Heuristic Analysis

Name : Kwangshin Oh
Udacity Email Address : kwangshin@gmail.com

Table of Contents

Heuristic Analysis

1. Part 1 - Planning problems

1-1. Result of non-heuristic planning solution searches

Here is the result of 3 problems for 3 search algorithms: (1) breadth_first_search, (2) depth_first_graph_search and (3) uniform_cost_search.

Search Algorithms	Metrics	air_cargo_p1	air_cargo_p2	air_cargo_p3
breadth_first_search	number of node expansions required	43	3343	14663
	number of goal tests	56	4609	18098
	new nodes	180	30509	129631
	time elapsed	0.02532546099973842	16.69981197800371	166.7516441129992
	optimality of solution	Yes (Plan length: 6)	Yes (Plan length: 9)	Yes (Plan length: 12)
depth_first_graph_search	number of node expansions required	12	582	627
	number of goal tests	13	583	628
	new nodes	48	5211	5176
	time elapsed	0.00583288700727280	3.594843339989893	5.281379441003082
	optimality of solution	No (Plan length: 12)	No (Plan length: 575)	No (Plan length: 596)
uniform_cost_search	number of node expansions required	55	4852	18235
	number of goal tests	57	4854	18237
	new nodes	224	44030	159716
	time elapsed	0.04080953099764883	66.59963337500812	621.9642140949873
	optimality of solution	Yes (Plan length: 6)	Yes (Plan length: 9)	Yes (Plan length: 12)

1-2. Result Analysis

1-2-1. breadth_first_search

The breadth first search in graph will search all the current level(depth) from root node, and then go to the next level. That is the reason that this search algorithm provides the optimal solution for all 3 problems.

However, as a cost, we need to expect the higher expense for number of node expansions required, number of goal tests, new nodes and time elapsed, compare to the depth_first_graph_search providing non-optimal solution.

Here is the command line output for reference.

Command Line Output

```
(aind) P35999:AIND-Planning kwangshin.oh$ python run_search.py -s 1 -p 1

Solving Air Cargo Problem 1 using breadth_first_search...

Expansions   Goal Tests   New Nodes
    43          56         180

Plan length: 6  Time elapsed in seconds: 0.025325460999738425
Load(C2, P2, JFK)
Load(C1, P1, SFO)
Fly(P2, JFK, SFO)
Unload(C2, P2, SFO)
Fly(P1, SFO, JFK)
Unload(C1, P1, JFK)

(aind) P35999:AIND-Planning kwangshin.oh$ python run_search.py -s 1 -p 2

Solving Air Cargo Problem 2 using breadth_first_search...

Expansions   Goal Tests   New Nodes
   3343        4609       30509

Plan length: 9  Time elapsed in seconds: 16.69981197800371
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)
```

```
(aind) P35999:AIND-Planning kwangshin.oh$ python run_search.py -s 1 -p 3

Solving Air Cargo Problem 3 using breadth_first_search...

Expansions   Goal Tests   New Nodes
  14663       18098       129631

Plan length: 12  Time elapsed in seconds: 166.7516441129992
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)

(aind) P35999:AIND-Planning kwangshin.oh$
```

1-2-2. depth_first_graph_search

The depth first search in graph will search the next level(depth) node first, and then search the other nodes in the same level. As of this approach, they will find out the solution quickly most of time, but it does not guarantee that the solution is the optimal solution.

As you can see from the result, this algorithm provides the best performance for number of node expansions required, number of goal tests, new nodes and time elapsed among the 3 algorithms.

However this algorithm does not guarantee the optimal solution.

Here is the command line output for reference.

Command Line Output

```
(aind) P35999:AIND-Planning kwangshin.oh$ python run_search.py -s 3 -p 1

Solving Air Cargo Problem 1 using depth_first_graph_search...

Expansions   Goal Tests   New Nodes
    12          13          48

Plan length: 12  Time elapsed in seconds: 0.005832887007272802
Fly(P1, SFO, JFK)
```

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

(aind) P35999:AIND-Planning kwangshin.oh$ python run_search.py -s 3 -p 2

Solving Air Cargo Problem 2 using depth_first_graph_search...

Expansions   Goal Tests   New Nodes
   582         583         5211

Plan length: 575  Time elapsed in seconds: 3.594843339989893
Fly(P3, ATL, SFO)
Fly(P1, SFO, ATL)
Fly(P3, SFO, JFK)
Fly(P1, ATL, JFK)
Fly(P2, JFK, ATL)
Fly(P3, JFK, ATL)
Fly(P2, ATL, SFO)
Fly(P3, ATL, SFO)
Load(C1, P3, SFO)
~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~
Unload(C1, P3, JFK)
Fly(P2, SFO, JFK)
Fly(P3, JFK, ATL)
Fly(P2, JFK, ATL)
Fly(P3, ATL, SFO)
Fly(P1, JFK, ATL)
Fly(P2, ATL, SFO)
Fly(P1, ATL, SFO)
Fly(P3, SFO, ATL)
Unload(C3, P1, SFO)

(aind) P35999:AIND-Planning kwangshin.oh$ python run_search.py -s 3 -p 3

Solving Air Cargo Problem 3 using depth_first_graph_search...

Expansions   Goal Tests   New Nodes
   627         628         5176

Plan length: 596  Time elapsed in seconds: 5.281379441003082
```

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

(aind) P35999:AIND-Planning kwangshin.oh$
```

1-2-3. uniform_cost_search

This algorithm works similar as breadth first search algorithm. However when this algorithm finds out the next node, they will decide the next node based on the cost calculation, instead of always current level's node in breadth first search algorithm.

Of course, it will provide optimal solution, but the performance is worse than breadth first search algorithm based on the result. So if we are in the situation that we should provide the optimal solution, then we had better use the breadth first search algorithm.

Here is the command line output for reference.

Command Line Output

```
(aind) P35999:AIND-Planning kwangshin.oh$ python run_search.py -s 5 -p 1

Solving Air Cargo Problem 1 using uniform_cost_search...

Expansions   Goal Tests   New Nodes
    55          57          224

Plan length: 6  Time elapsed in seconds: 0.040809530997648835
Load(C1, P1, SFO)
Load(C2, P2, JFK)
```

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

(aind) P35999:AIND-Planning kwangshin.oh$ python run_search.py -s 5 -p 2

Solving Air Cargo Problem 2 using uniform_cost_search...

Expansions   Goal Tests   New Nodes
   4852        4854       44030

Plan length: 9  Time elapsed in seconds: 66.59963337500812
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(C1, P1, JFK)
Unload(C2, P2, SFO)

(aind) P35999:AIND-Planning kwangshin.oh$ python run_search.py -s 5 -p 3

Solving Air Cargo Problem 3 using uniform_cost_search...

Expansions   Goal Tests   New Nodes
  18235       18237       159716

Plan length: 12  Time elapsed in seconds: 621.9642140949873
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(P2, ORD, SFO)
Fly(P1, ATL, JFK)
Unload(C4, P2, SFO)
Unload(C3, P1, JFK)
Unload(C1, P1, JFK)
Unload(C2, P2, SFO)

(aind) P35999:AIND-Planning kwangshin.oh$
```

2. Part 2 - Domain-independent heuristics

2-1. Result of A* searches with heuristics

Here is the result of 3 problems for 3 A* searches with heuristics: (1) astar_search h_1, (2) astar_search h_ignore_preconditions and (3) astar_search h_pg_levelsum.

Search Algorithms	Metrics	air_cargo_p 1	air_cargo_p 2	air_cargo_p 3
astar_search h_1	number of node expansions required	55	4852	18235
	number of goal tests	57	4854	18237
	new nodes	224	44030	159716
	time elapsed	0.03401124197989702	51.70433558395598	479.6114926080336
	optimality of solution	Yes (Plan length: 6)	Yes (Plan length: 9)	Yes (Plan length: 12)
astar_search h_ignore_preconditions	number of node expansions required	41	1506	5118
	number of goal tests	43	1508	5120
	new nodes	170	13820	45650
	time elapsed	0.04395636700792238	13.760683299042284	97.08552193496143
	optimality of solution	Yes (Plan length: 6)	Yes (Plan length: 9)	Yes (Plan length: 12)
astar_search h_pg_levelsum	number of node expansions required	11	86	408
	number of goal tests	13	88	410
	new nodes	50	841	3758
	time elapsed	1.2193327589775436	117.67138353997143	803.3100059930002
	optimality of solution	Yes (Plan length: 6)	Yes (Plan length: 9)	Yes (Plan length: 12)

2-2. Result Analysis

2-2-1. astar_search h_1

This is the implementation of A* search algorithm without any special heuristic. Basically it has better performance than non-heuristic planning solution searches which is implemented and tested from Part 1.

Additionally this A* search algorithm provides the optimal solutions for all 3 problems.

Here is the command line output for reference.

Command Line Output

```
(aind) P35999:AIND-Planning kwangshin.oh$ python run_search.py -p 1 -s 8

Solving Air Cargo Problem 1 using astar_search with h_1...

Expansions   Goal Tests   New Nodes
    55          57          224

Plan length: 6  Time elapsed in seconds: 0.03401124197989702
Load(C1, P1, SFO)
Load(C2, P2, JFK)
Fly(P1, SFO, JFK)
Fly(P2, JFK, SFO)
Unload(C1, P1, JFK)
Unload(C2, P2, SFO)

(aind) P35999:AIND-Planning kwangshin.oh$ python run_search.py -p 2 -s 8

Solving Air Cargo Problem 2 using astar_search with h_1...

Expansions   Goal Tests   New Nodes
   4852        4854       44030

Plan length: 9  Time elapsed in seconds: 51.70433558395598
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(C1, P1, JFK)
Unload(C2, P2, SFO)

(aind) P35999:AIND-Planning kwangshin.oh$ python run_search.py -p 3 -s 8

Solving Air Cargo Problem 3 using astar_search with h_1...
```

```
Expansions   Goal Tests   New Nodes
  18235       18237       159716

Plan length: 12  Time elapsed in seconds: 479.6114926080336
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(P2, ORD, SFO)
Fly(P1, ATL, JFK)
Unload(C4, P2, SFO)
Unload(C3, P1, JFK)
Unload(C1, P1, JFK)
Unload(C2, P2, SFO)

(aind) P35999:AIND-Planning kwangshin.oh$
```

2-2-2. astar_search h_ignore_preconditions

This is the implementation of A* search algorithm with ignore preconditions heuristic. The heuristic looks simple and easy to implement, but it provides 5 times faster performance than A* search algorithm without any special heuristic for Problem 3.

Of course, this solution also provides the optimal solution for all 3 problems.

Here is the command line output for reference.

Command Line Output

```
(aind) P35999:AIND-Planning kwangshin.oh$ python run_search.py -p 1 -s 9

Solving Air Cargo Problem 1 using astar_search with
h_ignore_preconditions...

Expansions   Goal Tests   New Nodes
    41          43         170

Plan length: 6  Time elapsed in seconds: 0.04395636700792238
Load(C1, P1, SFO)
Fly(P1, SFO, JFK)
Unload(C1, P1, JFK)
Load(C2, P2, JFK)
Fly(P2, JFK, SFO)
Unload(C2, P2, SFO)

(aind) P35999:AIND-Planning kwangshin.oh$ python run_search.py -p 2 -s 9
```

```
Solving Air Cargo Problem 2 using astar_search with
h_ignore_preconditions...

Expansions   Goal Tests   New Nodes
   1506        1508       13820

Plan length: 9  Time elapsed in seconds: 13.760683299042284
Load(C3, P3, ATL)
Fly(P3, ATL, SFO)
Unload(C3, P3, SFO)
Load(C1, P1, SFO)
Fly(P1, SFO, JFK)
Unload(C1, P1, JFK)
Load(C2, P2, JFK)
Fly(P2, JFK, SFO)
Unload(C2, P2, SFO)

(aind) P35999:AIND-Planning kwangshin.oh$ python run_search.py -p 3 -s 9

Solving Air Cargo Problem 3 using astar_search with
h_ignore_preconditions...

Expansions   Goal Tests   New Nodes
   5118        5120       45650

Plan length: 12  Time elapsed in seconds: 97.08552193496143
Load(C2, P2, JFK)
Fly(P2, JFK, ORD)
Load(C4, P2, ORD)
Fly(P2, ORD, SFO)
Unload(C4, P2, SFO)
Load(C1, P1, SFO)
Fly(P1, SFO, ATL)
Load(C3, P1, ATL)
Fly(P1, ATL, JFK)
Unload(C3, P1, JFK)
Unload(C1, P1, JFK)
Unload(C2, P2, SFO)

(aind) P35999:AIND-Planning kwangshin.oh$
```

2-2-3. astar_search h_pg_levelsum

This is an implementation of A* algorithm with sum of the level cost heuristic.

For the all problems, this heuristic provides the less memory usage than the other A* search algorithm with lowest Expansions, Goal Tests and New Nodes. However there is a trade-off.

We need to spend more time to get the solution. Compare to the A* search algorithm with ignore precondition heuristic, this heuristic takes about 10 times more time.

Of course, this solution also provides the optimal solution for all 3 problems.

Here is the command line output for reference.

Command Line Output

```
(aind) P35999:AIND-Planning kwangshin.oh$ python run_search.py -p 1 -s
10

Solving Air Cargo Problem 1 using astar_search with h_pg_levelsum...

Expansions   Goal Tests   New Nodes
    11          13          50

Plan length: 6  Time elapsed in seconds: 1.2193327589775436
Load(C1, P1, SFO)
Fly(P1, SFO, JFK)
Load(C2, P2, JFK)
Fly(P2, JFK, SFO)
Unload(C1, P1, JFK)
Unload(C2, P2, SFO)

(aind) P35999:AIND-Planning kwangshin.oh$ python run_search.py -p 2 -s
10

Solving Air Cargo Problem 2 using astar_search with h_pg_levelsum...

Expansions   Goal Tests   New Nodes
    86          88         841

Plan length: 9  Time elapsed in seconds: 117.67138353997143
Load(C1, P1, SFO)
Fly(P1, SFO, JFK)
Load(C2, P2, JFK)
Fly(P2, JFK, SFO)
Load(C3, P3, ATL)
Fly(P3, ATL, SFO)
Unload(C3, P3, SFO)
Unload(C1, P1, JFK)
Unload(C2, P2, SFO)

(aind) P35999:AIND-Planning kwangshin.oh$ python run_search.py -p 3 -s
10

Solving Air Cargo Problem 3 using astar_search with h_pg_levelsum...

Expansions   Goal Tests   New Nodes
   408         410         3758
```

```
Plan length: 12  Time elapsed in seconds: 803.3100059930002
Load(C2, P2, JFK)
Fly(P2, JFK, ORD)
Load(C4, P2, ORD)
Fly(P2, ORD, SFO)
Load(C1, P1, SFO)
Fly(P1, SFO, ATL)
Load(C3, P1, ATL)
Fly(P1, ATL, JFK)
Unload(C4, P2, SFO)
Unload(C3, P1, JFK)
Unload(C1, P1, JFK)
Unload(C2, P2, SFO)

(aind) P35999:AIND-Planning kwangshin.oh$
```

3. Part 3 - Written Analysis

3.1 Optimal Plan

Here is the one of the optimal plan for give problems.

	Problem 1	Problem 2	Problem 3
Optimal Plan	Plan length: 6 Load(C1, P1, SFO) Fly(P1, SFO, JFK) Unload(C1, P1, JFK) Load(C2, P2, JFK) Fly(P2, JFK, SFO) Unload(C2, P2, SFO)	Plan length: 9 Load(C3, P3, ATL) Fly(P3, ATL, SFO) Unload(C3, P3, SFO) Load(C1, P1, SFO) Fly(P1, SFO, JFK) Unload(C1, P1, JFK) Load(C2, P2, JFK) Fly(P2, JFK, SFO) Unload(C2, P2, SFO)	Plan length: 12 Load(C2, P2, JFK) Fly(P2, JFK, ORD) Load(C4, P2, ORD) Fly(P2, ORD, SFO) Unload(C4, P2, SFO) Load(C1, P1, SFO) Fly(P1, SFO, ATL) Load(C3, P1, ATL) Fly(P1, ATL, JFK) Unload(C3, P1, JFK) Unload(C1, P1, JFK) Unload(C2, P2, SFO)

3.2 Non-heuristic Search Analysis

Search Algorithms	Metrics	air_cargo_p1	air_cargo_p2	air_cargo_p3
breadth_first_search	number of node expansions required	43	3343	14663
	time elapsed	0.025	16.700	166.752
	optimality of solution	Yes (Plan length: 6)	Yes (Plan length: 9)	Yes (Plan length: 12)
depth_first_graph_search	number of node expansions required	12	582	627
	time elapsed	0.006	3.595	5.281
	optimality of solution	No (Plan length: 12)	No (Plan length: 575)	No (Plan length: 596)
uniform_cost_search	number of node expansions required	55	4852	18235
	time elapsed	0.041	66.600	621.964
	optimality of solution	Yes (Plan length: 6)	Yes (Plan length: 9)	Yes (Plan length: 12)

As you can see the above table, we could choose the breadth first search or depth first graph search according to the situation.

If we don't need to provide optimal solution, in terms of memory (the number of node expansions) and performance (time elapsed), the depth first graph search would be the best choice.

However, we might need to provide the optimal solution from most of environments. In that case, we are unable to choose the depth first graph search, because it does not support the optimal solution. In this environment, the breadth first search would be the best choice from non-heuristic search algorithm. Although the uniform cost search also provides the optimal solution, but it uses more memory and spent more time to get the solution than the breadth first search from all 3 problems.

3.3 Heuristic Search Analysis

As we already know, there is a limitation for search algorithm in performance without good heuristic function. Basically the heuristic function will represent the distance from the start node to the goal node, and if we can find out the heuristic that is the heuristic for the distance without overestimate (called admissible heuristic), then we can use the A* search algorithm to find optimal solution.[1]

However, as you can see the "astar_search h_1" algorithm, it doesn't guarantee the performance without good heuristic, even if it is an A* algorithm.

Then what kind of heuristic we can use to get the better performance?

Let's consider this search problem as a graph having the start state and goal state. Of course the state could be represented as a node, and the actions moving from one state into another state would be edges in graph. So the real-world problem solving can be mapped to find the path connecting the start state and goal state in graph. Based on this assumption, how we can easily find out the path in graph?

We are going to find out the path, so what if we have more and more candidate path? Of course, finding the path will be easier with more paths. The edge in the graph means the actions, and we need to check the precondition of action before we draw the edge between 2 nodes. What if there is no precondition? We can add more edges to the graph. So if we ignore the precondition through dropping all preconditions from actions, then more edges will be added to the graph, and must be easier to find out the path which is a solution!

Search Algorithms	**Metrics**	**air_cargo_p1**	**air_cargo_p2**	**air_cargo_p3**
astar_search h_1	number of node expansions required	55	4852	18235
	time elapsed	0.034	51.704	479.611
	optimality of solution	Yes (Plan length: 6)	Yes (Plan length: 9)	Yes (Plan length: 12)
astar_search h_ignore_preconditions	number of node expansions required	41	1506	5118
	time elapsed	0.0440	13.761	97.086
	optimality of solution	Yes (Plan length: 6)	Yes (Plan length: 9)	Yes (Plan length: 12)

astar_search h_pg_levelsum	number of node expansions required	11	86	408
	time elapsed	1.219	117.671	803.310
	optimality of solution	Yes (Plan length: 6)	Yes (Plan length: 9)	Yes (Plan length: 12)

The result is too obvious for A* search algorithm with heuristic.

All the solutions support the optimal solution for all 3 problems. So we can ignore the optimality of solution from the judgement points.

In terms of speed performance, the ignore preconditions heuristic get the solution in 10 times faster time than the sum of the level cost heuristic. However sum of the level cost heuristic is able to find out the solution with 10 times less node expansion.

As a conclusion, there is no doubt we would choose the A* search with ignore preconditions heuristic, if we need to find out the solution quickly. However, if the environment has a limited memory condition, and the more time spending is acceptable, then we would better select the A* search with sum of the level cost heuristic.

3.4 Best Heuristic

Let's assume that we need to find out the optimal solution quickly, and let's compare heuristic and non-heuristic solutions together.

Search Algorithms	Metrics	air_cargo_p1	air_cargo_p2	air_cargo_p3
breadth_first_search	number of node expansions required	43	3343	14663
	time elapsed	0.025	16.700	166.752
	optimality of solution	Yes (Plan length: 6)	Yes (Plan length: 9)	Yes (Plan length: 12)
astar_search h_ignore_preconditions	number of node expansions required	41	1506	5118
	time elapsed	0.0440	13.761	97.086
	optimality of solution	Yes (Plan length: 6)	Yes (Plan length: 9)	Yes (Plan length: 12)

As a memory usage and performance, we would choose the A* search with ignore preconditions heuristic for these problems.

How about if we don't need to get the optimal solution, and want to find out the solution as soon as possible with minimum usage of memory?

Search Algorithms	Metrics	air_cargo_p1	air_cargo_p2	air_cargo_p3
depth_first_graph_search	number of node expansions required	12	582	627
	time elapsed	0.006	3.595	5.281
	optimality of solution	No (Plan length: 12)	No (Plan length: 575)	No (Plan length: 596)
astar_search h_ignore_preconditions	number of node expansions required	41	1506	5118
	time elapsed	0.0440	13.761	97.086
	optimality of solution	Yes (Plan length: 6)	Yes (Plan length: 9)	Yes (Plan length: 12)
astar_search h_pg_levelsum	number of node expansions required	11	86	408
	time elapsed	1.219	117.671	803.310
	optimality of solution	Yes (Plan length: 6)	Yes (Plan length: 9)	Yes (Plan length: 12)

If we don't need the optimal solution and need to find out the solution quickly in limited memory environment, then it would be better choose the non-heuristic depth first graph search algorithm. It uses less memory and has the best performance in time among all the search algorithms including heuristic searches for all 3 problems, although it doesn't support the optimal solution.

Appendix I

Problems

----

1. Air Cargo Problem 1
2. Air Cargo Problem 2
3. Air Cargo Problem 3

Search Algorithms

----

1. breadth_first_search
2. breadth_first_tree_search
3. depth_first_graph_search
4. depth_limited_search
5. uniform_cost_search
6. recursive_best_first_search h_1
7. greedy_best_first_graph_search h_1
8. astar_search h_1
9. astar_search h_ignore_preconditions
10. astar_search h_pg_levelsum