AIND-Planning Search

Heuristic function analysis

Cheng Wang

Provide an optimal plan for Problems 1, 2, and 3.

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))

Optimal Plan:

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

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))

Optimal Plan:

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(C3, P3, SFO)
Unload(C2, P2, SFO)
Unload(C1, P1, JFK)

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))

Optimal Plan:

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

Compare and contrast non-heuristic search result metrics (optimality, time elapsed, number of node expansions) for Problems 1,2, and 3. Include breadth-first, depth-first, and at least one other uninformed non-heuristic search in your comparison; Your third choice of non-heuristic search may be skipped for Problem 3 if it takes longer than 10 minutes to run, but a note in this case should be included.

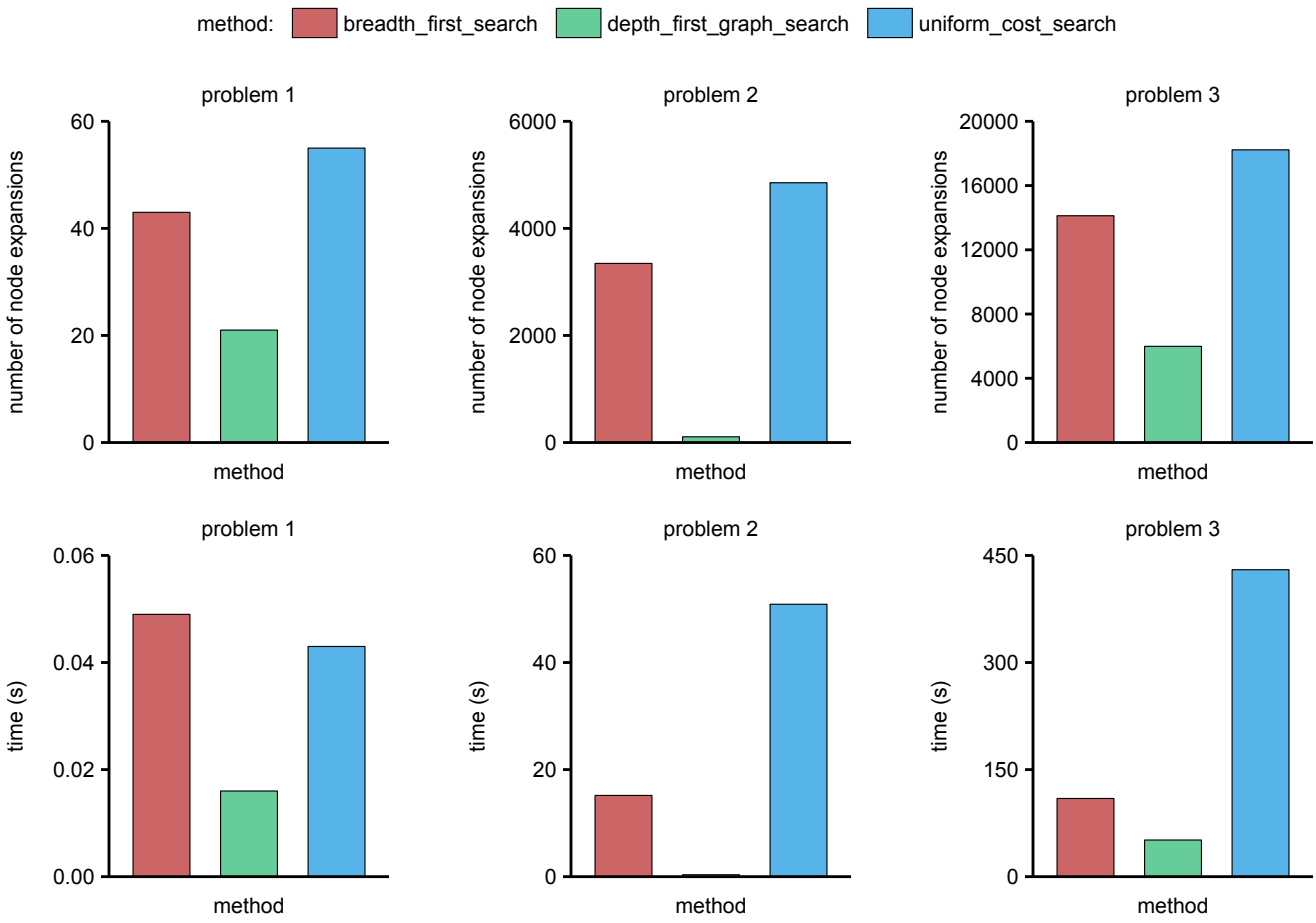

Figure 1. Comparison among three non-heuristic search method (breadth-first search, depth-first graph search, and uniform cost search) in number of node expansions and time for three problems.

In Figure 1, the number of node expansions and time elapsed are compared for each problem with three search method breadth-first search (BFS), depth-first graph search (DFGS), and uniform cost search (UCS). The other two approaches - breadth first tree search and depth limited search - are running more than 10 min for problem 2 and 3 and they are not included for comparison. The full data is shown in Table A1 at the end of this report.

Three metrics of searching algorithms (i.e. optimality, space complexity, and time complexity) can be analyzed in theory and in practice here. BFS and UCS are guaranteed to find the optimal solution given the condition in our problems in theory and the results in Table A1 confirm this property of these two algorithms. However, DFGS is not guaranteed to find the optimal solution and it could return the non-optimal solution first. DFGS does not return the optimal solutions for three problems here.[1,2]

The space complexity for this three algorithms in theory are in an order: DFGS ($O(bm)$) < BFS ($O(b^d)$) < UCS ($O(b^{d+1})$ when step costs are equal), where b is the branching factor, d is the depth of the shallowest solution, and m is maximum depth of the search tree.[1] From the results, we can see that the expansion of nodes for DFGS is the smallest among these three algorithms since the space complexity is linear for DFGS while the other two algorithms are exponential.

The time complexity for this three algorithms in theory are: DFGS is $O(b^m)$, BFS is $O(b^d)$, and UCS is $O(b^{d+1})$ when step costs are equal.[1] The BFS is faster than UCS and this can be inferred from problem 2 and 3. In problem 1, BFS and UCS are similar but the time is really short in millisecond time scale. The speed comparison for BFS and DFGS is complicate depending on *m* and *d* and the results indicate that DFGS is extremely faster than BFS for all three problems.

Overall, we can see that BFS and UCS can guarantee to find the optimal solution with exponential space complexity. DFGS cannot guarantee to find the optimal solution but it can get the solution with linear space complexity. For the problems with extremely high space complexity which are impossible to fit in the memory, DFGS is a good method for them.

Compare and contrast heuristic search result metrics using A with the "ignore preconditions" and "level-sum" heuristics for Problems 1, 2, and 3.*

A* search with ignore precondition and level-sum heuristic are shown in Table 1. A* with ignore preconditions returns optimal solution for all three problems, while A* with level-sum only return optimal solution for problem 1 before timeout (10 min). For problem 1, A* with ignore preconditions is faster but more node expansions than level-sum heuristic.

Table 1 A* with Ignore Precondition and Level Sum Heuristic

Problem	Method and Heuristic	Expansions	Time (s)	Optimal
1	astar_search with h_ignore_preconditions	41	0.052	Yes
1	astar_search with h_pg_levelsum	11	4.54	Yes
2	astar_search with h_ignore_preconditions	1506	15.922	Yes
2	astar_search with h_pg_levelsum	NA	NA	NA
3	astar_search with h_ignore_preconditions	5118	97.298	Yes
3	astar_search with h_pg_levelsum	NA	NA	NA

What was the best heuristic used in these problems? Was it better than non-heuristic search planning methods for all problems? Why or why not?

A* with ignore precondition is the fastest and the most space efficient search method across all problems with optimal solution returned. It is also better than the non-heuristic search method. Ignore precondition is an admissible heuristic and it relaxes the problem for A* to search the optimal solution with efficiently.

Reference

[1] Russell, S.J.; and Norving, P. *Artificial Intelligence: A Modern Approach* 3rd ed. (2009).

[2] Web: http://stackoverflow.com/questions/21371921/solving-a-puzzle-game-using-ai/21380001#21380001

Appendix

Table A1. Overall Results for Three Problems

	Heuristic	Expansions	Goal Tests	New Nodes	Plan length	Time (s)	Optimal
1	breadth_first_search	43	56	180	6	0.049	Yes
1	breadth_first_tree_search	1458	1459	5960	6	1.055	Yes
1	depth_first_graph_search	21	22	84	20	0.016	No
1	depth_limited_search	101	271	414	50	0.111	No
1	uniform_cost_search	55	57	224	6	0.043	Yes
1	recursive_best_first_search with h_1	4229	4230	17023	6	3.074	Yes
1	greedy_best_first_graph_search with h_1	7	9	28	6	0.006	Yes
1	astar_search with h_1	55	57	224	6	0.044	Yes
1	astar_search with h_ignore_preconditions	41	43	170	6	0.052	Yes
1	astar_search with h_pg_levelsum	11	13	50	6	4.54	Yes
2	breadth_first_search	3346	4612	30534	9	15.174	Yes
2	breadth_first_tree_search	NA	NA	NA	NA	NA	NA
2	depth_first_graph_search	107	108	959	105	0.357	No
2	depth_limited_search	NA	NA	NA	NA	NA	NA
2	uniform_cost_search	4853	4855	44041	9	50.883	Yes
2	recursive_best_first_search with h_1	NA	NA	NA	NA	NA	NA
2	greedy_best_first_graph_search with h_1	998	1000	8982	21	8.042	No
2	astar_search with h_1	4853	4855	44041	9	48.98	Yes
2	astar_search with h_ignore_preconditions	1506	1508	13820	9	15.922	Yes
2	astar_search with h_pg_levelsum	NA	NA	NA	NA	NA	NA
3	breadth_first_search	14120	17673	123964	12	109.546	Yes
3	breadth_first_tree_search	NA	NA	NA	NA	NA	NA
3	depth_first_graph_search	5990	5991	48926	2417	51.331	No
3	depth_limited_search	NA	NA	NA	NA	NA	NA
3	uniform_cost_search	18223	18225	158174	12	430.029	Yes
3	recursive_best_first_search with h_1	NA	NA	NA	NA	NA	NA
3	greedy_best_first_graph_search with h_1	5613	5615	48684	26	95.058	No
3	astar_search with h_1	18223	18225	158174	12	437.029	Yes
3	astar_search with h_ignore_preconditions	5118	5120	45475	12	97.298	Yes
3	astar_search with h_pg_levelsum	NA	NA	NA	NA	NA	NA