AIND Planning - Heuristic Analysis

In this document we analyze the performance of different searching algorithms on three similar planning problems. All three planning problems deal with cargo transportation, but have different start and goal states. Each problem consists of airports, planes, and cargos, of which there are a different combinations of each.

Part 1 analyzes the performance of *uninformed planning searches* (non-heuristic based planning) on the cargo transportation problems.

Part 2 analyzes the performance of the A* search algorithm, using domain independent heuristics on the same problems as in Part 1, but this time using a *planning graph* for node expansion and cost estimation.

The problems

Three initial state spaces are defined, stipulating at which airport each plane is, and at which airport each cargo is. These three different problems are referred to as `air_cargo_p1`, `air_cargo_p2` and `air_cargo_p3`.

Each problem has an optimal solution. We will use this to compare the optimality of the various searching algorithms.

See the *Problems Definition* section at the end of the document for a full description of the start state, goal state and optimal solution of each problem.

Part 1 - Uninformed heuristic analysis

In this section, we compare the performance of various *uninformed* search algorithms. An *uninformed search algorithm* is one that does not have access to any information other than the definition of the problem, and the ability to distinguish the present state from a goal state.

These search algorithms will be evaluated in terms of:

- **Expansions**: Number of nodes expanded to find the Solution
- **Goal Tests**: Number of states tested for a goal state
- **Plan Length**: The length of the solution
- **Optimality**: Is the optimal solution found?
- **Execution Time**: Time (in seconds) it took to find the proposed solution

Results:

`air_cargo_p1`:

Search Method	Expansions	Goal Tests	Plan Length	Execution Time
breadth_first_search	36	48	**6**	0.027
breadth_first_tree_search	1299	1300	**6**	0.897
depth_first_graph_search	12	13	12	0.009
depth_limited_search	97	246	50	0.089
uniform_cost_search	47	49	**6**	0.032
recursive_best_first_search	3818	3819	**6**	2.621
greedy_best_first_graph_search	**7**	**9**	**6**	**0.006**

`air_cargo_p2`:

Search Method	Expansions	Goal Tests	Plan Length	Executio Time
breadth_first_search	2093	2910	**9**	7.381
breadth_first_tree_search	--	--	--	--
depth_first_graph_search	**224**	**225**	203	**0.514**
depth_limited_search	148565	1183717	50	584.533
uniform_cost_search	3086	3088	**9**	7.791
recursive_best_first_search	--	--	--	--
greedy_best_first_graph_search	349	351	12	0.815

> Rows with -- indicate that the search algorithm took more than 10 minutes to execute

`air_cargo_p3`:

Search Method	Expansions	Goal Tests	Plan Length	Execution Time
breadth_first_search	12644	14409	**14**	67.514
breadth_first_tree_search	--	--	--	--
depth_first_graph_search	**242**	**243**	224	**0.923**
depth_limited_search	--	--	--	--
uniform_cost_search	14838	14840	**14**	46.449
recursive_best_first_search	--	--	--	--
greedy_best_first_graph_search	3615	3617	**30**	11.299

> Rows with -- indicate that the search algorithm took more than 10 minutes to execute

Analysis

From the results above, we can see that `depth_first_graph_search` and `greedy_best_first_graph_search` consistently outperform other searching algorithms in terms of expansions, goal tests and execution time. However, these algorithms do not provide an optimal solution to the problem. We can see that `breadth_first_search` and `uniform_cost_search` are the only algorithms that do provide an optimal solution.

Thus, if the requirement is to have the **shortest execution time**, and the **minimum number of expanded nodes**, the `depth_first_graph_search` algorithm is recommended for solving the air cargo planning problems. If the requirement is to have an **optimal solution** in the **least amount of time** and using **minimum memory**, the `breadth_first_search` algorithm is recommended. Although `uniform_cost_search` solves problem 3 in a shorter time, `breadth_first_search` solves problems 1 and 2 in a shorter time and would thus be our recommendation.

The motivation for the recommendations above can be confirmed by comparing the theoretical time and computational complexity, as well the completeness of an algorithm, using the following table, adapted from Peter& Norvig 3rd edition, section 3.4.7:

Search Method	Time Complexity	Computational Complexity	Complete?
breadth_first_search	$O(b^d)$	$O(b^d)$	Yes
depth_first_graph_search	$O(b^m)$	$O(bm)$	No
uniform_cost_search	$O(b^{1 + [C^*/\epsilon]})$	$O(b^{1 + [C^*/\epsilon]})$	Yes
greedy_best_first_graph_search	$O(b^m)$	$O(b^m)$	No

> Where b is the branching factor, d is the depth of the shallowest solution, m is the maximum depth of the search tree, C* is the cost of the optimal solution, and ϵ is the step cost.

Part 2 - Informed heuristic analysis

In this section we analyze the performance of the A* search algorithm using three different *informed* heuristics - heuristics which enable the algorithm to deduce some information about the current state, other than if it is a goal state or not.

The three heuristics that we will compare are:

- h_1 - Always return 1 (not a true informed heuristic, but used a benchmark for A* search)
- h_ignore_preconditions - Ignore the preconditions of the Actions
- h_pg_levelsum - Compute the sum of the levels at which each goal appears on the graph.

Results

`air_cargo_p1`:

Search Method	Expansions	Goal Tests	Plan Length	Execution Time	Opti
A* Search with h_1	47	49	6	0.032	True
A* Search with h_ignore_preconditions	37	39	6	**0.028**	True
A* Search with					

h_pg_levelsum	**11**	**13**	6	0.661	True

`air_cargo_p2` :

Search Method	Expansions	Goal Tests	Plan Length	Execution Time	Opti
A* Search with h_1	3086	3088	9	7.743	True
A* Search with h_ignore_preconditions	1049	1051	9	**2.827**	True
A* Search with h_pg_levelsum	**130**	**132**	9	114.039	True

`air_cargo_p3` :

Search Method	Expansions	Goal Tests	Plan Length	Execution Time	Opt
A* Search with h_1	14838	14840	14	47.026	True
A* Search with h_ignore_preconditions	**7969**	**7971**	14	**27.44**	True
A* Search with h_pg_levelsum	--	--	--	--	--

Analysis

From the data above, we can see that A* with `h_pg_levelsum` outperforms the rest, but unfortunately cannot calculate a solution within 10 minutes on problem 3, and thus we cannot consider this heuristic.

As for the other 2 heuristics, naturally `h_ignore_preconditions` is faster and more memory efficient than `h_1` , however, this heuristic is not practical in my opinion, because (as it states) it ignores some mechanics of the problem and is thus not feasible in reality.

Problem Definitions

`air_cargo_p1`

Initial State

```
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 State

```
Goal(At(C1, JFK) ∧ At(C2, SFO))
```

Optimal Solution

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

`air_cargo_p2`

Initial State

```
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 State

```
Goal(At(C1, JFK) ∧ At(C2, SFO) ∧ At(C3, SFO))
```

Optimal Solution

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

`air_cargo_p3`

Initial State

```
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 State

```
Goal(At(C1, JFK) ∧ At(C3, JFK) ∧ At(C2, SFO) ∧ At(C4, SFO))
```

Optimal Solution

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

```
Unload(C4, P2, SFO)
```