实验报告

徐毅 2016213585
曹高飞 2016213617
刘昆 2016213605

一、实验目的

本次实验主要研究并实现了**B06论文[1]**中的社区发现算法，并采用了真实的数据集在分布式集群上进行了相关的实验，并实现了数据可视化。

项目源码地址：https://github.com/MyXOF/ego-net

二、实验环境

本次实验运行在三节点的**Spark[2]**集群上，原始数据存储在**HDFS[3]**上。

操作系统：Ubuntu 14.04

处理器：Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz

内存：32GB x 3

Spark：2.0.2

Hadoop：2.6

Java：JDK 1.8

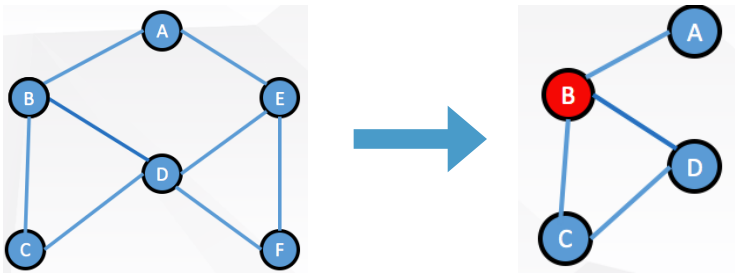
Python：3.5

数据来源：**Facebook[4]**，一共有4039个顶点，88234条边。

三、论文思路

3.1 ego-net定义

ego-net是指给定的一个图中，从一个给定的点出发，寻找和自身相关的一个子图。首先将和该点有边相连的点和相应的边加入到子图中，称这些点为“邻居点”。对于这些“邻居点”，如果这些点中之间有边相连，也把这些边加入到子图中。



在上图中，B的ego-net除了和它相连的A, C, D以外，由于C、D之间有边相连，因此，边(C, D)也在B的ego-net中。

举一个例子，在一个社交网络中，对于每一个用户而言，他所对应的ego-net图是包含所有他认识的人，以及在所有他认识人中各自相互认识的关系。

3.2 原始算法

论文算法基本的思想是，对于输入的一个图G(V, E)：

1. 找到图G中度数最小的点P，以及和它有边相邻的点的集合S，称这个集合S为邻居
2. 将S中的点和P相连的边加入到P的ego-net中
3. 对于集合S中任意不相同的两个点N1和N2，如果它们之间有边相连
    1. 将点P, N1和边(P, N1)加入到N2的ego-net中
    2. 将点P, N2和边(P, N2)加入到N1的ego-net中
    3. 将边(N1, N2)加入到P的ego-net中
4. 在图G中去掉点P和所有它的边
5. 如果G中还有点，重复步骤1，否则算法结束

Algorithm 2 Fast ego-network construction

```
Input: G(V,E)
Output: All ego-nets of nodes of G.
while V ≠ ∅ do
    u → node of minimal degree.
    for ∀v, z ∈ N(u) do
        if (v, z) ∈ E then
            Add (u, v) to S_z
            Add (v, z) to S_u
            Add (u, z) to S_v
        end if
    end for
    Delete node u and its adjacent edges.
end while
```

3.2 算法伪代码

3.3 并行算法

3.2节中描述的算法需要将整个图作为输入，每次找到度数最小的点，然后在他所有的邻居中遍历，算法流程比较简单，但是当输入的图比较大的时候，计算所花费的时间就会很长。因此提出了并行化的方式，基本思想是将图拆分成若干个子图，子图之间的可能会有边交叠，对于每个子图，用3.2节的算法计算各个点的ego-net，然后将每个点在不同子图中得到的ego-net进行合并，得到最终的ego-net。

将全图拆分成若干个子图的过程中，首先需要定义哈希函数h(t)和一个常数ρ，实验中需要预先知道图中点的个数，记为N。ρ的值为N开根号的结果向上取整，h(t) = t % ρ，那么对于每一个输入的点，h(t)将计算这个点被分到某一个区中，做好了以上的准备工作之后，算法就可以开始：

对于输入的每一条边(u, v)，u, v表示点的编号，进行以下操作：

1. 用哈希函数h(t)，计算各自的分区即 i = h(u)，j = h(v)
2. 如果 i = j
    1. 遍历集合{1,2…ρ}的数z，z != i
        1. 遍历集合{1,2…ρ}的数w，w != i, z
            1. 将i, z w从小到大排序，变成一个三元组(i', z', w')
            2. (i', z', w')作为新的子图的编号，(u, v)作为该子图的边输出
3. 如果 i != j
    1. 遍历集合{1,2…ρ}的数z，z != i, j
        1. 将i, j, z从小到大排序，变成一个三元组(i', j', z')
        2. (i', j', z')作为新的子图的编号，(u, v)作为该子图的边输出

Algorithm 3 Fast parallel ego-network construction

Map: Input: edge (u, v)
{Let $h(\cdot)$ be a universal hash function into $[0, \rho]$}
$i \leftarrow \lceil h(u) \rceil$
$j \leftarrow \lceil h(v) \rceil$
if $i == j$ **then**
  for $z \in \{1, 2, \ldots, \rho\} \wedge z \neq i$ **do**
    for $w \in \{1, 2, \ldots, \rho\} \wedge w \neq i, z$ **do**
      Output $(sorted(i, z, w), (u, v))$
    end for
  end for
else
  for $z \in \{1, 2, \ldots, \rho\} \wedge z \neq i, j$ **do**
    Output $(sorted(i, j, z), (u, v))$
  end for
end if
Reduce: Run Algorithm 2 on the input graph

3.3 算法伪代码

3.4 具体实现

实验中采用了Spark的计算框架，在3.3算法中对全图拆分之后，会生成若干个(K, V)对，K是一个三元组(i, j, w)，V是一条边(u, v)。在第一阶段会按照K值进行Reduce操作，将相同K值的合并成一个完整的子图，作为3.2算法的输入。

3.2算法中对于每个输入的子图中每一个点会计算一个自己的ego-net图，在最后好需要将这些ego-net进行一次合并，得到一个点的全局ego-net图。

论文中的主要工作在于将原有的算法并行化，但对于原始算法并没有做过多的改进。

3.5 实验结果

本次实验中主要考察了并行算法在不同的数据集上运行的效率。

顶点数	边数	输入文件	运行时间

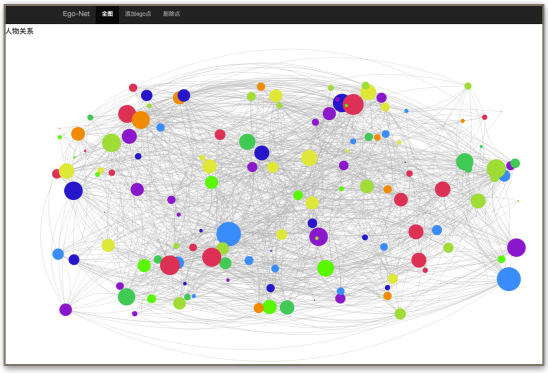
333	5038	36.4K	4s
534	9626	93.9K	5s
786	28048	273.8K	8s
747	60050	586K	22s
4039	88234	843K	84s

从实验的结果上可以看到，随着数据集的不断增大，运行时间并不是线性增加的。原因在于在第一步进行数据分区的时候，随着点数的增大，ρ的变化不是线性的，带来的结果就是分区的数量的变化不是线性的，那么在将全图拆分完成之后，子图的数量会变得很大，导致最后计算所花费的时间变得很大。

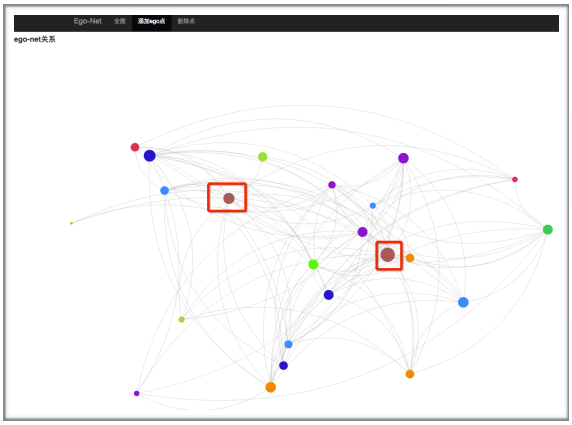
四、数据可视化

本次实验中还实现了ego-net的可视化，可以看每一个节点的ego-net以及多个节点的ego-net组合。同时还支持删除ego-net中的点和它所对应的边，看剩下的子图中各个点之间的关联。

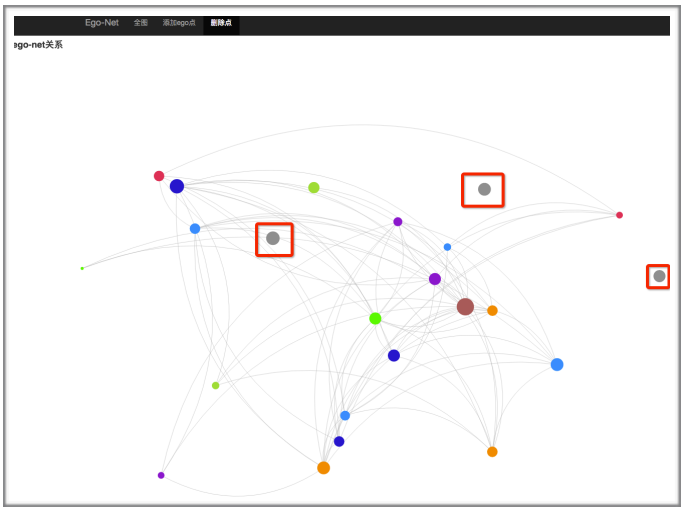
可视化过程中用到的是一个规模较小的数据集，大约1000条边，148个点。全图如下：



选取图中的两个点作为ego-net核心点(核心点用暗红色标出)并求它们的交集可以看到：



删除一些ego-net核心点和普通点(删除的点用灰色标出)可以看到：



五、总结

5.1 对于并行化计算的思考

实验中比较特殊的地方是ρ的选取，过小的ρ会导致子图数量过小，计算的并行化程度不高；ρ过大会导致子图数量过多，需要计算的中间结果变得很大，计算所花费的时间也会变大。

论文中采用了顶点数开根号的方式确定ρ值，并且采用了三元组的方式确定分区位置，这样理论上子图的数量是n^(3/2)，n是顶点数。但是如果采用二元组的方式确定ρ或者ρ是开三次方来确定，对于最后的实验结果应该没有太大的影响，主要影响的是运行时间和程序执行过程中的资源占用，运行的效果主要取决于具体的数据集。

对于一种有大量度数较低和少量度数较高的点的数据集，ρ值取小一些，尽管有很多点被划分到相同的区中，但是由于大部分点的度数较小，相对而言不会出现某些子图很大，某些很小的情况；但是对于一个各个点的度数相近的数据集，ρ去大一些就会比较好，因为这个数据切分的作用就相对明显，每个计算节点分到的数据量就会比较均匀，并行化程度就变高了。

5.2 对于推荐的一些思考

对于一个给定的点得到ego-net图，再将这个点和所相连的边去掉，这样原来的图中会构成若干个互不连通的子图，不同子图之间的点不应该相互推荐，而同一个子图内的点应该相互推荐。上述做法有点绝对化了，同一个子图内的点如果没有边相连，那么也不一定要互相推荐，应该基于两个点所对应的特征，看他们的交集。ego-net的作用应该是同一个子图内的点更需要被互相推荐；不同子图中的点之间，可能不需要互相推荐。

在实际社交网络的应用中，人们往往关注有用的推荐，对于人们不想要的推荐，关注点就没有那么强烈。所以如果要放到实际的产品应用中，可能应该把重点放在那些需要互相推荐可能性比较高的那些情况。因此，设计算法的时候还应该考虑用户的体验，有所侧重。

参考资料

[1]. Alessandro Epasto. Silvio Lattanzi. Vahab Mirrokni. Ismail Oner Sebe. Ahmed Taei. Sunita Verma. Ego-net Community Mining Applied to Friend Suggestion. 2015 VLDB.

[2]. Apache Spark. http://spark.apache.org

[3]. Apache Hadoop. http://hadoop.apache.org

[4]. ego-facebook. https://snap.stanford.edu/data/egonets-Facebook.html