Graph Planning
Air-Cargo Problem Heuristics

This report describes and compares two main approaches to solving a planning problem – one approach using uninformed graph-search algorithms, and the other using the A* graph search algorithm with heuristics derived from the problem description.

1 'Air-Cargo Problem': Description and Solutions

We're trying to solve 3 instances of what is known as the air-cargo problem, given by this action schema:

Action(Load(c, p, a),
PRECOND: $At(c, a) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$
EFFECT: $\neg At(c, a) \land In(c, p))$

Action(Unload(c, p, a),
PRECOND: $In(c, p) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$
EFFECT: $At(c, a) \land \neg In(c, p))$

Action(Fly(p, from, to),
PRECOND: $At(p, from) \land Plane(p) \land Airport(from) \land Airport(to)$
EFFECT: $\neg At(p, from) \land At(p, to))$

Note that the number of state spaces in this problem can be extended indefinitely – a plane can fly endlessly from one airport to another, or cargo can be repeatedly loaded and offloaded from a plane. In other words, the search graph of problem states is infinite.

1.1 Problem 1: 2 planes, airports and packages

The initial and goal states of this problem are:

Init($At(C1, SFO) \land At(C2, JFK) \land$
$At(P1, SFO) \land At(P2, JFK) \land$
$Cargo(C1) \land Cargo(C2) \land$
$Plane(P1) \land Plane(P2) \land$
$Airport(JFK) \land Airport(SFO))$

Goal($At(C1, JFK) \land At(C2, SFO))$

The problem can be solved with the following 6 steps:

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

1.2 Problem 2: 3 planes, airports and packages

Init($At(C1, SFO) \land At(C2, JFK) \land At(C3, ATL)$
$At(P1, SFO) \land At(P2, JFK) \land At(P3, ATL)$
$Cargo(C1) \land Cargo(C2) \land Cargo(C3)$
$Plane(P1) \land Plane(P2) \land Plane(P3)$

Airport(JFK) ∧ Airport(SFO) ∧ Airport(ATL))

Goal(At(C1, JFK) ∧ At(C2, SFO) ∧ At(C3, SFO))

The problem can be solved with the following 9 steps:

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

1.3 Problem 3: 2 planes, 4 airports and packages

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))

Goal(At(C1, JFK) ∧ At(C2, SFO) ∧ At(C3, JFK) ∧ At(C4, SFO))

This problem can be solved with the following 12 steps:

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

2 Algorithm Comparison

To make the comparison of the various algorithms and problems easier, results from the appendix, where an algorithm was able to find an optimal (i.e. shortest) solution, were normalised by the lowest/smallest/best result for their respective problem, and displayed graphically on the next page, as well as presented in the following table.

Problem	Algorithm			Normalised Memory	Normalised Time
1	1	○	Breadth-first	2.048	11.424
1	2	×	Breadth-first Tree	69.429	341.681
1	5	+	Uniform Cost	2.857	13.138
1	6	*	Recursive Best-first (Constant Heuristic)	201.381	1011.507
1	8	□	A* (Constant Heuristic)	2.857	15.076
1	9	◇	A* (Ignore Preconditions Heuristic)	2.381	1.000
1	10	☆	A* (Planning Graph LevelSum Heuristic)	1.333	145.274
2	1	○	Breadth-first	9.364	13.446
2	5	+	Uniform Cost	14.437	1.431
2	8	□	A* (Constant Heuristic)	14.437	1.418
2	9	◇	A* (Ignore Preconditions Heuristic)	6.910	1.000
2	10	☆	A* (Planning Graph LevelSum Heuristic)	1.000	188.648
3	1	○	Breadth-first	39.737	86.790
3	5	+	Uniform Cost	50.163	3.266
3	8	□	A* (Constant Heuristic)	50.163	3.477
3	9	◇	A* (Ignore Preconditions Heuristic)	20.022	2.022
3	10	☆	A* (Planning Graph LevelSum Heuristic)	1.000	220.268

Normalised results for all algorithms which were able to find the shortest solutions.

Shaded figures, or points on the chart joined by solid lines, indicate results which lie on the dominance (Pareto) frontier of their problem class – which means there were no algorithms which were able to find an optimal solution in both less time and fewer node expansions than these highlighted.

Looking at the chart and the dominance frontiers, we quickly note that the informed search algorithms are superior. Two informed algorithms appear on the frontier for all 3 problems: the “A* (Ignore Preconditions Heuristic)” and the “A* (Planning Graph LevelSum Heuristic)” algorithm. Using the ignore preconditions heuristic is the fastest, but uses more memory (in other words, performs more node expansions), while using the level-sum heuristic is slower, but uses less memory. The 3rd informed search algorithm (using the constant heuristic) is also always relative close to the dominance frontier.

Of the uninformed strategies, only the Breadth First search (BFS) and Uniform Cost search (UCS) algorithms produce results within the scale of the displayed graph. BFS is better (with respect to the given metrics of memory/time) than UCS (in the case of problem 1, BFS even lies on the Pareto front.) What's interesting to see, on the chart, is that the uninformed UCS algorithm has almost the same performance characteristics as the informed A* search with a constant heuristic.

In fact, BFS and UCS were the only algorithms that were a) able to terminate in a 'reasonable' amount of time and b) reliably find optimal plans, for all 3 problems. Other uninformed algorithms were soon overwhelmed by the exponential explosion of their search space (Breadth-first tree, Recursive Best First), weren't able to find optimal plans within their allocated resources (the Depth First and Depth Limited algorithms), or didn't reliably find the optimal solution (see the appendix for the observations on the Greedy Best First algorithm.)

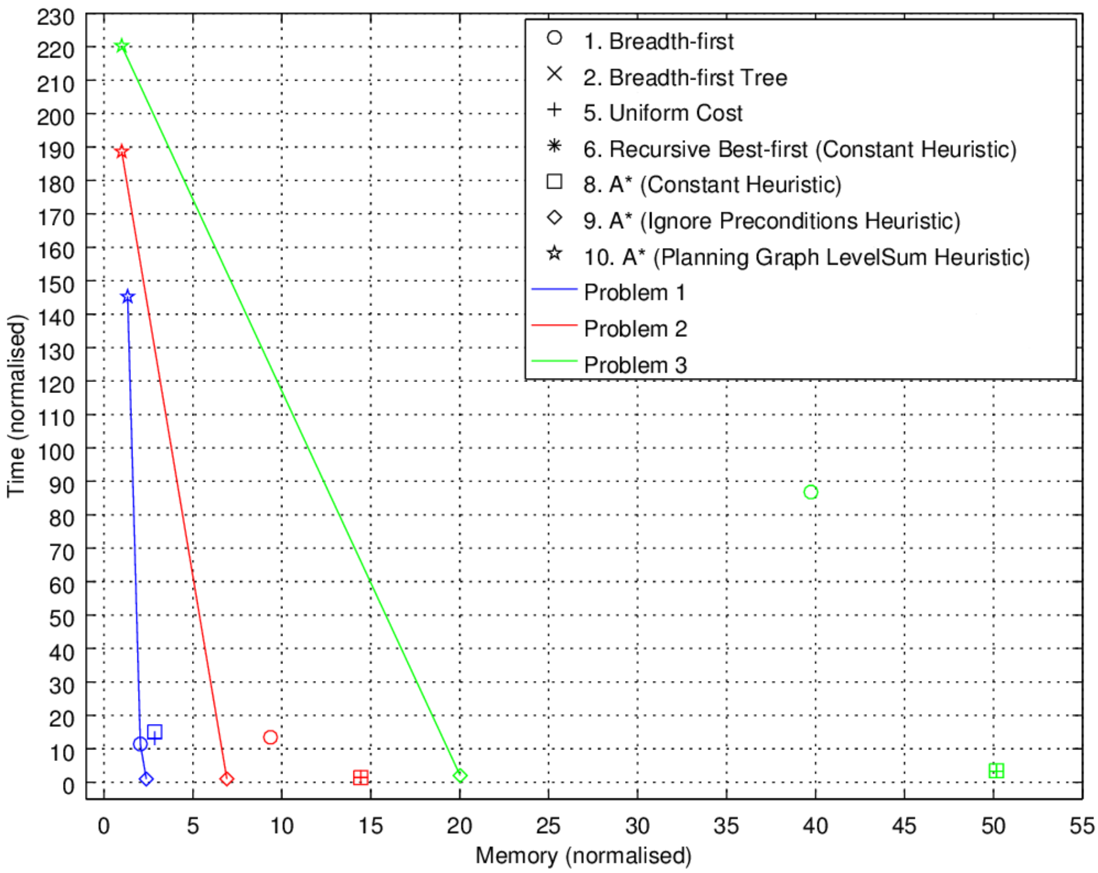

Chart showing normalised performance results for various algorithms on the 3 air-cargo problems, with the best-in-class algorithms/heuristics marked on the dominance frontier.

All the observations above are supported by references to sections from Russell, Norvig, *Artificial Intelligence: A Modern Approach*, (3rd ed, Prentice Hall, 2010), abbreviated as AIMA.

The observation that the informed, A* search strategies are more efficient than the uninformed ones is also noted in AIMA §3.5, with the statement "an informed search strategy can find solutions more efficiently than can an uninformed strategy.".

The fact that BFS performed better than UCS is also supported by Figure 3.21 in AIMA, which state that the time and space complexity of BFS is $O(b^d)$ (with *b* being the branching factor and *d* the depth of the shallowest solution) while UCS is $O(b^{1+\lfloor C^*/\epsilon \rfloor})$, with C* being the cost of the optimal solution.

At best, when all step costs are equal, that reduces to $O(b^{1+d})$, and UCS is similar to BFS. However, BFS stops as soon as it finds a goal, while UCS, even if it has found the same goal, will search all the remaining nodes at that goal's depth, to see if it can find a better solution. This explains why, in the chart, results for UCS are always to the right (more node expansions) than those for BFS.

The fact that BFS and UCS were the only uniformed algorithms to reliably finish, with an optimal solutions, is also confirmed by AIMA fig. 3.2, which states that those are the only two algorithms (of the ones considered here) that are optimal (return an optimal solution) and complete (can be guaranteed to finish if a solution exists.)

The fact that the performance of UCS is similar to A* with a constant heuristic, is because they are, essentially, the same algorithm. Both algorithms evaluate a path cost, *g(n)*, but A* incorporates an estimate of cost of the path from the current node to the goal, *h(n)*. Here *h(n)* is set to a constant. So UCS evaluates *g(n)*, while the A* algorithm evaluates *g(n) + C*. The extra computation would take a little more time, which is why the A* constant algorithm's results are always on, or slightly higher, those for UCS.

3 Conclusion

For the given problems, the A* search strategy, with the ignore preconditions heuristic or the LevelSum heuristic, perform better – the former is quicker, the latter uses less memory. Of all the uninformed search strategies examined, Breadth First Search is the preferred algorithm to use, with performance characteristics somewhere in-between those of the 2 informed heuristics discussed. There's no need to use the constant heuristic with the A* search algorithm for this problem, as the uninformed Uniform Cost Search algorithm has almost identical performance and finds the same optimal plan.

4 Appendix: Result Tables

The outputs of the 'run_search.py' script, for each of the 3 problems, are summarised in the tables below. A grey row, with no results, indicates that the search script took longer than 10 minutes on my computer to find a solution.

Some search algorithms involve an element of chance, so the results might vary in different runs. For example, for problem 1, on one run the Greedy Best First search (#7) produced an optimal (6-step) solution plan with only 7 node expansions, instead of the 56 expansions shown in the table. However, as the 56-expansion solution came up more frequently in runs, that is the result that is reported here. Similar behaviour was also observed for that A* (Constant Heuristic) algorithm.

#	Algorithm	Expansions	Goal Tests	New Nodes	Plan Length	Elapsed Time (s)
1	Breadth-first	43	56	180	6	0.041
2	Breadth-first Tree	1458	1459	5960	6	1.240
3	Depth-first Graph	21	22	84	20	0.018
4	Depth-limited	101	271	414	50	0.030
5	Uniform Cost	60	62	240	6	0.048
6	Recursive Best-first (Constant Heuristic)	4229	4230	17023	6	3.672
7	Greedy Best-first (Constant Heuristic)	56	58	224	9	0.007
8	A* (Constant Heuristic)	60	62	240	6	0.055
9	A* (Ignore Preconditions Heuristic)	50	52	206	6	0.004
10	A* (Planning Graph LevelSum Heuristic)	28	30	122	6	0.527

Performance characteristics for various algorithms while solving problem 1

#	Algorithm	Expansions	Goal Tests	New Nodes	Plan Length	Elapsed Time (s)
1	Breadth-first	3343	4609	30509	9	5.527
2	Breadth-first Tree					
3	Depth-first Graph	624	625	5602	619	2.048
4	Depth-limited	222719	2053741	2054119	50	19.006
5	Uniform Cost	5154	5156	46618	9	0.588
6	Recursive Best-first (Constant Heuristic)	61078392	6E+007	6E+008	9	5304.328
7	Greedy Best-first (Constant Heuristic)	4455	4457	40095	14	0.494
8	A* (Constant Heuristic)	5154	5156	46618	9	0.583
9	A* (Ignore Preconditions Heuristic)	2467	2469	22522	9	0.411
10	A* (Planning Graph LevelSum Heuristic)	357	359	3462	9	77.544

Performance characteristics for various algorithms while solving problem 2

#	Algorithm	Expansions	Goal Tests	New Nodes	Plan Length	Elapsed Time (s)
1	Breadth-first	14663	18098	129631	12	57.778
2	Breadth-first Tree					
3	Depth-first Graph	408	409	3364	392	0.666
4	Depth-limited					
5	Uniform Cost	18510	18512	161936	12	2.174
6	Recursive Best-first (Constant Heuristic)					
7	Greedy Best-first (Constant Heuristic)	15392	15394	134640	15	1.889
8	A* (Constant Heuristic)	18510	18512	161936	12	2.315
9	A* (Ignore Preconditions Heuristic)	7388	7390	65711	12	1.346
10	A* (Planning Graph LevelSum Heuristic)	369	371	3403	12	146.637

Performance characteristics for various algorithms while solving problem 3