UCD Smurfit Graduate School of Business

Network Software Modelling

Assignment One - Dijkstra's Algorithm

Author:
Conor Reid
(16202630 - MSc B.A Part-time)

Lecturer:
Dr. James McDermott

March 7, 2017

Dijkstra's Algorithm
Dijkstra's Algorithm is used for finding the shortest path (or that of lowest cost) between a initial point in a directed graph, and all others. More formally; Given an initial vertex s in a weighted directed graph $G = (V, E)$ where edges are non-negative, Dijkstra's Algorithm finds the path of lowest cost from s to all other vertices in G. This is sometimes referred to as the *single-source shortest path problem.*

The algorithm works as follows; Every node in a graph is assigned a tentative distance value, which zero for the initial node and infinity for all others. The initial node is marked the current node and thus visited, all others are marked as unvisited and are passed into a set. Visited nodes are not visited again. The neighbours of the current node are calculated for actual tentative distance. That is, the distance to the second node going through the first is calculated. If this distance is lower than the previously assigned tentative distance to this node then it overwrites the same. The process is repeated, assigning the next unvisited node with the lowest tentative distance as the current node. The algorithm stops when the unvisited set is empty.

Dijkstra's algorithm has time complexity $O(n^2)$ where n is the number of nodes in the graph.

Bidirectional Dijkstra's Algorithm
The bi-directional version of Dijkstra's algorithm can be used in an effort to "speed up" the searching process, by reducing the number of visited vertices. In the above explanation, the searching process happens in the direction of s (the initial node) to t (the target node). In this version, searching direction alternates between forward and backwards. That is, we alternate between searching from s to t and from t to s, the latter involving following edges backwards. We denote distances for forward search as $d_f(u)$ and distances for backward search as $d_b(u)$. The algorithm terminates when some vertex has been removed from the queues of both searches. Intuitively, we can view bi-directional search as the rowing of two bubbles around the initial vertex and target vertex, terminating when these bubbles intersect [1].

Interesting to note, is that this technique does not improve the worst case behaviour of the algorithm, or its big 0 notation. That is, bi-drectional searching still has time complexity of $O(n^2)$ (which is really $O(\frac{n^2}{2})$ which is $\sim O(n^2)$).

Dijkstra's Original Algorithm - Comparison
Time Complexity
Implementation
Run-Time behavior
Conclusions

References

[1] https://courses.csail.mit.edu/6.006/fall11/lectures/lecture18.pdf