Heuristic Analysis

Table of Contents

1. Abbreviations

- BFS: breadth-first search
- DFS: depth-first search
- UCS: uniform-cost search
- IP: ignore-preconditions heuristic
- LS: level-sum heuristic
- FIFO: first-in, first-out
- A*IP: A* search using IP
- A*LS: A* search using LS

2. Optimal Action Sequences

air_cargo_p1 (**6 actions**):

1. Load(C1, P1, SFO)
2. Fly(P1, SFO, JFK)
3. Load(C2, P2, JFK)
4. Fly(P2, JFK, SFO)
5. Unload(C1, P1, JFK)
6. Unload(C2, P2, SFO)

air_cargo_p2 (**9 actions**):

1. Load(C1, P1, SFO)
2. Fly(P1, SFO, JFK)
3. Load(C2, P2, JFK)
4. Fly(P2, JFK, SFO)
5. Load(C3, P3, ATL)
6. Fly(P3, ATL, SFO)
7. Unload(C3, P3, SFO)
8. Unload(C2, P2, SFO)
9. Unload(C1, P1, JFK)

air_cargo_p3 (**12 actions**):

1. Load(C2, P2, JFK)
2. Fly(P2, JFK, ORD)
3. Load(C4, P2, ORD)
4. Fly(P2, ORD, SFO)
5. Load(C1, P1, SFO)
6. Fly(P1, SFO, ATL)
7. Load(C3, P1, ATL)
8. Fly(P1, ATL, JFK)
9. Unload(C4, P2, SFO)
10. Unload(C3, P1, JFK)
11. Unload(C2, P2, SFO)
12. Unload(C1, P1, JFK)

3. Algorithm Performance on the Problems

The discussion and tables below compare the performance of BFS, DFS, UCS, A*IP, and A*LS. All algorithms except for DFS found an optimal solution to all problems.

Optimality of solution. Because the air-cargo graph is unweighted (all actions have the same cost), BFS is guaranteed to find the optimal sequence of actions. Inspection of the results in Table 1 below shows that no other algorithm finds a shorter sequence of actions than BFS. We also expect UCS and A* with an admissible heuristic to both find an optimal solution, and they do. While DFS is guaranteed to find some solution in a finite search space, it is not guaranteed to find an optimal solution. We see that DFS indeed does not find an optimal solution.

	air_cargo_p1	air_cargo_p2	air_cargo_p3
BFS	**6**	**9**	**12**
DFS	**20**	**619**	**392**
UCS	**6**	**9**	**12**
A*IP	**6**	**9**	**12**
A*LS	**6**	**9**	**12**

Table 1: optimality of solution. Comparing the optimality of solutions found by each algorithm across the three air-cargo problems. The **bold** cells show the number of actions in each solution (the fewer the actions, the better).

Time elapsed. In terms of time used, BFS was fastest on all problems among the algorithms that found an optimal solution. DFS, however, happened to take significantly less time, but in all cases DFS failed to find anywhere close to an optimal solution. Table 2 below shows the time used in each algorithm-problem pairing.

	air_cargo_p1	air_cargo_p2	air_cargo_p3
BFS	**0.04**	**20.24**	**173.02**
DFS	**0.03**	**4.82**	**2.36**
UCS	**0.06**	**73.68**	**644.43**
A*IP	**0.11**	**30.75**	**179.13**
A*LS	**3.81**	**351.53**	**2422.56**

Table 2: time elapsed. Comparing the time used by each algorithm while finding a solution to an air-cargo problem. The **bold** cells show the number of seconds elapsed during a run of an algorithm-problem pairing.

Node expansions. Just as DFS happened to take less time than all other searches, DFS also happens to expand fewer nodes than all but one search (A*LS). However, the solutions from DFS were all suboptimal. Among all searches that found optimal solutions, the A* searches, as expected, expanded the fewest nodes. A*LS was a standout, expanding significantly fewer nodes than all other searches across all problems. Table 3 below shows the number of nodes expanded in each algorithm-problem pairing.

	air_cargo_p1	air_cargo_p2	air_cargo_p3
BFS	**43**	**3343**	**14,663**
DFS	**21**	**624**	**408**
UCS	**55**	**4852**	**18,235**
A*IP	**41**	**1506**	**5118**
A*LS	**11**	**86**	**404**

Table 3: node expansions. Comparing the number of nodes expanded by each algorithm across the three air-cargo problems. The **bold** cells show the number of nodes expanded in the search for a solution (the fewer the nodes, the better).

4. Explanation of Performance Differences

A*LS versus A*IP. A*LS expands significantly fewer nodes than A*IP but requires more time. The fact that it takes more time makes sense: my implementation of IP is a simple greedy algorithm, whereas LS constructs and analyzes a planning graph. Therefore a run of IP can be expected to take more time than a run of LS.

It also makes sense that A*LS expands fewer nodes than A*IP. This is because IP is more relaxed than LS: 1) IP ignores the preconditions of actions, whereas the planning graph used in LS uses action preconditions in the generation of action levels; 2) my implementation of IP (following a tip from Russell-Norvig Ed-3 10.2.3) also ignores the fact that actions can undo each other's effects, whereas the planning graph used in LS takes this into account by modeling actions with mutexes. Because LS is less relaxed than IP, it returns a closer estimate of the true distance between a search node and the goal node, while still being admissible. And because IP, in general, gives us a better estimate of distance than LS, this allows our search to better discriminate between nodes that are on an optimal path and nodes that are not, which in turn means that our search has to expand fewer nodes.

A* versus BFS. Note that the ratio of time used by A* searches versus BFS shrinks as the problem size grows. A*IP took almost three times as many seconds as BFS did on problem 1,

but on problem 2 it was roughly 1.5 times as many seconds, and then on problem 3 only 1.04 times as many seconds. For A*LS versus BFS, the ratios were roughly 95, 17, and 14. On smaller problem sizes, the slower times for A*, compared to BFS, come from the extra time A* spends on each node in the search space. That is, using a priority queue, rather than FIFO queue, to store nodes on the frontier means that A* spends more time than BFS in putting a node on the frontier, and using a heuristic function to analyze each node on the frontier is an extra cost that comes with A* that is not present in BFS. However, the benefit of A* search (i.e., expanding fewer nodes in the search space) becomes apparent on larger problems, and this is reflected in the shrinking ratio of A* versus BFS's times elapsed. With a large enough search space, we could expect to see A*IP and A*LS both take less time than BFS.

BFS versus UCS. UCS search always takes significantly longer than BFS. This is mostly due to the extra expense that UCS incurs from using a priority queue, rather than FIFO queue, as its frontier. Furthermore, because our air-cargo problem is over an unweighted graph, we know that both BFS and UCS will both find optimal solutions, in which case BFS should be chosen over UCS due to BFS's faster runtime.