# Heuristic Analysis - Planning Search

Yun-Sheng Tu

In this project, we implement a planning search problem for Air Cargo System. There are two analysis results in this document: **non-heuristic planning** and **heuristic planning.**

## Planning problem

Here are all the 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))

There are three problems need to be solved:

- 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))

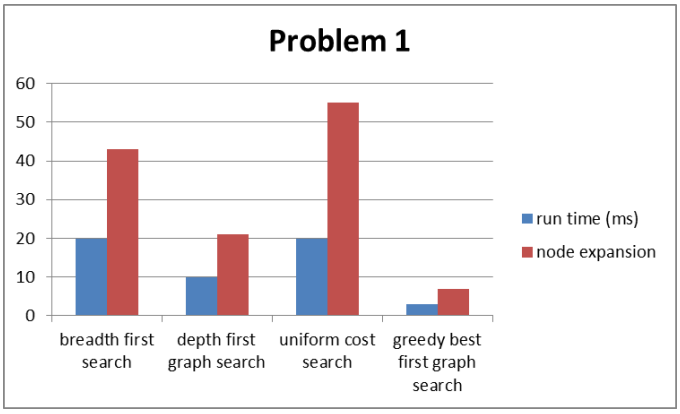
- Problem 3

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))

# non-heuristic planning result

- *problem 1*

| search strategy | best plan length | run time (sec) | node expansion | result |
|---|---|---|---|---|
| breadth first search | 6 | 0.020 | 43 | |
| breadth first tree search | 6 | 0.750 | 1458 | |
| depth first graph search | 20 | 0.010 | 21 | x |
| depth limited search | 50 | 0.070 | 101 | x |
| uniform cost search | 6 | 0.020 | 55 | |
| recursively best first search | 6 | 2.150 | 4226 | |
| greedy best first graph search | 6 | 0.003 | 7 | best time & memory |




In problem1, each search strategy tries to find its best plan. Node expansion means the number of nodes created. Therefore, it represents memory cost. In this result, **depth first graph search** and **depth limited search** didn't find the best plan. They are not good strategies even though they spend less time, because finding the best plan is more important. We think spend more time to get the best plan is acceptable. In this table, **greedy best first graph search** is the best strategy because it cost less time and memory usage.

**Optimal Plan**

Load(C1, P1, SFO)

Load(C2, P2, JFK)

Load(C3, P3, ATL)

Fly(P2, JFK, SFO)

Unload(C2, P2, SFO)

Fly(P1, SFO, JFK)

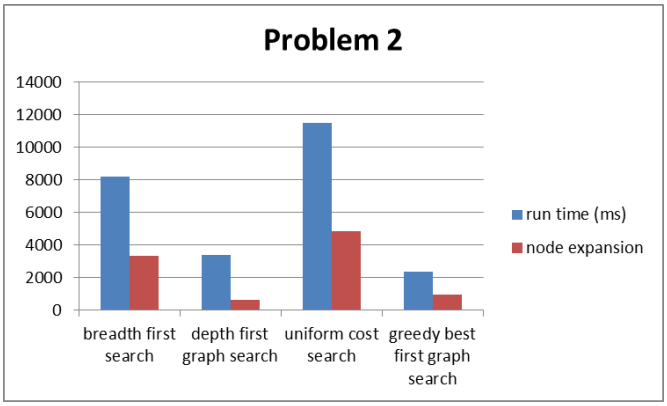
Unload(C1, P1, JFK)

Fly(P3, ATL, SFO)

Unload(C3, P3, SFO)

- *problem 2*

| search strategy | best plan length | run time (sec) | node expansion | result |
|---|---|---|---|---|
| breadth first search | 9 | 8.23 | 3343 | best time and mem |
| breadth first tree search | - | - | - | |
| depth first graph search | 619 | 3.39 | 624 | x |
| depth limited search | - | - | - | |
| uniform cost search | 9 | 11.50 | 4849 | |
| recursively best first search | - | - | - | |
| greedy best first graph search | 16 | 2.38 | 966 | x |




In problem 2, this is a more complex problem than problem 1. We disable the strategies which cost more than 10 minutes. Unfortunately, **greedy best first graph search** didn't found the best plan. **Breadth first search** found the best plan and cost less time and memory usage.

**Optimal Plan**

Load(C1, P1, SFO)

Load(C2, P2, JFK)

Load(C3, P3, ATL)

Fly(P2, JFK, SFO)

Unload(C2, P2, SFO)

Fly(P1, SFO, JFK)

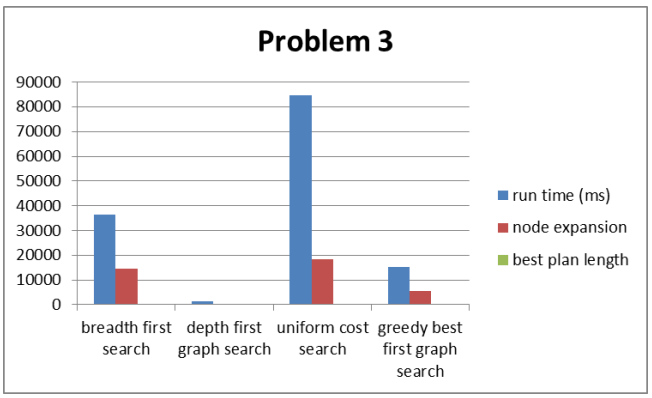
Unload(C1, P1, JFK)

Fly(P3, ATL, SFO)

Unload(C3, P3, SFO)

- *problem 3*

| search strategy | best plan length | run time (sec) | node expansion | result |
|---|---|---|---|---|
| breadth first search | 12 | 36.44 | 14663 | best time and mem |
| breadth first tree search | - | - | - | |
| depth first graph search | 392 | 1.52 | 408 | x |
| depth limited search | - | - | - | |
| uniform cost search | 12 | 84.80 | 18235 | |
| recursively best first search | - | - | - | |
| greedy best first graph search | 21 | 15.19 | 5462 | x |



Problem 3 is the most complex. Also, we disable the strategies which cost more than 10 minutes. **breadth first search** found the best plan and cost less time and memory usage.

**Optimal Plan**

Load(C1, P1, SFO)

Load(C2, P2, JFK)

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)

# heuristic planning result

Heuristic strategy can find solutions more efficiently than non-heuristic strategy. We compare A* search and add some different heuristics.

- *problem 1*

| search strategy | best plan length | run time (sec) | node expansion | result |
|---|---|---|---|---|
| A* search | 6 | 0.03 | 55 | |
| A* search with ignore precondition | 6 | 0.02 | 41 | best time |
| A* search with level sum heuristic | 6 | 0.59 | 11 | best memory |

- *problem 2*

| search strategy | best plan length | run time (sec) | node expansion | result |
|---|---|---|---|---|
| A* search | 9 | 9.99 | 4849 | |
| A* search with ignore precondition | 9 | 3.08 | 1443 | best time |
| A* search with level sum heuristic | 9 | 56.81 | 85 | best memory |

- *problem 3*

| search strategy | best plan length | run time (sec) | node expansion | result |
|---|---|---|---|---|
| A* search | 12 | 46.64 | 18235 | |
| A* search with ignore precondition | 12 | 14.03 | 4945 | best time |
| A* search with level sum heuristic | 12 | 249.44 | 290 | best memory |

**A* search with ignore precondition** estimates the minimum number of actions that must be carried out from the current state in order to satisfy all of the goal conditions. Therefore, it costs less time than original A* search.

**A* search with level sum heuristic** uses a planning graph representation of the problem state space to estimate the sum of all actions that must be carried out from the current state in order to satisfy each individual goal condition. It costs more time because it needs to create the planning graph. When the problem is larger, the graph grows exponentially so that the run time increases tremendously. After building the graph, it can use level sum to reach the goal efficiently, therefore, the number node expansion become lower.

# Conclusion

| search strategy | best plan length | run time (sec) | node expansion | result |
|---|---|---|---|---|
| breadth first search | 12 | 36.44 | 14663 | |
| A* search | 12 | 46.64 | 18235 | |
| A* search with ignore precondition | 12 | 14.03 | 4945 | best strategy |
| A* search with level sum heuristic | 12 | 249.44 | 290 | |

In the book *Artificial Intelligence: A Modern Approach, 3/e Chapter 10,* "An admissible heuristic can be derived by defining a relaxed problem that is easier to solve. The exact cost of a solution to this easier problem then becomes the heuristic for the original problem". We compare heuristic and non-heuristic strategy in problem 3. In this table, heuristic method **A* search with ignore precondition** has correct direction toward the goal rather than **breadth first search**. Therefore, it costs less time and memory space to reach the goal. When the problem become large, **A* search with ignore precondition** has better performance than other methods.