Build a Forward Planning Agent

After finishing the “ToDo” parts, I ran all the combination of problems and search methods with the results presented in table below. I used the PyPy binary for windows for the running the algorithms.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Problem** | **Search Method** | **Heuristic** | **Plan length** | **Time elapsed** | **Actions** | **Expansions** | **Goal Tests** | **New Nodes** |
| Air Cargo Problem 1 | breadth\_first\_search | | 6 | 0.026 | 20 | 43 | 56 | 178 |
| Air Cargo Problem 1 | depth\_first\_graph\_search | | 20 | 0.010 | 20 | 21 | 22 | 84 |
| Air Cargo Problem 1 | uniform\_cost\_search | | 6 | 0.034 | 20 | 60 | 62 | 240 |
| Air Cargo Problem 1 | greedy\_best\_first\_graph\_search | h\_unmet\_goals | 6 | 0.006 | 20 | 7 | 9 | 29 |
| Air Cargo Problem 1 | greedy\_best\_first\_graph\_search | h\_pg\_levelsum | 6 | 0.523 | 20 | 6 | 8 | 28 |
| Air Cargo Problem 1 | greedy\_best\_first\_graph\_search | h\_pg\_maxlevel | 6 | 0.225 | 20 | 6 | 8 | 24 |
| Air Cargo Problem 1 | greedy\_best\_first\_graph\_search | h\_pg\_setlevel | 6 | 0.649 | 20 | 6 | 8 | 28 |
| Air Cargo Problem 1 | astar\_search | h\_unmet\_goals | 6 | 0.034 | 20 | 50 | 52 | 206 |
| Air Cargo Problem 1 | astar\_search | h\_pg\_levelsum | 6 | 0.747 | 20 | 28 | 30 | 122 |
| Air Cargo Problem 1 | astar\_search | h\_pg\_maxlevel | 6 | 0.537 | 20 | 43 | 45 | 180 |
| Air Cargo Problem 1 | astar\_search | h\_pg\_setlevel | 6 | 1.032 | 20 | 33 | 35 | 138 |
| Air Cargo Problem 2 | breadth\_first\_search | | 9 | 0.330 | 72 | 3343 | 4609 | 30503 |
| Air Cargo Problem 2 | depth\_first\_graph\_search | | 619 | 0.589 | 72 | 624 | 625 | 5602 |
| Air Cargo Problem 2 | uniform\_cost\_search | | 9 | 0.618 | 72 | 5154 | 5156 | 46618 |
| Air Cargo Problem 2 | greedy\_best\_first\_graph\_search | h\_unmet\_goals | 9 | 0.039 | 72 | 17 | 19 | 170 |
| Air Cargo Problem 2 | greedy\_best\_first\_graph\_search | h\_pg\_levelsum | 9 | 1.383 | 72 | 9 | 11 | 86 |
| Air Cargo Problem 2 | greedy\_best\_first\_graph\_search | h\_pg\_maxlevel | 9 | 1.215 | 72 | 27 | 29 | 249 |
| Air Cargo Problem 2 | greedy\_best\_first\_graph\_search | h\_pg\_setlevel | 9 | 2.368 | 72 | 9 | 11 | 84 |
| Air Cargo Problem 2 | astar\_search | h\_unmet\_goals | 9 | 0.794 | 72 | 2467 | 2469 | 22522 |
| Air Cargo Problem 2 | astar\_search | h\_pg\_levelsum | 9 | 17.580 | 72 | 357 | 359 | 2426 |
| Air Cargo Problem 2 | astar\_search | h\_pg\_maxlevel | 9 | 48.493 | 72 | 2887 | 2889 | 26594 |
| Air Cargo Problem 2 | astar\_search | h\_pg\_setlevel | 9 | 90.274 | 72 | 1037 | 1039 | 9605 |
| Air Cargo Problem 3 | breadth\_first\_search | | 12 | 0.925 | 88 | 14663 | 18098 | 129625 |
| Air Cargo Problem 3 | depth\_first\_graph\_search | | 392 | 0.272 | 88 | 408 | 409 | 3364 |
| Air Cargo Problem 3 | uniform\_cost\_search | | 12 | 1.391 | 88 | 18510 | 18512 | 161936 |
| Air Cargo Problem 3 | greedy\_best\_first\_graph\_search | h\_unmet\_goals | 15 | 0.043 | 88 | 25 | 27 | 230 |
| Air Cargo Problem 3 | greedy\_best\_first\_graph\_search | h\_pg\_levelsum | 14 | 2.277 | 88 | 14 | 16 | 126 |
| Air Cargo Problem 3 | greedy\_best\_first\_graph\_search | h\_pg\_maxlevel | 13 | 1.542 | 88 | 21 | 23 | 195 |
| Air Cargo Problem 3 | greedy\_best\_first\_graph\_search | h\_pg\_setlevel | 17 | 7.355 | 88 | 35 | 37 | 345 |
| Air Cargo Problem 3 | astar\_search | h\_unmet\_goals | 12 | 1.289 | 88 | 7388 | 7390 | 65711 |
| Air Cargo Problem 3 | astar\_search | h\_pg\_levelsum | 12 | 28.886 | 88 | 369 | 371 | 3403 |
| Air Cargo Problem 3 | astar\_search | h\_pg\_maxlevel | 12 | 279.742 | 88 | 9580 | 9582 | 86312 |
| Air Cargo Problem 3 | astar\_search | h\_pg\_setlevel | 12 | 455.025 | 88 | 3423 | 3425 | 31596 |
| Air Cargo Problem 4 | breadth\_first\_search | | 14 | 4.669 | 104 | 99736 | 114953 | 944130 |
| Air Cargo Problem 4 | depth\_first\_graph\_search | |  | fail | 104 |  |  |  |
| Air Cargo Problem 4 | uniform\_cost\_search | | 14 | 7.602 | 104 | 113339 | 113341 | 1066413 |
| Air Cargo Problem 4 | greedy\_best\_first\_graph\_search | h\_unmet\_goals | 18 | 0.052 | 104 | 29 | 31 | 280 |
| Air Cargo Problem 4 | greedy\_best\_first\_graph\_search | h\_pg\_levelsum | 17 | 3.218 | 104 | 17 | 19 | 165 |
| Air Cargo Problem 4 | greedy\_best\_first\_graph\_search | h\_pg\_maxlevel | 17 | 2.888 | 104 | 56 | 58 | 580 |
| Air Cargo Problem 4 | greedy\_best\_first\_graph\_search | h\_pg\_setlevel | 23 | 25.606 | 104 | 107 | 109 | 1164 |
| Air Cargo Problem 4 | astar\_search | h\_unmet\_goals | 14 | 4.480 | 104 | 34330 | 34332 | 328509 |
| Air Cargo Problem 4 | astar\_search | h\_pg\_levelsum | 15 | 144.887 | 104 | 1208 | 1210 | 12210 |
| Air Cargo Problem 4 | astar\_search | h\_pg\_maxlevel | 14 | 2929.513 | 104 | 62077 | 62079 | 599376 |
| Air Cargo Problem 4 | astar\_search | h\_pg\_setlevel | 14 | 5414.834 | 104 | 22606 | 22608 | 224229 |

# Response to questions:

## Use a table or chart to analyze the number of nodes expanded against number of actions in the domain

Plot below show the number of expansions against the number of actions. The figure is plotted with logarithmic scale for y-axis. All methods show an exponential relation between expansions and actions (which shows as linear in a plot with logarithmic y-axis). This behavior is expected in a tree search problem.

When comparing different algorithms, *greedy\_best\_first* expands the lowest number of nodes by a significant margin, followed by *A\** method. For both *A\** and *greedy\_best\_first* the number of expansions is affected by the heuristic, and heuristics that better estimate the true cost require less expansion. The best heuristic in terms of node expansions was the *levelsum* followed by *setlevel*. The other two heuristics, *maxlevel* and *unmet\_goals*, had similar performance and resulted in slightly more expansions overall.

The *depth\_first* method was comparable with *A\** and performed much better compared to *breadth\_first* and *uniform\_cost* methods. However, it would yield very inefficient plans and failed to provide a meaningful plan for the largest problem (the plan was longer than 1000 steps)

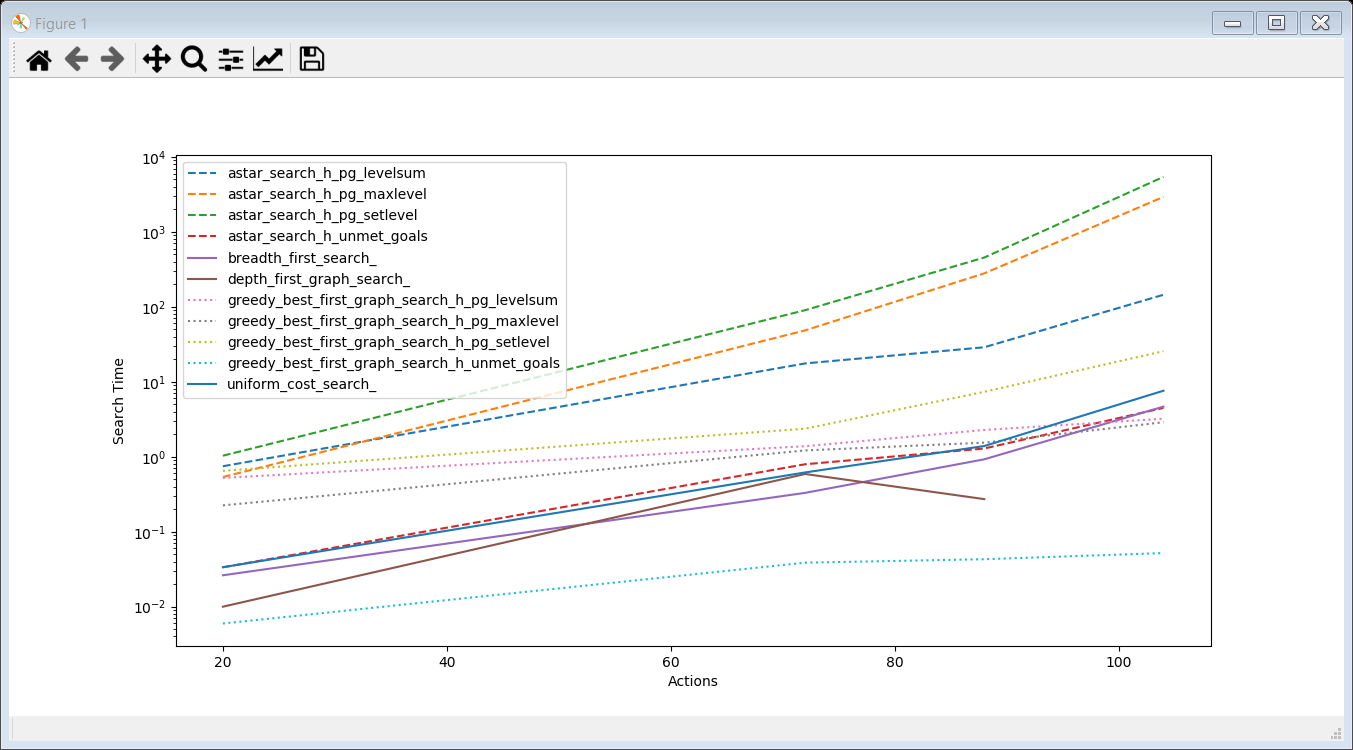
## Figure 4

## Use a table or chart to analyze the search time against the number of actions in the domain

The plot below shows the search time against the number of actions. Here we observe the search time growing exponentially with the number of actions (the y-axis has logarithmic scale).

When comparing different algorithms, the ones that do not rely on a heuristic generally perform faster. This shows that the cost of calculating the heuristic is significant compared to the search algorithm processing time. The exception is the *unmet\_goals* heuristic that seems to have minimal processing penalty.

The conclusion is that although using good heuristic reduce the number of node expansions, if the calculation of heuristic is more costly compared to checking the goal condition, the heuristic can significantly increase the search time. The only case that we would want to use a good heuristic irrespective of its processing time is when there is memory limitations and we need to minimize the number of node expansions.



## Use a table or chart to analyze the length of the plans returned by each algorithm on all search problems

Table below summarizes the performance of the algorithms for different problems. As expected, *Breadth First* and *Uniform Cost* algorithms find the optimal solution. The *Depth First* algorithm fails to find a reasonable solution in all cases. It searches deep in the tree and find the first plan that reaches a goal. The *Greedy Best First* finds the optimal solution for smaller problems, but for larger problems it fails to find the optimal solution. That being said, its solutions are reasonable and very close to the optimal solution. The *A\** algorithm finds the optimal plan, as long as the heuristic is smaller than the true cost of a plan. This is true for all heuristics except the *Level sum* heuristic. I believe it is the summation of levels for all goals that ends up overestimating the cost for a plan compared to its true cost. This especially happens for larger problems where there are many goals. The other heuristics properly provide an estimate cost for a plan that is lower than the true cost, hence the *A\** finds the optimal solution.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | Greedy Best First | | | | A\* | | | |
|  | Breadth First | Depth First | Uniform Cost | Unmet Goal | Level Sum | Max Level | Set Level | Unmet Goal | Level Sum | Max Level | Set Level |
| Air Cargo Problem 1 | 6 | 20 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| Air Cargo Problem 2 | 9 | 619 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 9 |
| Air Cargo Problem 3 | 12 | 392 | 12 | 15 | 14 | 13 | 17 | 12 | 12 | 12 | 12 |
| Air Cargo Problem 4 | 14 | >1000 | 14 | 18 | 17 | 17 | 23 | 14 | 15 | 14 | 14 |

## Which algorithm or algorithms would be most appropriate for planning in a very restricted domain (i.e., one that has only a few actions) and needs to operate in real time?

For small problems where we are not worried about exponential growth of the search space, *Breadth First* or *uniform cost* algorithms would work just fine. They are fast for small problems and do not rely on additional information needed for a heuristic function.

## Which algorithm or algorithms would be most appropriate for planning in very large domains (e.g., planning delivery routes for all UPS drivers in the U.S. on a given day)

For larger problems we need to employ a heuristic to ensure the search space remains manageable. If we are not worried to find the absolutely optimal solution, we can use the *Greedy Best First* algorithm to find a solution with minimal search in the space. It is important to ensure that the *heuristic* is simple and fast so that it does not impose significant computation cost. For the *Greedy Best First* algorithm, the heuristic does not need to be necessarily optimistic compared to the true cost of the plan. The best heuristic is the one that can provide relative ranking for different plans that is similar to the actual plan cost.

## Which algorithm or algorithms would be most appropriate for planning problems where it is important to find only optimal plans?

To find the optimal plan we need to choose between *Breadth First*, *Uniform Cost*, and *A\**. In general *A\** would be a better choice, as it can greatly reduce the search space. However, *A\** depends on the presence of a good and fast (compared to checking the goal condition) heuristic function. In the absence of such heuristic, we can use *Breadth First* or *Uniform Cost* algorithms. When the plan cost is just the depth of the tree, the two algorithms are similar and *Breadth First* would be simpler to implement. But when the edges of the graph have varying costs, *Uniform Cost* can perform faster with less expansion.