EL7363 Communication Networks II:

Design and Algorithms

Shortest Path Finding In Selfish Network

NEW YORK UNIVERSITY

TANDON SCHOOL OF ENGINEERING

TIANPEI CAI tc1996@nyu.edu

YUANZHI YAO yy1776@nyu.edu

Introduction

During this semester, we mainly talked about cooperate network modules, i.e. to achieve our objective, we can allocate each demand to the paths we want, and each demand will cooperate. This kind of modules may work well in computer networks, but will fail in many other networks, for example, the traffic network.

In a non-cooperate network, some new problem like "Braess paradox" arised. In order to do research on this kind of networks, the very first step may be "how to find the "shortest path"(actually the Nash equilibrium) for a selfish routing problem.

Keyword

non-cooperate network, selfish routing, BRITE, Braess’ Paradox, Nash equilibrium, dynamic programming.

Part 1. Project Definition

1.1 Problem Definition

In this project , a method to find the *Nash Equilibriums*(check 1.2 for detail) for selfish routing problems will be given. We will conduct it in two different type of networks: (1) Non-congestion network, where the congestion of each link are negligible, thus delay of each link is fixed. (2)Congestion network, where the congestion matters and a link's delay will be related to the load of it.

In both kind of network, we will find out the Nash Equilibrium for given demands, in topologies that generated randomly by BRITE (https://www.cs.bu.edu/brite/).

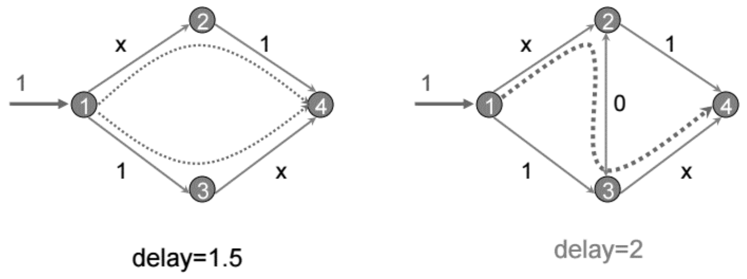
1.2 Background

Braess' Paradox

The Braess' Paradox Problem can be define in short as:

Adding extra capacity to a network when the moving entities selfishly choose their route can in some cases reduce overall performance.

This concept is somehow against human intuition. As showing below, after we add an ideal link to this network, the overall delay increasing.



Taking it into our daily life, a new high-way opened, people all want to leverage it, however, due to the selfish routing strategy, congestion became heavily on the connection roads. All of this problem is created by lacking of order.

Nash Equilibrium

Nash Equilibrium is a common definition in game theory. In this selfish routing problem, it's basically an allocation where everyone in the network will be satisfied, which means no one want to change its path.

Nash Equilibrium is a well defined name, for details, please refer to wiki or any resource.

1.3 Tools

Topology Generator: BRITE (https://www.cs.bu.edu/brite/)

Programming Language: Java

Collaborate Platform: GitHub (https://github.com/Tempay/ShortestPathFinding)

Part 2. Design Models

For shortest path routing, we considered the COST, which contains: propagation delays, unit costs, interface delays…, in our problem, the topology has already be defined by BRITE, so here we considered about the propagation delay.

In our project, as mentioned above, we designed two model to find the shortest path: Non-Congestion model and Congestion model.

2.1 Non-Congestion Model

In this model, we do not consider the additional delay cost by traffic volume on each link. For a link, we use the parameter "length" generated by BRITE as the fixed delay of it.

Constants

a_{ev}=1 if link e originaes at node v, 0 therwise

b_{ev}=1 if link e terminates in node v, 0 otherwise

s source for the desired shortest path

t target for the desired shortest path

h demand

ξ_e delay of link e

variables

δ_e =1 if link e is on the shortest path, 0 otherwise

objective

minimize $F = \Sigma_e \xi_e \delta_e$

constrains

$$\Sigma_e a_{ev} \delta_e - \Sigma_e b_{ev} \delta_e = \begin{cases} h & \text{if } v=s \\ 0 & \\ -h & \text{if } v=d \end{cases}$$

2.2 Congestion Model

In this model, we consider the delay cost by the traffic volume on each link, if there are more traffic on link, the delay will be greater. In this model. We assume the relation is linear, that is, the delay of a link e, will become $(\xi_e+C_eP_e)P_e$.

Again, we consider selfish routing, i.e. each player are trying to find a shortest path/strategy. Because a path's delay is depend on the load of the path, obviously the Nash Equilibrium will be a combined strategy, with multiple paths, and corresponding probability of choosing it.

Constants

a_{ev}=1 if link e originaes at node v, 0 therwise	
b_{ev}=1 if link e terminates in node v, 0 otherwise	
s	***source for the desired shortest path***
t	***target for the desired shortest path***
h	***demand***
ξ_e	***delay of link e***
Ce	***congestion delay factor, congestion on a link= ce*load(e)***

variables

P_e ***=The probability of choosing link e in the best strategy***

objective

minimize $F=\Sigma_e(\xi_e+C_eP_e)P_e$

constrains

$$\Sigma_e a_{ev}P_e - \Sigma_e b_{ev}P_e = \begin{cases} h & \text{if } v=s \\ 0 & \\ -h & \text{if } v=d \end{cases}$$

Part 3. Solution Techniques

3.1 BRITE

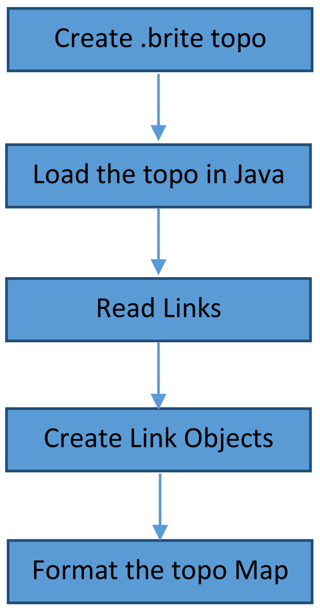
BRITE is an approaching towards universal topology generation, it supports multiple generation models including flat AS, flat Router and hierarchical topologies, and those models can be enhanced by assigning links attributes such as bandwidth and propagation delay.

This platform is implemented in Java and C++ with a user friendly GUI that user can easily create a network topology with expected size.

In this project, we requires the Java Environment.

3.2 Java IO for .brite File

After create a .brite file for topology, we have to load it into Java via Java IO tools.



This is how we load the network topology created by BRITE, due to the certain pattern of the .brite file, it's easy to parse out all the links and create a Map for all links. The next step is using DP to find out the shortest path for certain demand pair.

3.3 Algorithm for non-congestion Shortest Path Finding

We developed a very delicate and effective algorithm, dynamic programming and recursion has been used. We are trying to explain it in short, ***but there is no way to fully understand an algorithm without reading the code, let's read the code for details.***

Objective: find shortest path from s to t;

Step 1: build a map to keep shortest paths for each source to the same target;

Step2: if s is already in the map, return the value in the map

Step 3: if s is not in the map, find shortest path from each m to t, where m is connected to s. ***(recursion, goto objective (m,t))***

Step 4: find the shortest path among step 3, and save it in the map.

3.4 Algorithm for Congestion Nash-Equilibrium Finding

To find the Nash Equilibrium, we use repeat trials:

Objective: find the probability of choosing each path

```
for (i=1-100) {
        find the shortest path in current situation
        add delay to each link of the shortest path
        the probability of choosing this path+1%
}
```

Again, talk is cheap, let's read the code for details.

(https://github.com/Tempay/ShortestPathFinding)

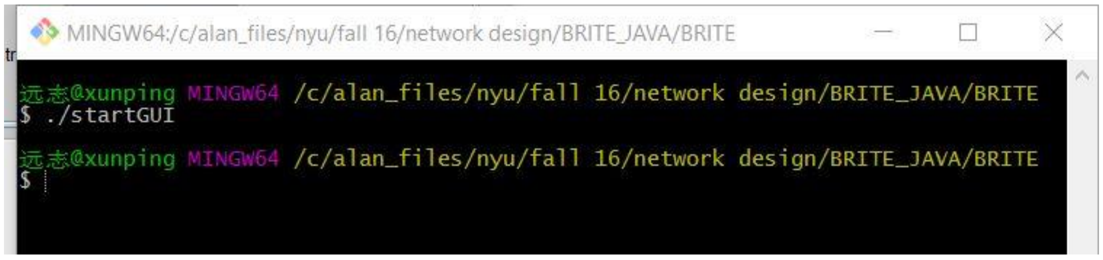
Part 4. Project Manual

4.1 Start with BRITE

4.1.1 Start with GUI

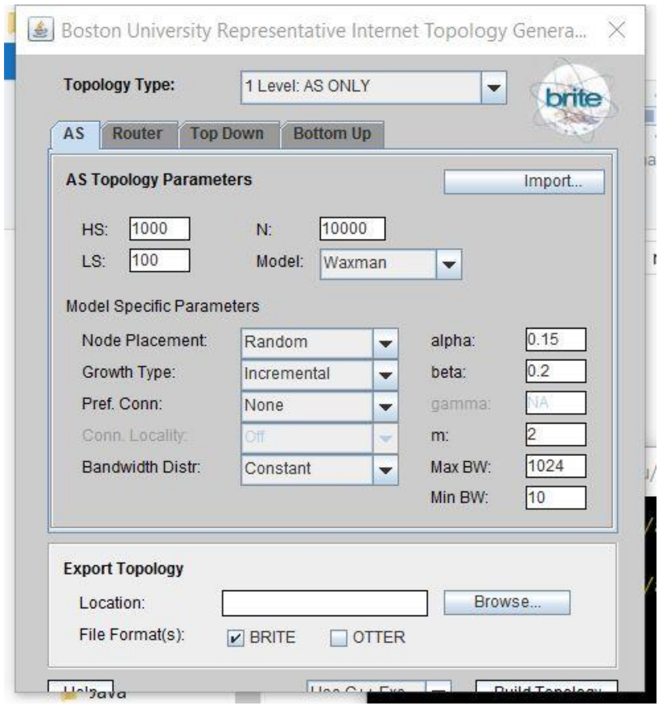
Running the BRITE GUI requires some dependency like C++ environment. BRITE has already create the Makefile for users, so make sure you have run *make* command at all BRITE, BRITE/GUI, BRITE/Java directory.

Then, at BRITE directory, type ./startGUI



```
MINGW64:/c/alan_files/nyu/fall 16/network design/BRITE_JAVA/BRITE

远志@xunping MINGW64 /c/alan_files/nyu/fall 16/network design/BRITE_JAVA/BRITE
$ ./startGUI

远志@xunping MINGW64 /c/alan_files/nyu/fall 16/network design/BRITE_JAVA/BRITE
$ 
```

The GUI interface will start automatically:



Boston University Representative Internet Topology Genera...

Topology Type: 1 Level: AS ONLY

AS Topology Parameters

HS: 1000 N: 10000
LS: 100 Model: Waxman

Model Specific Parameters

Node Placement: Random alpha: 0.15
Growth Type: Incremental beta: 0.2
Pref. Conn: None gamma: NA
Conn. Locality: Off m: 2
Bandwidth Distr: Constant Max BW: 1024
Min BW: 10

Export Topology

Location:

File Format(s): [✓ BRITE] OTTER

Adjust the variables and export location. (For this project, simply change the N, HS/LS is optional for the topo map size).

Press Build Topology. We realized that it won't generate .brite due to unknown bug, but it will generate GUI_GEN.conf file.

Skip to 3.1.3 and follow the guide.

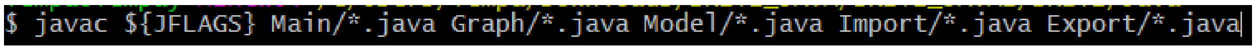
4.1.2 Start without GUI

Without the GUI, we still can generate .brite file we need. Following the steps:

Step 1

At SOME+PATH\BRITE\Java, type:

javac ${JFLAGS} Main/.java Graph/*.java Model/*.java Import/*.java Export/*.java*



```
$ javac ${JFLAGS} Main/*.java Graph/*.java Model/*.java Import/*.java Export/*.java
```

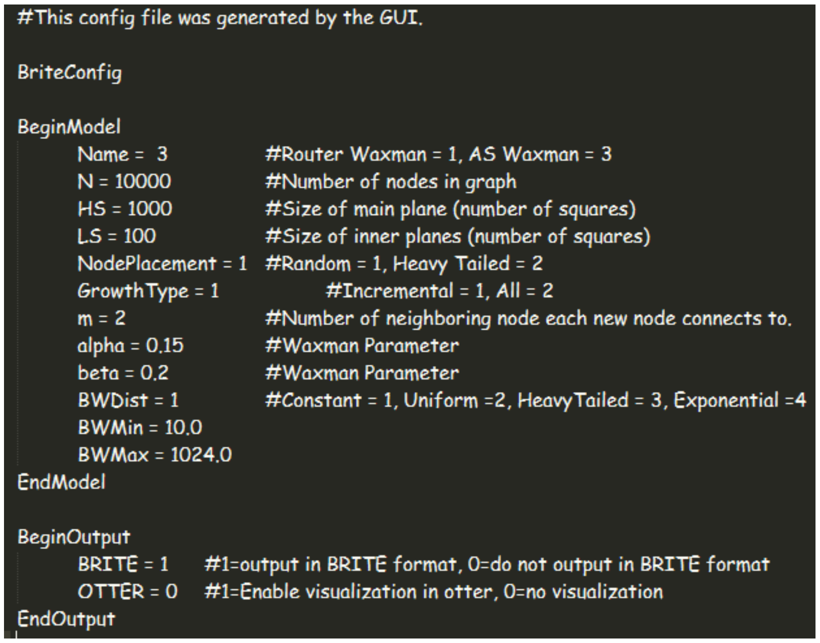
to compile the Java file.

Step 2:

Create a GUI_GEN.conf file as shown below.

(template here: https://gist.github.com/Tempay/f346ff819f3895905f05277d41469e6b)

Simply change the value for your need, for change the size of topo, we only need to modify N (number of nodes).



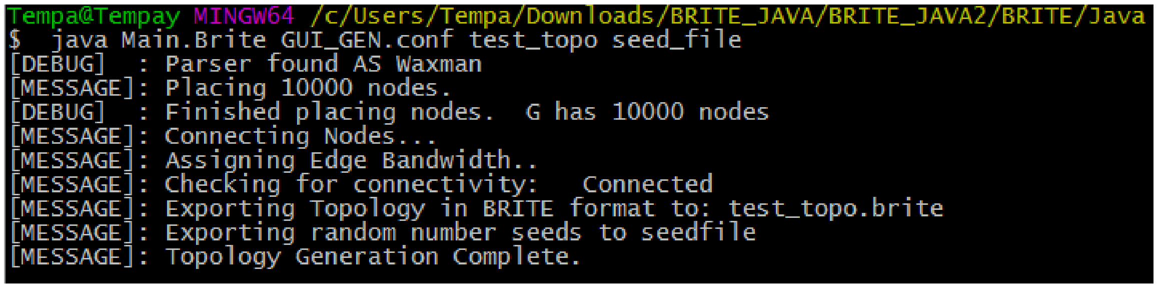
```
#This config file was generated by the GUI.

BriteConfig

BeginModel
	Name =  3		#Router Waxman = 1, AS Waxman = 3
	N = 10000		#Number of nodes in graph
	HS = 1000		#Size of main plane (number of squares)
	LS = 100		#Size of inner planes (number of squares)
	NodePlacement = 1	#Random = 1, Heavy Tailed = 2
	GrowthType = 1		#Incremental = 1, All = 2
	m = 2			#Number of neighboring node each new node connects to.
	alpha = 0.15		#Waxman Parameter
	beta = 0.2		#Waxman Parameter
	BWDist = 1		#Constant = 1, Uniform =2, HeavyTailed = 3, Exponential =4
	BWMin = 10.0
	BWMax = 1024.0
EndModel

BeginOutput
	BRITE = 1	#1=output in BRITE format, 0=do not output in BRITE format
	OTTER = 0	#1=Enable visualization in otter, 0=no visualization
EndOutput
```

4.1.3 Generate topo

At BRITE/Java directory, type:

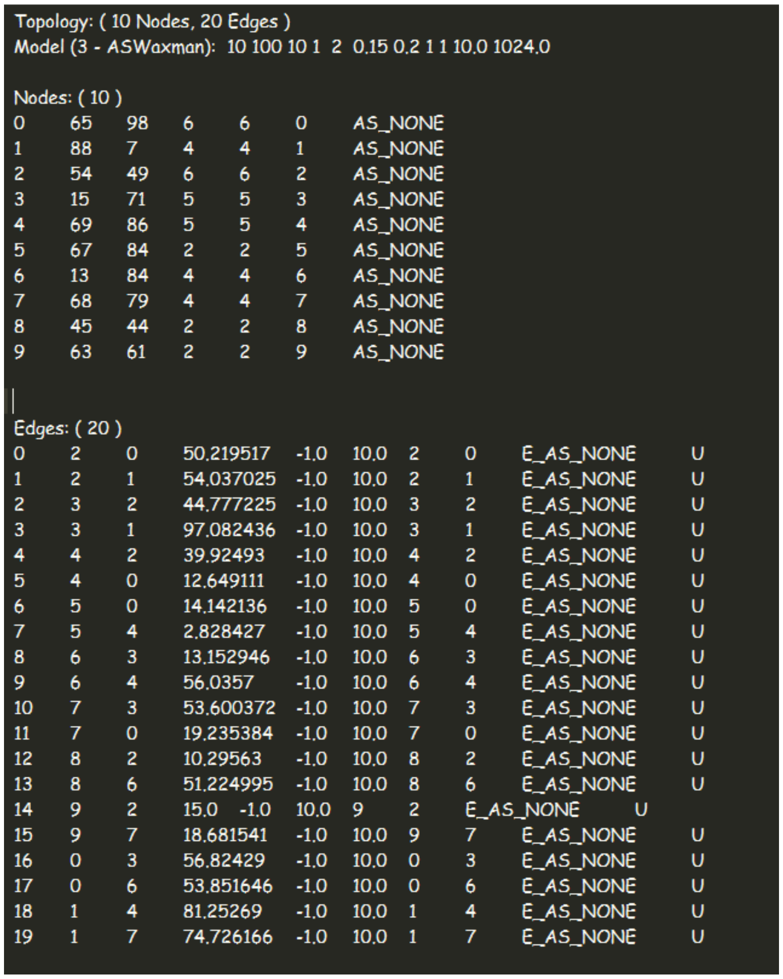
```
java Main.Brite GUI_GEN.conf {OUTPUT_FILE_NAME} seed_file
```

change the output file name for your need.



```
Tempa@Tempay MINGW64 /c/Users/Tempa/Downloads/BRITE_JAVA/BRITE_JAVA2/BRITE/Java
$  java Main.Brite GUI_GEN.conf test_topo seed_file
[DEBUG]   : Parser found AS Waxman
[MESSAGE]: Placing 10000 nodes.
[DEBUG]   : Finished placing nodes.  G has 10000 nodes
[MESSAGE]: Connecting Nodes...
[MESSAGE]: Assigning Edge Bandwidth..
[MESSAGE]: Checking for connectivity:   Connected
[MESSAGE]: Exporting Topology in BRITE format to: test_topo.brite
[MESSAGE]: Exporting random number seeds to seedfile
[MESSAGE]: Topology Generation Complete.
```

Now you get the .brite file looks like:



```
Topology: ( 10 Nodes, 20 Edges )
Model (3 - ASWaxman):  10 100 10 1  2  0.15 0.2 1 1 10.0 1024.0

Nodes: ( 10 )
0    65   98   6    6    0    AS_NONE
1    88   7    4    4    1    AS_NONE
2    54   49   6    6    2    AS_NONE
3    15   71   5    5    3    AS_NONE
4    69   86   5    5    4    AS_NONE
5    67   84   2    2    5    AS_NONE
6    13   84   4    4    6    AS_NONE
7    68   79   4    4    7    AS_NONE
8    45   44   2    2    8    AS_NONE
9    63   61   2    2    9    AS_NONE


Edges: ( 20 )
0    2    0    50.219517   -1.0   10.0   2    0    E_AS_NONE    U
1    2    1    54.037025   -1.0   10.0   2    1    E_AS_NONE    U
2    3    2    44.777225   -1.0   10.0   3    2    E_AS_NONE    U
3    3    1    97.082436   -1.0   10.0   3    1    E_AS_NONE    U
4    4    2    39.92493    -1.0   10.0   4    2    E_AS_NONE    U
5    4    0    12.649111   -1.0   10.0   4    0    E_AS_NONE    U
6    5    0    14.142136   -1.0   10.0   5    0    E_AS_NONE    U
7    5    4    2.828427    -1.0   10.0   5    4    E_AS_NONE    U
8    6    3    13.152946   -1.0   10.0   6    3    E_AS_NONE    U
9    6    4    56.0357     -1.0   10.0   6    4    E_AS_NONE    U
10   7    3    53.600372   -1.0   10.0   7    3    E_AS_NONE    U
11   7    0    19.235384   -1.0   10.0   7    0    E_AS_NONE    U
12   8    2    10.29563    -1.0   10.0   8    2    E_AS_NONE    U
13   8    6    51.224995   -1.0   10.0   8    6    E_AS_NONE    U
14   9    2    15.0   -1.0   10.0   9    2    E_AS_NONE    U
15   9    7    18.681541   -1.0   10.0   9    7    E_AS_NONE    U
16   0    3    56.82429    -1.0   10.0   0    3    E_AS_NONE    U
17   0    6    53.851646   -1.0   10.0   0    6    E_AS_NONE    U
18   1    4    81.25269    -1.0   10.0   1    4    E_AS_NONE    U
19   1    7    74.726166   -1.0   10.0   1    7    E_AS_NONE    U
```

4.2 Java IO

The next step is to load the .brite file in our Java program.

To simplify the process, we put the .brite file under the /ShortestPathFinding/src directory (without using Java package). And under this directory, start our Java program by the following steps.

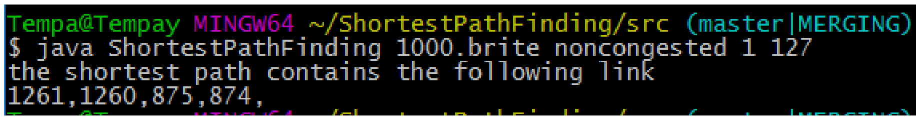
Please compile all .java file under /src before next step by javac *.java command

4.3 Non-Congestion Model

This is the model to find out shortest path without considering congestion (i.e. volume of traffic on links).

Open a new terminal and cd to /ShortestPathFinding/src type:

```
$ java ShortestPathFinding{somefile.brite} noncongested {src node} {dst node}
```



```
Tempa@Tempay MINGW64 ~/ShortestPathFinding/src (master|MERGING)
$ java ShortestPathFinding 1000.brite noncongested 1 127
the shortest path contains the following link
1261,1260,875,874,
```

Replace the file name to your own. Press Enter and the result will show in short.

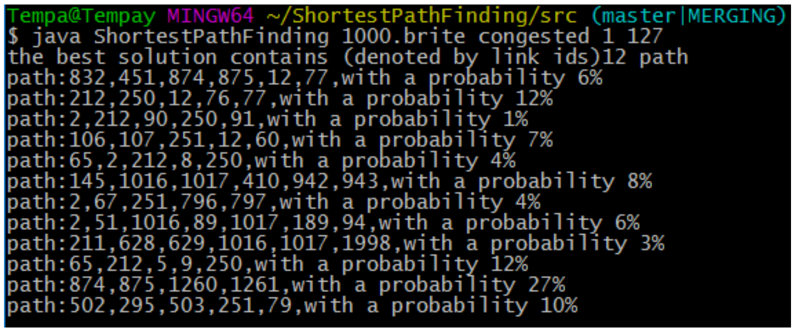
4.4 Congestion Model

In our project, we add a congestion model for shortest path routing with congestion condition (i.e., the shortest path will consider the current volume of traffic on links and the link delay will affect by the volume of traffic.)

Open a new terminal and cd to /ShortestPathFinding/src type:

```
$ java ShortestPathFinding{somefile.brite} congested {src node} {dst node}
```

Replace the file name to your own. Press Enter and the result will show in short.

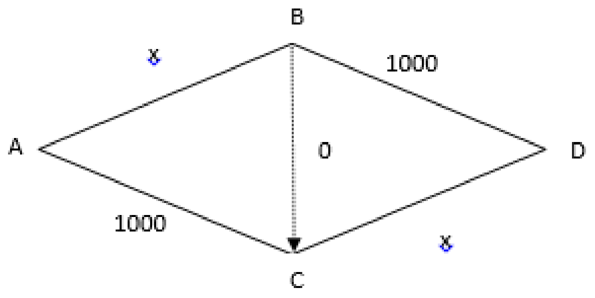


```
Tempa@Tempay MINGW64 ~/ShortestPathFinding/src (master|MERGING)
$ java ShortestPathFinding 1000.brite congested 1 127
the best solution contains (denoted by link ids)12 path
path:832,451,874,875,12,77,with a probability 6%
path:212,250,12,76,77,with a probability 12%
path:2,212,90,250,91,with a probability 1%
path:106,107,251,12,60,with a probability 7%
path:65,2,212,8,250,with a probability 4%
path:145,1016,1017,410,942,943,with a probability 8%
path:2,67,251,796,797,with a probability 4%
path:2,51,1016,89,1017,189,94,with a probability 6%
path:211,628,629,1016,1017,1998,with a probability 3%
path:65,212,5,9,250,with a probability 12%
path:874,875,1260,1261,with a probability 27%
path:502,295,503,251,79,with a probability 10%
```

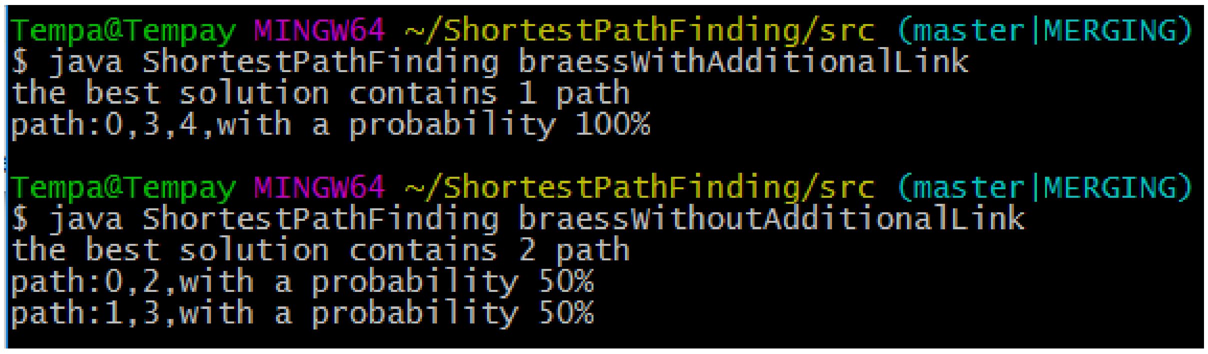
Part 5. Extension – Braess Paradox

As mentioned in the first part, we intended to find a solution to the Braess Paradox problem in future, so in this project, we applied a simple Braess paradox topo and test the result.

5.1 Braess Paradox – a simple topo



We can compare the result with additional link and without additional link.



```
Tempa@Tempay MINGW64 ~/ShortestPathFinding/src (master|MERGING)
$ java ShortestPathFinding braessWithAdditionalLink
the best solution contains 1 path
path:0,3,4,with a probability 100%

Tempa@Tempay MINGW64 ~/ShortestPathFinding/src (master|MERGING)
$ java ShortestPathFinding braessWithoutAdditionalLink
the best solution contains 2 path
path:0,2,with a probability 50%
path:1,3,with a probability 50%
```

It’s shows that with additional link, all traffic go through 1 path, which, we all know, will increasing the network delay.

In future, we can add some strategy like additional penalty for certain flow if it increasing the overall delay, that would be a solution to avoid the braess paradox phenomenon.

Part 6. Conclusion

Our project developed a model for Shortest Path Finding in a given topology within the selfish routing strategy. Which can help dealing with lots of topic we have discussed in lectures. This project also realized a Java programming that can work with any given topology generated by BRITE – A network topology generator, which enhanced our project's universality.

In our project, two different models are applied – Congestion network / Non – Congestion network. By using those two different models, we can analyze network from different angles, for some network, the increasing of traffic may not increase the overall delay significantly so we can use Non-Congestion model to analyze it, for others, the congestion model is more appropriated.

Furthermore, we applied our model to Braess Paradox problem and confirmed the additional delay that cost by opening an ideal link. All the reason is that the traffic find the Shortest Path in a selfish way, in a high level, we can say it's disorder.

Due to the limited time, we have not expand our discussion on Braess Paradox deeply, however, it could be a future direction to extend our project.