Heuristic Review

By Jason McDonald

For this project, we are given the following classical PDDL problems within the same domain and action schema. The problems vary in their initial state and goals

- Air Cargo Action Schema:

```
Action(Load(c, p, a),
        PRECOND: At(c, a) ∧ At(p, a) ∧ Cargo(c) ∧ Plane(p) ∧ Airport(a)
        EFFECT: ¬ At(c, a) ∧ In(c, p))
Action(Unload(c, p, a),
        PRECOND: In(c, p) ∧ At(p, a) ∧ Cargo(c) ∧ Plane(p) ∧ Airport(a)
        EFFECT: At(c, a) ∧ ¬ In(c, p))
Action(Fly(p, from, to),
        PRECOND: At(p, from) ∧ Plane(p) ∧ Airport(from) ∧ Airport(to)
        EFFECT: ¬ At(p, from) ∧ At(p, to))
```

- 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))
```

For our first set of experiments we ran uninformed planning searches for air_cargo_p1, air_cargo_p2, and air_cargo_p3 using breadth_first_search, depth_first_graph_search, and uniform_cost_search. The following results show optimality as plan length, goal tests, time elapsed, and the number of nodes expanded before the goal was reached.

air_cargo_p1	**Optimality**	**Goal Tests**	**Time Elapsed (s)**	**Node Expansions**
breadth_first_search	6	56	0.07	43
depth_first_graph_search	12	13	0.02	12
uniform_cost_search	6	57	0.09	55

air_cargo_2	**Optimality**	**Goal Tests**	**Time Elapsed (s)**	**Node Expansions**
breadth_first_search	9	4,672	25.05	3,401
depth_first_graph_search	346	351	2.34	350
uniform_cost_search	9	4,763	25.95	4,761

air_cargo_p3	**Optimality**	**Goal Tests**	**Time Elapsed (s)**	**Node Expansions**
breadth_first_search	12	17,947	136.52	14,491
depth_first_graph_search	1,878	1,949	24.59	1,948
uniform_cost_search	12	17,785	94.24	17,783

For the second set of experiments we ran informed planning searches for air_cargo_p1, air_cargo_p2, and air_cargo_p3 using astar_search with the h_1, h_ignore_preconditions, and h_pg_levelsum heuristics. The following results show optimality as plan length, goal tests, time elapsed, and the number of nodes expanded before the goal was reached.

air_cargo_p1	**Optimality**	**Goal Tests**	**Time Elapsed (s)**	**Node Expansions**
h_1	6	57	0.08	55
h_ignore_preconditions	6	43	0.06	41
h_pg_levelsum	6	13	1.15	11

air_cargo_p2	**Optimality**	**Goal Tests**	**Time Elapsed (s)**	**Node Expansions**
h_1	9	4,763	23.40	4,761
h_ignore_preconditions	9	1,452	7.26	1,450
h_pg_levelsum	9	88	188.25	86

air_cargo_p3	**Optimality**	**Goal Tests**	**Time Elapsed (s)**	**Node Expansions**
h_1	12	17,785	95.95	17,783
h_ignore_preconditions	12	5,005	27.72	5,003
h_pg_levelsum	12	313	1,028.54	311

An optimal plan for Problems 1, 2, and 3

(An) Optimal Plan		
air_cargo_p1	air_cargo_p2	air_cargo_p3
Load(C1, P1, SFO) Fly(P1, SFO, JFK) Load(C2, P2, JFK) Fly(P2, JFK, SFO) Unload(C1, P1, JFK) Unload(C2, P2, SFO)	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(C3, P3, SFO) Unload(C2, P2, SFO) Unload(C1, P1, JFK)	Load(C1, P1, SFO) Fly(P1, SFO, ATL) Load(C3, P1, ATL) Fly(P1, ATL, JFK) Load(C2, P2, JFK) Fly(P2, JFK, ORD) Load(C4, P2, ORD) Fly(P2, ORD, SFO) Unload(C4, P2, SFO) Unload(C3, P1, JFK) Unload(C2, P2, SFO) Unload(C1, P1, JFK)

Analysis of non-heuristic search results

For optimality both breadth_first_search and uniform_cost_search yielded optimal plan lengths for all problems while depth_first_graph_search failed to produce an optimal plan length. Depth_first_graph_search can only be expected to produce an optimal plan by happenstance if the first branch that contains a path that fulfills the goal also happens to do so in an optimal manner (https://youtu.be/JbJMxp3Lva4?t=1m08s). Goal test all performed similarly for both breadth_first_search and uniform_cost_search. By far the shortest compute time was for depth_first_graph_search and if all we required were *a* solution then depth_first_graph_search would actually be fine. We might even ask depth_first_graph_search to produce as many plans as it can in some amount of time and pick the one of the least length. This might even be necessary if the problem is memory limited, as we can see from the node expansion counts depth_first_graph_search also uses the least memory. Certainly though, for this exact problem, optimality is king.

Analysis of A* heuristic search results

For A* search using h_ignore_preconditions (pp. 376) and h_pg_levelsum (pp. 378) heuristics, plans with optimal length were found. The tradeoffs appear straight forward. The h_ignore_preconditions heuristic is fast yet memory intensive while h_pg_levelsum is much slower yet uses far less memory. This is expected as h_ignore_preconditions is a very relaxed heuristic and while it computes faster it is far less likely to 'point' in the right state-space direction and must, on average, explore more of the graph to find a solution. The heuristic h_pg_levelsum however is a better 'pointer' since it contains more of the problems constraints and, on average, points closer to a path that contains a goal state. Though the added rigors of h_pg_levelsum means the heuristic takes longer to compute then the more relaxed case.

Winner: h_ignore_preconditions

This heuristic runs with times comparable to the fastest non-heuristic search while simultaneously producing an optimal path. The node expansion counts stay near or lower than non-heuristic search while still finishing orders of magnitude faster than its heuristic rival h_pg_levelsum.