C++ Generic Graph Library Metric Analysis

Chenqin Xu
cx2198@columbia.edu

Kaimao Yang
ky2371@columbia.edu

Mingjie Zhao
mz2646@columbia.edu

April 30, 2018

Contents

1 Environment **2**

2 Test on real data **2**

3 Test on artificial data **2**

4 Conclusion **3**

OS	macOS High Sierra
Processor	Inter Core i7
GCC Version	7.3.0
Compling option	-O3
C++ standard	C++17

Table 1: Environment

	BFS Find Path	Single Source Shortest Path
Fixed dense graph	0.2602	96.1392
Dense graph	0.3597	102.8210
Fixed sparse graph	0.0054	15.4307
Sparse graph	0.0088	14.3237

Table 2: Test on real data

1 Environment

Our test environment are listed in table. 1.

2 Test on real data

We use data from Social circles provided by Facebook (accessed from `https://snap.stanford.edu/data/egonets-Facebook.html`). This dataset includes 4039 nodes and 88234 edges. According to definition, this is a sparse graph. We test our graph library based on this dataset running BFS find path algorithms to find whether two people know each other and shortest path algorithms to find the minimum connections needed to let two people know each other, and the result is shown in table. 3.

From the result we can find that the fixed graph saves 10% time than variable graphs, that means our design is useful when users do not need to chage the graph's size. The reason why it is quicker is that in variable graph we use a **unordered_map** to store the mapping, and each time we need to check this map. This also means,using a **vector** is more efficient that **unordered_map**. Therefore, our optimization works. Another thing we find is the sparse graph is much faster than dense graph, which is also expected because the increment operation we used to traverse the out edges. For dense graph, we need to check the validness of each edge, while in sparse graph all edges are valid.

3 Test on artificial data

It necessary for us to test our graph library on a dense graph. We generated a dense graph that has 400 nodes and 79601 edges, and ran the same algorithms.

	BFS Find Path	Single Source Shortest Path
Fixed dense graph	0.0110	0.6600
Dense graph	0.0108	0.6897
Fixed sparse graph	0.0084	0.5821
Sparse graph	0.0100	0.6112

Table 3: Test on real data

The result is shown in

From this result, we can find dense and sparse graphs have similar performance. We use adjacency matrix to implement dense graph. The main advantage is the quick remove and access to one specific edge between two vertices. In this two algorithms, we only go through all the vertices, all edges or adjacent edges, and that is why they have similar performance. However, we can expect better performance on dense graphs if we access edges very often.

4 Conclusion

From our result, we find that our classification is useful for different applications. Sometimes, it is not necessarily to use the general graphs, and a little simpler graph is enough and saves time and space resources. Also, using our library to write these algorithms is minute-work, so it will be very handy for developers.