Planning Search Heuristic Analysis

Aritifical Intelligence Nanodegree

March 18, 2017
Terence So

Introduction

This report compares the performance of a number of algorithms and heuristics used to solve the Air Cargo Problem defined in Planning Domain Definition Language (PDDL). The algorithms covered are:

- `breadth_first_search`
- `breadth_first_tree_search`
- `depth_first_graph_search`
- `depth_limited_search`
- `uniform_cost_search`
- `a_star_ignore_preconditions`
- `a_star_level_sum`

Air Cargo Problem

The goal of the classic air cargo problem is to deliver cargo using the available planes to its required airport destination, preferably with the minimum number of actions. There are 3 problems presented in this project. It is defined in classical PDDL with the following specifications:

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

```
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

```
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

```
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))
```

Non-heuristic Search

Problem 1 Results

Algorithm	Plan Length	Expansions	Goal Tests	New Nodes	Time Elapsed
breadth_first_search	6	43	56	180	0.0572
breadth_first_tree_search	6	1458	1459	5960	1.8333
depth_first_graph_search	12	12	13	48	0.0152
depth_limited_search	50	101	271	414	0.1502
uniform_cost_search	6	55	57	224	0.0725

Overall, `depth_first_graph_search` (DFGS) is the fastest algorithm. It is about 3x faster than `breadth_first_search` (BFS) and explores 3x fewer nodes. However, DFGS yields a suboptimal plan listed below:

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

On the other hand, BFS produces an optimal (shortest length) solution as follows:

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

This implies that DFGS will tend to find a solution faster than BFS especially if the problem requires longer sequence of actions to solve (more literals and more actions resulting in a deeper tree/graph). The downside is that it will include some superfluous states. This is because there are no restrictions other than it cannot visit any state more than once. BFS on the other hand, is conservative in that it will always explore each level completely before proceeding to the next, thus minimizing actions. This works well enough if all actions have equal cost.

The third in terms of performance is `uniform_cost_search` (UCS), which in fact is not very different from BFS in terms of implementation except that it accounts for cost instead of depth, so we can expect similar results if all actions have equal cost, as is in this case. It also arrives at an optimal solution.

The fourth, `depth_limited_search` (DLS) is somewhat interesting. It is not restricted to visiting nodes only once, so it has a tendency to see-saw back and forth between the same two states. It eventually finds a solution, but not without first reaching the maximum depth of search, as shown here:

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

.. Repeat Load-Unload 20x ..

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

Finally, the last algorithm - `breadth_first_tree_search` (BFTS) also arrives at an optimal solution but has the worst performance (32x worse than BFS). Removing the check for node visitation results in a lot of wasted time exploring the same nodes.

Problem 2 Results

Algorithm	Plan Length	Expansions	Goal Tests	New Nodes	Time Elapsed
breadth_first_search	9	3343	4609	30509	18.3881
breadth_first_tree_search	-	-	-	-	-
depth_first_graph_search	575	582	583	5211	3.8879
depth_limited_search	-	-	-	-	-
uniform_cost_search	9	4853	4855	44041	51.0000

For problem 2, the algorithms demonstrated similar albeit scaled results. Once again, DFGS arrived at a solution the quickest but it was not optimal (60x longer than necessary). BFS performed slower, but eventually found an optimal plan of length 9.

It is quite obvious that BFS scales poorly (exponentially in fact) to the number of literals and actions. This is even more pronounced with UCS, as it does additional operations comparing and minimizing costs. Both algorithms produced the following plan:

```
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)
```

BFTS and DLS were excluded due to unusually long run times.

Problem 3 Results

Algorithm	Plan Length	Expansions	Goal Tests	New Nodes	Time Elapsed
breadth_first_search	12	14663	18098	129631	121.9929
breadth_first_tree_search	-	-	-	-	-
depth_first_graph_search	596	627	628	5176	4.2825
depth_limited_search	-	-	-	-	-
uniform_cost_search	12	18223	18225	159618	381.6485

Not much can be said of the results of problem 3 except it shows further proof of the poor scalability of BFS as well as the sub-optimality of DFGS. This time BFS is now about 30x slower than DFGS, while the plan of DFGS is now 50x longer.

The optimal plan for Problem 3 is shown below:

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

Ideally, we would like to search with the speed of DGFS but the results of BFS. If possible, we can guide the search such that it is not as exhaustive as BFS, and at the same time it should take cost into account so that we can still find a plan with optimal length, unlike DGFS. This leads us into heuristic search in the next section.

Heuristic Search Results

P	Algorithm	Plan Lgth	Expansions	Goal Tests	New Nodes	Time Elapsed
1	breadth_first_search	6	43	56	180	0.0572
1	h_ignore_preconditions	6	41	43	170	0.0703
1	h_pg_levelsum	6	11	13	50	1.9528
2	breadth_first_search	9	3343	4609	30509	18.3881
2	h_ignore_preconditions	9	1506	1508	13820	15.2320
2	h_pg_levelsum	9	86	88	841	208.4576
3	breadth_first_search	12	14663	18098	129631	121.9929
3	h_ignore_preconditions	12	5118	5120	45650	96.5885
3	h_pg_levelsum	12	414	416	3818	1465.9183

It is fortunate that both heuristic searches produced the shortest possible plans for each problem. In these air cargo problems, suboptimal solutions are nonsensical since they are a waste of time and plane fuel, however fast we can derive them. For this reason, it makes sense that we only consider searches that produce optimal solutions. We shall use BFS as a benchmark to compare the performance of our heuristics searches.

In all three problems, both heuristic searches expand and explore far fewer nodes than BFS. The levelsum heuristic is by far the most efficient in terms of nodes visited. However, we can see that it is the slowest algorithm, magnitudes more so. This demonstrates that heuristic functions come at a cost. More often than not, accurate heuristics can save us node visits but can cost more cycles to compute. This is most apparent in Problem 3 levelsum heuristic, despite having only visited 1/35ths of the nodes as BFS, it took 12x as much time.

The overall winner is the ignore_preconditions heuristic. As optimal plan length increased, the heuristic slowly edged out BFS while arriving at the same solution.

Conclusion

Depth-first search, while fast, can produce suboptimal plans. On the other hand, Breadth-first search can generate optimal plans but is slow and scale poorly to the complexity of the problem.

Heuristics provide a middle ground, directing search towards the goal so that fewer nodes are explored. However, heuristic functions will come at a cost and *heavier* heuristics will slow down search despite being more accurate. Therefore, heuristics must be carefully chosen be empirically tested to ensure their benefits.