# Analysis of different search algorithms for PDDL-Problems

Kai Schos

August 25, 2017

## 1 Introduction

This document is an analysis and comparison of different search algorithms used on a classical planning problem, namely the 3 air-cargo-problems that can be found at https://github.com/udacity/AIND-Planning. Problem 1 is the easiest and problem 3 is the hardest of the set. In the first section, non-heuristic, also known as uninformed search algorithms are used on the problem. In the second part, an A* agent is used to search the planning space with different heuristics, which are compared to one another.

## 2 Analysis

### 2.1 Uninformed search

The three air-cargo-problems are searched with three different uninformed search algorithms. These algorithms were breath-first-search, depth-first-graph-search and depth-limited-search with a limit of 50 layers. Figures 1, 2, 3 and 4 show the results for the 3 problems and the 3 algorithms in 4 different metrics: The time it took the algorithm to complete, the number of new nodes, the number of expanded nodes and the number of goal tests.
In addition to these metrics it should be noted that only breadth-first-search finds an optimal solution, whereas depth-first-graph-search and depth-limited-search do not.
Overall, BFS is in general half an order of magnitude worse than DFS. Depth limited search did very poorly in searching plans defined in PDDL. It takes the worst of both worlds of BFS and DFS.
DFS has the advantage that, in theory, there is no upper bound on the depth of the search. As the possible actions are sampled from an unordered set, making them more or less random, and every effect of an action can be undone by a different action, it would be easy to prove that every problem in the air-cargo-domain can be solved by DFS given enough time.
BFS has the advantage that it finds the optimal path when there is one. Also, the shorter the optimal path, the faster BFS will find it. This contrasts DFS: Even if there is a very short path, DFS can go a long time without finding any solution because of the mutually exclusive actions.
DLS first searches to layer 50 and then basically degrades to a reversed BFS at layer 50. If it does not find a solution by accident on its way down, it will stay deep in the search tree before checking different, shorter solutions. If there were 2 actions taken that have to be undone when DLS hits its limiting layer, it will have to search at least (about) 2 layers at this depth to find a solution. Because we are so deep in the three, 2 layers are huge.

### 2.2 Informed search

In a second step, plans for the three air-cargo-problems were searched with the A search algorithm with two different heuristics: The 'ignore preconditions'-heuristic and a heuristic that makes use of a planning graph, simply summing the level cost of the literals.
Diagram 1 shows that the ignore-preconditions seems to be faster for smaller problems. This is caused by the long time it takes to create the planning graph. However, in larger problems, the time it takes to compute the planning graph degrades to a constant factor and both searches seem to run in approximately the same time ( They both do a search over a relaxed problem, and the two relaxed problems seem to be of the same difficulty ).
However, the heuristic that the planning graph provides is a lot more precise than the *ignore preconditions* heuristic. This results in less expansions, new nodes and goal tests, as shown in figures 2, 3 and 4.

## 3 Optimal Paths

Following are 3 paths that solve the 3 problems optimally. They were extracted with BFS, so they are guaranteed to be optimal (when the concerned metric is that of number of steps), according to AIMA, third edition, page 81.

### 3.1 Problem #1

6 Steps:

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

### 3.2 Problem #2

9 Steps:

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

### 3.3 Problem #3

12 Steps:

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

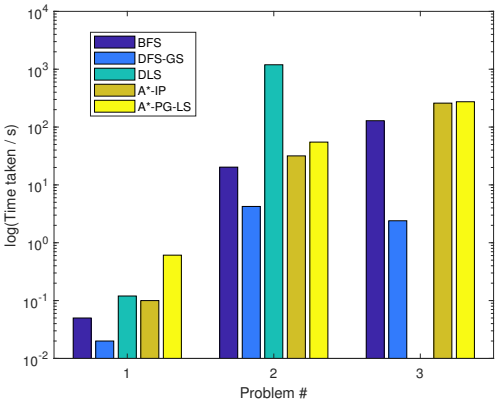


Figure 1: Logarithm of the time it took for the different algorithms to complete. No data exists for problem 3 using depth-limited-search because the computation time exceeded one hour.

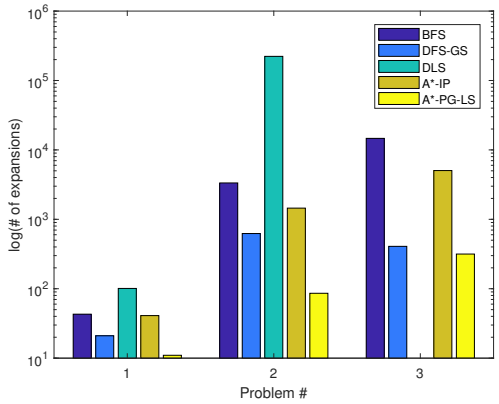


Figure 2: Logarithm of the number of nodes the algorithm expanded. No data exists for problem 3 using depth-limited-search because the computation time exceeded one hour.

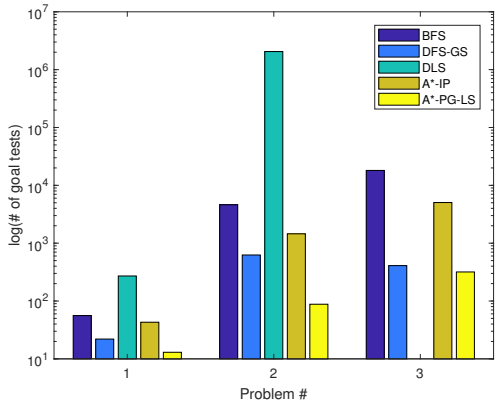


Figure 3: Logarithm of the number of goal tests the algorithm issued. No data exists for problem 3 using depth-limited-search because the computation time exceeded one hour.

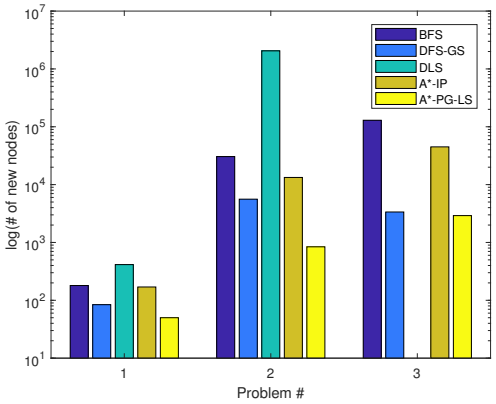


Figure 4: Logarithm of the number of new nodes that were generated by the algorithm. No data exists for problem 3 using depth-limited-search because the computation time exceeded one hour.