Michael Kytka – Dan Mitchell – Michael Curtis – David Kloss

Dr. Wolfe

Introduction to Algorithms

12/7/2015

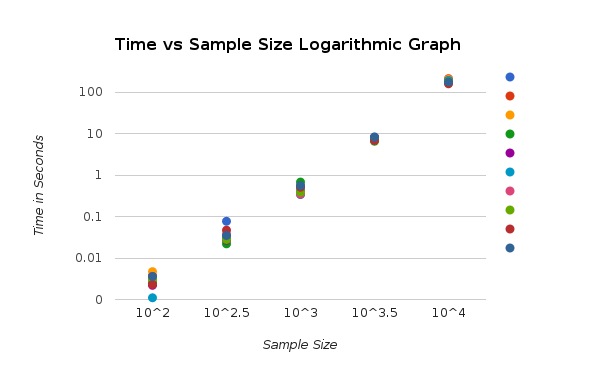
Project 2

The algorithm that we have implemented is the Shortest Path Faster Algorithm (SPFA). This algorithm is essentially an improvement over the Bellman-Ford Algorithm. Its purpose is to calculate the shortest paths from a single source to every other node. Like Bellman-Ford, SPFA can handle negative-weight edges. On the contrary, rather than exhaustively iterating over each edge for every vertex in a brute force manner, SPFA utilizes a queue for improved efficiency. A vertex is placed in the queue once it is relaxed. This potentially can remove unnecessary iterations that Bellman-Ford would have completed for the same graph. With this said, these algorithm have identical worst-case running times depending on the graph’s structure. SPFA is much more effective for graphs that are sparse and does poorly with those who are dense.[[1]](#footnote-1)

The structure of the algorithm is similar to Bellman-Ford in that each vertex’s distance estimation from the source is originally set to infinity and then relaxed. Upon termination, the vertices distances will accurately reflect the shortest distance to that node from the source. More specifically, the setup of the algorithm is identical. Except for the source, the estimated shortest distances are set to infinity. Afterwards, the source is added to the queue. In the while loop, the next node in the queue is pulled out. Each adjacent edge for this node is then relaxed. If a node is relaxed, it is likewise added to the queue. In this methodology, a vertex can be placed back in the queue multiple times depending on the amount of times it is relaxed.

In the worst-case scenario, every edge will be relaxed for each vertex in the graph. This case is O(|V||E|). The setup is O(V) since the for-loop iterates for every vertex and each iteration is constant time. Since O(|V||E|) dominates O(V), the worst-case running time is the former value.

We wrote the algorithm and problem set generation code in python. It was run on a standard GCC laptop running Arch Linux. The problem sets were generated in the manner described in the spec, choosing a random starting node and depth-first expansion from there. The problem sets were then run using the same starting node as was used for generation. Since the graph doesn’t have weighted edges, we made all graph edges weighted one.



One interesting aspect was how the spread of the running times converged as the problem sets got larger. This actually supports our given average running time of O(E) for this algorithm, as the spread of edges for the number of vertices taken is vastly different with a small number of vertices, but gets closer and closer the more vertices we used. The data we used was relatively sparse, so the algorithm ran quite fast, and thus was not close to the theoretical worst case of O(|V||E|).

Works Cited

Wu, Yonghui. "Algoirthms of Best Paths." Lecture.

1. Wu, Yonghui. "Algoirthms of Best Paths." Lecture. [↑](#footnote-ref-1)