Analysis of deterministic logistics planning problems for an Air Cargo transport system using a planning search agent

Introduction

We'll try to solve 3 different problems in an Air Cargo transport system using 10 different search agents. 7 being non heuristic:

- breadth_first_search (BFS): is a simple strategy in which the root node is expanded first, then all the successors of the root node are expanded next, then their successors, and so on.
- breadth_first_tree_search: traverses a tree using a queue to achieve breadth-first order.
- depth_first_graph_search (DFS): always expands the deepest node in the current frontier of the search tree.
- depth_limited_search: we supply depth-first search with a predetermined depth limit l.
- uniform_cost_search (UCS): demands the use of a priority queue. Recall that Depth First Search used a priority queue with the depth upto a particular node being the priority and the path from the root to the node being the element stored. The priority queue used here is similar with the priority being the cumulative cost upto the node. Unlike Depth First Search where the maximum depth had the maximum priority, Uniform Cost Search gives the minimum cumulative cost the maximum priority.
- recursive_best_first_search: Recursive Best-First Search is a search algorithm that acts in a way similar to the Standard Best-First Search. It is also very similar in design to the recursive depth-first search algorithm, although there is one significant difference. The RBFS algorithm works by keeping track of an upper bound. This upper bound allows the algorithm to "choose" better paths rather than continuing indefinitely down the same one. The upper bound is set to the f-value of the best alternative path from the ancestor of the current node. The current node is then expanded and the child nodes investigated. If all the child nodes exceed the upper bound then the current nodes f-value is set to the best child nodes f-value, and the best alternative path (i.e. the upper bound) is visited. Should one or more of the child nodes be less than the upper bound the node with the smallest f-value is visited and the upper bound is set to the lowest alternative (which could mean it does not change). One major flaw of this algorithm is that it can visit the same node several times.
- greedy_best_first_graph_search: a search with a heuristic that attempts to predict how close the end of a path is to a solution, so that paths which are judged to be closer to a solution are extended first. Using a greedy algorithm, expand the first successor of the parent. After a successor is generated:
    i. If the successor's heuristic is better than its parent, the successor is set at the front of the queue (with the parent reinserted directly behind it), and the loop restarts.
    ii. Else, the successor is inserted into the queue (in a location determined by its heuristic value). The procedure will evaluate the remaining successors (if any) of the parent.

3 being heuristic: heuristic can be derived by defining a relaxed problem that is easier to solve.

- astar_search with h_1: heuristic returning the cost of 1 (not a real heuristic).
- astar_search with h_ignore_preconditions: the ignore preconditions heuristic drops all preconditions from actions.
- astar_search with h_pg_levelsum: The level sum heuristic, following the subgoal independence assumption, returns the sum of the level costs of the goals.

We'll then analyze the results and propose the optimal solution for each problem.

Problem 1

Introduction

In this problem 1, there are 12 fluents, or state variables, which means our state space could be as large as 2to12 = 4096

Initial states 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))
```

Results

Non heuristics

Search Method	Expansions	Goal tests	New nodes	Plan length	Time spent
breadth_first_search	43	56	180	6	0.039
breadth_first_tree_search	1458	1459	596043	6	1.152
depth_first_graph_search	21	22	84	20	0.016
depth_limited_search	101	271	414	50	0.126
uniform_cost_search	55	57	224	6	0.051
recursive_best_first_search	4229	4230	17023	6	3.276
greedy_best_first_graph_search	7	9	28	6	0.005

Heuristics

Search Method	Expansions	Goal tests	New nodes	Plan length	Time spent
astar_search with h_1	55	57	224	6	0.054
astar_search with h_ignore_preconditions	41	43	170	6	0.081
astar_search with h_pg_levelsum	11	13	50	6	2.122

Optimal plan

The greedy_best_first_graph_search algorithm is the best algorithm generating an optimal plan:

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

Analysis

Many heuristics achieve the optimal length under a second of calculation time. This is the greedy best first search which achieve the best result. Let's notice that the breadth_first_search and depth_first_graph_search are very efficient too. With such a little state space, A* algorithms are not as efficient even if astar_search with h_ignore_preconditions performs very well.

Problem 2

Introduction

In this problem 2, there are 27 fluents, or state variables, which means our state space could be as large as 2to27 = 134 217 728

Initial states 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))
```

Results

Non heuristics

Search Method	Expansions	Goal tests	New nodes	Plan length	Time spent
breadth_first_search	3343	4609	30509	9	16.962
breadth_first_tree_search	ABORTED	ABORTED	ABORTED	ABORTED	ABORTED
depth_first_graph_search	624	625	5602	619	4.029
depth_limited_search	ABORTED	ABORTED	ABORTED	ABORTED	ABORTED
uniform_cost_search	4780	4782	43381	9	52.526
recursive_best_first_search	ABORTED	ABORTED	ABORTED	ABORTED	ABORTED
greedy_best_first_graph_search	598	600	5382	17	3.982

Heuristics

Search Method	Expansions	Goal tests	New nodes	Plan length	Time spent
astar_search with h_1	4780	4782	43381	9	52.049
astar_search with h_ignore_preconditions	1506	1508	13820	9	16.628
astar_search with h_pg_levelsum	86	88	841	9	230.865

Optimal plan

The heuristic a* with ignore preconditions is the one providing the optimal plan

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

Analysis

In this problem, the state space is bigger. We see breadth_first_tree_search, depth_limited_search and recursive_best_first_search which have been aborted because of their performance. breadth_first_search and uniform_cost_search achieve the optimal plan length, which is 9. In this case, this is the astar_search with h_ignore_preconditions heuristic which performs the best, even if breadth_first_search is very close.

Problem 3

Introduction

In this problem 3, there are 32 fluents, or state variables, which means our state space could be as large as 2to32 = 4 294 967 296

Initial states 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))
```

Results

Non heuristics

Search Method	Expansions	Goal tests	New nodes	Plan length	Time spent
breadth_first_search	14663	18098	12963	12	124.733
breadth_first_tree_search	ABORTED	ABORTED	ABORTED	ABORTED	ABORTED
depth_first_graph_search	408	409	3364	392	2.035
depth_limited_search	ABORTED	ABORTED	ABORTED	ABORTED	ABORTED
uniform_cost_search	17882	17884	156769	12	499.634
recursive_best_first_search	ABORTED	ABORTED	ABORTED	ABORTED	ABORTED
greedy_best_first_graph_search	4498	4500	39970	26	102.867

Heuristics

Search Method	Expansions	Goal tests	New nodes	Plan length	Time spent
astar_search with h_1	17882	17884	156769	12	491.435
astar_search with h_ignore_preconditions	5114	5116	45610	12	116.506
astar_search with h_pg_levelsum	404	406	3718	12	1617.218

Optimal plan

The heuristic a* with ignore preconditions is the one providing the optimal plan

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

Analysis

In this problem 3, we see once again good performance from breadth_first_search, only beaten by relaxing the problem with the heuristic astar_search with h_ignore_preconditions, achieving much less expansions and goal tests (but being less cost effective, the time spent is almost the same with breadth_first_search).

Conclusion

I think that for limited state spaces, non heuristic algorithms are efficient, as we've seen for these 3 problems. breadth_first_search, being quite a simple algorithm is very efficient in the 3 contexts. With state spaces increasing, I think that the heuristic with relaxed problems will prove to be much more efficient. The astar_search with h_ignore_preconditions limits the expansions while being CPU time efficient.