HEURISTIC ANALYSIS

**Planning Search Heuristic Analysis**

INTRODUCTION

In this project, we implemented 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** and compare their results/performance against several **uninformed non-heuristic search methods** (breadth-first, depth-first, etc.).

**By *Sean Alexander Frenn***

# ABSTRACT

In this project, we apply different planning algorithms to solve the air cargo planning problems using multiple search methods: breadth-first search (BFS), depth-first (DF) graph search, uniform cost search (UCS), and different variants of A star (A\*) search. An optimal plan has been obtained this problem by computing different methods and the performance of each search method have been analyzed. The results show us that DF graph search is the fastest but do not always obtain an optimal solution to the problem. A\* search on the other hand (used with a *h\_ignore\_preconditions* heuristic) was the fastest to discover an optimal solution to most of the air cargo problems we experimented with. The performance of the A\* with a *h\_pg\_levelsum* heuristic was almost identical to the performance of the UCS. We suggest using the DF graph search when solution speed is more important, and A\* *h\_ignore\_preconditions* when it is important to find an optimal plan to the problem. In general, if the depth of the greater than the width of it, and solutions are rare, DF graph search might take an extremely long time, but BFS could be faster. In the opposite scenario, a BF graph search might need too much memory, so it might be completely impractical. Experimentation would be the key to choosing the best method to use in any problem set.

# THE PROBLEM

Here we had three planning problems in the air cargo domain using the same 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))

Below are the initial states and goals of those problems:

|  |  |
| --- | --- |
| 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)) | 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)) |
| 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)) | |

We realized that we could reach the goals above by using different plans, considering the optimal plan lengths for each problem. Here are sample plans respecting the optimal length for each problem:

|  |  |  |
| --- | --- | --- |
| Problem 1:  Load(C1, P1, SFO)  Load(C2, P2, JFK)  Fly(P1, SFO, JFK)  Fly(P2, JFK, SFO)  Unload(C1, P1, JFK)  Unload(C2, P2, SFO) | Problem 2:  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, JF | Problem 3:  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) |

# RESULTS

## Uninformed Search Strategies Analysis

The uninformed search strategies or blind search have no additional information about states beyond what’s provided in the problem definition. They can only generate successors and differentiate between a goal state and a non-goal state. We present here our comparison of the performance of seven uniformed search strategies in terms of speed, memory usage and optimal length. More details are presented in the report tables which do not change the results of our analysis. To collect the different performance measures, we used the following commands:

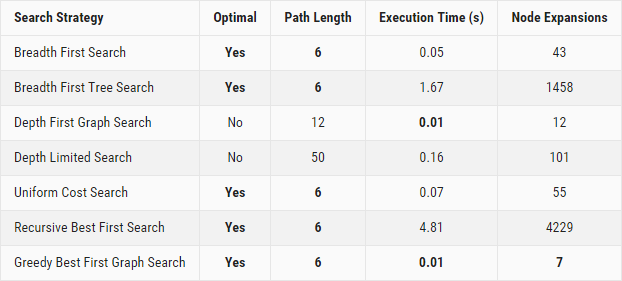
python run\_search.py -p 1 -s 1 2 3 4 5 6 7 >> run\_uninformed\_search\_results\_p1.txt

python run\_search.py -p 2 -s 1 3 5 7 >> run\_uninformed\_search\_results\_p2.txt

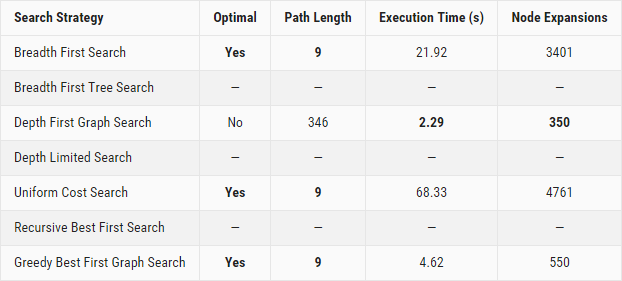
python run\_search.py -p 3 -s 1 3 5 7 >> run\_uninformed\_search\_results\_p3.txt

The data collection for Breadth First Tree Search, Depth Limited Search, and Recursive Best First Search exceeding 10 minutes, we did not report them as instructed by Udacity. Same with Problem 3 using the methods Breadth First Tree Search, Depth Limited Search, Uniform Cost Search, and Recursive Best First Search.

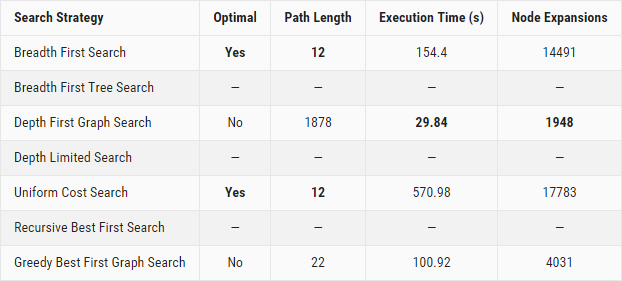
### Problem 1 Results



### Problem 2 Results



### Problem 3 Results



## Informed (Heuristic) Search Strategies Analysis

The informed search strategy uses a problem-specific knowledge beyond the definition of the problem to find solutions more efficiently than we could using an uninformed strategy. Here, we compare the performance of A\* Search using three different heuristics. And again, evaluate these methods in terms of speed, memory usage and optimal length.

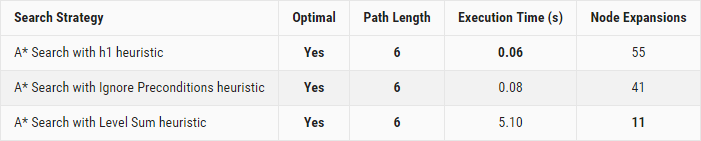
To collect the different performance measures, we used the following commands:

python run\_search.py -p 1 -s 8 9 10 >> run\_informed\_search\_results\_p1.txt

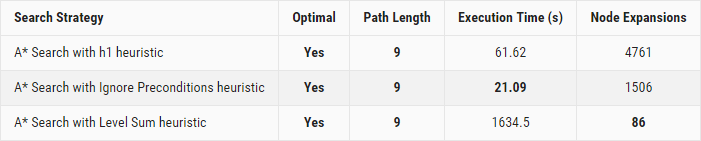
python run\_search.py -p 2 -s 8 9 10 >> run\_informed\_search\_results\_p2.txt

python run\_search.py -p 3 -s 8 9 >> run\_informed\_search\_results\_p3.txt

### Problem 1 Results



### Problem 2 Results



### Problem 3 Results



### Analysis

Among all the heuristics, only the h1 and Ignore Preconditions ones return the results within the first ten minutes of execution time.

We noticed that the A\* Search strategy with Ignore Preconditions heuristic was the fastest. But if we let search complete regardless of the execution time, we notice that A\* Search with Level Sum heuristic uses the least memory.

## Informed vs Uninformed Search Strategies

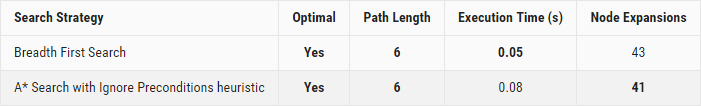
Among all the search strategies we experimented with, the results showed us that only Breadth First Search, Uniform Cost Search, and A\* Search (with all the three heuristics) generate the optimal length solution.

Depth First Graph Search is faster and uses lesser memory for uninformed search strategies, and uses less memory than Uniform Cost Search. And A\* Search with Ignore Preconditions heuristic is the faster and uses lesser memory for informed search strategies.

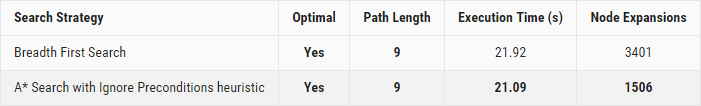
Those two strategies give us the best results in all our test cases.

Below are the comparison of the results we obtained for our problem sets.

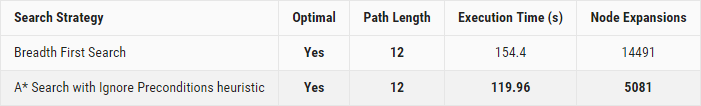
### Problem 1 Results



### Problem 2 Results



### Problem 3 Results



Based on the results we obtained, we would be more inclined to use for our Air Cargo problem the A\* Search with Ignore Preconditions heuristic for its speed and less memory consumption advantages.

# Conclusion

We illustrated in this document the benefits of using informed search strategies with custom heuristics over uninformed search techniques when searching for an optimal plan (based on speed and memory usage). A more subtle benefit of informed search strategies would be that we could perform a trade-off between speed and memory consumption by using different heuristics, which we haven’t been able to do using uninformed search strategies.

From the results exposed above, we can see that DF graph search obtained the fewest node expansions and was faster than the other methods. Unfortunately, this method has the largest path length compared to the other methods and couldn’t even reach an optimal solution for all the problems we experimented with as opposed to the other methods. As written in Russell and Norvig 2009, DF graph search methods are known to be nonoptimal.

On the other hand, BF graph search and UC graph search both obtained optimal and identical solutions. The main difference being that BF graph search had fewer node expansions and UC graph search was faster.

# REFERENCES

Russell, S. J. and P. Norvig (2009). Artificial intelligence: a modern approach (3rd edition).

https://stackoverflow.com/questions/3332947/when-is-it-practical-to-use-dfs-vs-bfs.