Heuristic analysis

Udacity Artificial Intelligence Nanodegree – Implement a planning search
Pierre Oberholzer – 17.12.2017

This report describes the difference observed between the different algorithms used to solve the planning search problem of the air cargo provided by Udacity.

Metrics

The function "run.search.py" provides following metrics:

- Path length from start to end goal
- Elapsed time: time needed to perform the search
- Expansions: number of calls to method "problem.actions" performed
  - ⇨ Provides update on how many times the "search network" was expanded
- Goal tests: the number of time the match of action to a goal is checked
- New nodes: number of calls to method "problem.result" performed
  - ⇨ Provides update on how many (action) nodes were added to the "search network"

Experiments

Due to their long computation time (> 1h), some of the searches were removed of the test. The following combinations were performed:

	Tested	**Removed (too long to perform)**
Problem 1	1,2,3,4,5,6,7,8,9,10	-
Problem 2	1,3,4,5,6,7,8,9,10	2,6
Problem 3	1,3,5,7,8,9,10	2,4,6

Analysis

Metrics comparison

The metrics gathered for "Problem 1" are reported on the right figure (Figure 1).

The first observation is that the metrics "Elapsed time" tends to correlates with "New nodes", whereas "Path length" tends to correlate with "Expansions". This shows that the size of the network, and not its update frequency, is fixing the computation time. An efficient search would hence benefit of working on a restricted amount of nodes.

Let's look at the different types of searches applied and proceed by elimination.

Search comparisons on "Problem 1"

It appears that two types of search are particularly inefficient in terms of elapsed time: the "recursive best first search" (nr. 6) and the "breadth first tree search" (nr. 2).

The "breadth first tree search" performs much worse than the "breadth first search" while

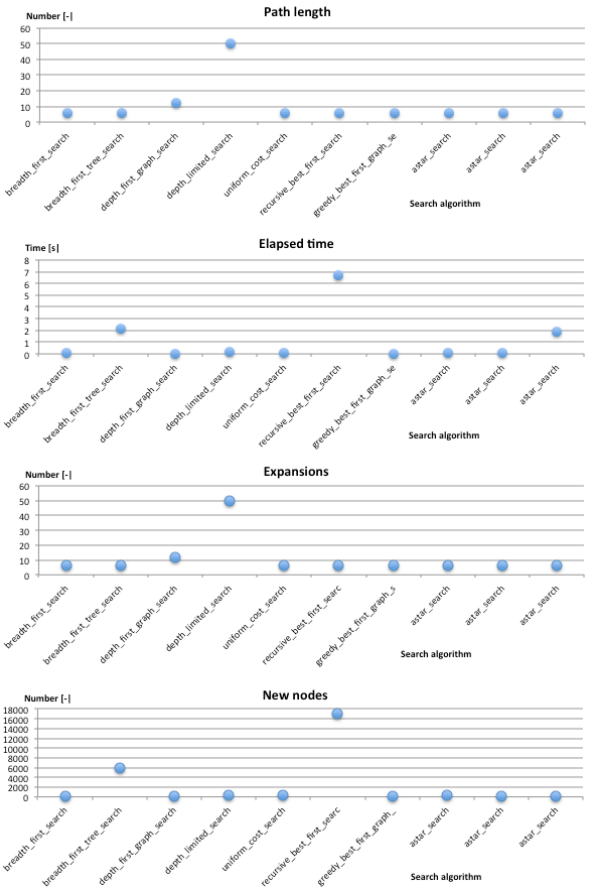

Figure 1 - Metrics "Problem 1"

having a similar construct. The code shows that the first one expands the frontier with all candidate nodes (`frontier.extend(node.expand(problem))`) whereas the second checks each candidate node individually in the new frontier (`for child in node.expand(problem)`). Since the breadth first algorithms expand their search on all possible candidates level by level, the full scan on the new frontier domain is very expensive, making "breadth first search" an expansive choice.

Similarly the "recursive best first search", the most inefficient algorithm in the selection, also makes use of a breadth search and also expand the frontier with all candidates at once, and then performs a reordering of all nodes (`successors.sort(key=lambda x:x.f)`) before recursively applying the search function.

Search comparisons on "Problem 2"

Similarly as for "Problem 1" the "recursive best first search" (nr. 2) and the "breadth first tree search" (nr. 6) were two computationally demanding and were removed from the test.

The third worse search seems hence to be "depth limited search" (nr. 4), which also looks to be related to the important number of nodes tested. The parameter "limit = 50" of the function should probably be minimized to obtain a better performance.

Search comparison for all problems

For Problem 3 the algorithm "depth limited search" (nr. 4) was also removed due to its computation time.

On the next figure (Figure 3) the computation times of all three problems are presented. They are scaled using "breadth_first_search" (nr. 1) as a reference.

Excluding now the algorithms discussed already (nr. 1,2,4,6), we see that the graph search based algorithms (nr. 3,5,7) appear to perform particularly well. As explained in the code comments, the graph search distinguishes in that it doesn't consider all the possibilities leading to a goal, but retain only one.

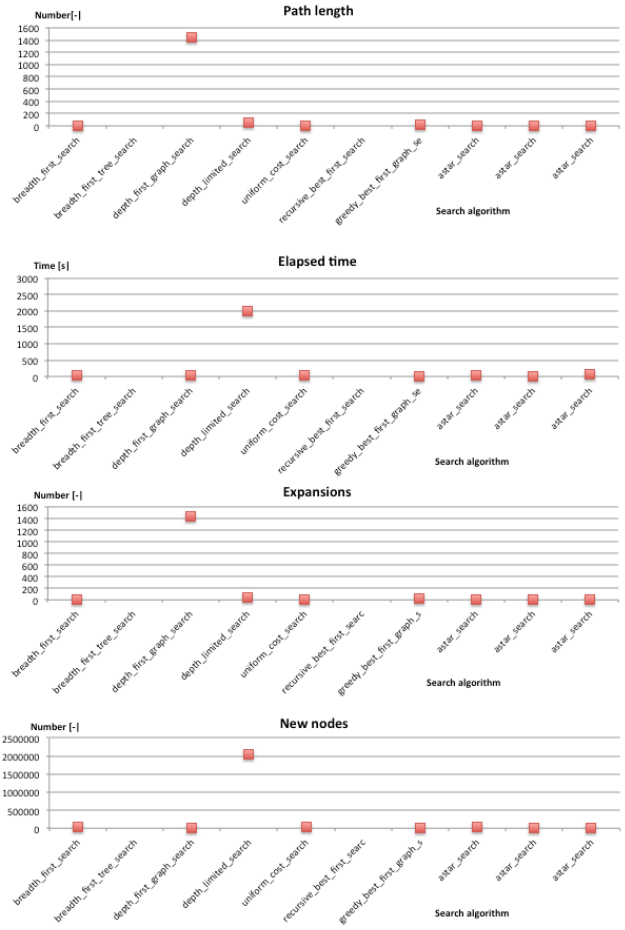

Figure 2 – Metrics "Problem 2"

One should emphasize at this point on the importance of the "path length" as another possible cost metrics. Depending on the problem, the cost of the solution found and its possible deviation from the optimal solution should be put into balance versus computational time. For example a route finding problem or a cargo planning problem will typically largely benefit of having dedicated cost function that might be reflected by the path length, for instance the real geographical distance or the number of different actions (load, fly, unload) respectively. Regarding "path length" in "Problem 2" (Figure 2), it can be seen that the "depth first graph search" (nr. 3) leads actually to a long path, while performing well regarding "elapsed time". It should therefore rather be rejected in the context of this problem type (cargo planning).

This leaves us with two "uninformed" (nr. 5,7) and three "informed" algorithms (nr. 8,9,10). We see that "informed" graph clearly depend on the heuristics used, the best performing heuristic being "ignore preconditions".

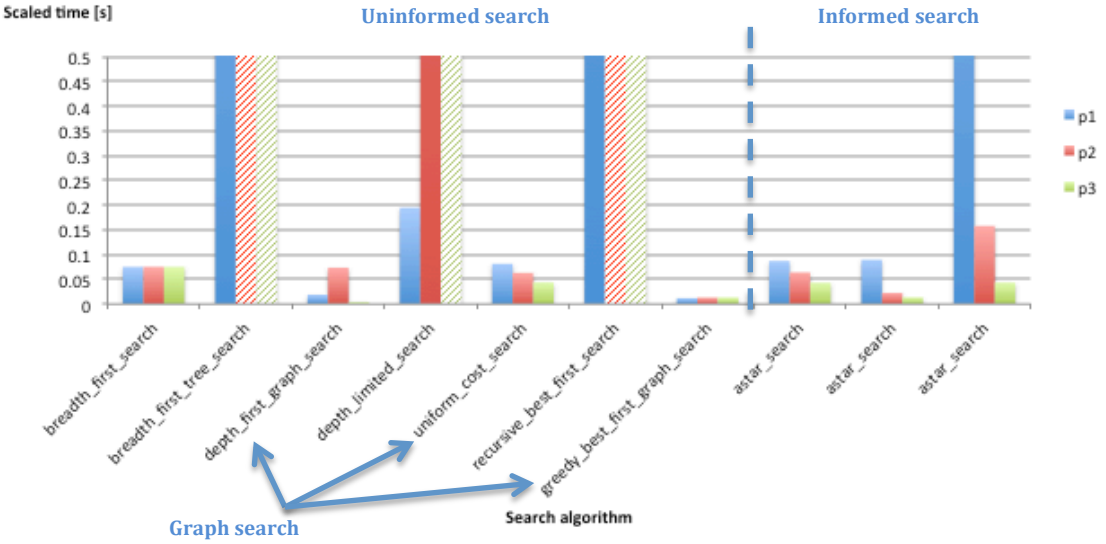

Figure 3 – Search comparison for all problems

From "elapsed time" perspective, the overall winner would be "greedy best first graph search" (nr. 7). However by looking at "Problem 3" metrics (Figure 4), one observes again the trade-off appearing between "path length" and "elapsed time", and hence concludes that "A* search – second heuristic" (nr. 9) would be the most promising challenger c" would be as reflecting a real cost metrics.

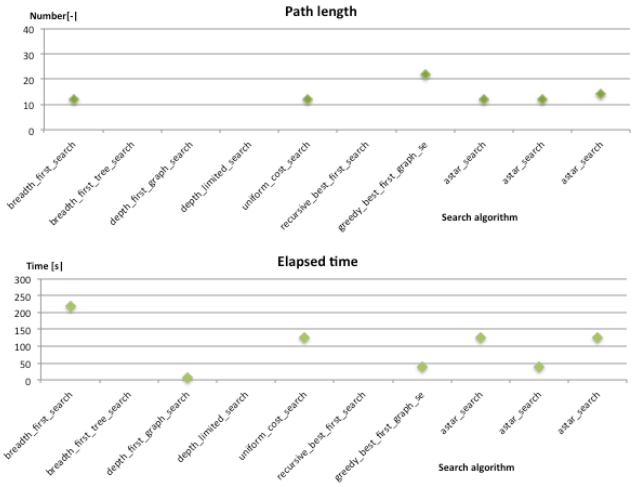

Figure 4 – Metrics "Problem 3"

Optimal sequence of actions

Based on above discussion, we retain "A* search – second heuristic" and show the paths found to the goal for the different problems.

Problem 1:

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

Problem 2:

```
Load(C3, P3, ATL)
Fly(P3, ATL, SFO)
Unload(C3, P3, SFO)
Load(C2, P2, JFK)
Fly(P2, JFK, SFO)
Unload(C2, P2, SFO)
Load(C1, P1, SFO)
Fly(P1, SFO, JFK)
Unload(C1, P1, JFK)
```

Problem 3:

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

Conclusion

After having compared the metrics retrieved by the "run_search" script, we discussed the different search algorithms, pointing out the difference between breadth and depth first algorithms, selected and full node expansion, to conclude that the searches "greedy best first graph search" (nr. 7) and "A* search – second heuristic" (nr. 9) offer the most promising results, for their short "elapsed time" and short "path length" respectively, the final choice having to be evaluated in light of the cost of the real system to be optimized.