Build a Forward Planning Agent

Alejandro Proano
alejandro.proano@gmail.com

Introduction

The goal of this project is to implement an agent that performs progression search to solve planning problems. We also evaluate the performance of different search algorithms and heuristics. We consider four different cargo problems:

1. Preconditions:
    - Planes: P1, P2
    - Cargos: C1, C2
    - Airports: JFK, SFO

   Initial locations:
    - At(C1, SFO)
    - At(C2, JFK)
    - At(P1, SFO)
    - At(P2, JFK)

   Goals:
    - At(C1, JFK)
    - At(C2, SFO)
2. Preconditions:
    - Planes: P1, P2, P3
    - Cargos: C1, C2, C3
    - Airports: JFK, SFO, ATL

   Initial locations:
    - At(C1, SFO)
    - At(C2, JFK)
    - At(C3, ATL)
    - At(P1, SFO)
    - At(P2, JFK)
    - At(P3, ATL)

   Goals:

- At(C1, JFK)
- At(C2, SFO)
- At(C3, SFO)

3. Preconditions:
   - Planes: P1, P2
   - Cargos: C1, C2, C3, C4
   - Airports: JFK, SFO, ATL, ORD

   Initial locations:
   - At(C1, SFO)
   - At(C2, JFK)
   - At(C3, ATL)
   - At(C4, ORD)
   - At(P1, SFO)
   - At(P2, JFK)

   Goals:
   - At(C1, JFK)
   - At(C2, SFO)
   - At(C3, JFK)
   - At(C4, SFO)

To solve each of the problems, the following search algorithms are evaluated.

1. Uninformed search:
   a. Breadth First Search
   b. Depth First Search
   c. Uniform Cost Search
2. Informed Search:
   a. Greedy Best First Graph Search
      - UnmetGoals heuristic
      - LevelSum heuristic
      - MaxLevel heuristic
      - SetLevel heuristic
   b. A* Search
      - UnmetGoals heuristic
      - LevelSum heuristic

- MaxLevel heuristic
- SetLevel heuristic

Implementations:

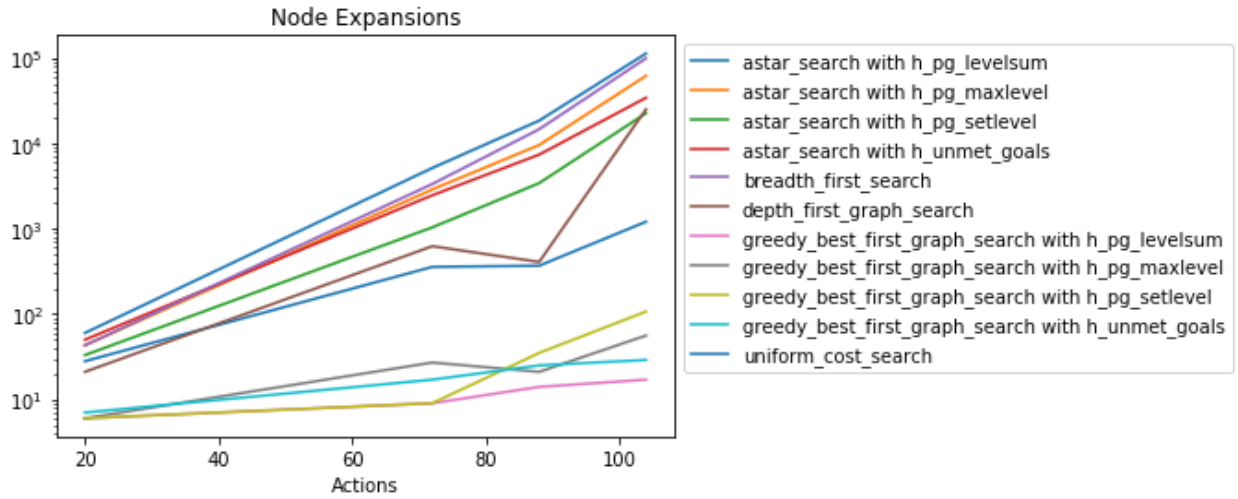
The code implementation of the following functions can be found in the attached ***my_planning_graph.py*** Python file.

Performance Evaluation:

We solve the four planning problems using the 11 different search algorithms. We first note that the number of actions taken for each problem are the following:

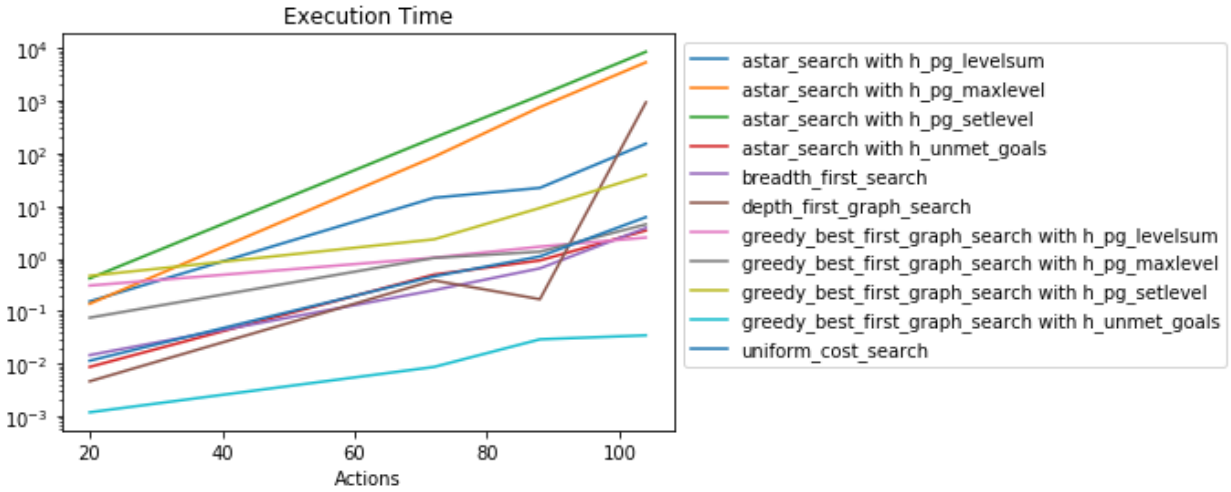
Cargo Problem 1	20
Cargo Problem 2	72
Cargo Problem 3	88
Cargo Problem 4	104

We first plot the number of expanded nodes against the number of actions. The vertical axis uses a logarithmic scale due to the exponential growth of the number of expanded nodes.



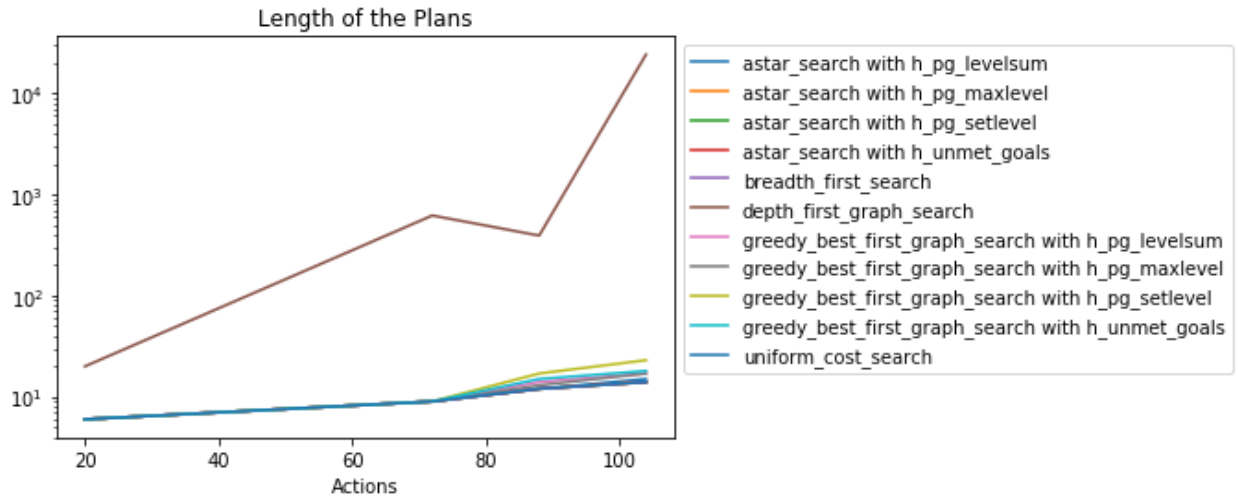
We note that the Greedy Best First Graph Search implementations lead to the lowest number of expanded nodes. The LevelSum heuristic implementation leading to the lowest results. We note that for this algorithm the number increases slower than for the other algorithms with the number of actions. In the case of the other search algorithms, we see a difference of at least 1 order of magnitude for any number of actions. Moreover, they show an exponential growth with respect to the number of actions.

We now consider the time taken by each algorithm to solve the problem. As before, we used the algorithmic scale on the vertical axis due to the exponential growth of the time metric.

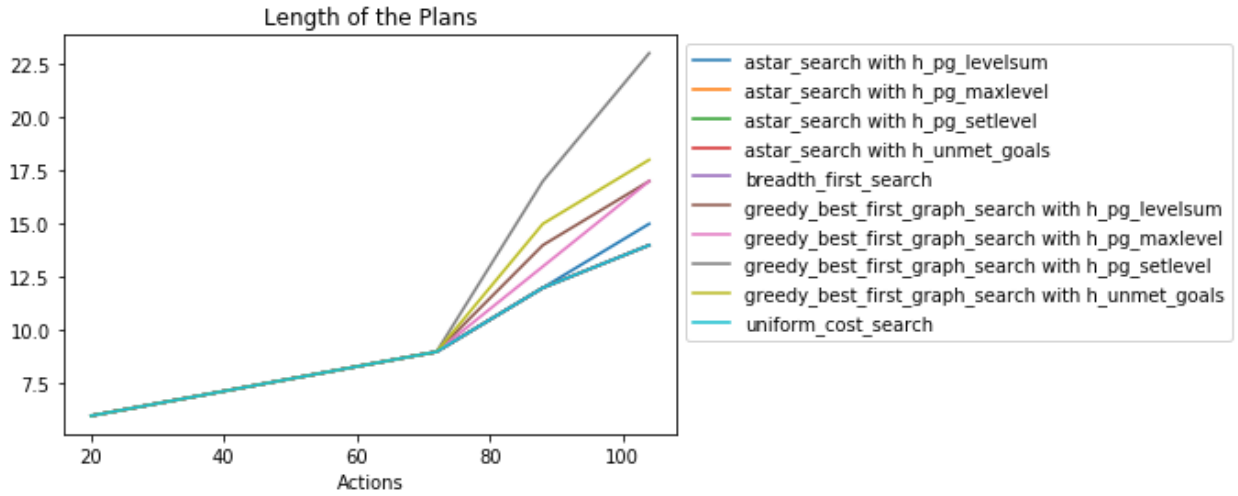


We note that the time increases exponentially. Seven of the implemented algorithms stay well below a 10 seconds runtime. Greedy Best First Graph Search with the UnmetGoals heuristic which achieves a run time of less than 100 milliseconds for all problems. On the other hand, the A* search with the SetLevel heuristic is the worst performing algorithm, doing less than 1 second for the first problem, going to as high as close to 10,000 seconds for the fourth problem. We note that A* search with MaxLevel and LevelSum are also low performant algorithms.

Finally, we consider the length of the plans returned by each algorithm as a function of the number of actions. We first show all the algorithms with the vertical axis using a logarithmic scale.



We note that Depth First Search is orders of magnitude higher than the rest of the algorithms. To understand the behavior of the rest, we remove the Depth First Search algorithm from the results and presented in the following figure.



We note that several algorithms have the same plan's length. All algorithms have the same performance for the first two problems. For the last two, we see a linear increase, getting to as high as a plan’s length of 23.

Conclusions

Based of the observed data from the four cargo problems, we note that for a real-time application where latency is sensitive, the Greedy Best First Graph Search with the UnmetGoals heuristic approach is the way to go. In terms of node expansions and path length, this algorithm presents an acceptable performance with respect to the others. We note that this algorithm will perform well for both very restricted domains and large domains.

If execution time is not a constraint, the Greedy Best First Graph Search with the LevelSum heuristic approach provides the best performance in terms of the number of node expansions. This is important since the node expansions are a representation of the amount of memory that solving the problem would require. In general, the Greedy Best First Graph Search implementations use less memory than the other algorithms.

Finally, in terms of the optimality of the solutions, we observe that Breath First Search, Uniform Cost Search, A* Search with UnmetGoals heuristic, A* Search with SetLevel heuristic, and the A* Search with MaxLevel heuristic find the optimal solution.