heuristic_analysis

July 19, 2017

1 Heuristic Analysis

1.1 Summary

In this project we have absolutely small data set and could estimate the results by ourselves. But I decided to get the help from Python to work out the pattern of investigation. Below the summary you can find the program pattern executed with the result of research. So, briefly:

As we could suppose, the time elapsed to solve the concrete problem depends on problem complexity, which implies the heuristics usage and test of more nodes for the better precision.

Heuristics tested:

- H_1: this is set to constant 1 – not a true heuristic.
- H_Ignore_Preconditions: this heuristic 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.
- H_PG_Levelsum: this 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.

Abbreviations:

```
  - BFS - Breadth First Search                         Pr     - Problem
  - BFT - Breadth First Tree Search                    S      - Search
  - DFG - Depth First Graph Search                     Opt    - Optimal
  - DLS - Depth Limited Search                         Com    - Completed
  - UCS - Uniform Cost Search                          Heu    - Heuristic
  - RBF - Recursive Best First Search with H_1         Exp    - Expansion
  - GBF - Greedy Best First Graph Search with H_1      Time_s - Time elapsed, sec
  - ASH - A* Search with H_1 (identical to UCS)        G_Test - Goal Test
  - AIP - A* Search with H_Ignore_Preconditions        N_Nod  - New Nodes
  - APL - A* Search with H_PG_Levelsum                 P_L    - Plan Length
```

The solution for the Problem#1:

```
- All algorithms worked well
```

Total better search algorithms : 8 Total worse search algorithms : 2

```
- Minimal Plan Path - 6
```

Algorithm DFG got the more worse result 20 Algorithm DLS got the more worse result 50

```
- Both variants of Depth Search are not optimal.
```

As we can see, the worst results were given by Depth First Graph Search and Depth Limited Search. But besides, this algorythms are not optimal for the current problems because of two reasons:

```
- The time complexity of depth-first graph search is bounded by the
size of the state space, which may be infinite in this problem. In
worth case, planes would fly without landing while they will have a fuel.
- They does not consider if a node is better than another, it simply
explores the nodes that take it as deep as possible, that is why got
the worst results.
```

All optimal non-heuristic searches like Breadth First Search, Breadth First Tree Search, Uniform Cost Search, Recursive Best First Search and Greedy Best First Graph Search as well as all A* heuristic searches have the same result for the Plan length, but different in all other points, with more nodes expansions and longer time elapsed. This makes sense since they are optimal, in this case given step costs are all identical. That also means there's no significant difference between Uniform Cost and Breadth First Search.

```
- Best algorythm GBF - Greedy Best First Graph Search with H_1
- Optimal sequence of actions
    - Step 1      Load(C1, P1, SFO)
    - Step 2      Load(C2, P2, JFK)
    - Step 3      Fly(P1, SFO, JFK)
    - Step 4      Fly(P2, JFK, SFO)
    - Step 5    Unload(C1, P1, JFK)
    - Step 6    Unload(C2, P2, SFO)
```

The solution for the Problem#2:

```
- Not all algorithms worked well
```

Algorithms BFT, DLS and RBF failed
Total better search algorithms : 5 Total worse search algorithms : 2

```
- Minimal Plan Path - 9
```

Algorithm DFG got the more worse result 619
Algorithm GBF got the more worse result 15

```
- Both variants of Depth Search are not optimal.
```

As we can see, the worst results were given by Depth First Graph Search. And Depth Limited Search even failed.

Among optimal non-heuristic searches only UCS and BFS got the best result unlike GBF.

A* heuristic searches have the same best result for the Plan length, but different in all other points. H_Ignore_Preconditions searches more nodes but got the result significantly faster. This makes sense since more nodes need to be explored when preconditions for an action are ignored. But with a simpler heuristic, it was able to compute faster compare to level sum, which is a more "accurate" but complex heuristic.

```
- Best algorythm AIP -  A* Search with H_Ignore_Preconditions
- Optimal sequence of actions
    - Step 1      Load(C3, P3, ATL)
    - Step 2      Fly(P3, ATL, SFO)
    - Step 3    Unload(C3, P3, SFO)
    - Step 4      Load(C2, P2, JFK)
    - Step 5      Fly(P2, JFK, SFO)
    - Step 6    Unload(C2, P2, SFO)
    - Step 7      Load(C1, P1, SFO)
    - Step 8      Fly(P1, SFO, JFK)
    - Step 9    Unload(C1, P1, JFK)
```

The solution for the Problem#3:

```
- Not all searches got the solution.
```

The algorithms BFT, DLS and RBF are among failed.

Total better search algorithms : 5 Total worse search algorithms : 2

```
- Minimal Plan Path - 12.
```

Algorithm DFG got the more worse result 392 Algorithm GBF from Problem 2 got the more worse result 22

As we can see, the worst result was given by Depth First Graph Search.

With non-heuristic and heuristic searches we have the same situation like in the Problem#2.

```
- Best algorythm AIP -  A* Search with H_Ignore_Preconditions
- Optimal sequence of actions
    - Step 1      Load(C2, P2, JFK)
    - Step 2      Fly(P2, JFK, ORD)
    - Step 3      Load(C4, P2, ORD)
    - Step 4      Fly(P2, ORD, SFO)
    - Step 5    Unload(C4, P2, SFO)
    - Step 6      Load(C1, P1, SFO)
    - Step 7      Fly(P1, SFO, ATL)
    - Step 8      Load(C3, P1, ATL)
    - Step 9      Fly(P1, ATL, JFK)
    - Step 10   Unload(C3, P1, JFK)
    - Step 11   Unload(C2, P2, SFO)
    - Step 12   Unload(C1, P1, JFK)
```

Final conclusion: the best heuristic for the current problems is A* Search with H_Ignore_Preconditions, but only for more complex problems: it achieved the goal 3 times faster, than the best non-heuristic UCS. For simple problem, it was working 5 times longer than non-heuristic GBF (look plot bars). H_Ignore_Preconditions is better here since it allows a more targeted search that "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". This resulted in less node expansions and goal tests, though the execution times were similar.

```
In [16]: import pandas as pd
         import matplotlib.pyplot as plt
         # Disable Anaconda warnings
         import warnings
         warnings.simplefilter('ignore')
         %pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

1.2 1. Key points

Definitions and key points from Stuart J. Russell, Peter Norvig (2010), Artificial Intelligence: A Modern Approach (3rd Edition):

- Breadth-first search is a simple strategy in which the root node is expanded first, then all the successors of the root node are expanded next, then their successors, and so on. In general, all the nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded.
- Instead of expanding the shallowest node, uniform-cost search expands the node n with the lowest path cost g(n). This is done by storing the frontier as a priority queue ordered by g. In addition to the ordering of the queue by path cost, there are two other significant differences from breadth-first search. The first is that the goal test is applied to a node when it is selected for expansion rather than when it is first generated. The reason is that the first goal node that is generated may be on a suboptimal path. The second difference is that a test is added in case a better path is found to a node currently on the frontier.
- Depth-first graph search always expands the deepest node in the current frontier of the search tree. The time complexity of depth-first graph search is bounded by the size of the state space (which may be infinite, of course). The depth limit search solves the infinite-path problem. Unfortunately, it also introduces an additional source of incompleteness if we choose l < d, that is, the shallowest goal is beyond the depth limit.
- Greedy best-first search tries to expand the node that is closest to the goal, on the grounds that this is likely to lead to a solution quickly. Thus, it evaluates nodes by using just the heuristic function; that is, f (n) = h(n).
- The most widely known form of best-first search is called A∗ search. It evaluates nodes by combining g(n), the cost to reach the node, and h(n), the cost to get from the node to the goal: f(n) = g(n) + h(n). Since g(n) gives the path cost from the start node to node n, and h(n) is the estimated cost of the cheapest path from n to the goal, we have f (n) = estimated cost of the cheapest solution through n. The algorithm is identical to uniform-cost search except that A∗ uses g + h instead of g.

1.3 2. Objective 1

TODO: Experiment and document: metrics of A* searches with these heuristics

- Run A* planning searches using the heuristics you have implemented on `air_cargo_p1`, `air_cargo_p2` and `air_cargo_p3`. Provide metrics on number of node expansions required, number of goal tests, time elapsed, and optimality of solution for each search algorithm and include the results in your report.

1.3.1 2.1. Metrics

All metrics of all searches with and without heuristics are collected in one file data.csv.

```
In [17]: data = pd.read_csv('data.csv')
         data.iloc[:,:10]
```

Out[17]:

	Pr	S	Opt	Com	Heu	Exp	Time_s	G_Test	N_Nod	P_L
0	1	BFS	Yes	Yes	No	43.0	0.0521	56.0	180.0	6.0
1	1	BFT	Yes	Yes	No	1458.0	1.2936	1459.0	5960.0	6.0
2	1	DFG	No	Yes	No	21.0	0.0268	22.0	84.0	20.0
3	1	DLS	No	Yes	No	101.0	0.1309	271.0	414.0	50.0
4	1	UCS	Yes	Yes	No	55.0	0.0548	57.0	224.0	6.0
5	1	RBF	Yes	Yes	No	4229.0	3.8351	4230.0	17023.0	6.0
6	1	GBF	Yes	Yes	No	7.0	0.01	9.0	28.0	6.0
7	1	ASH	Yes	Yes	Yes	55.0	0.0579	57.0	224.0	6.0
8	1	AIP	Yes	Yes	Yes	41.0	0.0569	43.0	170.0	6.0
9	1	APL	Yes	Yes	Yes	11.0	1.9697	13.0	50.0	6.0
10	2	BFS	Yes	Yes	No	3343.0	17.4116	4609.0	30509.0	9.0
11	2	BFT	Yes	No	No	NaN	>1200	NaN	NaN	NaN
12	2	DFG	No	Yes	No	624.0	4.2939	625.0	5602.0	619.0
13	2	DLS	No	No	No	NaN	>1200	NaN	NaN	NaN
14	2	UCS	Yes	Yes	No	4853.0	16.4781	4855.0	44041.0	9.0
15	2	RBF	Yes	No	No	NaN	>1200	NaN	NaN	NaN
16	2	GBF	Yes	Yes	No	998.0	3.2678	1000.0	8982.0	15.0
17	2	ASH	Yes	Yes	Yes	4853.0	16.3519	4855.0	44041.0	9.0
18	2	AIP	Yes	Yes	Yes	1450.0	5.8755	1452.0	13303.0	9.0
19	2	APL	Yes	Yes	Yes	86.0	172.7009	88.0	841.0	9.0
20	3	BFS	Yes	Yes	No	14663.0	126.5863	18098.0	129631.0	12.0
21	3	BFT	Yes	No	No	NaN	>1200	NaN	NaN	NaN
22	3	DFG	No	Yes	No	408.0	2.2993	409.0	3364.0	392.0
23	3	DLS	No	No	No	NaN	>1200	NaN	NaN	NaN
24	3	UCS	Yes	Yes	No	18223.0	69.9119	18225.0	159618.0	12.0
25	3	RBF	Yes	No	No	NaN	>1200	NaN	NaN	NaN
26	3	GBF	Yes	Yes	No	5578.0	21.4791	5580.0	49150.0	22.0
27	3	ASH	Yes	Yes	Yes	18223.0	70.1065	18225.0	159618.0	12.0
28	3	AIP	Yes	Yes	Yes	5040.0	23.1589	5042.0	44944.0	12.0
29	3	APL	Yes	Yes	Yes	325.0	848.4034	327.0	3002.0	12.0

2.1.1. First look Our goal is to find an optimal solution for each air cargo problem, which is a search algorithm to find the lowest path among all possible paths from start to goal, spending minimum time. Let's first find the length of the lowest path and estimate, how many searches got this result:

2.1.2. The length of the lowest path

```
In [1121]: args = data['P_L'] # data to estimate
```

```
In [1122]: lowest_path_length = get_path_length(args)
```

```
The length of the lowest path is  6
```

2.1.3. Failed algorithms

```
In [1123]: data = drop_fail_alg(data,args,lowest_path_length)
```

```
Algorithm BFT from problem 2 was failed and has been dropped out
Algorithm DLS from problem 2 was failed and has been dropped out
Algorithm RBF from problem 2 was failed and has been dropped out
Algorithm BFT from problem 3 was failed and has been dropped out
Algorithm DLS from problem 3 was failed and has been dropped out
Algorithm RBF from problem 3 was failed and has been dropped out
Total search algorithms for the best solution:  8
Total failed search algorithms :  6
```

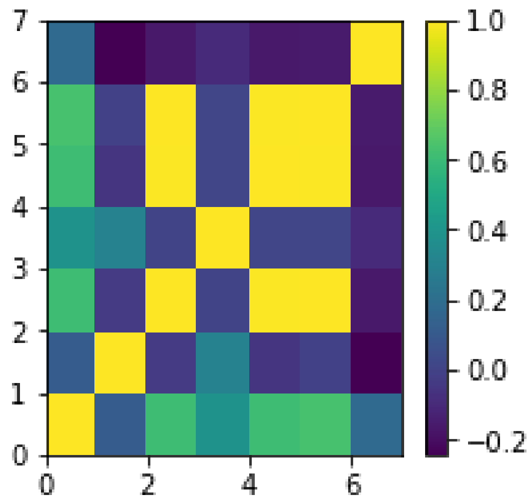
2.1.4. Data correlation

```
In [1124]: plan_cor = data.iloc[:,:]
           d = {'No' : 0, 'Yes' : 1}
           plan_cor['Heu'] = plan_cor['Heu'].map(d)
           cor=pd.DataFrame.corr(plan_cor)
           cor
```

Out[1124]:

	Pr	Heu	Exp	Time_s	G_Test	N_Nod	
Pr	1.000000	0.116335	0.613080	0.391319	0.612359	0.637589	0.18
Heu	0.116335	1.000000	-0.026324	0.309269	-0.051986	-0.007454	-0.24
Exp	0.613080	-0.026324	1.000000	0.010076	0.993479	0.996082	-0.15
Time_s	0.391319	0.309269	0.010076	1.000000	0.016752	0.017765	-0.09
G_Test	0.612359	-0.051986	0.993479	0.016752	1.000000	0.990879	-0.16
N_Nod	0.637589	-0.007454	0.996082	0.017765	0.990879	1.000000	-0.15
P_L	0.183310	-0.242152	-0.159783	-0.095074	-0.161016	-0.152519	1.00

```
In [1125]: plt.figure(figsize=(3,3))
           plt.pcolor(cor)
           plt.colorbar()
           plt.show()
```



2.1.5. First conclusions: As we could suppose, the time elapsed to solve the concrete problem depends on problem complexity, which implies the heuristics usage and test of more nodes for the better precision.

1.4 3. Objective 2

TODO: Include the following in your written analysis.

- Provide an optimal plan for Problems 1, 2, and 3.
- Compare and contrast non-heuristic search result metrics (optimality, time elapsed, number of node expansions) for Problems 1,2, and 3. Include breadth-first, depth-first, and at least one other uninformed non-heuristic search in your comparison; Your third choice of non-heuristic search may be skipped for Problem 3 if it takes longer than 10 minutes to run, but a note in this case should be included.
- Compare and contrast heuristic search result metrics using A* with the "ignore preconditions" and "level-sum" heuristics for Problems 1, 2, and 3.
- What was the best heuristic used in these problems? Was it better than non-heuristic search planning methods for all problems? Why or why not?
- Provide tables or other visual aids as needed for clarity in your discussion.

1.4.1 3.1. Problem 1

Let's get the main information about problem 1 sorted by heuristic usage with ascending respect to time elapsed:

```
In [1152]: problem1 = get_prob_data(data, 1)
           problem1.iloc[:,:10]
```

```
Out[1152]:    Pr    S  Opt  Com  Heu   Exp  Time_s  G_Test   N_Nod  P_L
           0   1  BFS  Yes  Yes    0  43.0  0.0521    56.0   180.0  6.0
```

1	1	BFT	Yes	Yes	0	1458.0	1.2936	1459.0	5960.0	6.0
2	1	DFG	No	Yes	0	21.0	0.0268	22.0	84.0	20.0
3	1	DLS	No	Yes	0	101.0	0.1309	271.0	414.0	50.0
4	1	UCS	Yes	Yes	0	55.0	0.0548	57.0	224.0	6.0
5	1	RBF	Yes	Yes	0	4229.0	3.8351	4230.0	17023.0	6.0
6	1	GBF	Yes	Yes	0	7.0	0.0100	9.0	28.0	6.0
7	1	ASH	Yes	Yes	1	55.0	0.0579	57.0	224.0	6.0
8	1	AIP	Yes	Yes	1	41.0	0.0569	43.0	170.0	6.0
9	1	APL	Yes	Yes	1	11.0	1.9697	13.0	50.0	6.0

3.1.1. First look

```
In [1153]: args = problem1['P_L']
           lowest_path_length = get_path_length(args)
```

```
The length of the lowest path is  6
```

```
In [1154]: problem1 = get_prob_data(data, 1)
           d = get_dict(problem1);
           problem1 = drop_worse_alg(problem1,args,d,lowest_path_length)
```

```
Algorithm DFG from Problem 2 got the more worse result 20
comparing to the best one with length = 6
Algorithm DLS from Problem 2 got the more worse result 50
comparing to the best one with length = 6
Total better search algorithms :  8
Total worse search algorithms :  2
```

3.1.2. First results All searches got the solution.

As we can see, the worst results were given by Depth First Graph Search and Depth Limited Search. But besides, this algorythms are not optimal for the current problems because of two reasons:

- The time complexity of depth-first graph search is bounded by the size of the state space, which may be infinite in this problem. In worth case, planes would fly without landing while they will have a fuel.
- They does not consider if a node is better than another, it simply explores the nodes that take it as deep as possible, that is why got the worst results.

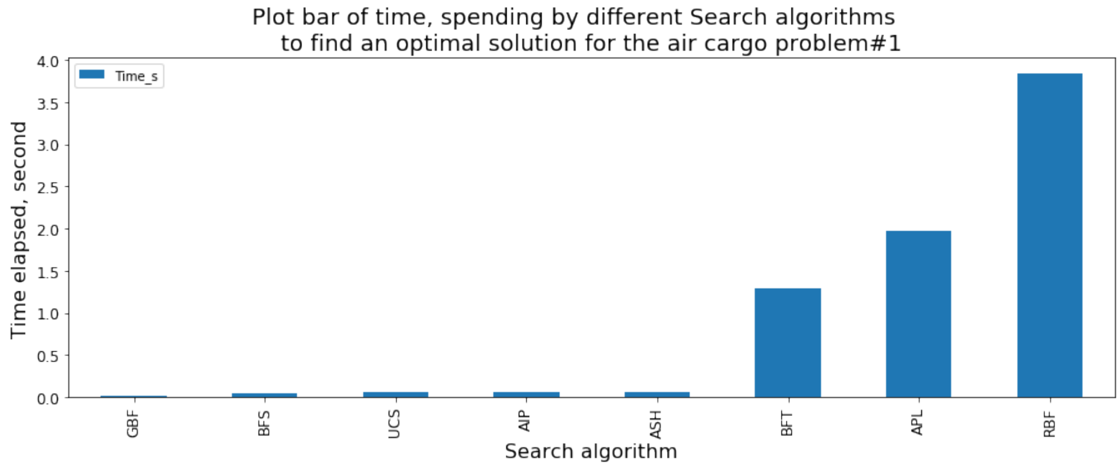
All optimal non-heuristic searches like Breadth First Search, Breadth First Tree Search, Uniform Cost Search, Recursive Best First Search and Greedy Best First Graph Search as well as all A* heuristic searches have the same result for the Plan length, but different in all other points. Let's compare and contrast them to get the best solution.

3.1.3. Result visualization

```
In [1155]: problem1 = drop_unnes_colum(problem1)
```

```
In [1156]: problem1.index=['BFS','BFT','UCS','RBF','GBF','ASH','AIP','APL']
```

```
In [1157]: plot_bar(problem1,1)
```



3.1.4. Best solution for the Problem#1

```
In [1136]: solution(1,"GBF",lowest_path_length)
```

```
Out[1136]: Step 1       Load(C1, P1, SFO)
           Step 2       Load(C2, P2, JFK)
           Step 3       Fly(P1, SFO, JFK)
           Step 4       Fly(P2, JFK, SFO)
           Step 5     Unload(C1, P1, JFK)
           Step 6     Unload(C2, P2, SFO)
           Name: 6, dtype: object
```

1.4.2 3.2. Problem 2

```
In [1137]: problem2 = get_prob_data(data, 2)
```

```
In [1138]: problem2.iloc[:,:10]
```

Out[1138]:

	Pr	S	Opt	Com	Heu	Exp	Time_s	G_Test	N_Nod	P_L
10	2	BFS	Yes	Yes	0	3343.0	17.4116	4609.0	30509.0	9.0
12	2	DFG	No	Yes	0	624.0	4.2939	625.0	5602.0	619.0
14	2	UCS	Yes	Yes	0	4853.0	16.4781	4855.0	44041.0	9.0
16	2	GBF	Yes	Yes	0	998.0	3.2678	1000.0	8982.0	15.0
17	2	ASH	Yes	Yes	1	4853.0	16.3519	4855.0	44041.0	9.0
18	2	AIP	Yes	Yes	1	1450.0	5.8755	1452.0	13303.0	9.0
19	2	APL	Yes	Yes	1	86.0	172.7009	88.0	841.0	9.0

3.2.1. First look

```
In [1139]: args = problem2['P_L']
           lowest_path_length = get_path_length(args);
```

```
The length of the lowest path is  9
```

```
In [1140]: problem2 = get_prob_data(data, 2)
           d = get_dict(problem2);
           problem2 = drop_worse_alg(problem2,args,d,lowest_path_length)
```

```
Algorithm DFG from Problem 2 got the more worse result 619
comparing to the best one with length = 6
Algorithm GBF from Problem 2 got the more worse result 15
comparing to the best one with length = 6
Total better search algorithms :  8
Total worse search algorithms :  2
```

3.2.2. First results Not all searches got the solution. The algorithms BFT, DLS and RBF are among failed.

As we can see, the worst result was given by Depth First Graph Search. But besides, this algorythm is not optimal for the current problems because of two reasons:

- The time complexity of depth-first graph search is bounded by the size of the state space, which may be infinite in this problem. In worth case, planes would fly without landing while they will have a fuel.
- It does not consider if a node is better than another, it simply explores the nodes that take it as deep as possible

Among optimal non-heuristic searches only Uniform Cost Search and Breadth First Search got the best result unlike Greedy Best First Graph Search.

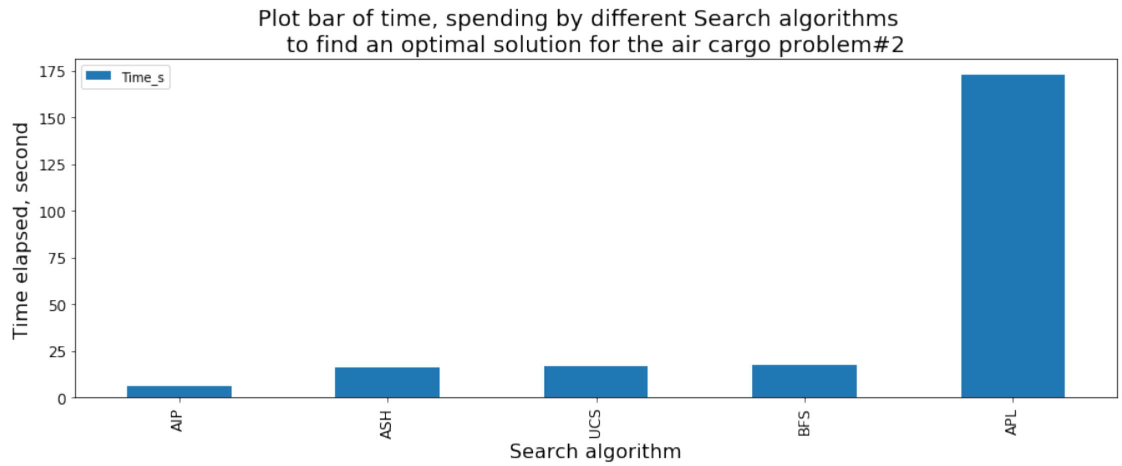
A* heuristic searches have the same best result for the Plan length, but different in all other points. Let's compare and contrast them to get the best solution.

3.2.3. Result visualization

```
In [1141]: problem2 = drop_unnes_colum(problem2)

In [1142]: problem2.index=['BFS','UCS','ASH','AIP','APL']

In [1143]: plot_bar(problem2,2)
```



3.2.4. Best solution for the Problem#2

In [1144]:
```
solution(2,"AIP",lowest_path_length)
```

Out[1144]:
```
Step 1       Load(C3, P3, ATL)
Step 2       Fly(P3, ATL, SFO)
Step 3     Unload(C3, P3, SFO)
Step 4       Load(C2, P2, JFK)
Step 5       Fly(P2, JFK, SFO)
Step 6     Unload(C2, P2, SFO)
Step 7       Load(C1, P1, SFO)
Step 8       Fly(P1, SFO, JFK)
Step 9     Unload(C1, P1, JFK)
Name: 18, dtype: object
```

1.4.3 3.3. Roblem 3

In [1145]:
```
problem3 = get_prob_data(data,3)
problem3.iloc[:,:10]
```

Out[1145]:

	Pr	S	Opt	Com	Heu	Exp	Time_s	G_Test	N_Nod	P_L
20	3	BFS	Yes	Yes	0	14663.0	126.5863	18098.0	129631.0	12.0
22	3	DFG	No	Yes	0	408.0	2.2993	409.0	3364.0	392.0
24	3	UCS	Yes	Yes	0	18223.0	69.9119	18225.0	159618.0	12.0
26	3	GBF	Yes	Yes	0	5578.0	21.4791	5580.0	49150.0	22.0
27	3	ASH	Yes	Yes	1	18223.0	70.1065	18225.0	159618.0	12.0
28	3	AIP	Yes	Yes	1	5040.0	23.1589	5042.0	44944.0	12.0
29	3	APL	Yes	Yes	1	325.0	848.4034	327.0	3002.0	12.0

3.3.1. First look

```
In [1146]: args = problem3['P_L']
           lowest_path_length = get_path_length(args);
```

```
The length of the lowest path is  12
```

```
In [1147]: problem3 = get_prob_data(data, 3)
           d = get_dict(problem3);
           problem3 = drop_worse_alg(problem3,args,d,lowest_path_length)
```

```
Algorithm DFG from Problem 2 got the more worse result 392
comparing to the best one with length = 6
Algorithm GBF from Problem 2 got the more worse result 22
comparing to the best one with length = 6
Total better search algorithms :  8
Total worse search algorithms :  2
```

3.3.2. First Result Not all searches got the solution. The algorithms BFT, DLS and RBF are among failed.

As we can see, the worst result was given by Depth First Graph Search. But besides, this algorythm is not optimal for the current problems because of two reasons:

- The time complexity of depth-first graph search is bounded by the size of the state space, which may be infinite in this problem. In worth case, planes would fly without landing while they will have a fuel.
- It does not consider if a node is better than another, it simply explores the nodes that take it as deep as possible

Among optimal non-heuristic searches only Uniform Cost Search got the best result unlike Breadth First Search and Greedy Best First Graph Search.

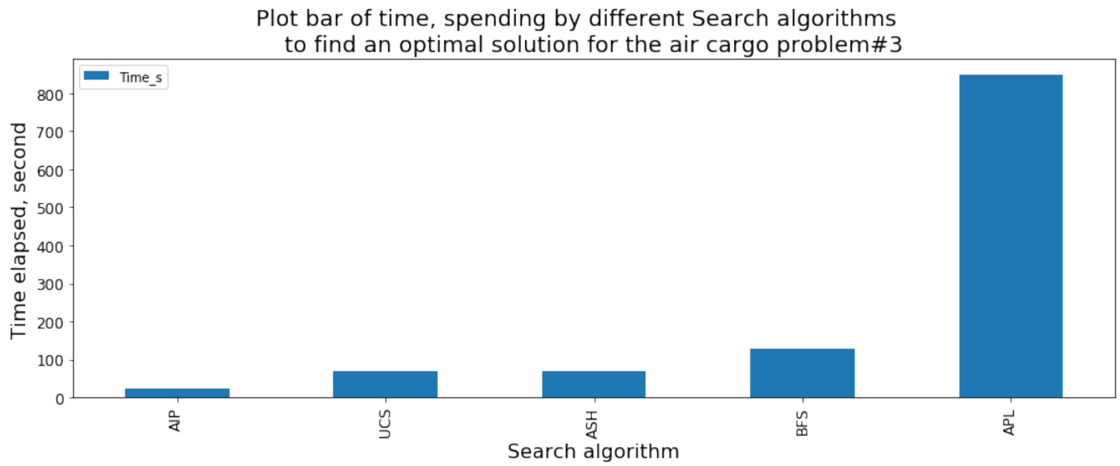
A* heuristic searches have the same best result for the Plan length, but different in all other points. Let's compare and contrast them to get the best solution.

3.3.3. Result visualization

```
In [1148]: problem3 = drop_unnes_colum(problem3)
```

```
In [1149]: problem3.index=['BFS','UCS','ASH','AIP','APL']
```

```
In [1150]: plot_bar(problem3,3)
```



3.3.4. Best solution for the Problem#3

```
In [14]: solution(3,"AIP",lowest_path_length)
```

```
Step 1        Load(C2, P2, JFK)
Step 2        Fly(P2, JFK, ORD)
Step 3        Load(C4, P2, ORD)
Step 4        Fly(P2, ORD, SFO)
Step 5      Unload(C4, P2, SFO)
Step 6        Load(C1, P1, SFO)
Step 7        Fly(P1, SFO, ATL)
Step 8        Load(C3, P1, ATL)
Step 9        Fly(P1, ATL, JFK)
Step 10     Unload(C3, P1, JFK)
Step 11     Unload(C2, P2, SFO)
Step 12     Unload(C1, P1, JFK)
Name: 28, dtype: object
```