网络的鲁棒性

几何意义

如果在移走少量节点后网络中的绝大部分接待你仍是连通的，那么就称该网络的连通性对节点故障具有鲁棒性。

量化：弹性系数

$$R_i = \frac{S_i}{S_0}, i = 1, 2, ..., n$$

其中，S_i 表示第 i 个子图的最大连通子图的节点数，S_0 为原网络图的最大连通子图的节点数。

计算方法

$R = Roustness(G(V,E),n)$
a. 对于给定的点集和边集，初始化图 $g_0 = G(V,E)$ b. 求出原图 g_0 的最大连通子图的节点数 S_0 （调用 graphx 的函数 connectedComponents[1]得到连接子图的情况） c. 对于 $i < n$ ： c.1 根据点属性（流失概率）做降序排序，选择前 $\frac{i}{n}$ 个作为将要删除的点，得到删除后的点集 c.2 根据 c.1 得到的点集，用原来的图调用 outerJoinVertices 并调用 graphx 中的 subgraph[2]函数筛选剩下的点，得到删除后的子图 g_i c.3 求子图 g_i 的最大连通子图的节点数 S_i （同 b） c.4 计算 S_i / S_0 ，存入 returnList d. 返回 returnList

[1] connectedComponents 返回 graph 对象，保留最大连通子图中最小的 VertexId 和所有边的属性
[2] subgraph 返回的是 Graph，删除顶点不存在点集的边

补充 EdgeIntersect

```
/**
* 以点集为基准，对边集取交集
* 删除顶点不存在点集的边
*
* @param vertexRDD
* @param edgeRDD
* @tparam VD
* @tparam ED
* @return
*/
def EdgeIntersect[VD, ED](vertexRDD: VertexRDD[VD], edgeRDD:
EdgeRDD[ED]): RDD[Edge[ED]] = {
val vertexTmp = vertexRDD.map(x => (x._1, 1)).distinct
val edgeDstRDD = edgeRDD.map(row => (row.dstId, row))
val edgeFilter = edgeDstRDD.leftOuterJoin(vertexTmp).filter(row =>
row._2._2 != None).map(row => row._2._1)
val edgeSrcRDD = edgeFilter.map(row => (row.srcId, row))
edgeSrcRDD.leftOuterJoin(vertexTmp).filter(row => row._2._2 !=
None).map(row => row._2._1)
}
```

主要思路

1. 将点集去重；
2. 对边集的 dst 点以点集为基准，去掉不存在点集里的边
3. 对 step2 中得到的边集的 src 点以点集为基准，去掉不存在点集里的边