Comparison of Python Search Algorithms

Final Report

Danielle Shackley and Brendan Dao

Wentworth Institute of Technology
Classical AI
Professor Frank Kreimendahl

Summer 2021

I. Problem description

A. *Problem motivation*: Our project motivation comes from our previous work on assignments in this class. We had a group meeting and discussed a few different AI topics that we might be able to implement. After multiple searches and looking into different topics and Github code examples, we concluded we would be able to produce the best work with something we were already familiar with and had experience working with already.

B. *Problem description:* The problem we are exploring is how effective the Greedy Search and weighted A* algorithms are in comparison to other search methods. The other search methods we are going to use as a comparison include ones we implemented in previous Assignments (depth-first search, iterative depth search, a-star, etc).

C. *Related to AI:* This project is related to AI because it uses fundamental techniques to build intelligent systems implementing algorithms.

D. *Importance / Interest:* Since we already implemented other search algorithms in A1, we thought it is interesting to look at a new algorithm we didn't implement to compare its effectiveness.

II. Solution methods

A. *Possible solutions:* In order to do a comparison of metrics on these algorithms we first need to implement all of them and get them running and producing the same outputs. We made a project repository on Github (https://github.com/shackleydatwit/finalProject) for easy team collaboration and then copied it all over to the given repository (https://github.com/shackleydatwit/finalProject). First we are going to evaluate our current implementations of algorithms we are going to analyze. We fixed any errors we had in the algorithms based on feedback from our assignments. Next we combined the multiple different algorithms into one Python project. Lastly we did some research on the two new algorithms we wanted to add by reading the sections of the book. After adding these algorithms we performed multiple runs of them and gathered some averaged metrics to analyze.

B. *Benefits / drawbacks:* Some benefits of this approach were that we already implemented some of the algorithms we plan to use. This also helps because we know that they are correct and producing the right outputs and functioning properly. With feedback on our previous assignments we can update our code and have confidence in their performance. A drawback to this approach is that our approach is specific to our scenario. We are collecting times and nodes produced by running different algorithms but this might not truly reflect which of these is the "most efficient." We have limited input files and trials used in our analysis.

III. Analysis

A. *Algorithm description:* Below are the algorithms we implemented and their descriptions.

a. <u>Depth first search:</u> This algorithm searches to the deepest level of the search tree and then backs up to the next deepest node that still has unexpanded successors. Typically it is implemented as a tree-like search and it does not keep track of reached states. This algorithm is not cost optimal and we do not expect it to compare well to the efficiency of the other algorithms.

b. <u>Iterative deepening search:</u> This algorithm repeatedly applies depth limited search with increasing limits. It tries all values, starting with 0, then 1, 2 and so on. This algorithm terminates when either a solution is found, it reaches the cutoff value (which is a failure because it could not find a solution within the cutoff value), or fails because it found there was no solution.

c. <u>Uniform cost search:</u> Actions have different costs, with this algorithm we use the idea of a search spreading out in waves of uniform path cost. It considers all paths systematically in order of increasing costs.

d. <u>A* search:</u> This algorithm is a best-first search that uses an evaluation function ($f(n) = g(n) + h(n)$). $g(n)$is the path cost from the initial state to node n and $h(n)$ is the estimated cost of the shortest path from $n$to a goal state. Overall $f(n)$is the estimated cost of the best path that runs from $n$to the goal state.

e. <u>Weighted A* search:</u> To lower the number of explored nodes we can accept solutions that are suboptimal. We can use the A* search but with a weighted heuristic that weighs more heavily in order to counteract us overestimating and missing the optimal solution. For this search we use the evaluation function $f(n) = g(n) + W \times h(n)$for some $W > 1$.

f. <u>Greedy search:</u> This search builds a path by always choosing the next node that offers the most obvious and immediate benefit

B. *Algorithm Metrics*: Below is a table of the explored algorithms and their associated: Completeness status, worst time complexities.

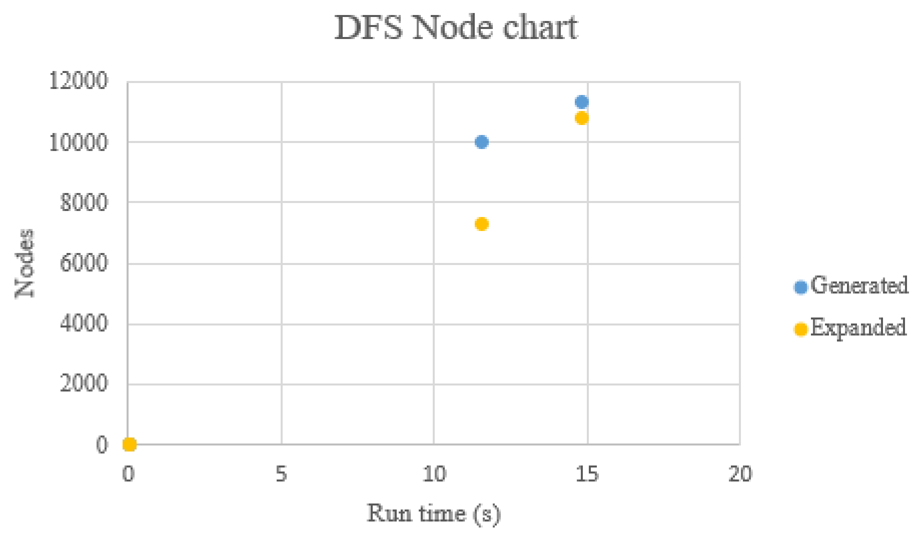
Algorithm	Completeness	Worst Case Time Complexity
Depth first	Complete	$O(n^m)$
Iterative deepening	Complete	$O(b^d)$
Uniform cost	Complete	$O(b^{1+[C^*/\epsilon]})$

A*	Complete	$O(b^d)$
Weighted A*	Complete	O(E)
Greedy	Not Complete	O(nlogn)

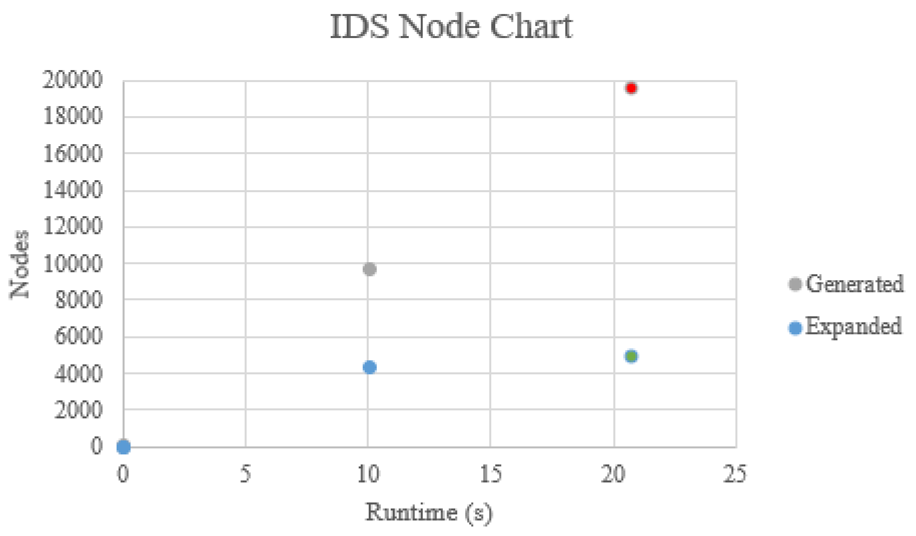
C. Approach: We went back and forth a few times deciding on what the best way to compare these algorithms was. We decided doing multiple time trials and exploring their nodes generated and expanded would be the best to numerically and visually see.

IV. Evaluation

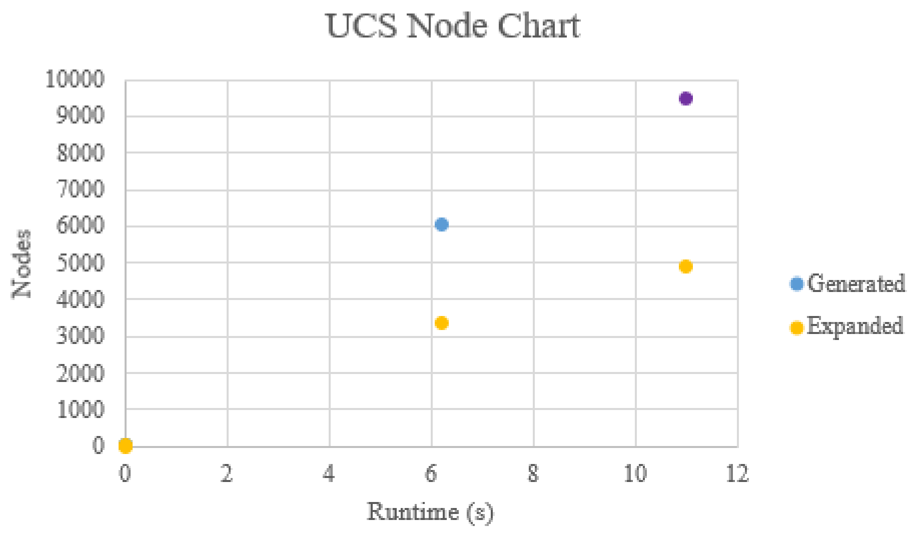
A. *Evaluation Methodology*: In order to perform our overall comparison between the different algorithms we made different graphs comparing different metrics. The graphs are included below:



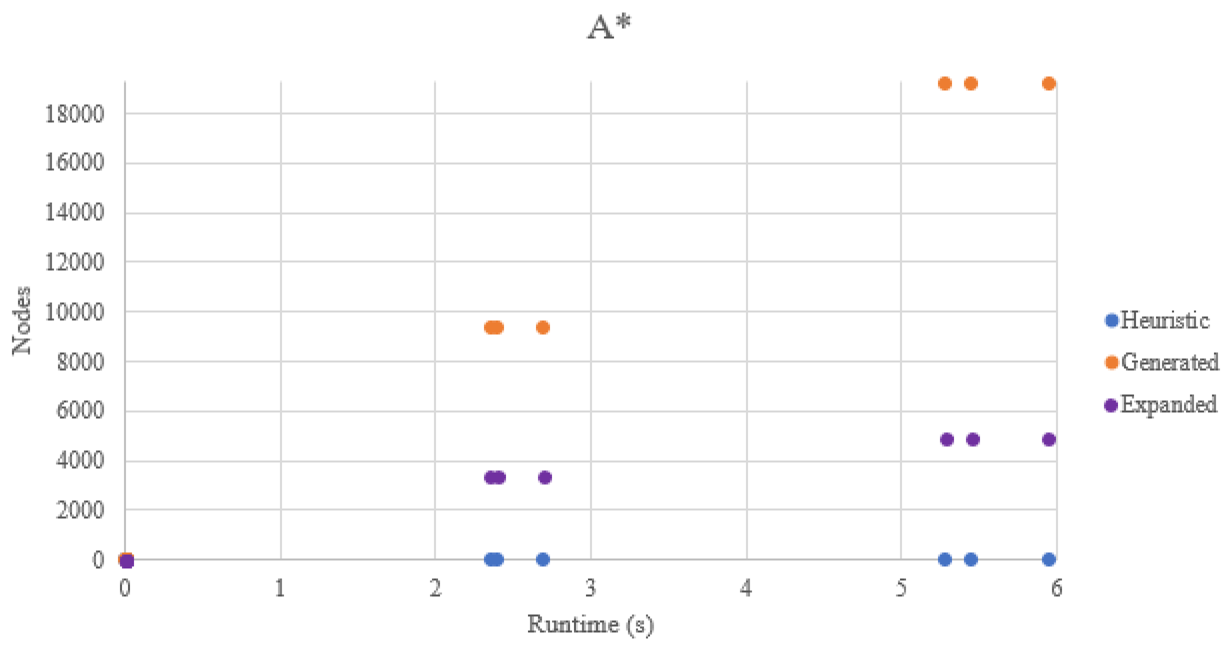
Graph 1: This is the DFS algorithm plotted with its runtime (Seconds) vs the nodes generated and expanded upon. As the time and node numbers increase, the input files are also increasing. We ran this algorithm with the small1, small2, large1 and large2 inputs files.



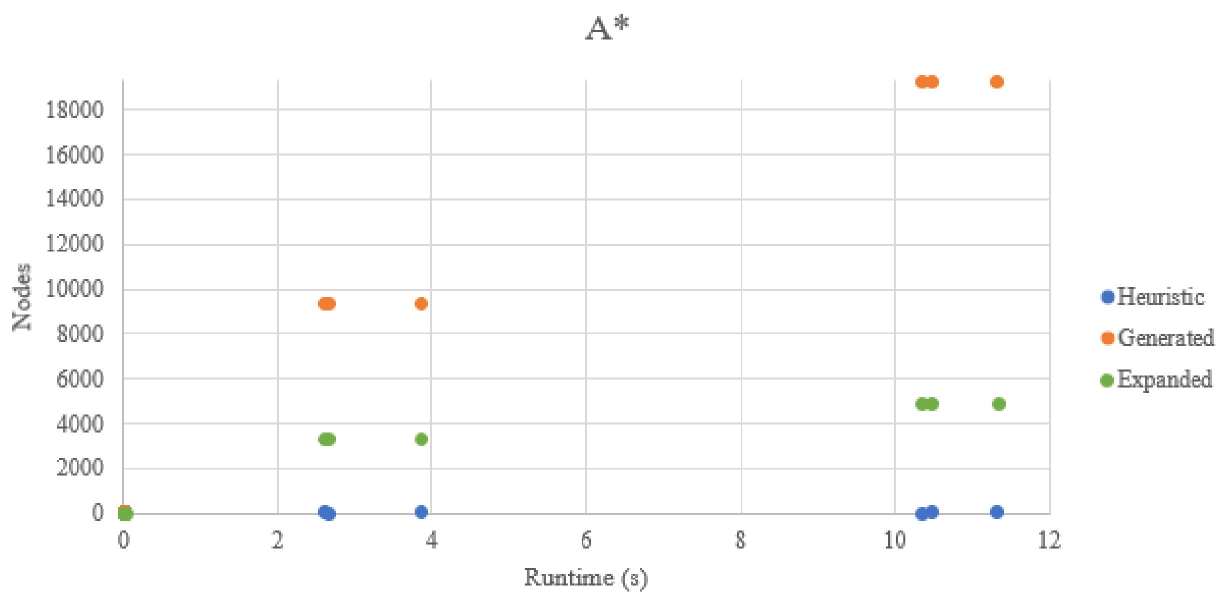
Graph 2: This is the IDS algorithm plotted with its runtime (Seconds) vs the nodes generated and expanded upon. This algorithm followed a similar pattern to the DFS with its time and nodes increasing as the input files got larger except for the large1 input file. The large1 input file time took the longest as well as the nodes generated and expanded were the largest (These are denoted by the red (generated) and green (expanded) data points).



Graph 3: This is the UCS algorithm plotted with its runtime (Seconds) vs the nodes generated and expanded upon. This algorithm followed a similar pattern to the IDS with its time and nodes increasing as the input files got larger except for the large1 input file. The large1 input file took the longest as well as the nodes generated was the largest (This is denoted by the purple (generated) data point).

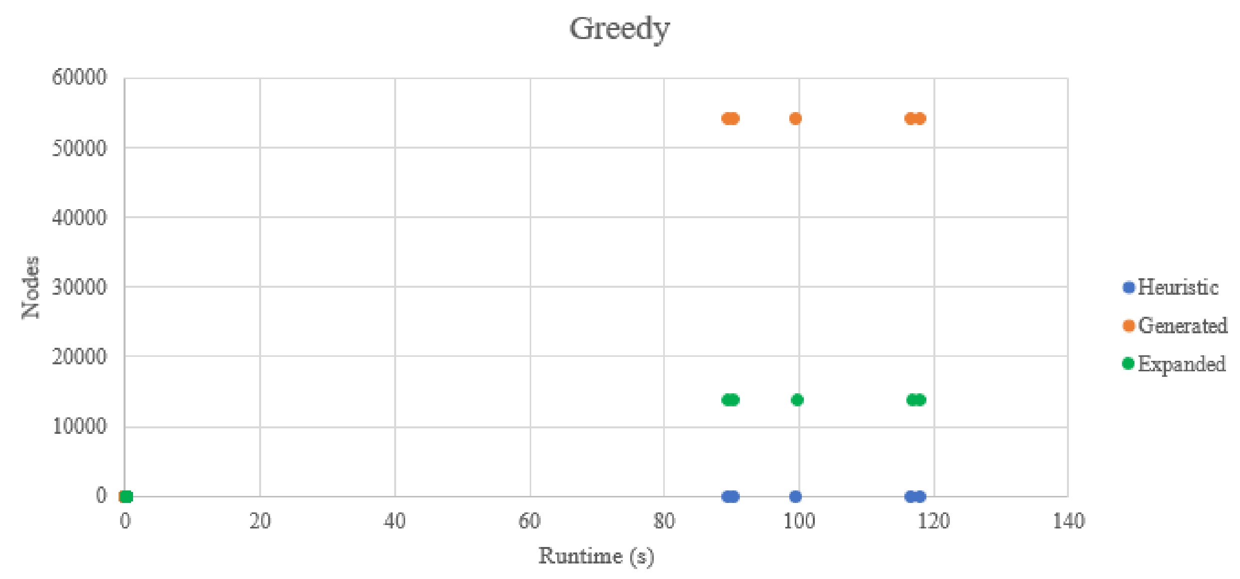


Graph 4: This is the A*algorithm plotted with its runtime (Seconds) vs the nodes generated and expanded upon. The data points on the 0 value of the Y axis (blue) are the heuristic value that was used to get the node values. This chart looked at heuristic values between 1 and 5.

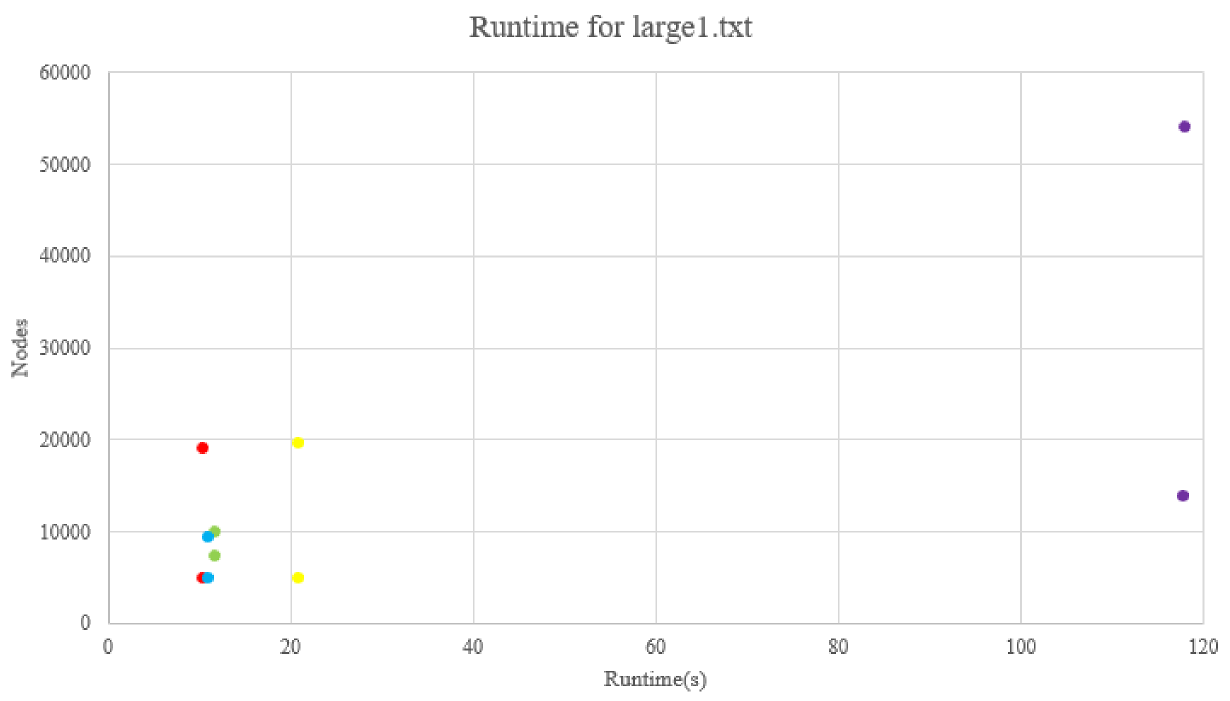


Graph 5: This is the A*algorithm plotted with its runtime (Seconds) vs the nodes generated and expanded upon. The data points on the 0 value of the Y axis (blue) are the heuristic value that was used to get the node values. This chart looked at heuristic values between 10 and 25.

We separated the two A* graphs because it was easier to examine the variety in the data. We noticed with all of the data the drastic jump in the number of nodes being generated and expanded on based on the input files. This makes sense that these two variables are connected (file size to number of nodes).



Graph 6: This is the Greedy algorithm plotted with its runtime (Seconds) vs the nodes generated and expanded upon. We noted that the nodes generated and expanded stayed the same despite changing the heuristic as long as the file input stayed the same. When the different files were inputted that is when we noticed a change in nodes.



Legend:
A* - Red
DFS - Green
IDS - Yellow
UCS - Blue
Greedy - Purple

Graph 7: This is an overall comparison of the 5 algorithms we implemented. This graph shows the nodes generated and expanded with the input file large1.txt. We chose this file because it was most of the algorithms worst run time case for the trials we performed. In all cases the top data point for each algorithm pair is the generated value and the bottom is the expanded.

B. *Results*: We agreed that the results are as predicted. We thought greedy would have the worst run times which proved to be true. We also thought that A* would have the best results. We deemed it “best” because it has the lowest run time with a relatively high number of nodes generated and expanded.
C. *Expansion*: For this project we could expand by adding more algorithms to our comparison, and creating a script that runs and records our times and nodes numbers for us. Instead of doing it manually, that way we could collect a lot more data to analyze.

V. Resources

[1] S. Russell and P. Norvig, Artificial Intelligence - A Modern Approach. Pearson, 2003.

Note: The data of our time trials can be found in the included excel file titled “Algorithm Data”.