Air Cargo Planning – Written Analysis

04/17/17
Daniele Pestilli

This project consisted of three cargo planning problems. The first involved simply switching cargo from one airport to the other. The second involved dispatching two pieces of cargo to the SFO airport and one to the JFK airport, given three airports (ATL, JFK, SFO). The third, and most complex, involved four airports (ATL, JFK, ORD, SFO) with one package at each. These had to be redistributed with two packages at JFK and two at SFO.

The purpose of this project was to assess the various search techniques and form a judgement regarding their applicability to the aforementioned planning graphs.

Optimal Plan (Problem 1)

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

Optimal Plan (Problem 2)

Load(C1, P1, SFO)
Load(C2, P2, JFK)
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)

Optimal Plan (Problem 3)

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

Nodes Expanded

Algorithm	Problem 1	Problem 2	Problem 3
Breadth-first	43	3343	14663
Depth-first graph	12	476	1511
Greedy best first graph	7	990	5614
h_1	55	4852	18235
h_ignore_preconditions	41	1506	5118
h_pg_levelsum	41	1245	–

Goal Tests

Algorithm	Problem 1	Problem 2	Problem 3
Breadth-first	56	4609	18098
Depth-first graph	13	477	1512
Greedy best first graph	9	992	5616
h_1	57	4854	18237
h_ignore_preconditions	43	1508	5120
h_pg_levelsum	43	1247	–

Time Elapsed

Algorithm	Problem 1	Problem 2	Problem 3
Breadth-first	0.052	22.08	195.367
Depth-first graph	0.011	3.34	21.969
Greedy best first graph	0.006	10.29	190.781
h_1	0.053	74.32	627.79
h_ignore_preconditions	0.064	24.23	143.40
h_pg_levelsum	17.021	15726.18	–

Plan Length

Algorithm	Problem 1	Problem 2	Problem 3
Breadth-first	6	9	12
Depth-first graph	12	466	1442
Greedy best first graph	6	21	22
h_1	6	9	12
h_ignore_preconditions	6	9	12
h_pg_levelsum	6	9	–

Analysis

Breadth-first search seems to have consistently done better in terms of finding optimal plan lengths, while depth-first graph search resulted in rather large plan lengths. This is because breadth-first search, with its LIFO (last in first out) queue algorithm is guaranteed to find the shallowest goal state, whereas depth-first search, with its FIFO (first in first out) stack algorithm returns the first goal state it encounters, as explained in Peter Norvig's seminal book, *Artificial Intelligence: A Modern Approach*[1]. The advantage of depth-first search is that it's speedier in finding a resolution and consumes less memory with respect to breadth-first. Breadth-first search also resulted in a surprisingly large number of node expansions (14,663 compared to depth-first's 1,511 and greedy best first's 5,614). Greedy best first performs relatively well in terms of plan length and execution time for simple problems, but becomes increasingly worse as the problem becomes more complex. This is because the greedy heuristic strategy is not perfectly optimal, and prefers to take the biggest bite possible out of the remaining cost to reach a goal, without taking into consideration whether this will be an optimal solution in the long run.

It is expected that the heuristic search results all returned equal plan lengths for the cargo problems. However, the pg_levelsum heuristic search took longer than 10 minutes to execute for problem 3, but it can be assumed that once the planning graph is generated it would also find an optimal solution. This is a good improvement over the non-heuristic search results. The simple h_1 heuristic expanded a staggering 18,235 nodes for problem 3 and took over 10 minutes of execution time. The pg_levelsum heuristic is powerful in that the expanded nodes and goal tests are significantly fewer than in h_1 and also smaller than the h_ignore_preconditions heuristic. However, building the graph takes a very large amount of time and would be impractical for real world, time-critical scenarios. The best heuristic appears to be h_ignore_preconditions, as it is both efficient in terms of time and of expanded nodes. This is because dropping action preconditions from the problem is guaranteed to lead to an admissible heuristic function, as explained by Bonet and Geffner[2]. The downside of h_ignore_preconditions is that in a worst case scenario, computing the preconditions can potentially be as hard as solving the original problem.

While heuristic searches are good for returning optimal plan lengths, the number of expanded nodes can often be greater than those expanded by a non-heuristic search, which results in poor performance. In fact, non-heuristic searches consistently return results more speedily.

1 Norvig, Peter. "Problem-solving." Artificial Intelligence: A Modern Approach. By Stuart J. Russell. N.p.: Prentice Hall, 1995. N. pag. Web. 17 Apr. 2017. <http://stpk.cs.rtu.lv/sites/all/files/stpk/materiali/mi/artificial%20intelligence%20a%20modern%20approach.pdf>.

2 Bonet, Blai, and Héctor Geffner. "Planning as Heuristic Search." Artificial Intelligence 129 (2001): 5-33. Web. 17 Apr. 2017. <http://www.cs.toronto.edu/~sheila/2542/s14/A1/bonetgeffner-heusearch-aij01.pdf>.