HEURISTIC ANALYSIS

Dinesh Bhat
bvdinesh@outlook.com

Table of Contents

Implementing an Air Cargo transport system using a planning search agent

For this application, I have used different search agents to solve the deterministic logistics planning problems for an Air Cargo transport system using a planning search agent. This solution provides the non-heuristic and domain Domain-independent heuristics comparison, justification and analysis about search agents. To solve the planning search agent there are several non-heuristic search methods (breadth-first, depth-first, etc.) are used.

The Planning Problems

There were three problems in the Air Cargo domain, that use the same action schema defined, but different initial states and goals.

Air Cargo 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 initial state 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))
```

- Problem 2 initial state 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))
```

- Problem 3 initial state 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))
```

The above problem can be defined with actions: Load, Unload and Fly. So, the problems can be represented using optimal sequence of actions for problems 1,2 and 3 are shown below

Problem 1	Problem 2	Problem 3
Load (C1, P1, SFO) Load (C2, P2, JFK) Fly (P1, SFO, JFK) Unload (C1, P1, JFK) Fly (P2, JFK, SFO) Unload (C2, P2, SFO)	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)	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) Fly (P2, ORD, SFO) Unload (C4, P2, SFO) Unload (C3, P1, JFK) Unload (C2, P2, SFO) Unload (C1, P1, JFK)

- ## Uninformed/blind Search Strategies

Per ref (2), The uninformed/blind search while searching no clue whether one non-goal state is better than any other. The total search space is looked for the solution. All they can do is generate successors and distinguish a goal state from a non-goal state.

The following are the uninformed search

1. Breadth First Search
2. Depth First Search
3. Depth Limited Search

In this section, I have captured metrics on number of node expansions required, number of goal tests, time elapsed, and optimality of solution for each search algorithm. The various search metrics are captured were collected using the following commands:

```
python run_search.py -p 1 -s 1 2 3 4 5 6 7 >> run_uninformed_search_results_p1.txt
python run_search.py -p 2 -s 1 3 5 7 >> run_uninformed_search_results_p2.txt
python run_search.py -p 3 -s 1 3 5 7 >> run_uninformed_search_results_p3.txt
```

In below table which are all execution time, expansions and smaller path are better, mentioned as Yes in the Is Optimal column.

Uninformed Problem 1 Results

Search Types	Expansions	Goal Tests	New Node	Plan Length	Time Elapsed (s)	Is Optimal
Breadth first search	43	56	180	6	0.034	**Yes**
Breadth first tree search	1458	1459	5960	6	1.1	**Yes**
Depth first graph search	12	13	48	12	0.01	No
Depth limited search	101	271	404	50	0.11	No
Uniform cost search	55	57	224	6	0.05	**Yes**
Recursive best first search with h_1	4229	4230	17029	6	3.18	**Yes**
Greedy best first graph search with h_1	7	9	28	6	0.005	**Yes**

Uninformed Problem 2 Results

Search Types	Expansions	Goal Tests	New Node	Plan Length	Time Elapsed (s)	Is Optimal
Breadth first search	3401	4672	31049	9	15.71	**Yes**
Depth first graph search	350	351	3142	346	1.66	No
Uniform cost search	4761	4763	43206	9	50.85	**Yes**
Greedy best first graph search with h_1	550	552	4950	9	3.53	**Yes**

Uninformed Problem 3 Results

Search Types	Expansions	Goal Tests	New Node	Plan Length	Time Elapsed (s)	Is Optimal
Breadth first search	14491	17947	128184	12	109.52	**Yes**
Depth first graph search	1948	1949	16253	1878	22.3	No
Uniform cost search	17783	17785	155920	12	441.96	**Yes**
Greedy best first graph search with h_1	4031	4033	35794	22	73.06	No

Analysis

The above 3 problems, **Breadth First** Search and **Uniform Cost Search** are the only two uninformed search strategies that yield an optimal action plan under the 10mn time limit. When it comes to execution speed and memory usage, Depth First Graph Search is the fastest and uses the least memory. However, it does not generate an optimal action plan (problem 1: plan length of 12 instead of 6, problem 2: plan length of 346 instead of 9, problem 3: plan length of 1878 instead of 12).

Informed/Directed Search Strategies

Per ref (3), The informed/directed search strategy will have some information about problem space(heuristic) is used to compute preference among the children for exploration and expansion.
The following are the informed search

1. Best First Search
2. Problem decomposition
3. A*

In this section, we compare the performance of A* Search using three different heuristics. Here again, we evaluate these strategies in terms of speed, memory usage and optimality.

The various search metrics are captured were collected using the following commands:

```
python run_search.py -p 1 -s 8 9 10 >> run_informed_search_results_p1.txt

python run_search.py -p 2 -s 8 9 10 >> run_informed_search_results_p2.txt

python run_search.py -p 3 -s 8 9 10 >> run_informed_search_results_p3.txt
```

In below table which all execution time, expansions and smaller path are better mentioned as Yes in the Optimal column.

- ## Informed Problem 1 Results

Search Types	Expansions	Goal Tests	New Node	Plan Length	Time Elapsed (s)	Is Optimal
A* search with h1 heuristic	55	57	224	6	0.045	**Yes**
A* search with heuristic ignore preconditions	41	43	170	6	0.053	**Yes**
A* search with heuristic level sum	11	13	50	6	4.366	**Yes**

- ## Informed Problem 2 Results

Search Types	Expansions	Goal Tests	New Node	Plan Length	Time Elapsed (s)	Is Optimal
A* search with h1 heuristic	55	57	224	9	49.526	**Yes**
A* search with heuristic ignore preconditions	1506	1508	13820	9	16.5	**Yes**
A* search with heuristic level sum	86	88	841	9	923.58	**Yes**

- ## Informed Problem 3 Results

Search Types	Expansions	Goal Tests	New Node	Plan Length	Time Elapsed (s)	Is Optimal
A* search with h1 heuristic	17783	17785	155920	12	430.389	**Yes**
A* search with heuristic ignore preconditions	5081	5083	45292	12	98.47	**Yes**
A* search with heuristic level sum	404	406	3718	12	7923.54	**Yes**

- ## Analysis

While all heuristics yield an optimal action plan, only the h1 and Ignore Preconditions heuristics return results within the 10mn max execution time. Of the two strategies mentioned above, A* Search with Ignore Preconditions heuristic is the fastest. Per Ref [1] If we let search run to completion on our machine, A* Search with Level Sum heuristic uses the least memory, but its execution time is much slower.

Per ref 4), If we compare uninformed and informed search strategies Depth First Graph Search is faster and uses less memory than Uniform Cost Search. As for informed search strategies, A* Search with Ignore Preconditions heuristic is the fastest and uses the least memory.

From the comparison A* Search with Ignore Preconditions heuristic would be the best choice overall for our Air Cargo problem because it is faster and uses less memory.

- ## Description and Analysis Performance Comparison

Comparison of Problems, Algorithms and Heuristics	Type	Comparison (* with per ref 4] and per ref [1]
Problem 1 BFS Vs DFGS	Uninformed	**Optimality**: BFS is guaranteed and optimal to find the shortest path with plan length of 6. DGFS is not optimal since there is more than one goal state, so it has plan length of 12. **Time Elapsed**: DGFS is faster than BFS **Node Expansion**: BFS expands > 2x more nodes than DGFS, since DGFS only uses storage space of n nodes instead of 2n nodes in BFS when not tracking the explored set. **Justification**: In the Tri-City Search Problem, we considered the 2 12th State Space of Problem 1.
Problem 1 with DFGS vs UCS	Uninformed	**Optimality**: Same as Problem 1 with BFS vs DFGS, change the result with UCS. **Time Elapsed**: BFS finds goal lesser time than UCS **Node Expansions**: UCS expands more nodes than BFS **Justification***: BFS finds the shortest path in terms of the least number of steps, but it will not find the shortest path in terms of the shortest total cost (sum of step costs). UCS takes longer and expands more nodes than BFS since even after finding a path to the goal state it continues searching to try and find a cheaper path that also reaches the goal state.

Problem 1 with DFGS vs UCS	Uninformed	**Optimality**: Same as Problem 1 with BFS vs DFGS change the result with UCS. **Time Elapsed**: DFGS finds goal faster than BFS. **Node Expansions**: UCS expands more nodes than DFGS since DFGS only uses Storage Space of n nodes instead of 2 n nodes in UCS when not tracking the explored set. **Justification***: Same justification as above
Problem 2 with BFS vs DFGS	Uninformed	**Optimality**: Both BFS and UCS are optimal with plan length of 9 **Time Elapsed**: BFS finds goal faster than UCS. Node Expansions: UCS expands more nodes than BFS. **Justification***: Same justification as for Problem1 with BFS vs UCS.
Problem 2 with DFGS vs UCS	Uninformed	**Optimality**: Both BFS and UCS are optimal with plan length of 9. **Time Elapsed**: BFS finds goal faster than UCS. **Node Expansions**: UCS expands more nodes than BFS. **Justification***: Same justification as for Problem 1 with BFS vs UCS.
Problem 3 with BFS vs DGFS	Uninformed	**Optimality**: BFS is optimal and guaranteed to find the shortest path with plan length of 9. DFGS is not optimal since there is more than one goal, so it has a plan length of 346 (>35 times longer). **Time Elapsed**: DFGS finds goal 100x faster than BFS. **Node Expansions**: BFS expands more nodes than DFGS since DFGS only uses Storage Space of n nodes instead of 2n nodes in BFS when not tracking the explored set.

		Justification*: Same justification as in Problem 1 with BFS vs DFGS, but in P2 where there are more goal states Since there are more than two nodes, the issue of exploring intersecting nodes repeatedly (duplicates) is apparent.
Problem 3 with BFS vs UCS	Uninformed	**Optimality**: Both BFS and UCS are optimal with plan length 12 **Time Elapsed**: BFS finds goal faster than UCS. **Node Expansions**: UCS expands more nodes than BFS. **Justification***: Same justification as for Problem 1 with BFS vs UCS.
Problem 3 with DFGS vs UCS	Uninformed	**Optimality**: Same as Problem 3 with BFS vs DFGS but replace BFS with UCS. **Time Elapsed**: DFGS finds goal faster than UCS. **Node Expansions**: UCS expands more nodes than DFGS since DFGS only uses Storage Space of n nodes instead of 2n nodes in UCS when not tracking the explored set. **Justification***: Same justification applies as in P2 with BFS vs DFGS, but replacing results of BFS with UCS.
Problem 1 with A*S HIP vs A*S HPGLS	Informed	**Optimality**: A*S algorithm is not guaranteed to be optimal but when used in combination with HIP and HPGLS heuristics, constraints are applied and optimal plan lengths of 6 are found. **Time Elapsed**: A*S HIP finds goal faster than A*S HPGLS. **Node Expansions**: A*S HIP expands more nodes than A*S HPGLS. **Justification***: The HPGLS heuristic uses a Planning Graph but just estimates the sum of all actions that

		must be carried out from the current state to satisfy each individual goal condition, but the HIP heuristic finds the goal faster since it ignores preconditions required for an action to be executed to make the problem easier in order to estimate the minimum number of actions that must be carried out from the current state in order to satisfy all of the goal conditions.
Problem 2 with A*S HIP vs A*S HPGLS	Informed	**Optimality**: A*S algorithm will always find the lowest cost path to the goal dependent on whether the heuristic estimate function h for a state is less than the true cost of the path to the goal through that state, so it is therefore not guaranteed to be optimal but when used in combination with HIP and HPGLS heuristics, constraints are applied and optimal plan lengths of 9 are found. **Time Elapsed**: A*S HI heuristic finds goal 12x faster than A*S HPGLS **Node Expansions**: A*S HI heuristic expands >17x more nodes than A*S HPGLS. **Justification***: Same justification as for Problem 1 with A*S HI heuristic vs A*S HPGLS.
Problem 3 with A*S HI heuristic vs A*S HPGLS	Informed	**Optimality**: A*S algorithm is not guaranteed to be optimal but when used in combination with the HI heuristics, constraints are applied and optimal plan length of 12 is found. No solution was found when using the HPGLS even after 10 minutes. **Time Elapsed**: Unable to compare as no solution was found when using HPGLS even after 10 minutes. **Node Expansions**: Unable to compare as no solution was found when using HPGLS even after 10 minutes.

		Justification*: The reason for HPGLS taking so long is explained, which highlights the importance of using an approach like HIP that minimizes the set of actions to remove redundant actions from consideration instead of HPGLS that uses a Planning Graph that estimates the sum of all actions including non-minimal ones that makes the search procedure unnecessarily slow.

- ## Final Note

The results above clearly illustrate the benefits of using informed search strategies with custom heuristics over uninformed search techniques when searching for an optimal plan. The benefits are significant both in terms of speed and memory usage. Another, benefit is that one can customize the trade-off between speed and memory usage by using different heuristics, which is simply not possible with uninformed search strategies.

- ## Reference

1. Stuart J. Russell, Peter Norvig, Artificial Intelligence: A Modern Approach 3rd Edition
2. https://www.ics.uci.edu/~welling/teaching/ICS171Fall05/Search041005.ppt
3. https://www.slideshare.net/ameykerkar/informed-and-uninformed-search-strategies
4. Chapter Planning and Chapter Search, video from Artificial Intelligence Nano degree program.