Finding Widest K-shortest Paths in Network Using Genetic Algorithm

Yu Shao, Liu
劉昱劭
0416235
c252541@gmail.com

1 Abstract

In this interconnected world, more and more usage of multimedia application is engaged. Most of the multimedia applications require large amount of bandwidth during the communication between source and destination. In this paper, we propose an algorithm that apply genetic algorithm to determine the widest k-shortest paths from source to destination.

2 Introduction

Widest path problem is to find a path between two designated nodes in a weighted graph, trying to maximize the weight of the minimum-weight edge in the path.

In networking, finding widest path is a common problem because many services need to guarantee bandwidth in transmission. However, there are other issues in real world networking. For example, due to unpredictability of networks, finding the multiple paths as redundancies instead of finding only the shortest one is needed for fault tolerance. Put latency into concern, the path should be as short as possible.

Many algorithms aim to find either shortest path or widest path. Dijkstra algorithm and its variant. We apply genetic algorithm on path finding problem, trying to find k paths to meet both two objectives, that is, maximize the bottleneck bandwidth and minimize the path length.

3 Problem Representation

In genetic algorithm, each possible solution is represented as an individual. Every individual will have its own chromosome that represent itself. In this work, integer array is used to represent an individual.

s	$\mathbf{n_1}$	$\mathbf{n_2}$	**…**	**…**	$\mathbf{n_{m-1}}$	$\mathbf{n_m}$	**d**

Figure 1: **Chromosome that represents the path between source and destination**

4 Method

4.1 Initializing Graph

There are some classical models for generating random graphs given the number of vertex n. The most commonly used model is $G(n,p)$ proposed by Gilbert, with the running time $\Theta(n^2)$, where p represents the probability of edge occurrence. [5]. Another model $G'(n,m)$ proposed by Erdős and Rényi, with the running time $O(m \log m)$, where $m = \binom{n}{2}$ [6]. Both are not efficient enough to generate large graphs. Faster algorithm implemented in NetworkX framework [8] generates different size of random weighted graphs used as test cases. This algorithm is introduced by Batagelj and Brandes, with the running time $O(n + k)$, where $k = p\,n\,(n - 1) / 2$ [7].

Initialization in this research is done using random weight assignment. The values of weights are sampled from F-distribution with parameters d1 of 1 and d2 of 1 (Figure 5). This distribution is used to simulate how the real network looks like. Which consists of high-bandwidth backbone and majorly low-bandwidth connections.

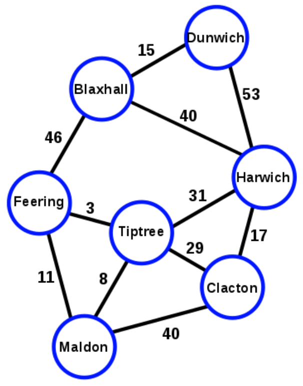


Figure 2: **Example of input graph**
Source: https://en.wikipedia.org/wiki/Widest_path_problem

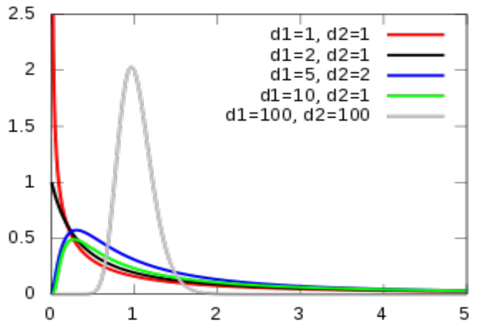
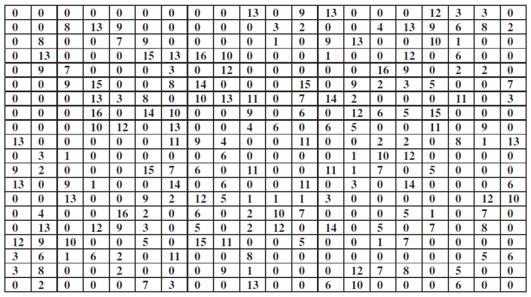


Figure 3: The probability density function of **F-distribution.** Source: https://en.wikipedia.org/wiki/F-distribution



0	0	0	0	0	0	0	0	0	13	0	9	13	0	0	0	12	3	3	0
0	0	8	13	9	0	0	0	0	0	3	2	0	0	4	13	9	6	8	2
0	8	0	0	7	9	0	0	0	0	1	0	9	13	0	0	10	1	0	0
0	13	0	0	0	15	13	16	10	0	0	0	1	0	0	12	0	6	0	0
0	9	7	0	0	0	3	0	12	0	0	0	0	0	16	9	0	2	2	0
0	0	9	15	0	0	8	14	0	0	0	15	0	9	2	3	5	0	0	7
0	0	0	13	3	8	0	10	13	11	0	7	14	2	0	0	0	11	0	3
0	0	0	16	0	14	10	0	0	9	0	6	0	12	6	5	15	0	0	0
0	0	0	10	12	0	13	0	0	4	6	0	6	5	0	0	11	0	9	0
13	0	0	0	0	0	11	9	4	0	0	11	0	0	2	2	0	8	1	13
0	3	1	0	0	0	0	0	6	0	0	0	0	1	10	12	0	0	0	0
9	2	0	0	0	15	7	6	0	11	0	0	11	1	7	0	5	0	0	0
13	0	9	1	0	0	14	0	6	0	0	11	0	3	0	14	0	0	0	6
0	0	13	0	0	9	2	12	5	1	1	1	3	0	0	0	0	0	12	10
0	4	0	0	16	2	0	6	0	2	10	7	0	0	0	5	1	0	7	0
0	13	0	12	9	3	0	5	0	2	12	0	14	0	5	0	7	0	8	0
12	9	10	0	0	5	0	15	11	0	0	5	0	0	1	7	0	0	0	0
3	6	1	6	2	0	11	0	0	8	0	0	0	0	0	0	0	0	5	6
3	8	0	0	2	0	0	0	9	1	0	0	0	12	7	8	0	5	0	0
0	2	0	0	0	7	3	0	0	13	0	0	6	10	0	0	0	6	0	0

Figure 4: **Example of initialized network adjacency matrix [1]**

4.2 Initializing Population

Initial population affect result significantly. A good population provides diverse individuals which leads to larger search space for the network to begin with. Length of individual should be considered as well. The longer the path, the higher chance the path(individual) contains a bottleneck effecting total bandwidth. However, its offspring is more diverse.
In this research, random walk is used to initialize the population and each individual is created by this fashion. Individuals are guaranteed no cycle is contained and the path is long enough.

4.3 Crossover Operation

Crossover in using one-point crossover. A cut point is selected, two parents will be elected, producing two children with exchanged parts from two parents.
The special parts are the way the cut point is selected, due to the choices the point can be selected (approximately the possible way to choose 1 point in each parent chromosomes). Cutting point is calculated using uniformly random generated number that is capped between 0(zero) and the maximum number of possible cut available. Possible cut available is equal to the multiple of length of each parent chromosomes minus 1. The location will the pinpoint the possible location for the cutting point.

There is high chance where the point selected will produce defective child which has disconnected path or cycle in its path. Therefore, further checking is needed. If defective child is detected, cutting point will be re-calculated. To prevent this operation cannot be completed, maximum attempt is limited. And the child will not be generated.

Parent 1:

s	$p_{1,1}$	$p_{1,2}$	$p_{1,3}$	...	$p_{1,m-1}$	$p_{1,m}$	d

Child:

s	$p_{1,1}$	$p_{1,2}$	$p_{2,3}$	...	$p_{2,m-1}$	$p_{2,m}$	d

Parent 2:

s	$p_{2,1}$	$p_{2,2}$	$p_{2,3}$	...	$p_{2,m-1}$	$p_{2,m}$	d

Figure 5: **Example of crossover operation**

4.4 Mutation Operation

Mutation is also used in this research to increase search space of the network. A node is randomly chosen from the path. Its previous and next node are inspected. They both have connected nodes on their own. After intersecting those nodes from these two sets and randomly choose a node to replace original chosen node. This is the mutation operation on a single node.

If the path is too short i.e. The path contains less than two nodes, the operation will not be committed. In this research, about 50% of nodes (except the source and destination) are mutated. And some of them may be mutated multiple times. The operation is now completed and the new generated individual is bound to be a legal path.

Parent 1:

s	n_1	n_2	...	...	n_{m-1}	n_m	d

Child:

s	n_1	n_k	...	...	n_{m-1}	n_m	d

Figure 6: **Example of mutation operation**

4.5 **Cycle Check**

Individual may contain one or multiple cycles after crossover or mutation operation. The path is still a legal path (i.e. Destination can still be reached from source). However, this will have major impact to the experiment. First, the nodes between cycle may be connected with low bandwidth, which is totally unnecessary and harmful. Second, after each generation, new crossover and mutation operations are applied. New cycles may be generated, and it will have impact on performance.

To prevent this. A check has been done if the path contains any cycle after possible operation (e.g. random walk, crossover etc.) By doing so, overall fitness value should rise and path will be simplified. Theoretically, getting rid of any cycle will limit the diversity of search space. As there may exist better path cannot be discovered by any means of current implementation. But that is not this research is focused on.

4.6 **Fitness**

In order to determine which individual should be disposed, fitness is calculated and mapped to each individual.
Each individual is subject to:

$$max\ f_1(x) = min\{\, bandwidth(e) \mid e \in individual \,\}$$
$$min\ \ f_2(x) = |individual|$$

$f_1(x)$ is the bottleneck bandwidth in the individual.
$f_2(x)$ represents the number of nodes in the individual, or the hops in the corresponding path.

As mentioned, bottleneck bandwidth is the major concern. ε - constraint method is used to deal with these two objective, and take $f_1(x)$ as measurement in survival selection process.
After that, use of $f_2(x)$ as constraint to rank top k individuals, which represents top k shortest paths.

5 **Experiments**

The experiments are designed to verify correctness of the proposed method (Sec 6.1), show the solutions found (sec 6.2), and determine the parameters by empirical results (Sec 6.3).
Some constants don't have impact on the experiments. We use same constants to do the experiments.
By setting source to 0, destination to 1, and k to 5. GA will find top 5 shortest path with maximum bottleneck bandwidth from node 0 to node 4

5.1 **Correctness Verification**

This section describes how this work verifies the correctness of our work. A random weighted graph with 30 nodes (Figure 7.) and edge creation probability p is set to 0.1. This case is relatively small so that DFS could traverse all possibilities in few seconds and get the optimal solutions.

Comparing the ground truth obtained by DFS with results of GA (Table 1) shows the top 15 paths obtained by DFS. From Table 2 to Table 4 shows the top 5 paths obtained by GA with different population size. We found that all results obtained by GA are in the ground truth, which implies the proposed method could find legal good paths.

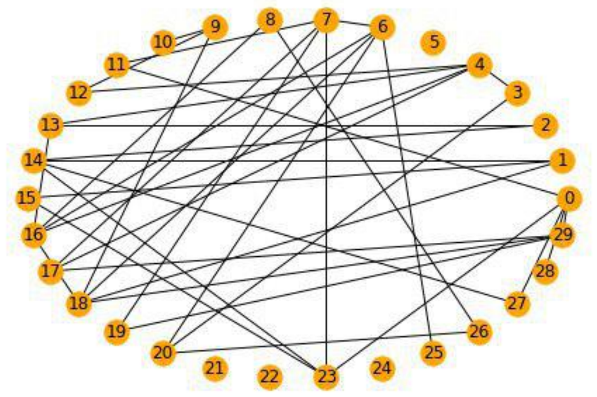


Figure 7: A random weighted graph G(n, p), n=30, p=0.1, bandwidth is removed for better visualization, the bandwidth matrix can be accessed at:
https://github.com/AilurusUmbra/evolutionary_computation /blob/master/node30.csv

	fitness	individual	hop
0	541	[0, 11, 7, 23, 15, 1]	6
1	541	[0, 23, 15, 1]	4
2	482	[0, 11, 7, 23, 14, 1]	6
3	482	[0, 23, 14, 1]	4
4	291	[0, 11, 7, 17, 29, 18, 6, 20, 3, 4, 16, 13, 2,...	17
5	291	[0, 11, 7, 17, 4, 3, 20, 6, 18, 16, 13, 2, 14,...	16
6	291	[0, 11, 7, 17, 29, 18, 6, 20, 3, 4, 16, 13, 2,...	15
7	291	[0, 23, 7, 17, 29, 18, 6, 20, 3, 4, 16, 13, 2,...	15
8	291	[0, 11, 7, 17, 4, 3, 20, 6, 18, 16, 13, 2, 14, 1]	14
9	291	[0, 23, 7, 17, 4, 3, 20, 6, 18, 16, 13, 2, 14, 1]	14
10	291	[0, 11, 7, 17, 29, 18, 16, 13, 2, 14, 23, 15, 1]	13
11	291	[0, 11, 7, 17, 4, 16, 13, 2, 14, 23, 15, 1]	12
12	291	[0, 11, 7, 17, 29, 18, 16, 13, 2, 14, 1]	11
13	291	[0, 23, 7, 17, 29, 18, 16, 13, 2, 14, 1]	11
14	291	[0, 11, 7, 17, 4, 16, 13, 2, 14, 1]	10

Table 1: Top 15 paths obtained by DFS

	fitness	individual	hop
0	541	[0, 11, 7, 23, 15, 1]	6
1	541	[0, 23, 15, 1]	4
2	482	[0, 11, 7, 23, 14, 1]	6
3	482	[0, 23, 14, 1]	4
4	264	[0, 29, 17, 7, 23, 14, 1]	7

Table 2: 5 paths obtained by GA with population size of 10

	fitness	individual	hop
0	541	[0, 11, 7, 23, 15, 1]	6
1	541	[0, 23, 15, 1]	4
2	482	[0, 11, 7, 23, 14, 1]	6
3	482	[0, 23, 14, 1]	4
4	291	[0, 11, 7, 17, 4, 16, 13, 2, 14, 1]	10

Table 3: 5 paths obtained by GA with population size of 50

	fitness	individual	hop
0	541	[0, 11, 7, 23, 15, 1]	6
1	541	[0, 23, 15, 1]	4
2	482	[0, 11, 7, 23, 14, 1]	6
3	482	[0, 23, 14, 1]	4
4	291	[0, 11, 7, 17, 4, 3, 20, 6, 18, 16, 13, 2, 14, 1]	14

Table 4: 5 paths obtained by GA with population size of 100

5.2 **Pareto Front**

This section describes the Pareto front, which shows the optimal solutions found by the proposed method.

Pareto front represents the optimal solutions considering all objective functions. In this research, points on the Pareto front represent the paths that shorter than their neighbor, or larger bandwidth than their neighbor. The red line in following figures (Figure 8. to Figure 9..) shows the Pareto front in different experiments

5.2.1 **G(n, p), n = 1000, p = 0.1**

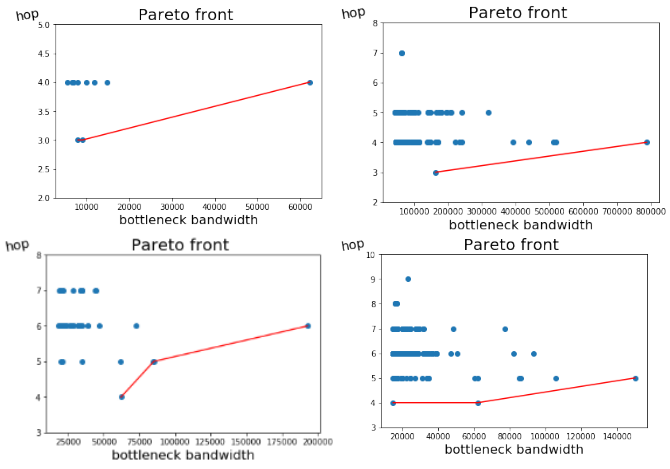


Figure 8: The Pareto front by GA with pop_size 10, 50, 100, 200

5.2.2 **G(n, p), n = 10000, p = 0.1**

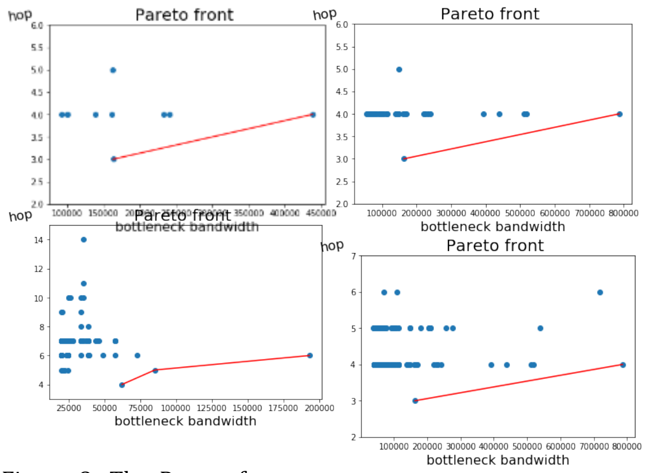


Figure 8: The Pareto front
by GA with pop_size 10, 50, 100, 200

6.3 Population Size

This section describes how the population size influence on the generation needed. (Figure 16. to 18.) shows the results of G(n, 0.1) with 100, 1000, and 10000 nodes. The results imply:

1. Complex graph needs more generation to create better individuals.
2. Larger population size speed up the time needed of the creation of individual we want.

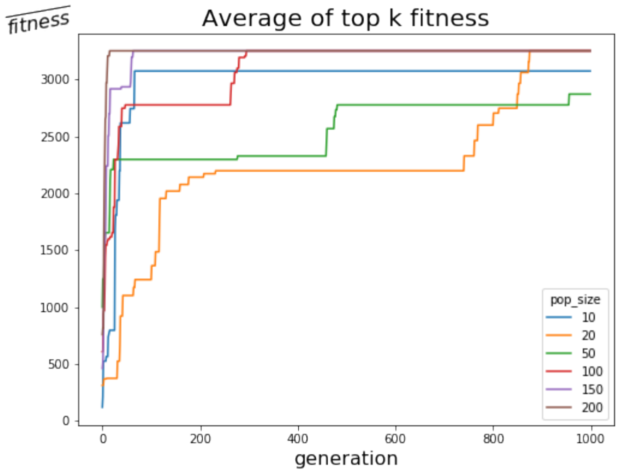


Figure 16: Average fitness of k individuals to generation (100 nodes)

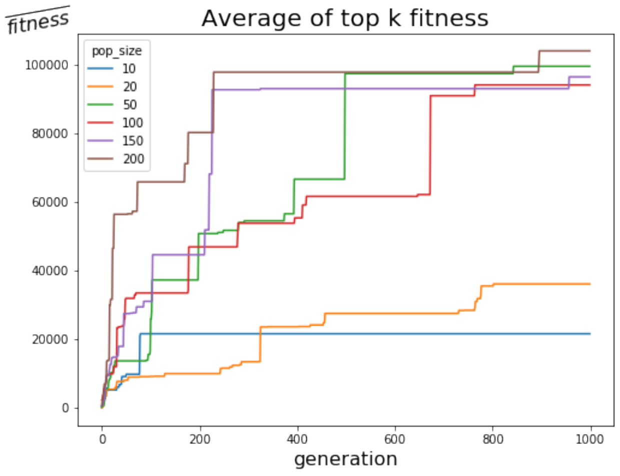


Figure 17: Average fitness of k individuals to generation (1000 nodes)

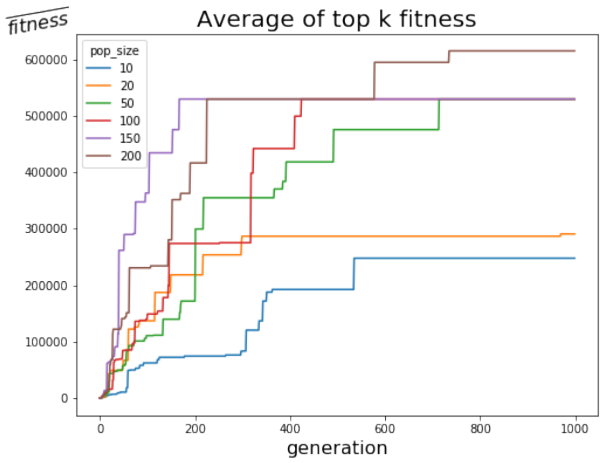


Figure 18: Average fitness of k individuals to generation (10000 nodes)

6 Conclusion

Brute force method can only handle network with nodes around 30 ~ 40. And the time needed will increase drastically when the scale grows larger. The proposed method can find several good enough solutions to widest k-shortest paths in a reasonable time.

This work could also be considered a template of path finding problem. As long as change the objective function, GA will have different behavior. For example, to minimize

$$f_3(x) = \sum_{e \in individual} weight(e)$$

this work will become the classical shortest path finding.

References

[1] Ahmed Younes Hamed (2010). A genetic algorithm for finding the k shortest paths in a network. Egyptian Informatics Journal, 11, 75-79.
[2] Bako A, Kas P. Determining the k-th shortest path by matrix method. Szigma 1977;10:61–6 [In Hungarian].
[3] Kumar N, Ghosh RK. Parallel algorithm for finding first K-Shortest paths. Comput Sci Inform 1994;24(3):21–8.
[4] Yen JY. Another algorithm for finding the K shortest-loop less network paths. In: Proc. 41st mtg. operations research society of America, vol. 20; 1972. p. B/185.
[5] E. N. Gilbert, Random Graphs, Ann. Math. Stat., 30, 1141 (1959).
[6] P Erdős and A. Rényi, On Random Graphs, Publ. Math. 6, 290 (1959)..
[7] Vladimir Batagelj and Ulrik Brandes, "Efficient generation of large random networks", Phys. Rev. E, 71, 036113, 2005.
[8] NetworkX frameworks, URL: https://networkx.github.io/