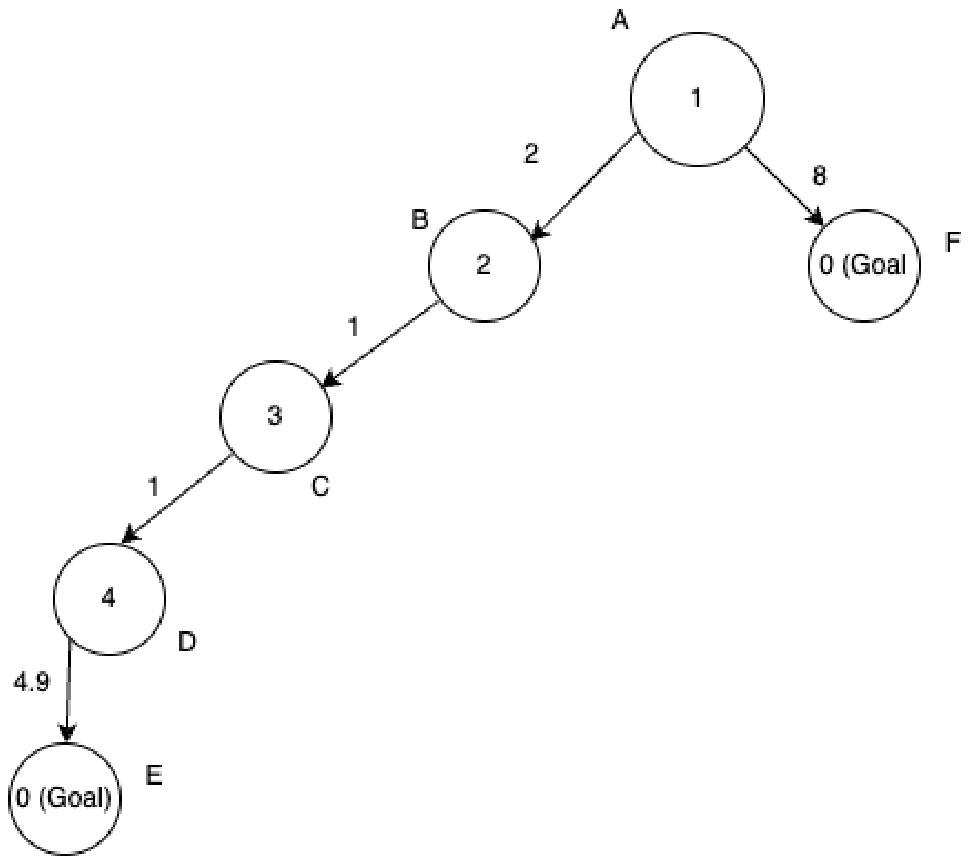

Figure 4: Integer H IDA*

Threshold 0: 1; Expand A, Generate B, Generate F.
Threshold 1: 4: Expand A, Generate B; Expand B, Generate C. Generate F from A.
Threshold 2: 6: Expand A, Generate B; Expand B, Generate C; Expand C, Generate D. Generate F from A.
Threshold 3: 7: Expand A, Generate B; Expand B, Generate C; Expand C, Generate D; Expand D, Generate E. Generate F from A.
Threshold 4: 8: Expand A, Generate B; Expand B, Generate C; Expand C, Generate D; Expand D, Generate E; Expand E, Goal. Generate F from A.
Clearly E is a suboptimal goal (cost 8.9) while F is the optimal goal (cost 8).

5 Part 5

Lemma: If WIDA* completes an iteration without solving the problem, then some node on the optimal path must have been generated but not expanded during the previous iteration.

Theorem: If the heuristic used is admissible, then any solution found by WIDA* is guaranteed to have a cost of no more than wC^*.

Proof: There are two cases to consider:

1. **Case 1: The solution is found during the first iteration.** The first threshold is $T_1 = wh(n_I) \leq wh^*(n_I)$ since h is admissible, where h^* denotes the true cost to the goal. Since n_I is on the optimal

path, we know that $h^*(n_I) \leq C^*$. Therefore, $wh^*(n_I) \leq wC^*$, and by transitivity, we have $wh(n_I) \leq wC^*$. This satisfies the theorem, as the cost of the solution found is bounded by wC^*.

2. **Case 2: The solution is found during iteration $k > 1$ with threshold T_k.** Since no solution was found during iteration $k-1$, Lemma 1 implies that there exists a node n_i on the optimal path $p^* = [n_0, \ldots, n_j]$ that was generated but not expanded in iteration $k-1$. By the definition of WIDA* thresholds, we have:

$$T_k \leq wf(n_i)$$

where $f(n_i)$ is the f-value of n_i on the optimal path p^*. Since $f(n_i) \leq C^*$ and T_k is set to the minimum f-value among unexpanded nodes, we conclude that $T_k \leq wC^*$. Therefore, the solution found in this iteration will also have a cost of at most wC^*.

Thus, in both cases, any solution found by WIDA* is bounded by wC^*, which proves the theorem.

6 Part 6

a) Configurations: WA*: no reexpansion, high-g tie-breaking, no random operator ordering. WIDA*: no random operator ordering. Here is the line chart:

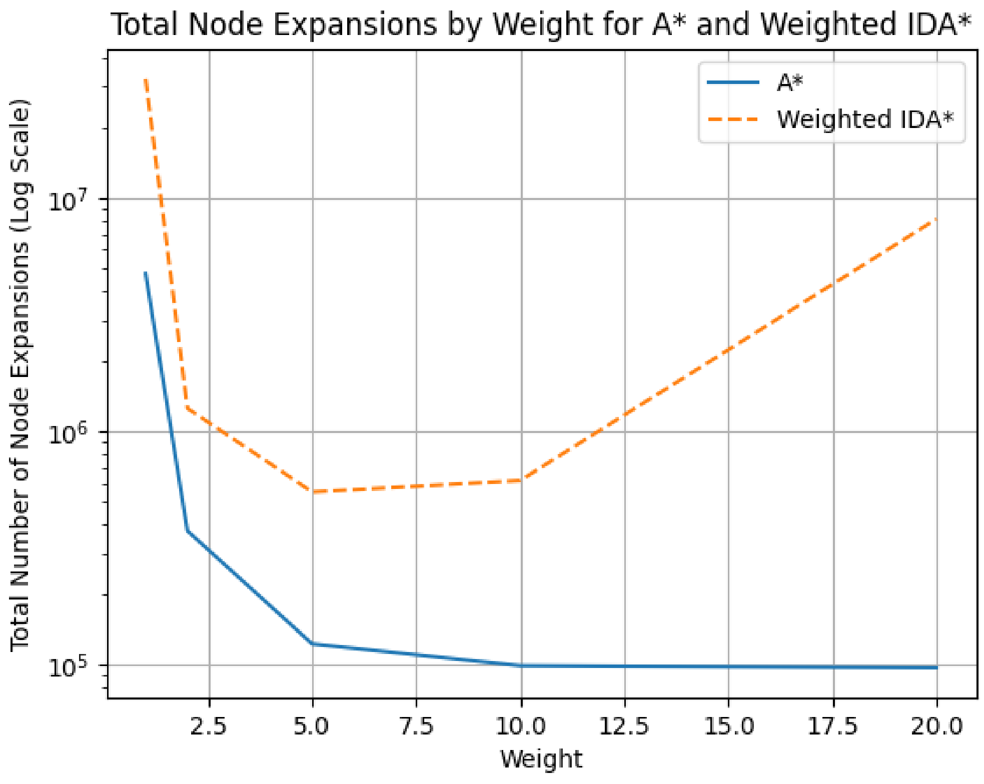

Figure 5: Line Chart Comparing Number of Node Expansions For Weights WA* and WIDA*

We observe that the total number of node expansions decreases for WA* as the weight increases, while for WIDA*, the number of node expansions initially drops and then rises. Generally, WA* expands fewer nodes than WIDA* across the tested weights. The reduction in node expansions for WA* can be attributed to the increased weight, which makes WA* more greedy, leading it to explore fewer nodes. In contrast, the initial decrease followed by an increase in expansions for WIDA* may be explained by the threshold value rising too quickly, occasionally causing the search to overshoot the goal.

b) Here is the line chart:

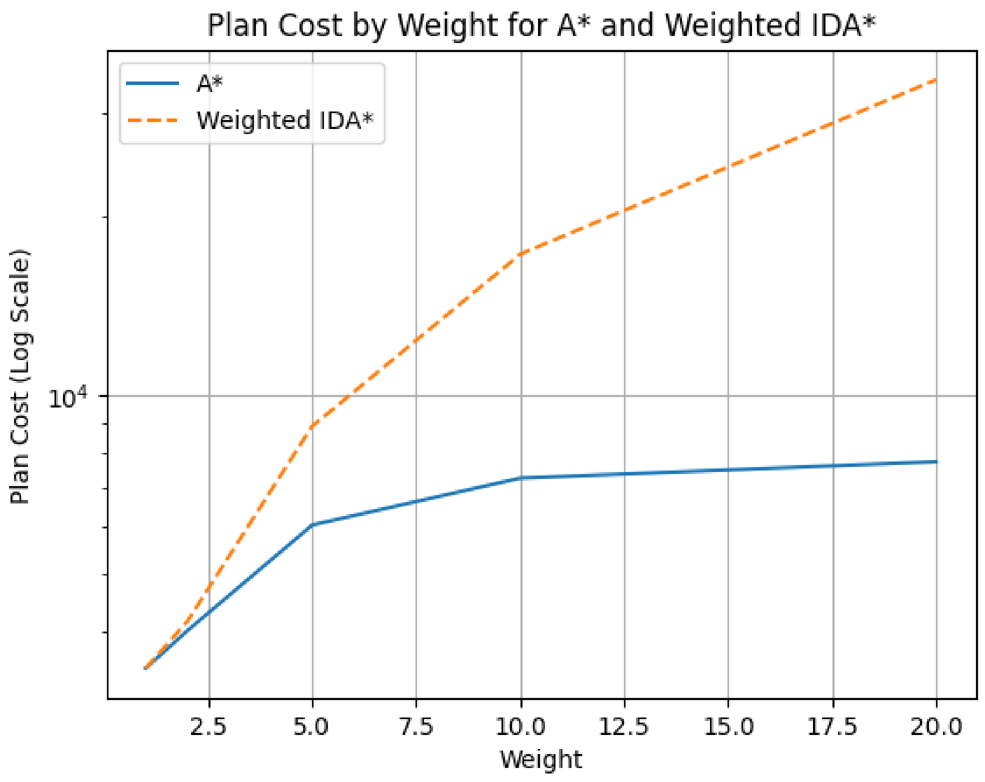

Figure 6: Line Chart Comparing Solution Cost For Weights WA* and WIDA*

We observe that the solution cost increases for both WA* and WIDA* as the weight increases. The solution cost for WA* is generally lower than that of WIDA*. This rise in solution cost can be attributed to the increased greediness of the algorithms, which leads to finding more suboptimal solutions as weight grows. Additionally, the overshooting behavior of WIDA* causes it to produce even more suboptimal solutions than WA*, resulting in comparatively higher solution costs.