AIND - Implementing a planning search

Robson Cardoso

April 05, 2017

Contents

1. Introduction **1**

2. Algorithms and metrics **2**
2.1. Problem 1 2
2.2. Problem 2 4
2.3. Problem 3 5

3. Summing-up **6**

1. Introduction

This project aims to solve deterministic logistics planning problems for an Air Cargo transport system using a planning search agent. Optimal plans for each problem will be computed and domain-independent heuristics will be implemented.

We're going to solve problems in the Air Cargo domain considering the following action schema:

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

The following three different problems will be considered:

Problem 1

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

Problem 2

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

Problem 3

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

2. Algorithms and metrics

We'll use uninformed and informed planning search algorithms to solve the three problems previously mentioned. The following algorithms will be considered:

Uninformed planning search algorithms:

- Breadth-first search;
- Breadth-first tree search;
- Depth-first graph search;
- Depth-limited search;
- Uniform-cost search;
- Recursive best-first search with H1;
- Greedy best-first graph search with H1.

Informed planning search algorithms:

- A* search with H1
- A* search with ignore preconditions
- A* with levelsum

Additionally, we'll provide metrics for each algorithm on number of node expansions required, number of goal tests, time elapsed, and optimality of solution.

2.1. Problem 1

This section discuss about the results when previously mentioned search algorithms are applied to Problem 1.

Greedy best-first graph search with H1 algorithm presents optimal results in the sense that it achieves the goal with less node expansions, less goal tests, less new nodes, and generates the shortest plan. By design, Greedy best-first search uses only its heuristic function to determine which node is popped from the frontier (implemented as a priority queue such that the lowest evaluation is expanded first). Here, as the heuristic function H1 returns a constant value (1), the priority queue works as a FIFO, i. e., the first expanded nodes are popped first from the frontier. In practice, this means that the algorithm always expands the oldest node in the frontier, suggesting that the results we got here are problem-dependent (depends on domain, action schema, initial state, etc.) and that this algorithm is likely to perform sub-optimal for other problems.

A with levelsum* presents the second best performance overall although it is the second slowest algorithm. *A* search with ignore preconditions* and *A* search with H1* showed similar performance although the former performed slightly better.

We can observe that *A* search with H1* showed the same results as *Uniform-cost search* algorithm. This is consistent to the fact that the heuristic function H1 used by the A* search returns a constant value which implies the search relies exclusively on the path cost to select a node for expansion. It turns out that this is the approach used by *Uniform-cost search* algorithm, explaining the similar results we got here.

Although *Depth-first graph search* is the second fastest algorithm – taking only three times the time required by the optimal algorithm to generate a plan, it produces a relatively long plan. This algorithm tries to

expand the deepest node in the frontier first, explaining why it generates a long plan. In practice, this plan is unlikely to be chosen as a long plan will take more time to be executed in the real world.

Breadth-first search and *Uniform-cost search* showed similar metrics, but the former showed a better performance. For this problem, *Breadth-first search* performs better because every action taken from any node n has equal step-cost and this algorithm always expands the shallowest unexpanded node, resulting in less node expansions and a faster execution time.

Finally, we can observe that *Depth-limited search* produced the longest plan and *Recursive best-first search with H1* showed the worst performance.

Search algorithm	Time	Plan length	Expansions	Goal tests	New nodes
Breadth-first search	0.031	6	43	56	180
Breadth-first tree search	1.004	6	1458	1459	5960
Depth-first graph search	0.015	20	21	22	84
Depth-limited search	0.105	50	101	271	414
Uniform-cost search	0.045	6	55	57	224
Recursive best-first search with H1	2.955	6	4229	4230	17023
Greedy best-first graph search with H1	0.005	6	7	9	28
A* search with H1	0.053	6	55	57	224
A* search with ignore preconditions	0.078	6	41	43	170
A* with levelsum	1.447	6	11	13	50

Following we can see the plan produced by the optimal algorithm, *Greedy best-first graph search with H1*, as well as the plans produced by *Breadth-first search* and *Depth-first graph search* algorithms. We can observe that *Greedy best-first graph search with H1* and *Breadth-first search* produced very similar plans but *Breadth-first search* expands much more nodes to achieve the goal. In contrast, *Depth-first graph search* produced a long plan with many redundant steps.

Greedy best-first graph search with H1 (optimal):

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

Breadth-first search:

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

Depth first-graph search:

Fly(P1, SFO, JFK)
Fly(P2, JFK, SFO)

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

2.2. Problem 2

This section presents the results when the same search algorithms are applied to Problem 2.

This time the best performance and optimality is given to *A* with levelsum* which achieves the goal with less expansions, less goal tests and less new nodes. In contrast, it requires more time to be executed than the other algorithms as level cost estimation requires more computation and adds execution time. *A* search with ignore preconditions* and *A* search with H1* achieved lower execution times but required a lot more node expansions and goal tests. This time, *Greedy best-first graph search with H1* presented a very poor performance which corroborates with our assumption stated in the previous section.

Again we can observe that *Uniform-cost search* and *A* search with H1* showed almost identical results for the same reason as before, and that *Breadth-first search* performed better than *Uniform-cost search* as expected.

Among uninformed search algorithms, *Depth-first graph search* presented less node expansions and goal tests, small execution time, but generated a very long plan with many redundant steps.

The algorithms *Breadth-first tree search*, *Depth-limited search*, and *Recursive best-first search with H1* took too much time (more than 10 minutes) to run and it was not possible to collect their results.

Search algorithm	Time	Plan length	Expansions	Goal tests	New nodes
Breadth-first search	16.808	9	3343	4609	30509
Breadth-first tree search	-	-	-	-	-
Depth-first graph search	4.013	619	624	625	5602
Depth-limited search	-	-	-	-	-
Uniform-cost search	53.54	9	4853	4855	44041
Recursive best-first search with H1	-	-	-	-	-
Greedy best-first graph search with H1	8.807	21	998	1000	8982
A* search with H1	53.38	9	4853	4855	44041
A* search with ignore preconditions	20.59	9	1506	1508	13820
A* with levelsum	143.23	9	86	88	841

Following we can see the plan produced by the optimal algorithm, *A* with levelsum*, as well as the plan produced by *Breadth-first search*. The plan produced by *Depth-first graph search* is not presented in this report as it is too long to be shown here.

We can observe that *A* with levelsum* and *Breadth-first search* produced very similar plans but again *Breadth-first search* expands much more nodes to achieve the same goal.

A* with levelsum (optimal):

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)

Breadth-first search:

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)

2.3. Problem 3

This section presents the results when all search algorithms are applied to Problem 3.

Again *A* with levelsum* showed the optimal results although it required more time to be executed. *A* search with ignore preconditions* and *A* search with H1* generated plans with optimal length but required more time to be executed and a lot more node expansions and goal tests to achieve the goal. *Greedy best-first graph search with H1* again generated a sub-optimal plan and required a lot of node expansions.

Again *Uniform-cost search* and *A* search with H1* showed almost identical results, *Breadth-first search* performed better than *Uniform-cost search*, and *Depth-first graph search* was fast but generated a long and redundant plan.

Finally, the algorithms *Breadth-first tree search*, *Depth-limited search*, and *Recursive best-first search with H1* took too much time (more than 10 minutes) to run and it was not possible to collect their results.

Search algorithm	Time	Plan length	Expansions	Goal tests	New nodes
Breadth-first search	125.783	12	14663	18098	129631
Breadth-first tree search	-	-	-	-	-
Depth-first graph search	2.11	392	408	409	3364
Depth-limited search	-	-	-	-	-
Uniform-cost search	513.59	12	18223	18225	159618
Recursive best-first search with H1	-	-	-	-	-
Greedy best-first graph search with H1	130.18	22	5578	5580	49150
A* search with H1	487.80	12	18223	18225	159618
A* search with ignore preconditions	131.49	12	5118	5120	45650

Search algorithm	Time	Plan length	Expansions	Goal tests	New nodes
A* with levelsum	946.53	12	414	416	3818

Following we can see the plan produced by the optimal algorithm, *A* with levelsum*, as well as the plan produced by *Breadth-first search*. The plan produced by *Depth-first graph search* is not presented in this report as it is too long to be shown here.

We can observe that *A* with levelsum* and *Breadth-first search* produced very similar plans but *Breadth-first search* expands much more nodes to achieve the same goal. A worth mentioning fact is that both plans allow for more than one cargo to be loaded into the same plane.

A* with levelsum (optimal):

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(C4, P2, SFO)
Unload(C3, P1, JFK)
Unload(C2, P2, SFO)
Unload(C1, P1, JFK)

Breadth-first graph search:

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)

3. Summing-up

Among uninformed search algorithms, considering metrics on execution time, number of expansions, number of goal tests, and plan length, *Breadth-first search* consistently showed the best results. Comparing to *Uniform-cost search*, *Breadth-first search* performs less node expansions because it applies the goal test to each node when it is generated rather than when it is selected for expansion. *Depth-first graph search* consistently generated long and redundant plans, but presented small execution times when compared to the other uninformed search algorithms due to its simpler implementation.

A with levelsum* achieved the best performance overall when compared to other informed search algorithms considering metrics on number of node expansions, number of goal tests, and plan length, but it consistently requires more time to be executed. This is the cost to calculate the sum of the level costs to achieve the goal. Additionally, *A* with levelsum* showed best results for all problems when compared to *A* search with ignore preconditions*.

Therefore, the best heuristic used for all problems was **levelsum** paired with **A*** algorithm. As a bottom line, according to *Norvig* and *Russel*, search algorithms need a good heuristic to be efficient.