**Edge-Betweenness Analysis**

**Shortest-path graph**

[Definition of shortest path][1] Consider a directed network  with an arc length  associated with each arc . The network has a distinguished node *s*, called the *source*. The length of a directed path is defined as the sum of the lengths of arcs in the path. The shortest path problem is to determine for every non-source node  a shortest length directed path from node *s* to node *i*.

[Definition of shortest-path graph][1] In the shortest path problem, one may wish to determine a shortest path from the source node *s* to all other (n-1) nodes. Combining all the shortest paths together will form a shortest-path graph. It should be emphasized that the shortest-path graph for every source node will have the same number of nodes of the original graph if it is undirected. However, it is possible that there are some nodes unreachable for the source node *s* in a directed graph, thus the corresponding shortest-path graph may be smaller than the original graph.

[Create the shortest-path graph by depth first traversal]Original directed graph (the number inside the boxes are the weight for corresponding nodes, equal to the shortest distance to the source node, similar in all the graphs): Let the source node’s weight to be zero, while all the other node weights to be infinite, since they have not been reached yet. A stack is used to track the depth-first traversal, with the source node pushed into it.

4

6

2

2

2

1

1

3

7

0











|  |
| --- |
|  |
|  |
|  |
|  |
|  |
| 1 |

Select the top of the stack (node 1) as current node, there are two edges started from node 1 and terminated at node 2 and node 3, respectively, node 2 is with smaller index, push 2 into the stack. Since , so the weight of node 2 should be updated to 6. Shown in next picture.

4

6

2

2

2

1

1

3

7

0



6







|  |
| --- |
|  |
|  |
|  |
|  |
| 2 |
| 1 |

Select the top of the stack (node 2) as current node, there are two edges started from node 2 and terminated at node 3 and node 4, respectively, node 3 is with smaller index, push 3 into the stack. Since , so the weight of node 3 should be updated to 8. Shown in next picture.

4

6

2

2

2

1

1

3

7

0



6





8

|  |
| --- |
|  |
|  |
|  |
| 3 |
| 2 |
| 1 |

Select the top of the stack (node 3) as current node, there are two edges started from node 3 and terminated at node 4 and node 5, respectively, node 4 is with smaller index, push 4 into the stack. Since , so the weight of node 4 should be updated to 9. Shown in next picture.

4

6

2

2

2

1

1

3

7

0



6

9



8

|  |
| --- |
|  |
|  |
| 4 |
| 3 |
| 2 |
| 1 |

Select the top of the stack (node 4) as current node, there are one edge started from node 4 and terminated at node 6, push 6 into the stack. Since , so the weight of node 6 should be updated to 16. Shown in next picture.

4

6

2

2

2

1

1

3

7

0



6

9

16

8

|  |
| --- |
|  |
| 6 |
| 4 |
| 3 |
| 2 |
| 1 |

There is no new edge started from the top of the stack (node 6), pop 6 from the stack. Now the top of the stack is node 4, there are no new edges started from node 4, pop 4 from the stack. Now the top of the stack is node 3, there is a new edge (3, 5). Since , so the weight of node 5 should be updated to 10, and node 5 should be pushed into the stack. Shown in next picture.

4

6

2

2

2

1

1

3

7

0

10

6

9

16

8

|  |
| --- |
|  |
|  |
| 5 |
| 3 |
| 2 |
| 1 |

Select the top of the stack (node 5) as current node, there are one edge started from node 5 and terminated at node 6. push 6 into the stack. Since , so the weight of node 6 should be updated to 13, while 6 should be pushed into the stack. Shown in next picture.

4

6

2

2

2

1

1

3

7

0

10

6

9

13

8

|  |
| --- |
|  |
| 6 |
| 5 |
| 3 |
| 2 |
| 1 |

There is no new edge started from the top of the stack (node 6), pop 6 from the stack. Now the top of the stack is node 5, there are no new edges started from node 5, pop 5 from the stack. Now the top of the stack is node 3, there are no new edges started from node 3, pop 3 from the stack. Now the top of the stack is node 2, there is a new edge (2, 4). Since , so the weight of node 4 should be updated to 8, and node 4 should be pushed into the stack. Shown in next picture.

4

6

2

2

2

1

1

3

7

0

10

6

8

13

8

|  |
| --- |
|  |
|  |
|  |
| 4 |
| 2 |
| 1 |

Node 4 is revisited, thus all the steps later than node 4 should be repeated. Select the top of the stack (node 4) as current node, there is one edge started from node 4 and terminated at node 6, but since , so the weight of node 6 should be kept to 13, thus there is no new edge started from the top of the stack (node 4), pop 4 from the stack. Now the top of the stack is node 2, there are no new edges started from node 2, pop 2 from the stack. Now the top of the stack is node 1, there is one new edge started from node 1 and terminated at node 3, since , the weight of node 3 should be updated to 4, while 3 should be pushed into the stack. Shown in next picture.

4

6

2

2

2

1

1

3

7

0

10

6

8

13

4

|  |
| --- |
|  |
|  |
|  |
|  |
| 3 |
| 1 |

Select the top of the stack (node 3) as current node, there are two edges started from node 3 and terminated at node 4 and node 5, respectively, node 4 is with smaller index, since , push 4 into the stack and update the weight of node 4 to 5. Shown in next picture.

4

6

2

2

2

1

1

3

7

0

10

6

5

13

4

|  |
| --- |
|  |
|  |
|  |
| 4 |
| 3 |
| 1 |

Select the top of the stack (node 4) as current node, there are one edge started from node 4 and terminated at node 6. Since , so the weight of node 6 should be updated to 12, with 6 pushed into the stack. Shown in next picture.

4

6

2

2

2

1

1

3

7

0

10

6

5

12

4

|  |
| --- |
|  |
|  |
| 6 |
| 4 |
| 3 |
| 1 |

There is no new edge started from the top of the stack (node 6), pop 6 from the stack. Now the top of the stack is node 4, there are no new edges started from node 4, pop 4 from the stack. Now the top of the stack is node 3, there is a new edge started from node 3 and terminated at node 5. Since , so the weight of node 5 should be updated to 6, and node 5 should be pushed into the stack. Shown in next picture.

4

6

2

2

2

1

1

3

7

0

6

6

5

12

4

|  |
| --- |
|  |
|  |
|  |
| 5 |
| 3 |
| 1 |

Select the top of the stack (node 5) as current node, there are two new edges started from node 5 and terminated at node 4 and 6, respectively. Since , so the weight of node 4 should be kept to 5. Since , so the weight of node 6 should be updated to 9, while 6 should be pushed into the stack. Shown in next picture.

4

6

2

2

2

1

1

3

7

0

6

6

5

9

4

|  |
| --- |
|  |
|  |
| 6 |
| 5 |
| 3 |
| 1 |

There is no new edge started from the top of the stack (node 6), pop 6 from the stack. Now the top of the stack is node 5, there are no new edges started from node 5, pop 5 from the stack. Now the top of the stack is node 3, there is no new edge started from node 3, pop 3 from the stack. Now the top of the stack is node 1, there is no new edge started from node 1, pop 1 from the stack. Now the stack is empty, the algorithm is finished successfully.

The shortest-path graph of node 1 should includes all the edges (i, j) that fulfill , in which ,  and  are weights of node i, node j and edge (i, j), respectively. Thus from the figure above one can get the shortest-path graph shown as follow:

4

6

2

1

3

0

6

6

5

9

4



If node weight of all upstream notes is known, weight of the node is equals to min (node weight + path weight) for all into-node-direction paths. In the above diagram:

1. Since node-1 weight is known at the starting point, w(node-3) = 4 and w(node-2) = 6
2. Then w(node-5) = 6
3. At this point, weight of all upstream nodes (2,3,5) of node-4 are known, w(node-4) = min (7, 5, 8) = 5, path 3->4 selected
4. For node-6, w = min (9, 14) = 9. Path 5->6 selected

Assumptions:

1. No loop situation
2. No into-node-direction path connected to the starting node. If no, how to deal with the situation?

**Edge-betweenness[2]**

Please be aware that the edge-betweenness is used to measure how many shortest paths passed through a certain edge, thus all the edge-betweennesses can be calculated simultaneously by overlaying the contributions for all the shortest-path graphs (each node has its own shortest-path graph).

There are two types of shortest-path graphs, discussed separately as follow:

* Only a single shortest path from a source node (s) to all other reachable nodes

That is to say, the shortest-path graph forms a tree structure, shown in next figure.

Obviously, all the edges preceding those leave nodes in the figure above can only pass one shortest path, thus those edges can be assign a value equal to 1, shown as follow:

1

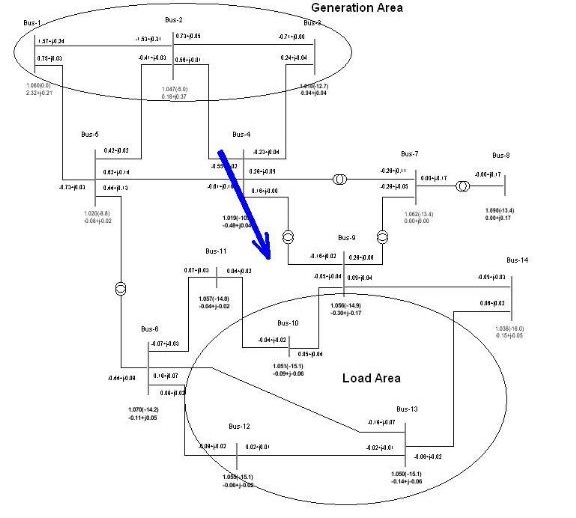
1

1

Now consider of those non-leaf nodes. For example, node E. We have already know that there is only a single shortest path from node S to node E, while the number of shortest paths passed through node E to other subsequent nodes (node F in this case) is also known, thus the number of shortest-paths through edge (B, E) is 1+1=2.

* There might be more than one shortest path to some nodes, for example, two paths happened to be with the same length and even more special, they happened to be the shortest two

It is much more complex, however, the program can also deal with such conditions by the help of ref[2].

  
--------------------------------------------------------------------------  
What I have done in pan-European network as well as the InterPSS UCTE sample case is to analyze the so-called "edge-betweenness" for all the branches within the network. The edge-betweenness is an index to describe how many shortest paths (of all possible vertex pairs) are passing current branch. Those branches with higher edge betweenness values should be more important than other branches.  
  
In AC power system, electric current (or electrical power) will be always trying to find shortest paths to go, thus the edge-betweenness index has inherent advantage to evaluate related things. However, it is also important to consider the "weight" of the branch.  
  
I have successfully write my own code to calculate such edge betweenness from Eurostag load flow results as well as InterPSS AclfNetwork object, with the help of two classical references:  
  
1. M. E. J. Newman and M. Girvan, Finding and evaluating community structure in networks, Physical Review E69, 026113 (2004)  
2. R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, Network Flows: Theory, Algorithms, and Applications, Prentice Hall, Upper Saddle River,  New Jersey (1993)  
  
Now I am analyzing the statistic properties of all the edge betweennesses for bulk power systems under both normal condition and critical condition. However, it highly depends on Tony's CPF algorithm, because right now I have no efficient tool to reach the critical point of a power system.  
  
Furthermore, the edge betweenness calculation may be easy to be "grid-enabled", because the calculation is done by:  
1. For a certain node, create its shortest-path digraph;  
2. Calculate the contribution of this shortest-path digraph to all the branches' edge-betweenness;  
3. Repeat step 1~2 for every node.  
  
You can see that the calculation of different nodes are totally independent with each other, thus it should be not too difficult to be transplanted into a grid-computing environment.

Reference

1. R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, Network Flows: Theory, Algorithms, and Applications, Prentice Hall, Upper Saddle River,  New Jersey (1993)
2. M. E. J. Newman and M. Girvan, Finding and evaluating community structure in networks, Physical Review E69, 026113 (2004)