Planning Graph Project Analysis

May 1, 2017

1 Data Wrangling Process

Here we define a few functions to assist us with extracting the captured data from the log files

In [1]:
```
import re
import pandas as pd
def get_alg(search_algs):
    r = re.compile(r".+using\s([a-z_]+)")
    l = [m.group(1) for line in search_algs for m in [r.match(line)] if m]
    return l
def get_plan_values(plan_lines):
    r = re.compile(r'.+:\s([0-9]+).+:\s([0-9\.]+)')
    l = [(m.group(1),m.group(2)) for line in plan_lines for m in [r.match(l
    return l

def create_dataframe(results,problem_name):
    results[0][3] = results[0][3] + ' with h_1'
    results[0][4] = results[0][4] + ' with h_ignore_preconditions'
    results[0][5] = results[0][5] + ' with h_pg_levelsum'
    d = {'Algorithm':results[0],'Problem':problem_name}
    plan_lengths = [p[0] for p in results[1]]
    exec_times = [p[1] for p in results[1]]
    expansions = [v[0] for v in results[2]]
    goal_tests = [v[1] for v in results[2]]
    new_nodes = [v[2] for v in results[2]]
    d['Plan Length'] = plan_lengths
    d['Execution Times(sec)'] = exec_times
    d['Expansions'] = expansions
    d['Goal Tests'] = goal_tests
    d['New Nodes'] = new_nodes
    df = pd.DataFrame(d)

    return df
```

In [2]:
```
def extract_data(filename):
    with open(filename, 'r') as tf:
        lines = tf.readlines()
```

```
            search_algs = [l.strip() for l in lines if "Solving Air Cargo" in l
            search_algs = get_alg(search_algs)
            plan_lines = [l.strip() for l in lines if "Plan length" in l]
            plan_lines = get_plan_values(plan_lines)
            node_vals = [l.strip().split() for l in lines if '       ' in l]
            return [search_algs,plan_lines,node_vals]
```

Now we extract the data and place them into list data structures

```
In [3]: l1 = extract_data("aircargo1.txt")
        l2 = extract_data("aircargo2_results.txt")
        l3 = extract_data("aircargo3_results.txt")
```

Finally we place the data into Panda DataFrames for table reprensentation

```
In [4]: df1 = create_dataframe(l1,'Air Cargo 1')
        df2 = create_dataframe(l2,'Air Cargo 2')
        df3 = create_dataframe(l3, 'Air Cargo 3')
```

```
In [5]: df = pd.concat([df1,df2,df3])
```

Now we want to split the data into heuristic and non-heuristic results

```
In [6]: a_star_algs = ['astar_search with h_1', 'astar_search with h_ignore_precond
        df_non_heuristic = df[df['Algorithm'].isin(a_star_algs) == False]
        df_heuristic = df[df['Algorithm'].isin(a_star_algs) == True]
```

2 Optimal Plan for Air Cargo Problem 1

The optimal plan for **Air Cargo Problem 1** is:
Load(C1, P1, SFO)
Load(C2, P2, JFK)
Fly(P2, JFK, SFO)
Unload(C2, P2, SFO)
Fly(P1, SFO, JFK)
Unload(C1, P1, JFK)

3 Optimal Plan for Air Cargo Problem 2

The optimal plan for **Air Cargo Problem 2** is:
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(C1, P1, JFK)
Unload(C2, P2, SFO)
Unload(C3, P3, SFO)

4 Optimal Plan for Air Cargo Problem 3

The optimal plan for **Air Cargo Problem 3** is:
Load(C1, P1, SFO)
Fly(P1, SFO, ATL)
Load(C3, P1, ATL)
Fly(P1, ATL, JFK)
Unload(C1, P1, JFK)
Load(C2, P2, JFK)
Fly(P2, JFK, ORD)
Load(C4, P2, ORD)
Fly(P2, ORD, SFO)
Unload(C2, P2, SFO)
Unload(C3, P1, JFK)
Unload(C4, P2, SFO)

5 Non -heuristic Search Analysis

DFS graph search had the minimum value in Execution Time, Expansions, Goal Tests and New Nodes categories but had the longest plan length in all three problems. BFS had the second shortest execution time, but achieved the optimal plan length. The difference in execution time for BFS was negligible for Cargo problem 1, but was significant for Problems two and three, running 4x and 55x longer than DFS respectively. Uniform cost search found an optimal plan, but at high execution time cost, running 10x and 178x longer than DFS respectively for problems two and three. The execution time seems to be proportional to the number of expansions and new nodes created which is expected.

```
In [7]: df_non_heuristic

Out[7]:                  Algorithm  Execution Times(sec) Expansions Goal Tests  \
        0      breadth_first_search  0.055707235361405406         43         56
        1  depth_first_graph_search  0.025222794381489857         21         22
        2       uniform_cost_search   0.08031563052932626         55         57
        0      breadth_first_search     33.87593957213068       3343       4609
        1  depth_first_graph_search     8.196923054605156        624        625
        2       uniform_cost_search     83.07583435626536       4853       4855
        0      breadth_first_search    222.20152550204733      14663      18098
        1  depth_first_graph_search     4.799967402708392        408        409
        2       uniform_cost_search     714.4653903376337      18151      18153

          New Nodes Plan Length      Problem
        0       180           6  Air Cargo 1
        1        84          20  Air Cargo 1
        2       224           6  Air Cargo 1
        0     30509           9  Air Cargo 2
        1      5602         619  Air Cargo 2
        2     44041           9  Air Cargo 2
        0    129631          12  Air Cargo 3
```

1	3364	392	Air Cargo 3
2	159038	12	Air Cargo 3

6 Heuristic Search Analysis

The "ignore preconditions" heuristic had better performance in all metrics except the plan length than the "level-sum" heuristic. The Execution times for the "ignore preconditions" were 4x, 64x and 50x smaller than the execution times of the "level-sum heuristic for Problems one, two and three respectively. It is also worthy to note that the number of expansions, goal tests, and new nodes were the same for the"h_1" heuristic and the "level-sum" heuristic for all problems, yet the the "h_1" heuristic took significantly less time to execute.

```
In [8]: df_heuristic
```

Out[8]:

	Algorithm	Execution Times(sec)	Expansior
3	astar_search with h_1	0.058018912402458794	5
4	astar_search with h_ignore_preconditions	0.054803129851993794	4
5	astar_search with h_pg_levelsum	2.073189452984167	5
3	astar_search with h_1	83.93119112807535	485
4	astar_search with h_ignore_preconditions	23.19300241962357	150
5	astar_search with h_pg_levelsum	1481.0403865083053	485
3	astar_search with h_1	524.804982009623	1815
4	astar_search with h_ignore_preconditions	141.41130358019808	511
5	astar_search with h_pg_levelsum	7068.988206521763	1815

	Goal Tests	New Nodes	Plan Length	Problem
3	57	224	6	Air Cargo 1
4	43	170	6	Air Cargo 1
5	57	224	6	Air Cargo 1
3	4855	44041	9	Air Cargo 2
4	1508	13820	9	Air Cargo 2
5	4855	44041	9	Air Cargo 2
3	18153	159038	12	Air Cargo 3
4	5120	45650	12	Air Cargo 3
5	18153	159038	12	Air Cargo 3

7 Best Heuristic

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?

The best heuristic used in these problems was the "ignore preconditions" heuristic. It acheived the optimal plan length with lower values in all metrics in respect to the BFS search strategy. I would think that the A* search with ignore preconditions would perform better than BFS because A* has an evaluation metric that determines which nodes to expand first. As discussed in Section 10.2.3 of the AIMA book, by relaxing the preconditions metric we are able to compute an estimate on how many actions are needed to achieve the goal in an efficient manner. By performing node expansion based on this evaluation function, we can avoid having to expand nodes that won't lead us to achieving the goal in the most optimal way. BFS, unlike A*, expands all nodes at each level without using any knowledge about whether exploring the sub-graph from this node will lead to the goal state as soon as possible. Thus this could cause the expansion of nodes that may not lead us to the goal state.

```
In [9]: best_algs = ['breadth_first_search', 'astar_search with h_ignore_preconditi
        df_best = df[df['Algorithm'].isin(best_algs) == True]
        df_best
```

```
Out[9]:                                   Algorithm  Execution Times(sec) Expansion
        0                      breadth_first_search  0.055707235361405406         4
        1                 depth_first_graph_search  0.025222794381489857         2
        4  astar_search with h_ignore_preconditions  0.054803129851993794         4
        0                      breadth_first_search     33.87593957213068       334
        1                 depth_first_graph_search     8.196923054605156        62
        4  astar_search with h_ignore_preconditions     23.19300241962357       150
        0                      breadth_first_search    222.20152550204733      1466
        1                 depth_first_graph_search     4.799967402708392        40
        4  astar_search with h_ignore_preconditions    141.41130358019808       511

          Goal Tests  New Nodes  Plan Length      Problem
        0         56        180            6  Air Cargo 1
        1         22         84           20  Air Cargo 1
        4         43        170            6  Air Cargo 1
        0       4609      30509            9  Air Cargo 2
        1        625       5602          619  Air Cargo 2
        4       1508      13820            9  Air Cargo 2
        0      18098     129631           12  Air Cargo 3
        1        409       3364          392  Air Cargo 3
        4       5120      45650           12  Air Cargo 3
```