HO CHI MINH UNIVERSITY OF TECHNOLOGY
COMPUTER SCIENCE AND ENGINEERING

SYSTEM PERFORMANCE EVALUATION

STUDENT REPORT

Student's name	: Nguyen Duy Tinh
Student's ID	: 1852797
Lecturer	: PhD. Tran Van Hoai

OCTOBER 1, 2020

Services

Dijkstra's algorithm, like Bellman-ford algorithm, finds the shortest path from an arbitrary vertex to all remaining vertices of a given G = (V, E) graph in which Dijkstra's algorithm is not applied. can be used for graphs with negative weights.

The Bellman-Ford algorithm is only applicable to directional and weighted G = (V, E) graphs where some edges may have negative weights. This algorithm is considered to be slower than Dijkstra's algorithm because it has to deal with edges with negative weights.

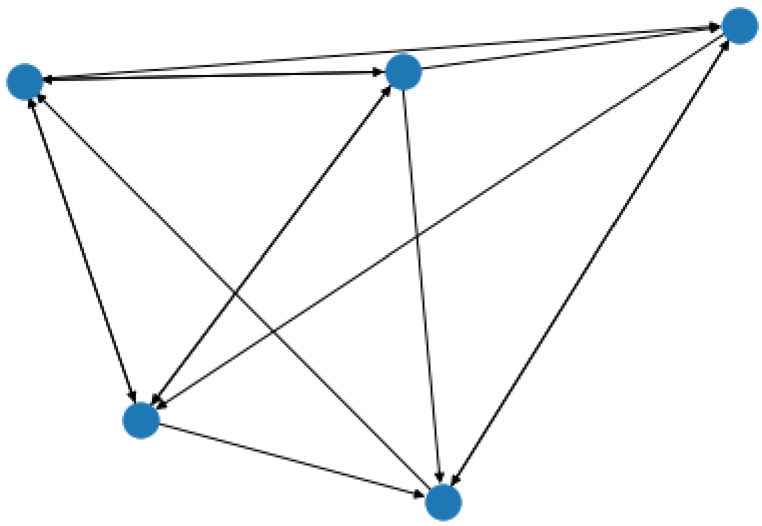


Figure 1.1

Besides, Johnson's algorithm aims to find the shortest path of all vertices of a graph G = (V, E) with direction, weight and without negative cycle.

All three algorithms find the shortest path, but because of the difference in the number of considered vertices, this report only considers the case where all three algorithms are applicable: applying Dijkstra algorithm, Bellman-Ford algorithm and Johnson algorithm to find the shortest path between pairs of vertices in the same graph G = (V, E) with a given direction and weight.

Performance metrics

Factors that influence the performance of an algorithm include the number of vertices and edges, which determine whether the execution time is fast or slow, and they are expressed as two variables in the complexity of each algorithm. maths: Dijkstra algorithm has the complexity $O(n^2 + m)$; Bellman-Ford algorithm has the complexity $O(m.n)$; Johnson algorithm has the complexity: $O(m + n.\log(n))$.

	Vertices	Dijkstra	Bellman-Ford	Johnson
1	20	2995700	4997600	3997600
2	20	3994800	7998100	3997500
3	20	1002300	3995000	2998200
4	20	4993800	13994900	20986000
5	20	3997600	7998100	3994800
6	20	999400	2998200	999400
7	20	1000500	3997100	1998900
8	20	2002200	3994200	2001600
9	20	3015900	5976100	3997600
10	20	998800	1999200	998700

Figure 2.1

Thus, the number of vertices and edges will be two variables to evaluate the operation time of each function. In this problem, the number of edges will be assigned by the randint () function to be objective and the number of vertices will in turn be numbers in the Sample set:

```
Sample = [5, 20, 50, 70, 100, 150, 200]
```

In this report, python and R are two languages used to evaluate the performance of these algorithms. To bring up a common problem, the two shortest path functions `nx.all_pairs_dijkstra_path(H)` and `nx.all_pairs_bellman_ford_path(H)` of the networkx test will be used to match all intent pairs instead of a vertex. The feature of these two functions is to need a variable representing the probability of an edge appearing between two vertices and in this article, this variable will be assigned to the random () function of the random library so that the collected samples will have multiple properties. be as objective as possible.

	Vertices	Dijkstra	Bellman-Ford	Johnson
70	170	1243287800	3109221200	1514130200
71	200	6279400200	16934292400	8136336400
72	200	5673751100	15629041000	7779538500
73	200	1562106100	4182602000	2965297700
74	200	2238717800	6384341500	3938739600
75	200	4721291100	12713712500	5488858100
76	200	2497572600	6586242500	3954714400
77	200	1635061200	4946165800	2629494600
78	200	732581100	2469585100	1159333700
79	200	3044255700	9207722000	4222579300
80	200	1168331500	3029262800	1523127300

Figure 2.2

As shown in Figure 2.2, it can be seen that 80 samples have been obtained corresponding to `Sample = [5, 20, 50, 70, 100, 150, 200]` by this calculation method.

Factors and their ranges

Three time variables are set for estimating the execution times of the three algorithms, which are collected at the time that the total number of vertices is 5, 20, 50, 70, 100, 150, 200. In each case , they take 10 samples:

```
time1 = [[0 for x in range(w)] for y in range(h)]
time2 = [[0 for x in range(w)] for y in range(h)]
time3 = [[0 for x in range(w)] for y in range(h)]
```

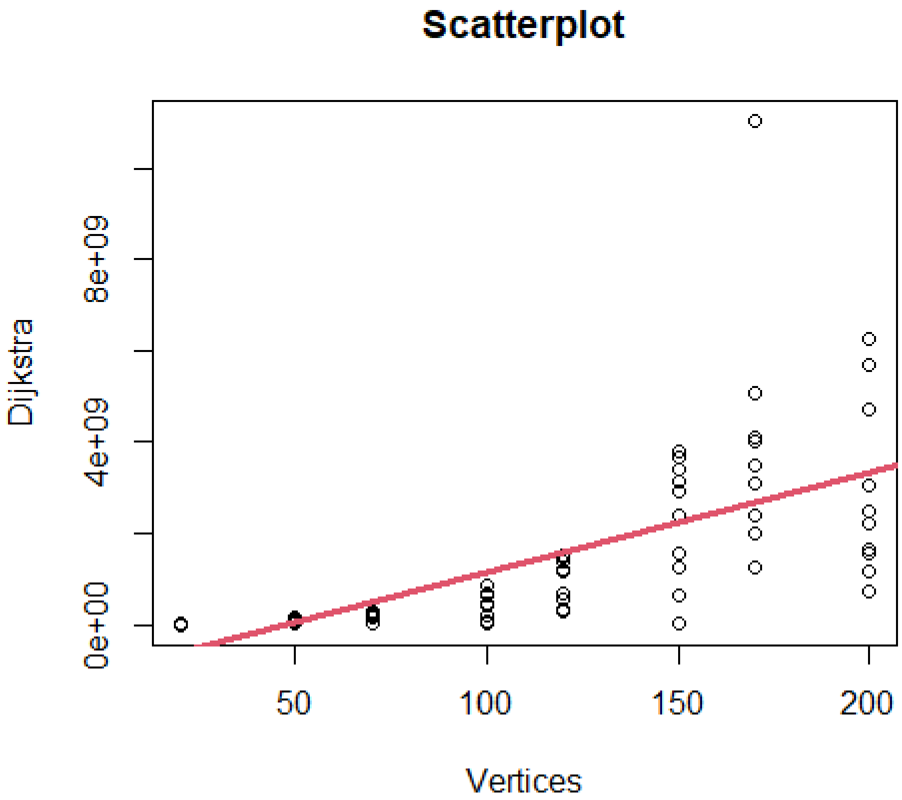
Variables such as an edge's weights or the probability that an edge exists are taken at random to ensure objectivity:

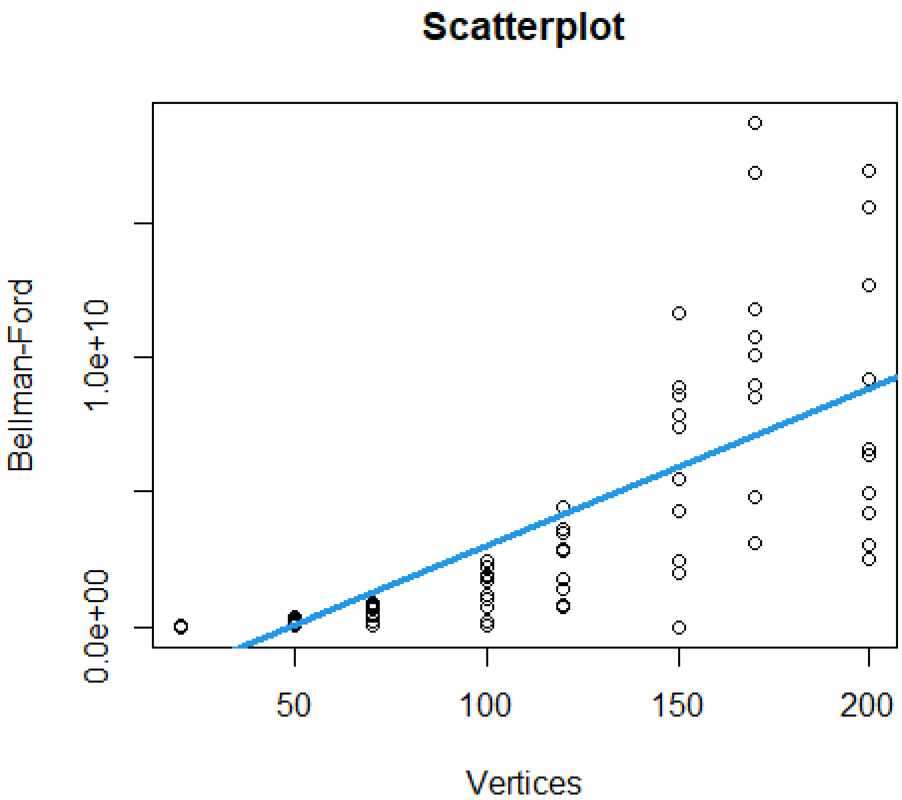
```
H.add_weighted_edges_from([(u,v,rd.randint(1, 100))])
```

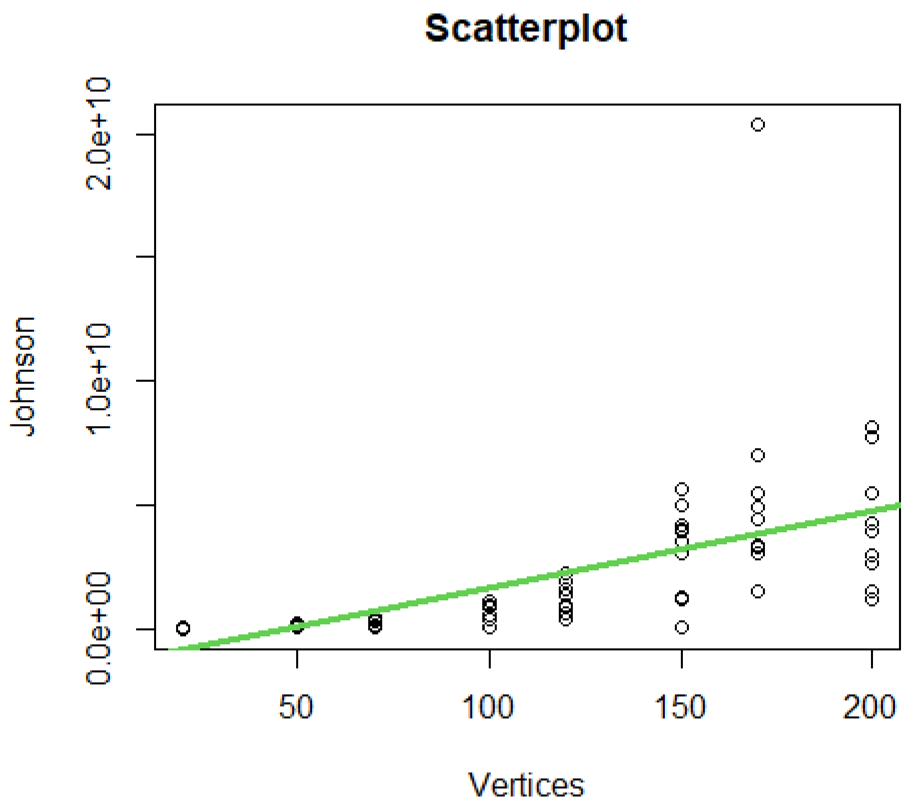
```
H = nx.binomial_graph(nonode, rd.random(),None,True)
```

Evaluation technique

The selected criteria for evaluation in this problem is the speed of the algorithms. Based on the linear regression model, we can plot the patterns on the graph and find the regression line of the execution time of all three algorithms as a function of the number of edges:







Workload

From the above chart and calculated, we find the slope of the time function of the algorithms according to the given number of edges according to the linear regression model.

```
Call:
lm(formula = Dijkstra ~ Vertices)

Residuals:
       Min         1Q     Median         3Q        Max 
-2.619e+09 -6.894e+08 -1.946e+08  6.199e+08  8.343e+09 

Coefficients:
              Estimate Std. Error t value Pr(>|t|)    
(Intercept) -1.060e+09  3.364e+08  -3.150  0.00231 ** 
Vertices     2.206e+07  2.707e+06   8.149 4.83e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Figure 5.1: Dijkstra algorithm's running time

```
lm(formula = `Bellman-Ford` ~ Vertices)

Residuals:
       Min         1Q     Median         3Q        Max
-6.369e+09 -1.886e+09 -4.297e+08  1.685e+09  1.164e+10

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -2.849e+09  7.681e+08  -3.709 0.000387 ***
Vertices     5.844e+07  6.180e+06   9.456 1.41e-14 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Figure 5.2: Bellman-Ford algorithm's running time

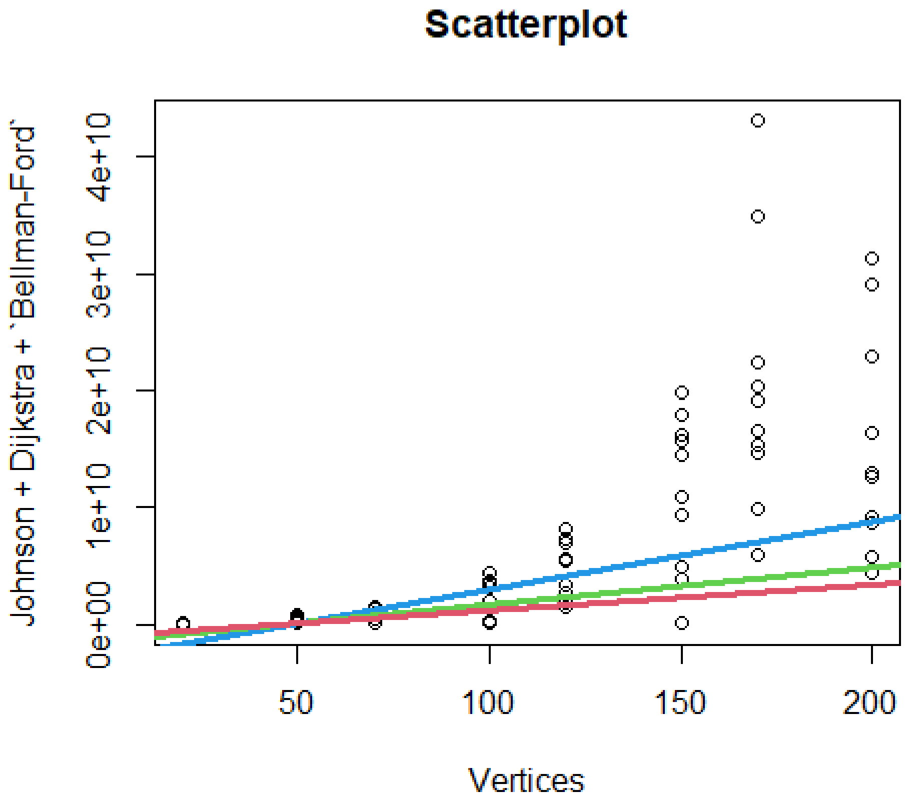
```
lm(formula = Johnson ~ Vertices)

Residuals:
       Min         1Q     Median         3Q        Max
-3.613e+09 -8.220e+08 -2.769e+08  8.202e+08  1.654e+10

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -1.522e+09  5.547e+08  -2.744  0.00753 **
Vertices     3.147e+07  4.463e+06   7.052 6.27e-10 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Figure 5.3: Johnson algorithm's running time

By graph below, we conclude that Dijkstra algorithm has the most optimal performance followed by Johnson function because it is a combination algorithm between Dijkstra and Bellman-Ford; The worst performing algorithm in this article is Bellman-Ford because it takes extra steps to consider negative weights that are inherently omitted in this problem. In the graph below, Johnson's line is denoted green, Bellman-Ford is blue and red is Dijkstra's:


Scatterplot
Johnson + Dijkstra + `Bellman-Ford`
0e+00
1e+10
2e+10
3e+10
4e+10
50
100
150
200
Vertices