Report of Project: Forward-Planning Agent

Israel G. de Oliveira

January 6, 2019

1 Introduction

This report is about the building a Forward-Planning Agent, project part from Artificial Intelligence Nanodegree [1]. The objective is implement functions to able a agent to solve planning problems based on symbolic logic and classical search (progression search), doing experiments with different search algorithms and heuristics [2]. This project is based on book Artificial intelligence: a modern approach [3].

1.1 Planning Problems

The planning problems used is based on the example Air Cargo Problem, 10.1.1 from [3], with four sized variants [2], in according with the Table 1.

Table 1: Air Cargo Problems

Problem nº	Planes	Cargos	Airports
1	2	2	2
2	3	3	3
3	2	4	4
4	2	5	4

One way to solve this problems is use a graph representation connecting all logic literals and possible actions. With a graph, it is possible to use a search algorithm to find a solution [3]. Not only, for some search algorithms, it is possible to use a heuristic method to estimate the cost for each action (connection in graph), in some cases, improving the finding for a solution (less time expending on searching).

1.2 Search Algorithms

The search algorithms proposed to use are Breadth First Search (BFS), Depth First Graph Search (DFS), Uniform Cost Search (UCS), Greedy Best First Graph Search (GBFGS) and A* Search [2]. All this algorithm is already implemented with the framework provided by Udacity, available in [2].

BFS is a instance of the tree search with FIFO (First In, First Out) queue. This algorithm is complete (completeness aspect). The solution is not necessarily optimal: because when a objective node is found at a (depth) level d it is not necessarily the best objective node. The total new nodes is $O(b^d)$ with b successors.

Considering the same step cost 1 for all nodes, we have the BFS. For achieve the UCS, we do a extended version considering a step cost function $g(n)$, for a n-node, then is possible to achieve a optimal solution. Basically, expanding the node with the lowest cost. There are more two differences from BFS: the first is when the objective node is selected for expansion, and not when it is generated, the second is the additional test, to find a possible better path for a objective node. The solution is, generally, optimal. The completeness is not guaranteed, because the number of steps in a path is not relevant, only your cost. If there is a infinity sequence of actions with zero cost (like *NoOp* actions), the search is not finite. The complexity is $O(b^{1+\lfloor C*/\epsilon \rfloor})$, with the optimal solution cost $C*$ and lowest action cost ϵ. If all actions have the same cost, the complexity is $O(b^{d+1})$. In this case, UCS expands all nodes of the same depth, even though the solution has already been found (at this depth).

DFS is a instance of graph (tree) search with a LIFO (Last in, First out) queue. This algorithm search the nodes without repetition (and redundant paths). The graph version is finite, but the tree not. The complexity is $O(b^M)$, with m the max depth of any node. Observe that m can be infinite, even greater than the state space. The advantage in use this algorithm is the space complexity: $O(bm)$, using much less memory. As the algorithm returns the solution when it finds the first objective node, ignoring other possible best nodes, the solution is not, necessarily, optimal.

GBFGS try expand the node nearest from the objective, with the idea of finding the solution quickly. In this way, only the heuristic function is considered. Even with a finite states, the completeness is not guaranteed. The complexity is $O(b^{\epsilon d})$, with m the max depth of state-space. The solution is not, necessarily, optimal because the greedy aspect, trying always get closer to node objective, choosing paths larger than possible others with intermediate nodes further from the node objective.

A* (so-called *a-star*), differing from UCS only by the cost function, considering plus a heuristic cost function $h(n)$ from a n-node to the objective node, resulting the cost function $f(n) = g(n) + h(n)$. Satisfying certain conditions, the completeness and the optimal solution is guaranteed. The complexity is $O(b^{\epsilon d})$, where the d the detph solution and ϵ is the heuristic error. Basically, Basically, the performance depends on the quality of the heuristics used.

1.3 Heuristics

Basically, with a good heuristic is possible to achieve a better performance using a search algorithm. All descriptions below are copied from the [2].

- **LevelCost**: "The level cost is a helper function used by MaxLevel and LevelSum. The level cost of a goal is equal to the level number of the first literal layer in the planning graph where the goal literal appears."
- **MaxLevel**: "The max-level heuristic simply takes the maximum level cost of any of the goals; this is admissible, but not necessarily accurate" [3].
- **LevelSum**: "The level sum heuristic, following the subgoal independence assumption, returns the sum of the level costs of the goals; this can be inadmissible but works well in practice for problems that are largely decomposable" [3].
- **SetLevel**: "The set-level heuristic finds the level at which all the literals in the conjunctive goal appear in the planning graph without any pair of them being mutually exclusive" [3].

2 Search Algorithms Results

For all experiments, was used a Intel(R) Xeon(R) CPU E3-1241 v3 @ 3.50GHz, on Ubuntu 18.04.1 LTS, Linux kernel 4.15.0-43-generic and PyPy 6.0.0 with GCC 4.8.2. All results, without the `my_planning_graph.py`, is availiable in [4].

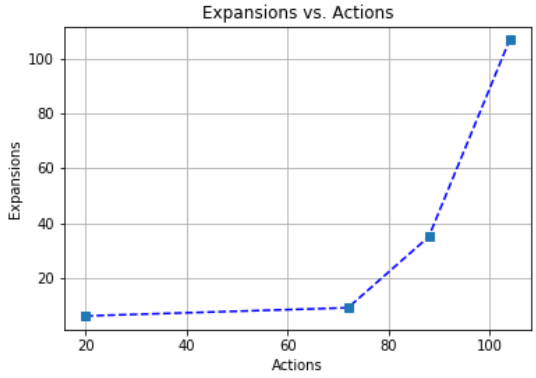

Figure 1: Search Method GBFGS w. SetLevel

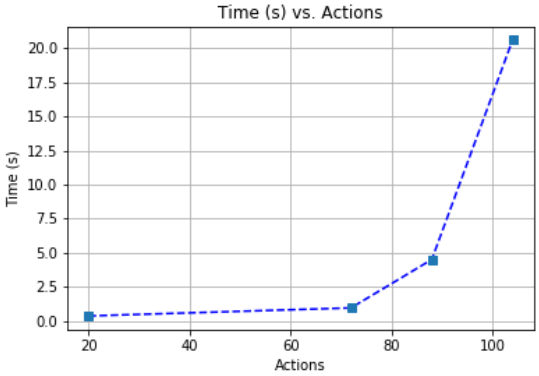

Figure 2: Search Method GBFGS w. SetLevel

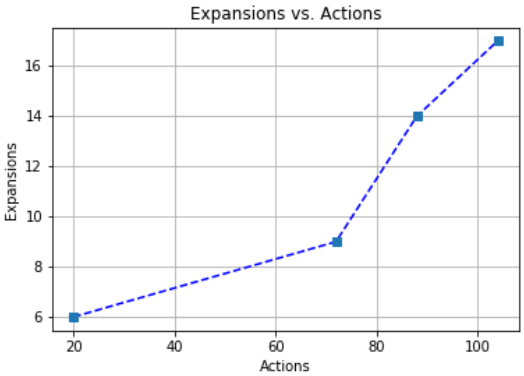

Figure 3: Search Method GBFGS w. LevelSum

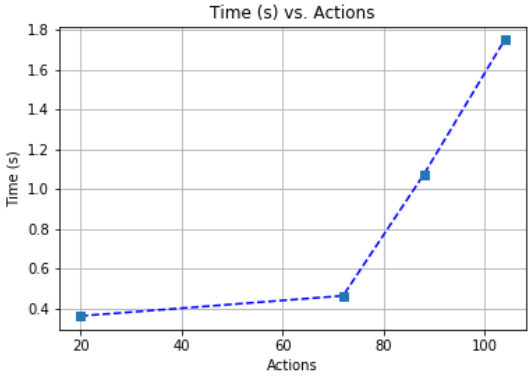

Figure 4: Search Method GBFGS w. LevelSum

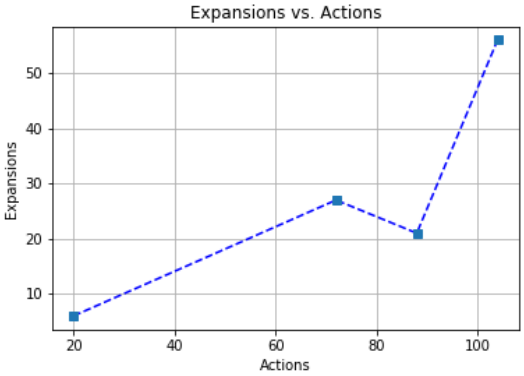

Figure 5: Search Method GBFGS w. MaxLevel

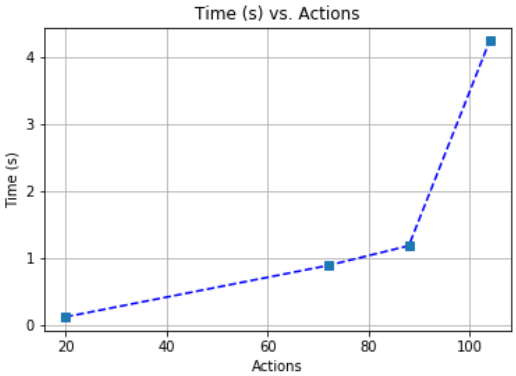

Figure 6: Search Method GBFGS w. MaxLevel

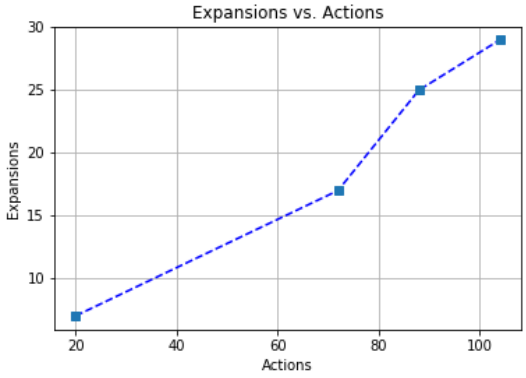

Figure 7: Search Method GBFGS w. Unmet Goals

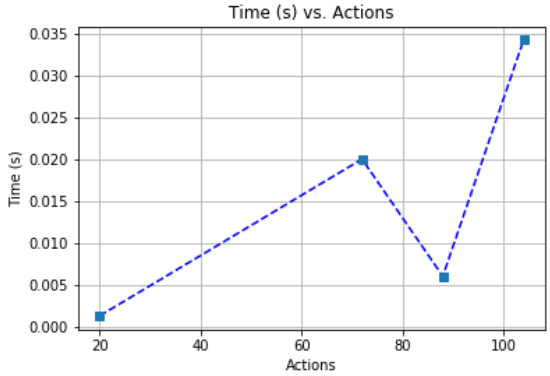

Figure 8: Search Method GBFGS w. Unmet Goals

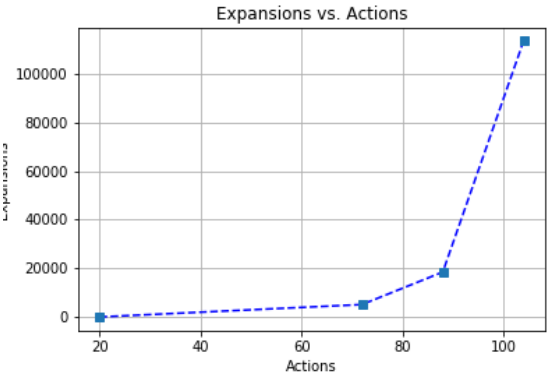

Figure 9: Search Method UCS

2.1 Greedy Best First Graph Search with SetLevel

2.2 Greedy Best First Graph Search with SetLevel

2.3 Greedy Best First Graph Search with LevelSum

2.4 Greedy Best First Graph Search with LevelSum

2.5 Greedy Best First Graph Search with MaxLevel

2.6 Greedy Best First Graph Search with MaxLevel

2.7 Greedy Best First Graph Search with Unmet Goals

2.8 Greedy Best First Graph Search with Unmet Goals

2.9 Uniform Cost Search

2.10 Uniform Cost Search

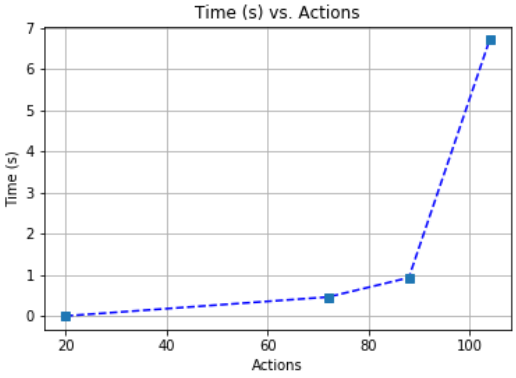

Figure 10: Search Method UCS

2.11 Depth First Graph Search

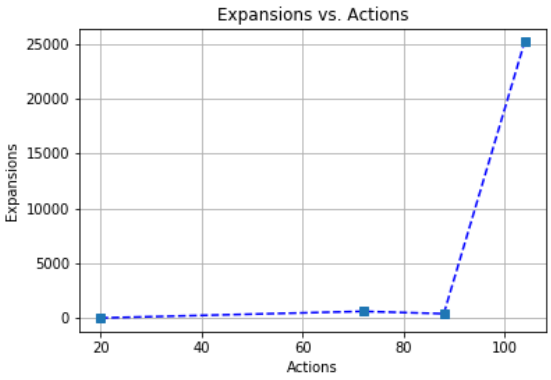

Figure 11: Search Method DFGS

2.12 Depth First Graph Search

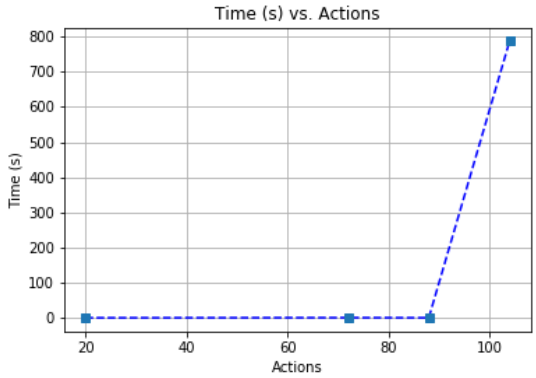

Figure 12: Search Method DFGS

2.13 A* Search with SetLevel

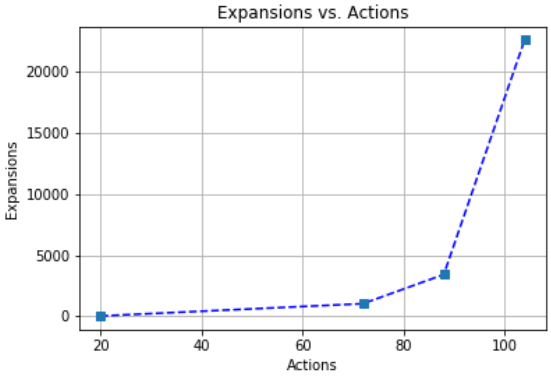

Figure 13: Search Method AS w. SetLevel

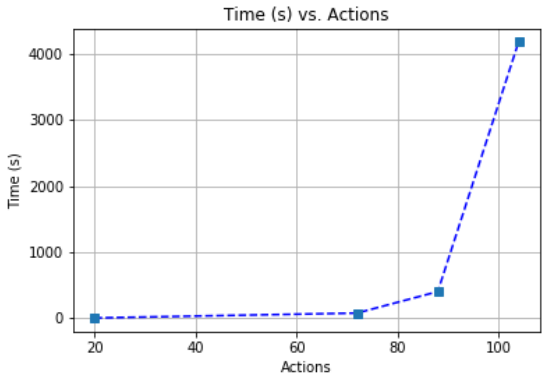

Figure 14: Search Method AS w. SetLevel

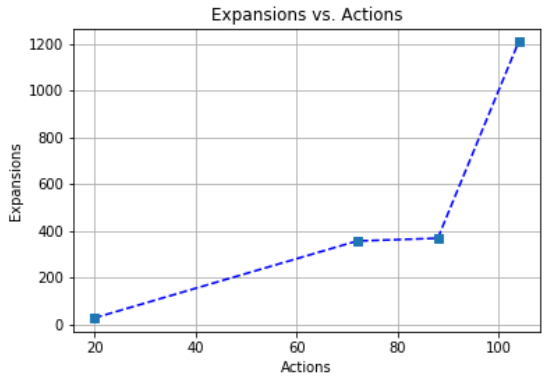

Figure 15: Search Method AS w. LevelSum

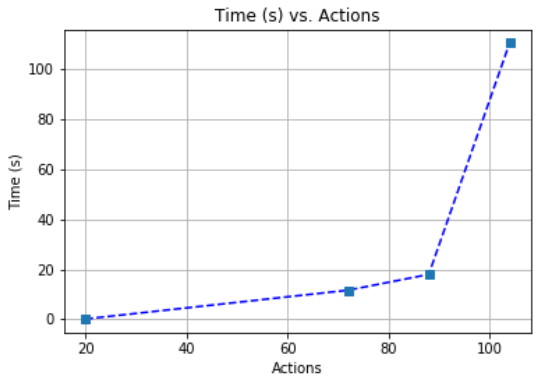

Figure 16: Search Method AS w. LevelSum

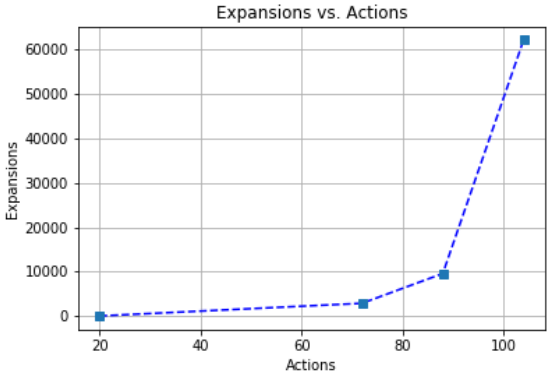

Figure 17: Search Method AS w. MaxLevel

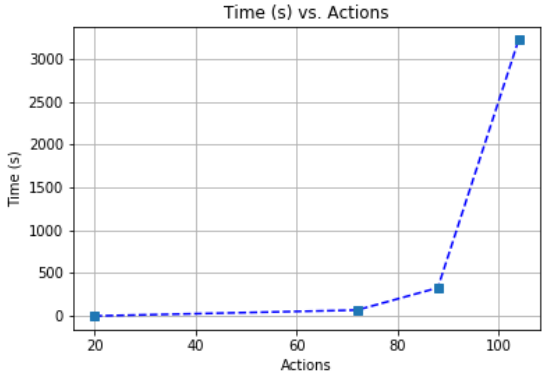

Figure 18: Search Method AS w. MaxLevel

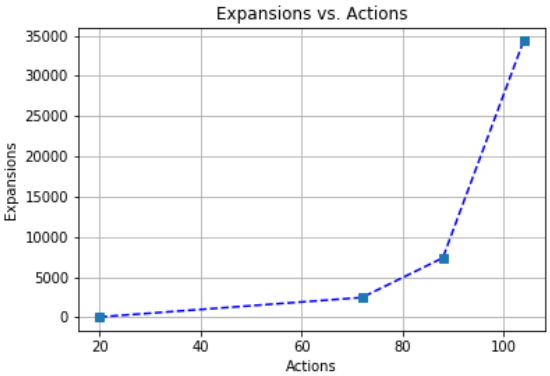

Figure 19: Search Method AS w. Unmet Goals

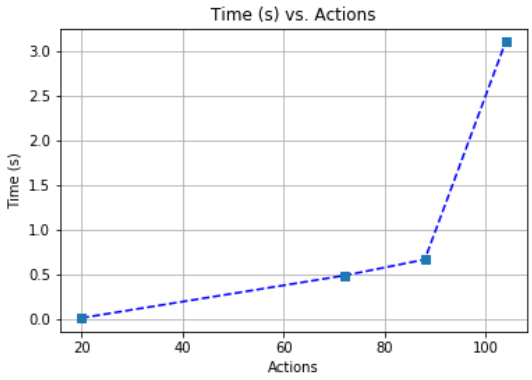

Figure 20: Search Method AS w. Unmet Goals

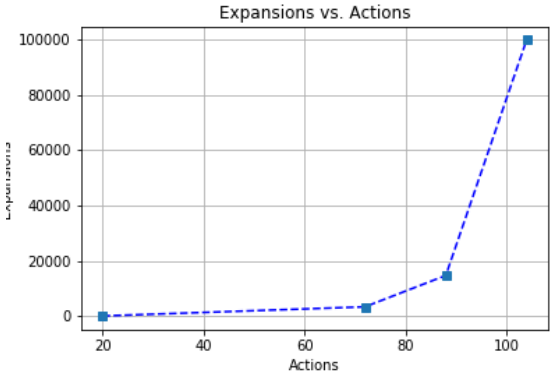

Figure 21: Search Method BFS

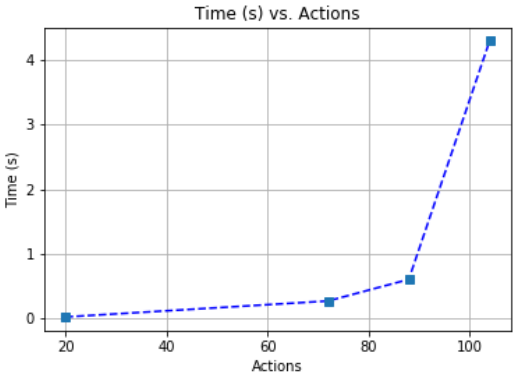

Figure 22: Search Method BFS

2.14 A* Search with SetLevel

2.15 A* Search with LevelSum

2.16 A* Search with LevelSum

2.17 A* Search with MaxLevel

2.18 A* Search with MaxLevel

2.19 A* Search with Unmet Goals

2.20 A* Search with Unmet Goals

2.21 Breadth First Search

2.22 Breadth First Search

2.23 Search Method vs. Plan length

Table 2: Air Cargo Problem 1

Search Method	Plan length
BFS	9
DFGS	619
UCS	9
GBFGS w. Unmet Goals	9
GBFGS w. LevelSum	9
GBFGS w. MaxLevel	9
GBFGS w. SetLevel	9
AS w. Unmet Goals	9
AS w. LevelSum	9
AS w. MaxLevel	9
AS w. SetLevel	9

Table 3: Air Cargo Problem 2

Search Method	Plan length
BFS	14
DFGS	24132
UCS	14
GBFGS w. Unmet Goals	18
GBFGS w. LevelSum	17
GBFGS w. MaxLevel	17
GBFGS w. SetLevel	23
AS w. Unmet Goals	14
AS w. LevelSum	15
AS w. MaxLevel	14
AS w. SetLevel	14

3 Conclusion

t

Table 4: Air Cargo Problem 3

Search Method	Plan length
BFS	6
DFGS	20
UCS	6
GBFGS w. Unmet Goals	6
GBFGS w. LevelSum	6
GBFGS w. MaxLevel	6
GBFGS w. SetLevel	6
AS w. Unmet Goals	6
AS w. LevelSum	6
AS w. MaxLevel	6
AS w. SetLevel	6

Table 5: Air Cargo Problem 4

Search Method	Plan length
BFS	12
DFGS	392
UCS	12
GBFGS w. Unmet Goals	15
GBFGS w. LevelSum	14
GBFGS w. MaxLevel	13
GBFGS w. SetLevel	17
AS w. Unmet Goals	12
AS w. LevelSum	12
AS w. MaxLevel	12
AS w. SetLevel	12

References

[1] Udacity, "Artificial Intelligence Nanodegree Program Resources," https://github.com/udacity/artificial-intelligence, 2019, [Online; accessed 6-Jan-2019].

[2] ——, "Projects: 2. Classical Planning," https://github.com/udacity/artificial-intelligence/tree/master/Projects/2_Classical%20Planning, 2019, [Online; accessed 6-Jan-2019].

[3] S. Russell and P. Norvig, "Artificial intelligence: a modern approach. 3rd," *Essex, UK: Parentice Hall*, p. 1152, 2009.

[4] I. de Oliveira and Udacity, "Artificial Intelligence Nanodegree Projects and Exercises," https://github.com/ysraell/aind, 2019, [Online; accessed 6-Jan-2019].