ECE608 Computational Models and Methods
Assignment

Cheng Ju Lee

Part 1

For the two given problems, I select the single-source shortest path problem. Then, I select two basic algorithms, Dijkstra's algorithm and Bellman-Ford algorithm, which were taught in class to solve the problem.

First we look at their complexities. Dijkstra's algorithm uses greedy method to choose the vertex with the smallest distance in each iteration and there are |V| iterations, so it takes O(V^2). As for Bellman-Ford algorithm, it relaxes every edge in the graph per iteration. With a total of |V| - 1 iterations, its complexity is O(VE). Therefore, my first thought was that the decision on what algorithm to use to solve the shortest path problem depends on the number of vertices and edges in a graph.

Before describing the algorithm for decision, we make sure that all edges are nonnegative. Otherwise, we'll just use Bellman-Ford algorithm to solve the problem since it has the ability to detect negative cycles. Now assume we have randomly generated a graph G, which contains |V| vertices and |E| edges. Below is the pseudocode:

```
selectAlgo(V, E)

1       For each edge e ∈ E do
2           if weight(e) < 0
3               then return "Bellman-Ford"
4       If |V| < |E|
5           then return "Dijkstra"
6       else
7           return "Bellman-Ford"
```

When |V| < |E|, V^2 < VE. As a result, we choose Dijkstra's algorithm, which has a complexity of O(V^2). On the contrary, when |V| >= |E|, V^2 >= VE. Then, we'll choose Bellman-Ford algorithm. This is a purely intuitive decision making. To find out actually which algorithm runs faster in these situations, we'll have to write a program to implement the algorithms. This will be done in the next part.

Part 2

Generate Graph:

Here, we randomly generate graph with given numbers of vertices and edges. With a given number of vertices |V|, we can determine that the maximum number of edges in the generated directed graph will be $|V|^2$ - V. So if |E| edges are going to be generated. Then, we simply sample $|E|$ numbers from 1 to $|V|^2$ - V. Let's say we sampled a number e. We'll add an edge from v to w, where v = e / (V-1) and w = e % (V-1). This is how I generate a random graph. As for the weight of an edge, it is also randomly generated from a given range of numbers. In my program, I set it from 1 to 10.

Dijkstra's Algorithm:

In order to make the algorithm runs faster. The algorithm will stop running if it finds out the smallest distance of a vertex found in one iteration is infinite, which means no more vertices are reachable from the source and the algorithm can terminate.

Bellman-Ford Algorithm:

The algorithm runs a total of |V| - 1 iterations to relax all edges. However, sometimes the relaxation has already been completed before it runs to the final iteration, which means we can terminate the algorithm earlier. In the implementation, we stop the algorithm when in one iteration, there is no label changes.

Implementation:

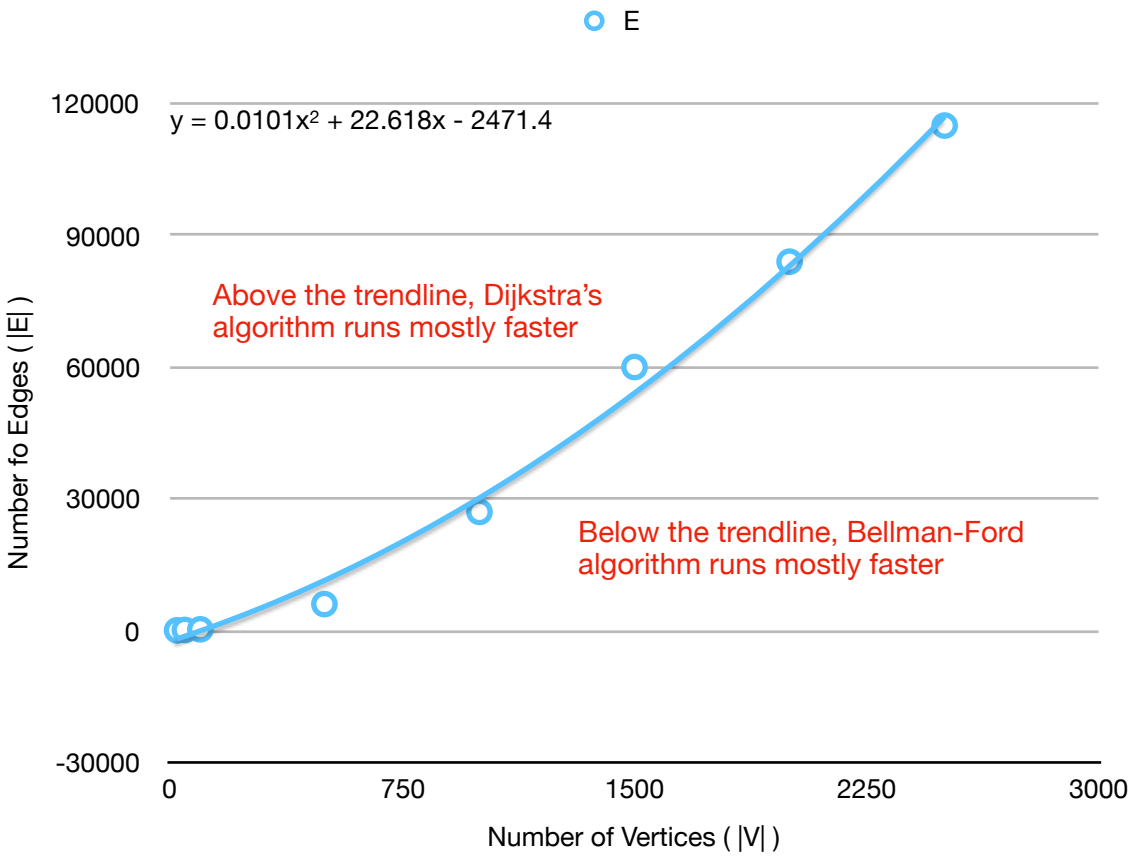
To find out in what situation one algorithm outperforms the other, we pick |V| = 25, 50, 100, 500, 1000, 1500, 2000, 2500. Them, we randomly generate graphs contain these number of vertices and each with different number of edges by the method described above.

We discover that after the number of edges exceeds a certain number, Dijkstra's algorithm will mostly run faster than Bellman-Ford algorithm. Before that, Bellman-Ford algorithm performs better. Table 1 below shows the exact number of vertices where the above situation happens.

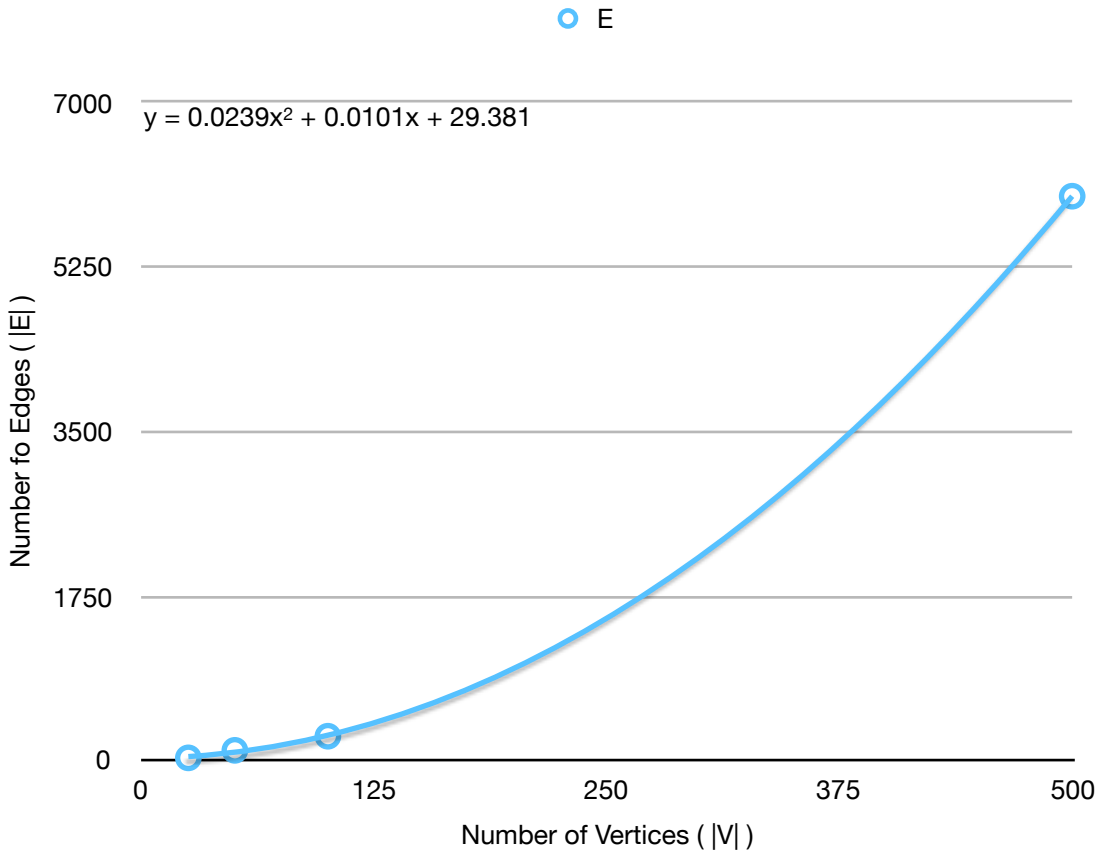
Table 1

V	25	50	100	500	1000	1500	2000	2500
E	31	111	261	6001	27001	60001	84001	115001

For the purpose of finding the approximate number of edges, we apply a trendline.



The trendline fits well to the data, but when the number of vertex is smaller than 104, the approximate number of edges will fall below 0, which we can not let it happen. Thus, we apply another trendline that only fits to a small portion of data for graphs with smaller number of vertices.



According to this new trendline, we can apply it to graphs with smaller number of vertices.

Improved Algorithm:

Because of the implementation, we can slightly improve the algorithm to make it more efficient. The improved pseudocode:

selectAlgo(V, E)

```
1       For each edge e ∈ E do
2          if weight(e) < 0
3             then return “Bellman-Ford”
4       if |V| < 250
5          if |E| > 0.0239 | V |^2 + 0.0101|V| + 29.381
6             then return “Dijkstra”
7          else
8             then return “Bellman-Ford”
9       else
10         if |E| > 0.0101 | V |^2 + 22.618|V| - 2471.4
11            then return “Dijkstra”
12         else
13            return “Bellman-Ford”
```

With this newly improved algorithm, we’ll test if it is as efficient as we think it will be. This time, we choose |V| = 100, 500, 1000. Table 2 on the next page shows the test result. The first two columns are the number of vertices and edges. The third and fourth column shows the running time for both algorithms. The fifth one shows which algorithm outperformed the other and the last column is the algorithm that the designed algorithm chose to solve the problem.

Table 2

V	E	Dijkstra(sec)	Bellman-Ford(sec)	Which is faster	Choose
100	50	5E-06	4E-06	Bellman-Ford	Bellman-Ford
100	550	8.5E-05	0.000116	Dijkstra	Dijkstra
100	1050	8.4E-05	0.000123	Dijkstra	Dijkstra
100	1550	9.8E-05	0.000173	Dijkstra	Dijkstra
100	2050	0.000109	0.000228	Dijkstra	Dijkstra
100	2550	0.000126	0.000214	Dijkstra	Dijkstra
100	3050	0.000183	0.000409	Dijkstra	Dijkstra
100	3550	0.000139	0.000375	Dijkstra	Dijkstra
100	4050	0.000149	0.000329	Dijkstra	Dijkstra
100	4550	0.000165	0.000409	Dijkstra	Dijkstra
500	50	7E-06	6E-06	Bellman-Ford	Bellman-Ford
500	2550	0.001214	0.000568	Bellman-Ford	Bellman-Ford
500	5050	0.00235	0.001085	Bellman-Ford	Bellman-Ford
500	7550	0.001386	0.0012	Bellman-Ford	Bellman-Ford
500	10050	0.001545	0.001319	Bellman-Ford	Bellman-Ford
500	12550	0.001241	0.001602	Dijkstra	Dijkstra
500	15050	0.001311	0.001917	Dijkstra	Dijkstra
500	17550	0.001407	0.001843	Dijkstra	Dijkstra
500	20050	0.001464	0.005527	Dijkstra	Dijkstra
500	22550	0.001527	0.00228	Dijkstra	Dijkstra
1000	50	1.2E-05	5E-06	Bellman-Ford	Bellman-Ford
1000	5050	0.004097	0.001062	Bellman-Ford	Bellman-Ford
1000	10050	0.005393	0.002827	Bellman-Ford	Bellman-Ford
1000	15050	0.004494	0.003397	Bellman-Ford	Bellman-Ford
1000	20050	0.004746	0.003492	Bellman-Ford	Bellman-Ford
1000	25050	0.004876	0.004033	Bellman-Ford	Bellman-Ford
1000	30050	0.004487	0.004147	Bellman-Ford	Bellman-Ford
1000	35050	0.006691	0.005651	Bellman-Ford	Dijkstra
1000	40050	0.004893	0.00556	Dijkstra	Dijkstra
1000	45050	0.004989	0.005918	Dijkstra	Dijkstra

The result is pretty good. It has a precision of 96.67% in choosing the faster algorithm to solve the problem.