Implement a Planning Search

Synopsis

In this project I created a planning search agent to solve an air cargo logistics problem defined in classical PDDL (Planning Domain Definition Language). I implemented and compared different non-heuristic based progression search algorithms and two domain-independent heuristics to be used with the A* search algorithm.

Problem description

The three problems that needed to be solved are shown below. They are defined in PDDL and are increasingly more complicated due to an increased number of variables (planes, cargo, and airports) and goal state requirements.

- Problem 1 initial state and goal:

```
Init(At(C1, SFO) ∧ At(C2, JFK)
        ∧ At(P1, SFO) ∧ At(P2, JFK)
        ∧ Cargo(C1) ∧ Cargo(C2)
        ∧ Plane(P1) ∧ Plane(P2)
        ∧ Airport(JFK) ∧ Airport(SFO))
Goal(At(C1, JFK) ∧ At(C2, SFO))
```

- Problem 2 initial state and goal:

```
Init(At(C1, SFO) ∧ At(C2, JFK) ∧ At(C3, ATL)
        ∧ At(P1, SFO) ∧ At(P2, JFK) ∧ At(P3, ATL)
        ∧ Cargo(C1) ∧ Cargo(C2) ∧ Cargo(C3)
        ∧ Plane(P1) ∧ Plane(P2) ∧ Plane(P3)
        ∧ Airport(JFK) ∧ Airport(SFO) ∧ Airport(ATL))
Goal(At(C1, JFK) ∧ At(C2, SFO) ∧ At(C3, SFO))
```

- Problem 3 initial state and goal:

```
Init(At(C1, SFO) ∧ At(C2, JFK) ∧ At(C3, ATL) ∧ At(C4, ORD)
        ∧ At(P1, SFO) ∧ At(P2, JFK)
        ∧ Cargo(C1) ∧ Cargo(C2) ∧ Cargo(C3) ∧ Cargo(C4)
        ∧ Plane(P1) ∧ Plane(P2)
        ∧ Airport(JFK) ∧ Airport(SFO) ∧ Airport(ATL) ∧ Airport(ORD))
Goal(At(C1, JFK) ∧ At(C3, JFK) ∧ At(C2, SFO) ∧ At(C4, SFO))
```

Solutions

The optimal solutions found by at least one of the search algorithms:

- Problem 1 solution in 6 steps:

```
Load(C1, P1, SFO)
Load(C2, P2, JFK)
Fly(P1, SFO, JFK)
Fly(P2, JFK, SFO)
Unload(C1, P1, JFK)
Unload(C2, P2, SFO)
```

- Problem 2 solution in 9 steps:

```
Load(C1, P1, SFO)
Fly(P1, SFO, JFK)
Load(C2, P2, JFK)
Fly(P2, JFK, SFO)
Load(C3, P3, ATL)
Fly(P3, ATL, SFO)
Unload(C1, P1, JFK)
Unload(C2, P2, SFO)
Unload(C3, P3, SFO)
```

- Problem 3 solution in 12 steps:

```
Load(C2, P2, JFK)
Fly(P2, JFK, ORD)
Load(C4, P2, ORD)
Fly(P2, ORD, SFO)
Load(C1, P1, SFO)
Fly(P1, SFO, ATL)
Load(C3, P1, ATL)
Fly(P1, ATL, JFK)
Unload(C1, P1, JFK)
Unload(C2, P2, SFO)
Unload(C3, P1, JFK)
Unload(C4, P2, SFO)
```

Results

Non-heuristic search

Tables 1-3 show the comparison between the following different non-heuristic search methods: breadth-first, depth-first, and uniform cost. The methods greedy best-first graph search with h1 and A* search with h1 also belong in this category as both of them are using the constant h1 heuristic, which, since it is constant, is not a true heuristic.

Problem 1						
Method	**Expansions**	**Goal Tests**	**New Nodes**	**Plan length**	**Time**	**Optimality**
Breadth first search	43	56	180	6	0.04	✔
Depth first graph search	21	22	84	21	0.01	✘
Uniform cost search	55	57	224	6	0.05	✔
Greedy best first graph search with h1	7	9	28	6	0.005	✔
A* search with h1	55	57	224	6	0.05	✔
A* search with ignore preconditions	41	43	170	6	0.06	✔
A* search with level sum	12	14	54	6	4.02	✔

Table 1: Problem 1

Problem 2						
Method	**Expansions**	**Goal Tests**	**New Nodes**	**Plan length**	**Time**	**Optimality**
Breadth first search	3343	4609	30107	9	14.96	✔
Depth first graph search	1811	1812	15303	831	10.21	✘
Uniform cost search	4780	4782	42670	9	46.75	✔
Greedy best first graph search with h1	478	480	4182	21	2.64	✘
A* search with h1	4780	4782	42670	9	47.07	✔
A* search with ignore preconditions	1506	1508	13739	9	15.7	✔
A* search with level sum	86	88	841	9	530.8	✔

Table 2: Problem 2

Problem 3						
Method	**Expansions**	**Goal Tests**	**New Nodes**	**Plan length**	**Time**	**Optimality**
Breadth first search	14663	18098	129631	12	112.44	✔
Depth first graph search	408	409	3364	392	1.9	✘
Uniform cost search	17532	17534	153777	12	416.06	✔
Greedy best first graph search with h1	3376	3378	30099	26	58.59	✘
A* search with h1	17532	17534	153777	12	416.83	✔
A* search with ignore preconditions	5102	5104	45488	12	94.34	✔
A* search with level sum	404	406	3718	12	2576.5	✔

Table 3: Problem 3

The tables are colour coded with blue showing the best, white the intermediate and red the worst search method for the respective metric (#nodes expanded, #goal tests, #new nodes, plan length, and time). The optimality metric shows if the solution is optimal in length, that is, it consists of the minimum possible number of steps required to go from the initial to the goal state.
A first observation is that depth first and greedy best first generally perform best when it comes to all heuristics (apart from plan length) but deliver a non-optimal solution. While greedy best first delivers a plan with some additional unnecessary steps, the depth first solution is significantly longer than the ideal one. It can also be noted that A* search with a constant heuristic is identical to the uniform cost search.

For the following comparison, I ignore greedy best first due to non-optimality and A* with h1 since it is identical to the uniform cost method. I include depth first even though it's solution is non optimal, because it is by far the fastest algorithm to produce a solution. So in a case of negligible execution time, this method might still be useful. Figures 1-3 show the performance of the search methods relative to each other.

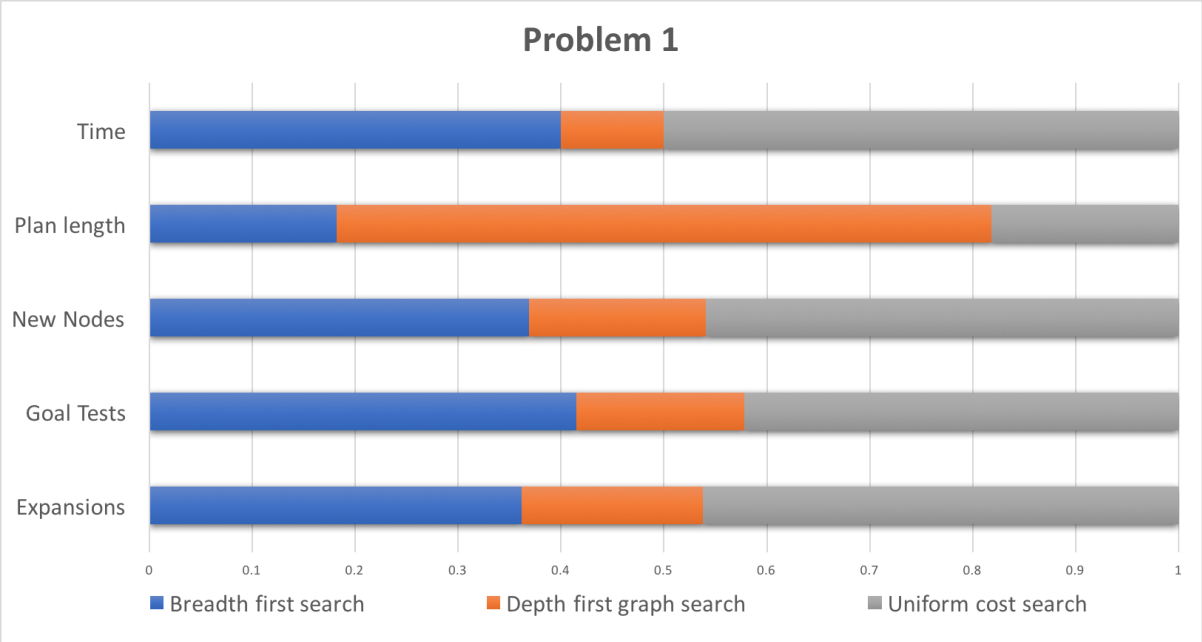

Figure 1: Problem 1, non-heuristic methods

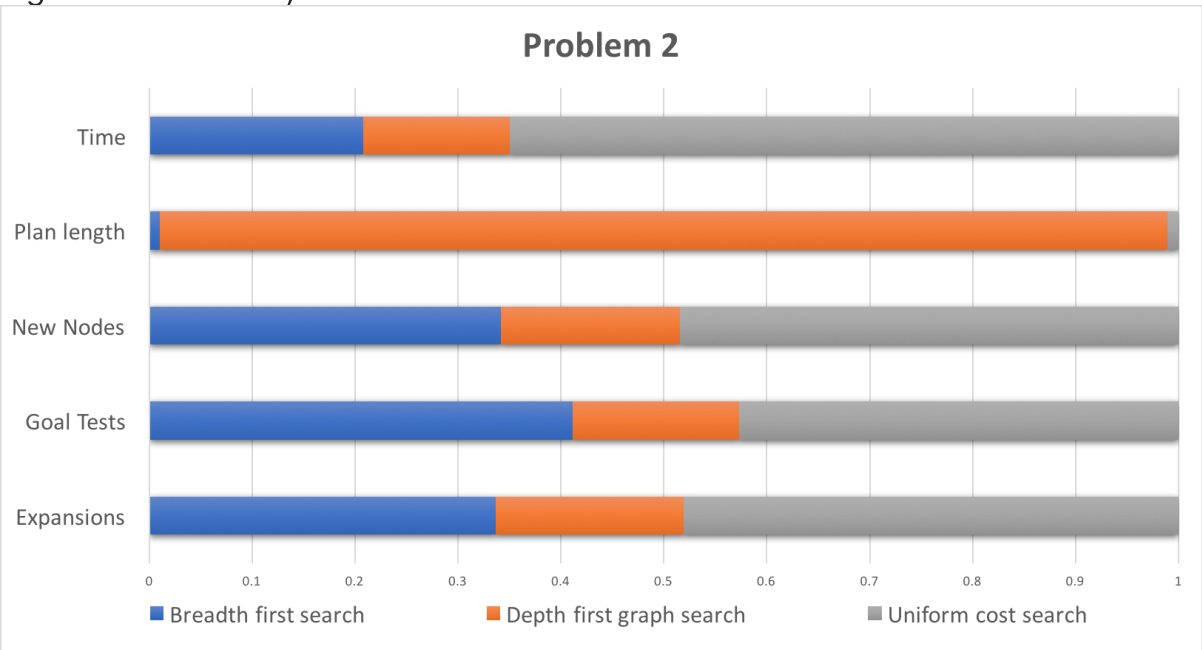

Figure 2: Problem 2, non-heuristic methods

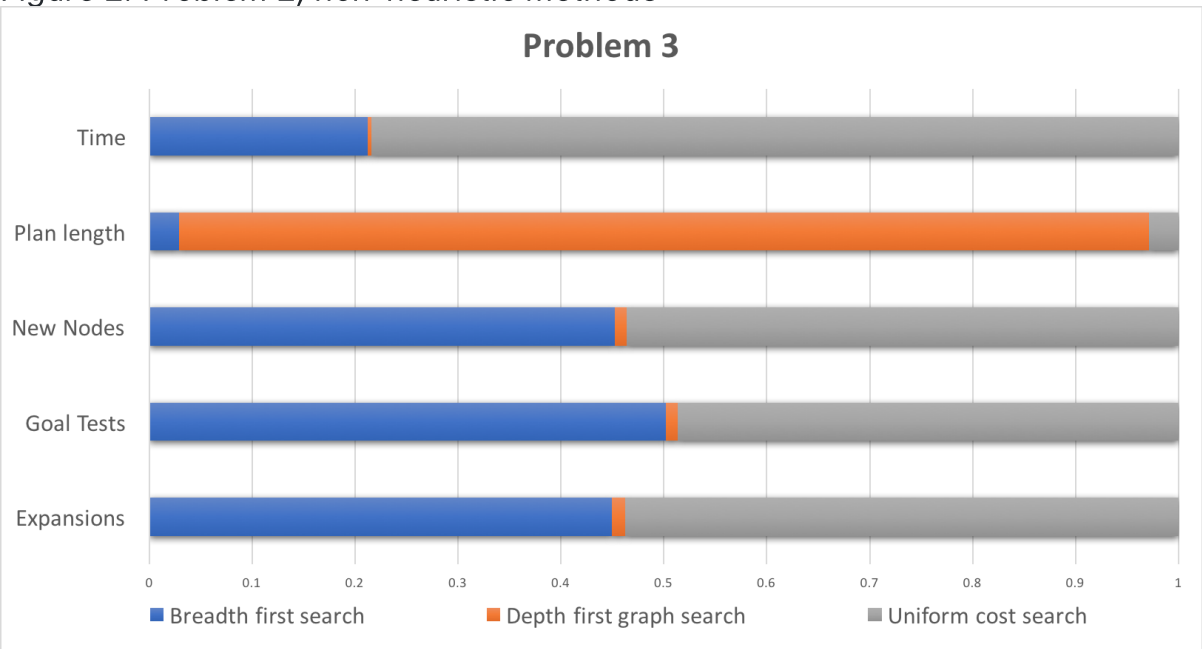

Figure 3: Problem 3, non-heuristic methods

Depth first search delivers non-optimal results but expands and evaluates the fewest nodes and comes to a solution in the least amount of time. The comparison between breadth first and uniform cost search is more interesting since both of them deliver an optimal result. Uniform cost takes longer and tends to create, expand and goal test more nodes. Going from the simple problem 1 to the more complicated problem 3, the difference becomes smaller though. This leads me to believe that as the problems get more complex, uniform cost search will start to create, expand and goal test fewer nodes than breadth first, although it might still require more time to come to a solution.

Domain-independent heuristics

The previously shown tables 1-3 also include the results for the two heuristic based methods, A* search with ignore preconditions and A* search with level sum. Both methods deliver optimal solutions. The runtime of A* search with ignore preconditions is comparable to the non-heuristic methods, whereas the runtime for A* search with level sum is significantly higher. When it comes to node creation, expansion and goal testing though, A* search with level sum even beats the depth first search most of the time while A* search with ignore preconditions gives intermediate results.

Figures 4-6 show a relative comparison between the two heuristics methods.

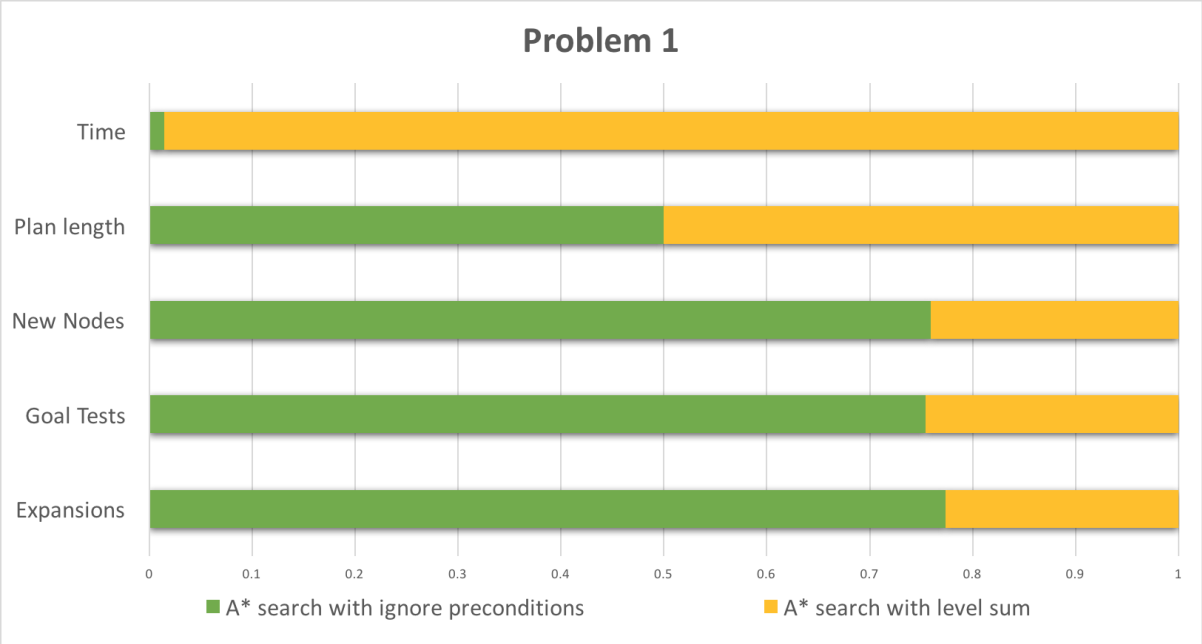

Figure 4: Problem 1, heuristic methods

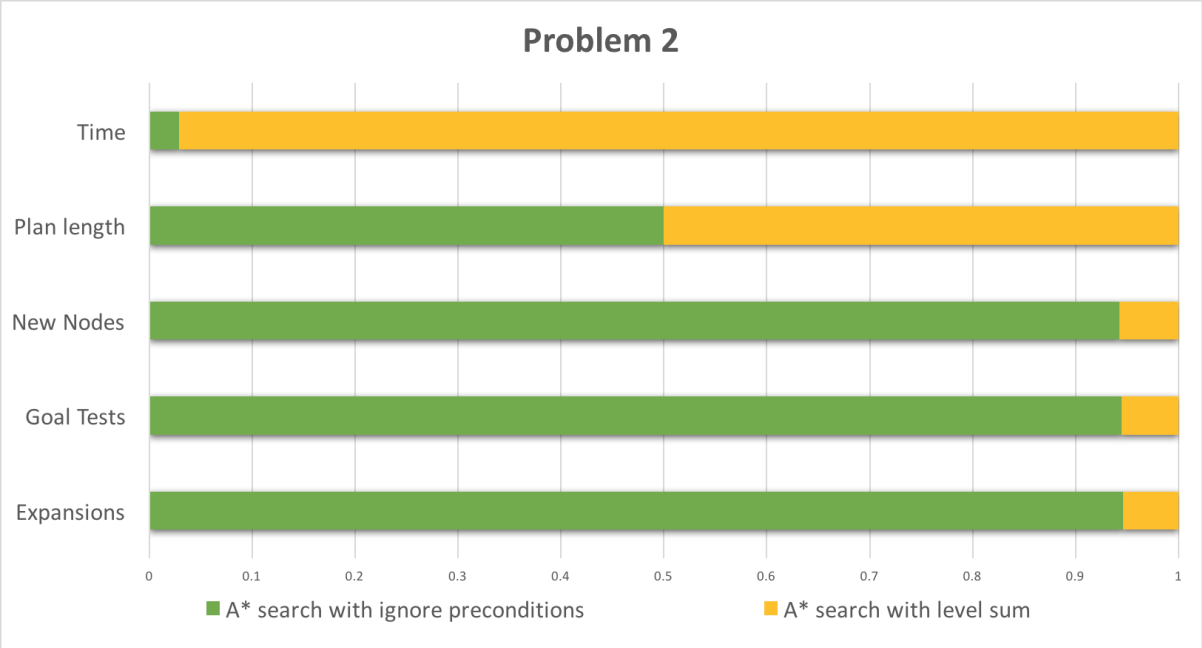

Figure 5: Problem 2, heuristic methods

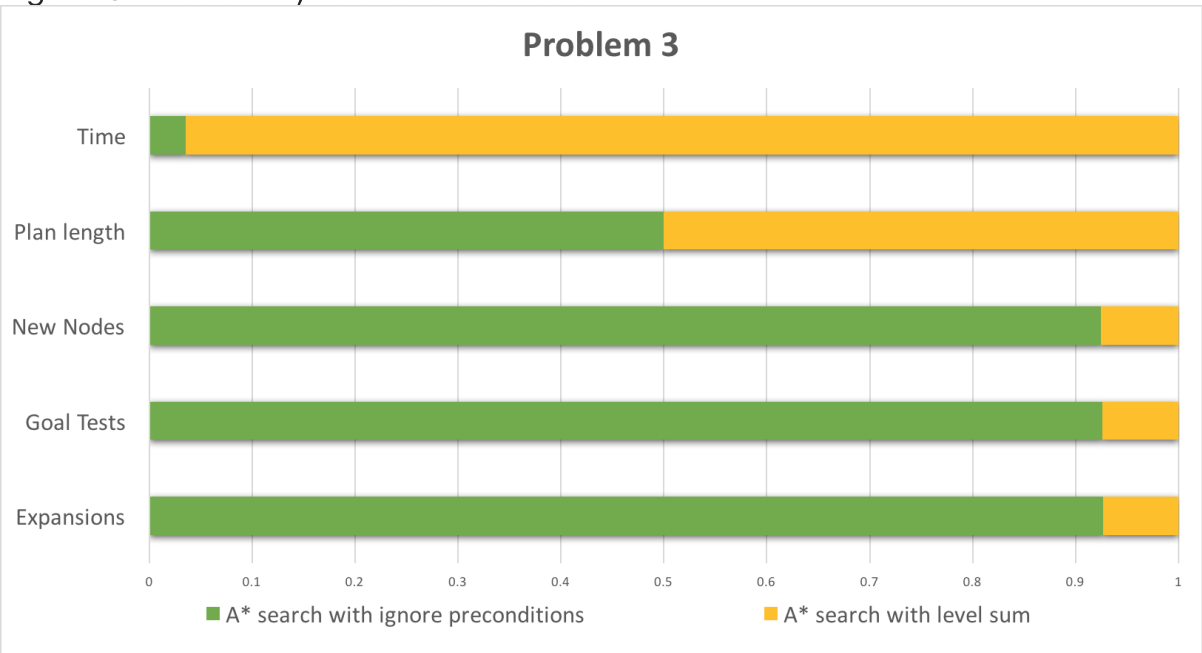

Figure 6: Problem 3, heuristic methods

It is obvious that the A* search with ignore preconditions is faster to compute but requires more node creations, expansions and goal tests. The more complicated the problem gets, the better A* search with level sum performs though. The runtime difference gets less but the difference in nodes created, expanded and goal tested also gets less.

Conclusions

In the scope of the current problem and with the implementation of the planning graph, A* search with ignore preconditions is the better heuristic. However, the results and intuition suggests that going to more complicated problems or further optimising the planning graph implementation will lead to a performance increase of A* search with level sum heuristic, making it the method of choice. This is expected since the level sum heuristic requires more computation but gives an estimate that is very close to the actual result, therefore prioritising nodes close to the ideal solution, avoiding unnecessary node expansion and testing.

When it comes to very simple problems, picking a non-heuristic based method is the best option. Calculating a heuristic is more costly than simply and quickly moving through the nodes. It is very quick to implement and run as long as the problem is simple and node creation is quick. As can be seen by the problem 3 results, more complicated problems benefit from a good heuristic based method though. The runtime is shorter than the non-heuristic based methods and there is a significant decrease in nodes created, expanded and goal tested. If the nodes become more complicated, this will lead to an even bigger benefit for the heuristic based method.