Planning Search Heuristic Analysis
By Amita Kapoor

In this project, we implement a planning search agent to solve deterministic logistics planning problems for an Air Cargo transport system. We use a **planning graph** and **automatic domain-independent heuristics with A* search**. As part of heuristic analysis we compare our result against different **uninformed non-heuristic search methods**.

We should solve three planning problems:

Planning Problem 1: It is defined as:

Initial state: 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 State: At(C1, JFK) ∧ At(C2, SFO)

Solution:

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

Planning Problem 2: It is defined as:

Initial state: 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 State: At(C1, JFK) ∧ At(C2, SFO) ∧ At(C3, SFO)

Solution:

- 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)

Planning Problem 3: It is defined as:

Initial state: 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(JFK)

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

Solution:

- 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)

Starting from the initial state there are many possible paths that lead to goal state. The solution listed above is the optimal solution to each air cargo problem.
Let us now explore the performance of different search techniques and heuristics to solve this problem.

Planning Problem 1: Analysis:

Uninformed searches (Ones using no Heuristics):						
Search Algorithm	**Expansions**	**Goal Tests**	**New Nodes**	**Plan Length**	**Time Taken**	**Optimal**
breadth_first_search	43	56	180	6	0.0241	Yes
breadth_first_tree_search	1458	1459	5960	6	0.7344	Yes
depth_first_graph_search	12	13	48	12	0.0063	No
depth_limited_search	101	271	414	50	0.0691	No
uniform_cost_search	55	57	224	6	0.02926	Yes
Informed searches (use some form of heuristic function)						
recursive_best_first_search with h_1	4229	4230	17029	6	2.1864	Yes
greedy_best_first_graph_search with h_1	7	9	28	6	0.0038	Yes (locally)
astar_search with h_1	55	57	224	6	0.0307	Yes
astar_search with h_ignore_preconditions	41	43	170	6	0.0303	Yes
astar_search with h_pg_levelsum	11	13	50	6	0.7610	Yes

***Analysis*:** A good search strategy is one that is optimal, fast and takes less memory. In our case the fastness of each algorithm is measured though time taken, and memory requirement is decided by the node expansions. An algorithm is optimal if it generates the best possible solution. In the case of Problem 1 the best solution is of Path length 6.

From the table above we can see that except depth first search and depth limited search all other algorithms considered for analysis are optimal. Depth first search algorithm expands the least number of nodes (48 in present case) but does not yield an optimal path. The least time is taken by Greedy best first search.

If we compare the three A* algorithms each using different heuristics, then based on the compromise between memory and time, the A* with Heuristic (ignore preconditions) "*which estimates the minimum number of actions that must be carried out from the current state in order to satisfy all of the goal conditions by ignoring the preconditions required for an action to be executed*" is the best.

Planning Problem 2: Analysis:

Uninformed searches (Ones using no Heuristics):						
Search Algorithm	**Expansions**	**Goal Tests**	**New Nodes**	**Plan Length**	**Time Taken**	**Optimal**
breadth_first_search	3343	4609	30509	9	11.55	Yes
breadth_first_tree_search						
depth_first_graph_search	582	583	5211	575	2.53	No
depth_limited_search						
uniform_cost_search	4852	4854	44030	9	9.88	Yes
Informed searches (use some form of heuristic function)						
recursive_best_first_search with h_1						
greedy_best_first_graph_search with h_1	990	992	8910	17	2.03	No
astar_search with h_1	4852	4854	44030	9	9.93	Yes
astar_search with h_ignore_preconditions	1450	1452	13303	9	3.51	Yes
astar_search with h_pg_levelsum	86	88	841	9	148.37	Yes

Analysis:
In the case of Problem 2 the best solution is of Path length 9.

From the table above we can see that except depth first search and depth limited search all other algorithms considered for analysis are optimal. A* algorithm with Heuristic defined as the "sum of all actions that must be carried out from the current state in order to satisfy each individual goal condition" expands the least number of nodes (841 in present case) but take computational time much higher than the rest. The least time was taken by Greedy best first search, though it failed to give an optimal solution. {The table excludes results for Breadth first tree search, depth limited search and recursive best first search, the execution of these despite the using the updated version went beyond 10 minutes on my system}

If we compare the three A* algorithms each using different heuristics, then based on the compromise between memory and time, the A* with Heuristic (ignore preconditions) *which estimates the minimum number of actions that must be carried out from the current state in order to satisfy all of the goal conditions by ignoring the preconditions required for an action to be executed* is the best

Planning Problem 3: Analysis:

Uninformed searches (Ones using no Heuristics):						
Search Algorithm	**Expansions**	**Goal Tests**	**New Nodes**	**Plan Length**	**Time Taken**	**Optimal**
breadth_first_search	14663	18098	129631	12	80.4	Yes
breadth_first_tree_search						
depth_first_graph_search	687	628	5176	596	2.6	No
depth_limited_search						
uniform_cost_search	18235	18237	159716	12	42.2	Yes
Informed searches (use some form of heuristic function)						
recursive_best_first_search with h_1						
greedy_best_first_graph_search with h_1	5614	5616	8910	22	13.1	No

astar_search with h_1	18235	18237	159716	12	42.10	Yes
astar_search with h_ignore_preconditions	5040	5042	44944	12	13.57	Yes
astar_search with h_pg_levelsum	318	320	2934	12	796.3	Yes

Analysis: In the case of Problem 3 the best solution is of Path length 12.

From the table above we can see that except depth first search and depth limited search all other algorithms considered for analysis are optimal. A* algorithm with Heuristic defined as the "sum of all actions that must be carried out from the current state in order to satisfy each individual goal condition" expands the least number of nodes (2934 in present case) but take computational time much higher than the rest. The least time was taken by depth first search, though it failed to give an optimal solution. {The table excludes results for Breadth first tree search, depth limited search and recursive best first search, the execution of these despite the using the updated version went beyond 10 minutes on my system}

If we compare the three A* algorithms each using different heuristics, then based on the compromise between memory and time, the A* with Heuristic (ignore preconditions) "*which estimates the minimum number of actions that must be carried out from the current state in order to satisfy all of the goal conditions by ignoring the preconditions required for an action to be executed"* is the best.

Conclusion

From above we can conclude the search algorithms using informed search are better as compared to uninformed search. The benefits are significant both in terms of speed and memory usage. Moreover, from the performance of three different heuristics with A* we learn that it is possible to customize the trade-off between speed and memory usage by using different heuristics.