# Charts

Table 1. Data Results

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Type** | **Time** | **Nodes Expanded** | **Maximum Depth** | **Steps** | **Starting Value** | **Target Value** | **Operations** |
| *greedy* | *0.0109634* | *286* | *95* | *95* | *5* | *100* | *+1 -1 -1* |
| *iterative* | *-1* | *-1* | *-1* | *-1* | *5* | *100* | *+1 -1 -1* |
| *greedy* | *0.0002257* | *25* | *8* | *8* | *-5* | *100* | *\*5 +10 -5* |
| *iterative* | *0.0003204* | *112* | *4* | *4* | *-5* | *100* | *\*5 +10 -5* |
| *greedy* | *0.0329787* | *526* | *105* | *105* | *-5* | *100* | *+1 +1 +1 +1 +1* |
| *iterative* | *-1* | *-1* | *-1* | *-1* | *-5* | *100* | *+1 +1 +1 +1 +1* |
| *greedy* | *0.0013804* | *85* | *21* | *21* | *5* | *2080* | *\*5 +50 /3 -15* |
| *iterative* | *0.1134698* | *35021* | *8* | *8* | *5* | *2080* | *\*5 +50 /3 -15* |
| *greedy* | *-1* | *-1* | *-1* | *-1* | *-50* | *235* | *\*5 +17 /3 -5* |
| *iterative* | *0.0126061* | *3969* | *6* | *6* | *-50* | *235* | *\*5 +17 /3 -5* |

\*-1 is infinity

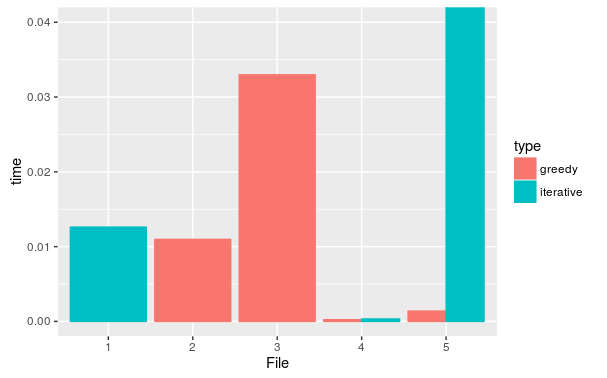


Figure 1. Time Comparison

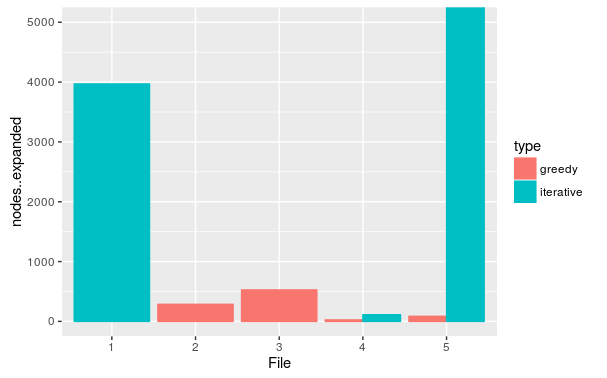


Figure 2. Node Expansion Comparison

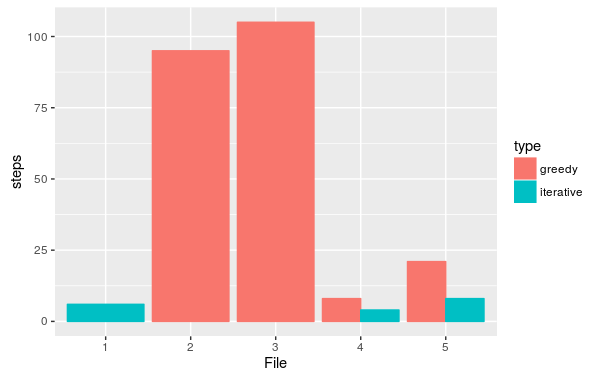


Figure 3. Steps Comparison

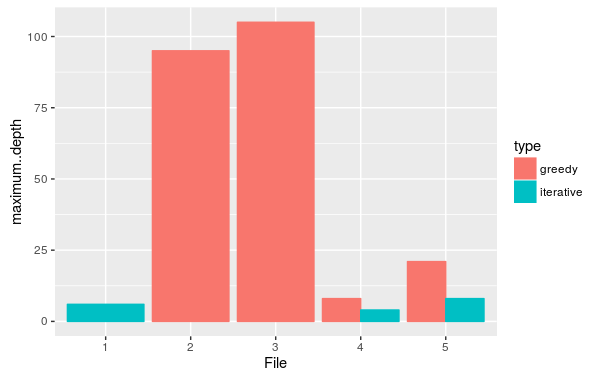


Figure 4. Depth Comparison

# Performance Analysis

The heuristic function h(n) that we chose to use was the absolute difference between the node and the goal; the smaller the value, the closer the node is to the goal state. We considered each operation was of the same cost. The order that children were created was based on the order that the operations were initially inputted.

### Number of Steps Required

The number of steps in the final solution for greedy search was found to be at least the number of steps generated by iterative deepening. Because iterative deepening is complete and optimal, it always returns the fewest number of steps possible to reach the solution. Figure 3 shows the results of the steps required to reach the goal for each input to compare the searches. It is apparent that if a solution does exist, iterative will produce a path that has either equal or less steps than greedy.

### Number of Nodes Expanded

In general, the number of nodes expanded to find the solution path by the iterative search was much greater than the same metric for greedy search. If greedy search could find a solution, it would do so in many fewer explored than iterative search. Figure 2 shows the node expansion comparison and it is obvious that iterative expands many more nodes than greedy to obtain a solution. Figure 1 shows the time comparison which agrees with Figure 2 since the more nodes expanded, the more time the program will take in order to perform the action.

### Effective Branching Factor

The effective branching factor of iterative deepening search is equal to the number of operators given as arguments. For greedy search, the answer is more complicated; where the effective branching factor is the number of children generated by a typical node, the EBF is 1. Of course, a node can generate more children, but greedy search tries to find the solution as quickly as possible so it only expands nodes that are close to the goal. Typically there is an operation that puts the current node closer to the goal so there isn’t a need to traverse to a previous node and start a new branch unless a solution doesn’t exist down the initial branch.

# Conclusion

Each search algorithm has its own strengths and weaknesses, which can be used to achieve different things in a particular problem. From this activity, it was easy to see that there is definitely a trade off between time, space, and optimality. For greedy search, finding a solution (if it existed) was significantly faster in exchange for a less likely optimal solution. It expanded fewer nodes and consequently took lesser space. In the case of iterative deepening, if a solution exists, it will eventually find it. Not only that, but it will also find the most optimal path to it. However, iterative takes much more time and space in order to reach the solution. In the case where a solution has a large depth, iterative will consume more resources than greedy to even get close to it.

At the end it was interesting to see the statistics for each set of inputs for each algorithm.

# References

Pseudo-code was used from:

* <https://en.wikipedia.org/wiki/Best-first_search>
* <https://en.wikipedia.org/wiki/Iterative_deepening_depth-first_search>

Borrowed time manager from a post from stackoverflow:

* <http://stackoverflow.com/questions/366682/how-to-limit-execution-time-of-a-function-call-in-python>