Shortest Path Algorithms Comparison Report

Hsin Chen

DIKH: Dijkstra's Algorithm with min-heap implementation

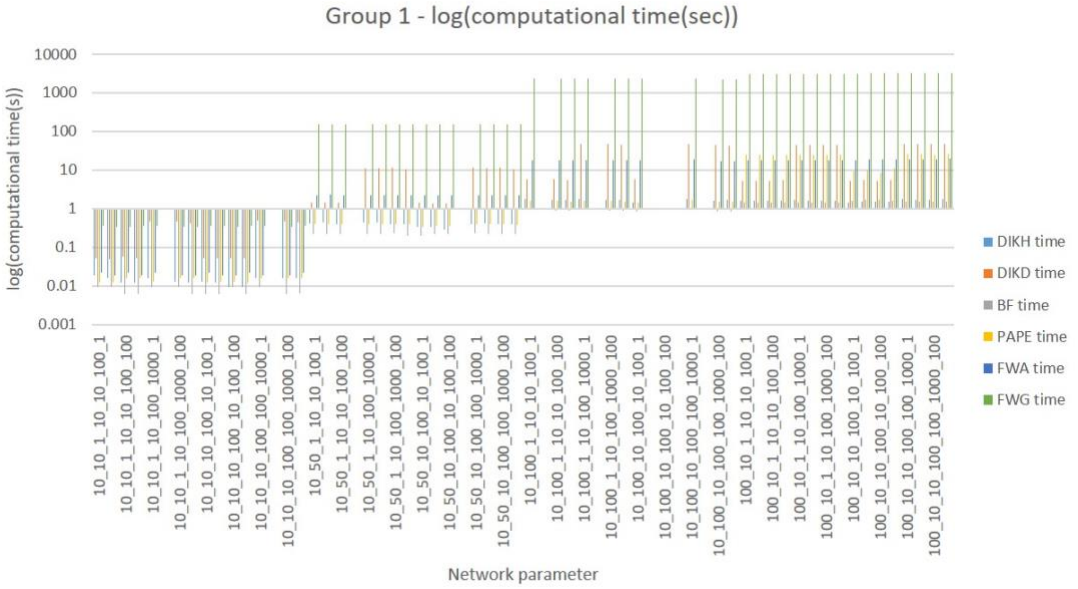
DIKD: Dijkstra's Algorithm with Dial's Implementation

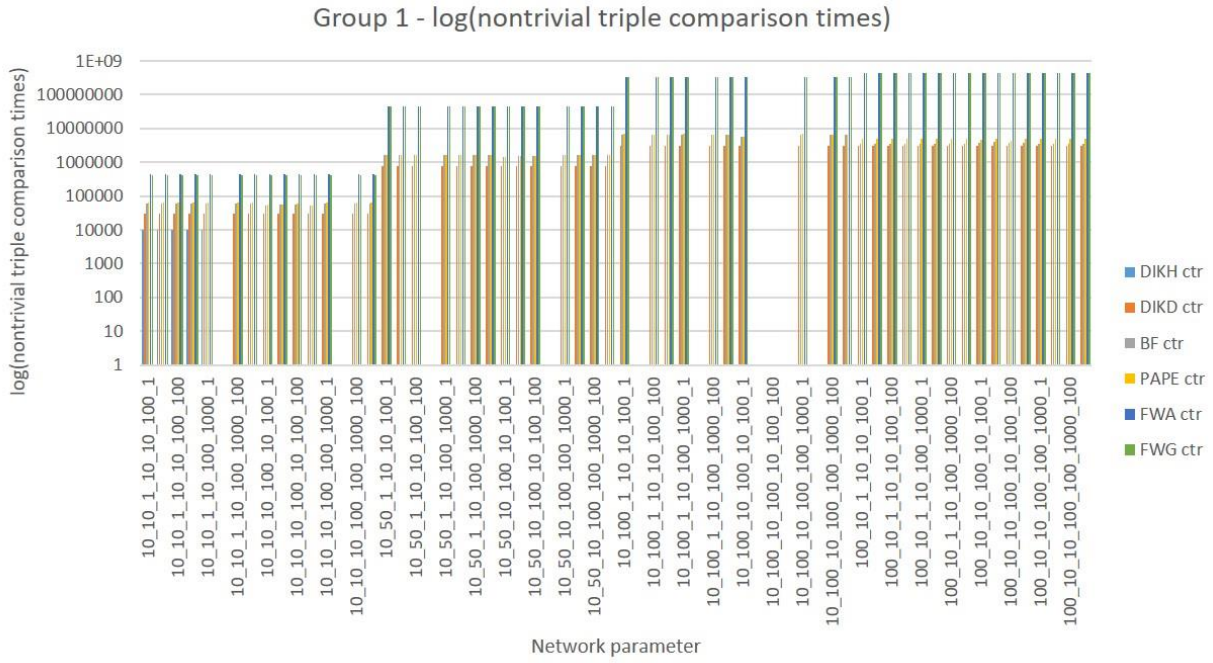
BF: Bellman-Ford Algorithm (FIFO modified Label Correcting)

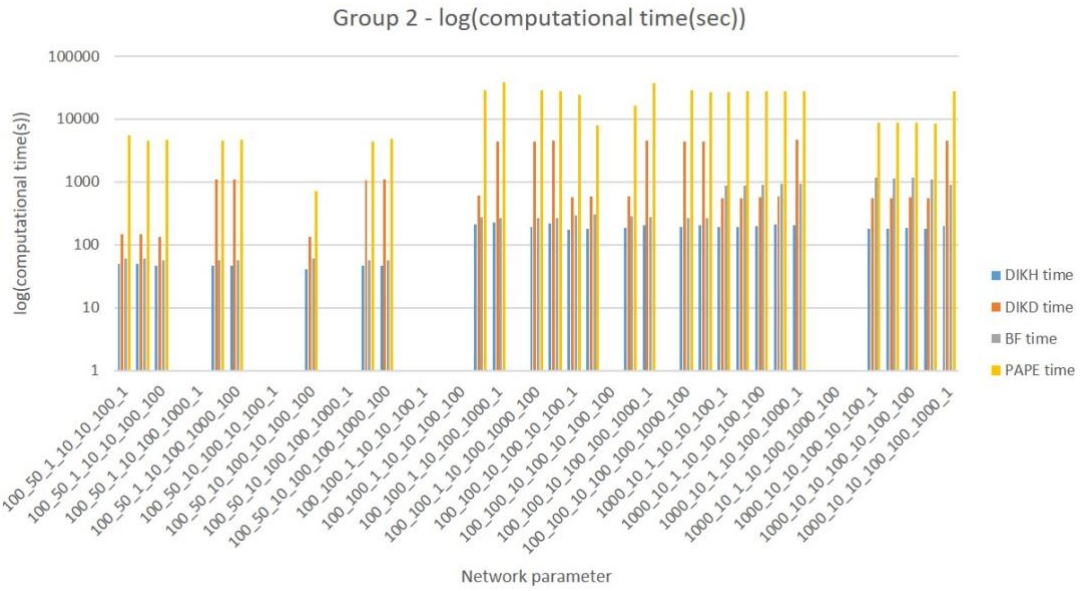
PAPE: Pape Algorithm (modified Label Correcting with dequeue implementation)

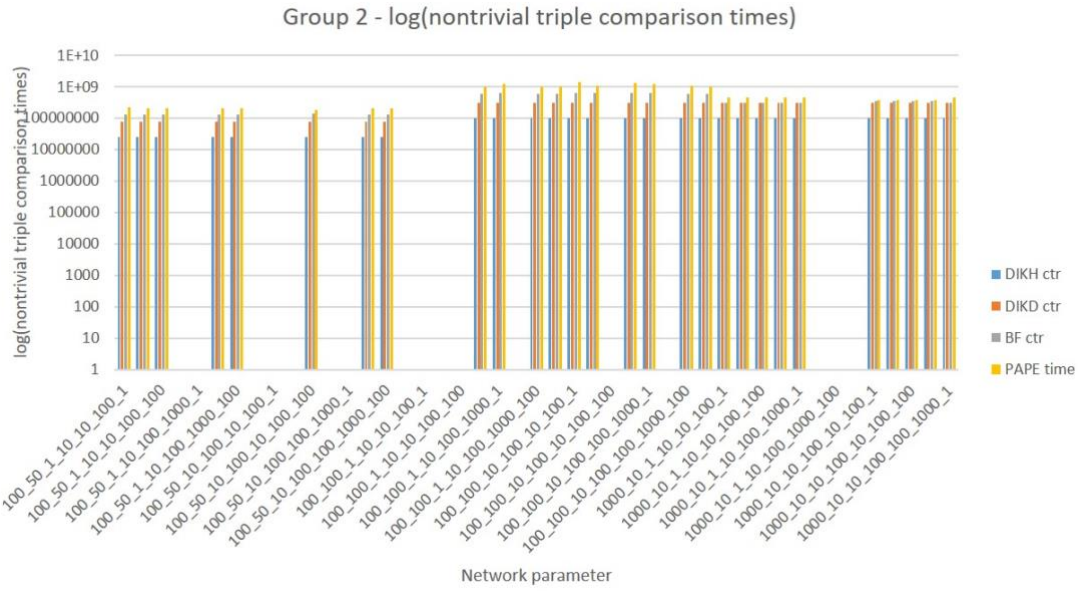
FWA: Floyd-Warshall Algebraic algorithm

FWG: Floyd-Warshall Graphical algorithm









Environment of Experiments:

CPU type: Intel Core i5-7500

RAM size: 16GB

HD size: 1TB

OS type: Windows 7

Compiler version: gcc version 7.3.0 (GCC)

The network generator used is “spweb2”.

It took about 9 days to get computational data shown above. Because FWA and FWG are too time-consuming, I divided all networks into 2 groups so Group 2 only tested the DIKH, DIKD, BF and PAPE in order to save some time and test larger networks.

For the smaller networks in Group 1, all 6 algorithms run really fast. As the networks getting larger, FWG takes a really long time to do the search. For some networks that are not very large, DIKD's performance is the second worse; the result may be caused by that the costs of arcs are big, compared to the scale of the network, so it takes much time to scan through the buckets. For the larger networks in Group 1, DIKH and BF perform well; DIKD, PAPE, FWA's performance are similar and a little worse; FWG takes a lot of time.

For Group 2, in general, PAPE's performance is the worse, and DIKD's is the second worse. The reason of DIKD's performance might be the same as the one mentioned above, while the reason why PAPE is time-consuming might be about the implementation problem. When implementing PAPE, I made it put the node in front of the LIST after distance update without checking if the node had existed in the LIST. The reason why I did that was that it would take O(n) to check if the node existed and another O(n) to remove it and update indices. I thought just putting it into the LIST and spending a little more time doing triple comparisons would be better than scanning and deleting the existent node, but it needed further testing. That's also why for some networks, PAPE's number non-trivial triple comparison times is 10 times larger than DIKH.

PAPE and FWG's computational time seem to be longer than it's supposed to be; it might be because of implementations problems or the network's type.