
金庸的江湖*

冷冲 131220043, 王雨阳 131220047, 黄玟钦 131220045 指导老师: 黄宜华

Jinyong's Society*

Abstract: The greatest swordsman novel writer Jinyong has created lots of well-known heroes for us, but we seldom know how his whole society builds and what are the relations with many different characters. In this survey, we collect Jinyong's famous 14 novels to mine the laws in them. We use Mapreduce as the main tool to do the practical jobs for our survey. At last, we use gephi to make the data and relations more compatible to be viewed in eyes. We benefit a lot from this experiment and we actually dig out many practical and contributinal laws.

Key words: martial art; Jinyong; Mapreduce; LPA; pagerank

摘要: 著名的武侠小说作家金庸先生为我们塑造了无数知名的英雄形象,但是我们很少知晓他的整个武侠社会是如何构成以及那些不同角色之间的关系是怎么样搭建的。在这个实验当中,我们收集了金庸先生的14部著名小说来挖掘其中的规律。我们使用 Mapreduce 作为主要工具来完成实验的具体工作。最后,我们使用 gephi 来使得数据变得可视化友好。在实验中我们收获了很多同时也确实找出了一系列实际的而有建设性的规律。

关键词: 武侠, 金庸, Mapreduce, LPA, pagerank

1 实验目的

对金庸先生的14部知名小说的人物进行数据的整理挖掘,利用 Mapreduce 以及 LPA, pagerank 等算法对实际问题进行处理,从而挖掘出金庸的“江湖网”中的数据状况与相应的人物规律,从中得到一些经验和规律,将可视化结果展示出来。

2 实验分工

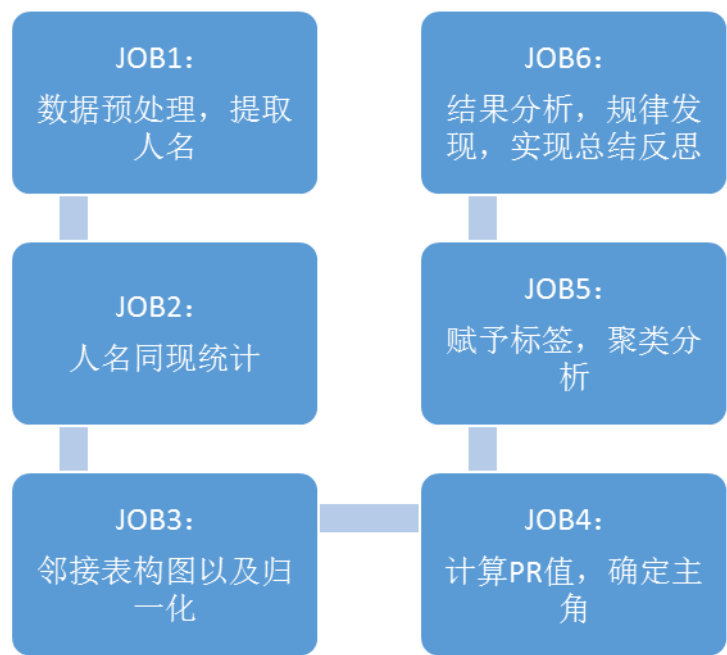
王雨阳: 主要编写代码和算法问题的优化处理

冷冲(组长): 开题报告、gephi 绘图、结果分析以及部分实验报告编写

黄玟钦: 部分数据优化处理和部分实验报告编写

3 实验过程

3.1 实验的整体设计流程：



整个实验的实现过程，大部分根据实验任务的介绍以及开题报告的流程进行，大概就是根据上面六个步骤按部就班的进行实验，对各个步骤里面的算法和细节进行了一些优化和处理，对实验过程中的中间结果以及实验最后的阶段性成果我们有自己创造性的认识和思考，总的来说，实验的整个过程比较顺利也让我们受益匪浅。

3.2 JOB1--数据预处理

本任务中，Map 阶段输出键值对为<人名#……人名#，#>，Reduce 阶段不工作，输出不变。

从原始的未分段的金庸小说，透过给定的金庸武侠小说人名表进行分词。使用 Ansj_seg 工具，经过分词，判断是否为人名表中的词，若被分出来的词属于人名列表中的名字，则将其提取出来。在任务一中 Map 部分主要用来提取所有出现过的人名，而 reduce 部分不工作。

在本任务中我们使用了全局文件复制方法来实现 Map 端连接，因为人物名表的数据源文件数据量较小、能够存放在单个节点的内存中，我们使用全局文件复制方法，把这个人物名列表复制到每个 Map 节点上。

```

public void setup(Context context) {
    try {
        @SuppressWarnings("deprecation")
        Path[] cacheFiles=context.getLocalCacheFiles();
        String line;
        if (cacheFiles!=null&&cacheFiles.length>0){
            BufferedReader dataReader=new BufferedReader
                (new FileReader(cacheFiles[0].toString()));
            try{
                while((line=dataReader.readLine())!=null){
                    UserDefineLibrary.insertWord(line, "name", 1000);
                }
            } finally{
                dataReader.close();
            }
        }
    } catch (IOException e){
        System.err.println("Exception reading DistributedCache: "+e);
    }
}

```

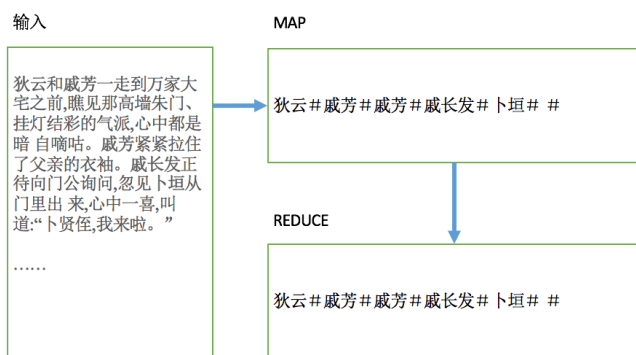
另外由于 Ansj_seg 工具提供了可自定义的词典进行分词，我们把给定的金庸人物名列表作为分词工具的自定义词典，并设置为最高优先级。

```

public void map(Object key,Text value, Context context)
    throws IOException, InterruptedException{
    List<Term> parse = FilterModifWord.modifResult(ToAnalysis.parse(value.toString()));
    StringBuilder str=new StringBuilder();
    for (Term term:parse){
        if (term.getNatureStr().equals("name")){
            str.append(term.getName());
            str.append("#");
        }
    }
    if (str.length(>0))
        context.write(new Text(str.toString()),new Text("#"));
}

```

具体的实现样例如下所示：



3.3 JOB2--特征抽取：人物同现统计

本任务中，Map 阶段输出键值对为<人名#人名，1>，Reduce 阶段输出键值对<人名#人名，共现次数>。

使用的算法是在单词共现的基础上进行修改，是基于单词共现算法统计在同一段落中，两个名字都出现的次数，并进行统计。将任务一中透过 `AnsJ_seg` 工具进行分词后的结果作为输入，并把出现在同一段落中的名字进行统计输出。这里我们利用 `MapReduce` 实现了单词共现算法中的一中称作 `pairs` 的算法，先是利用 `MAP` 端对共现的词进行提取，将每个出现过的计为 1，将输出结果进行分类，使用 `REDUCE` 端进行结合，将相同的 `pair` 出现的进行累加，累加后所得到的输出即为任务二的输出。由于这里所要进行操作的不是基本数据类型，我们定义了一个自定义 `WordPair` 类，该类实现了 `WritableComparable` 接口，另外我们充血自定义类的 `hashCode()` 方法，使得相同的 `WordPair` 主键都被发送到相同的 `Reduce` 节点去。

具体的实现如下所示：

`Map` 部分将出现在该行中的人名两两进行组合，并将每一个的数量计数为 1：

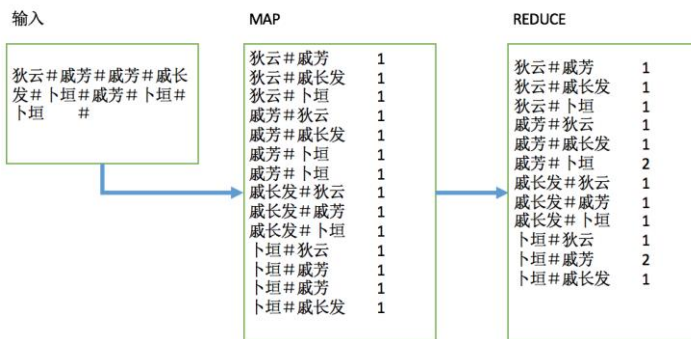
```
public void map(Object key,Text value, Context context)
    throws IOException, InterruptedException{
    String[] names=value.toString().split("#|\\t");
    Set<String> set=new HashSet<>();
    for (String name:names) set.add(name);
    for (String name1:set){
        for (String name2:set){
            if (!name1.equals(name2))
                context.write(new Text(name1+"#"+name2),new IntWritable(1));
        }
    }
}
```

`Reduce` 部分我们做了些许优化，重定义了 `combiner` 的方法，将 `Combiner` 定义为将 `Map` 工作完得到的当前段落的相同的 `key` 值部分进行 `value` 值的求和。而 `Reduce` 则进行对所有段落中出现的相同 `Key` 值的元素进行求和：

```
public static class Job2Combiner extends Reducer<Text,IntWritable,Text,IntWritable>{
    public void reduce(Text key,Iterable<IntWritable> values,Context context)
        throws IOException, InterruptedException{
        int sum=0;
        for (IntWritable value:values){
            sum+=value.get();
        }
        context.write(key, new IntWritable(sum));
    }
}

public static class Job2Reducer extends Reducer<Text,IntWritable,Text,IntWritable>{
    public void reduce(Text key,Iterable<IntWritable> values,Context context)
        throws IOException, InterruptedException{
        int sum=0;
        for (IntWritable value:values){
            sum+=value.get();
        }
        context.write(key, new IntWritable(sum));
    }
}
```

具体的实现例子如下所示：



3.4 JOB3—人物关系图构建与特征归一化

本任务中，Map 阶段输出键值对为<人名，人名#共现次数>，Reduce 阶段输出键值对<人名，|人名，权重……|人名，权重>。

当获取了人物之间的共现关系之后,我们根据共现关系,生成人物之间的关系图。人物关系图使用邻接表的形式表示,方便后面的 PageRank 计算。在人物关系图中,人物是顶点,人物之间的 互动关系是边。人物之间的互动关系靠人物之间的共现关系确定。如果两个人之间具有共现关系,则 两个人之间就具有一条边。两人之间的共现次数体现出两人关系的密切程度,反映到共现关系图上就 是边的权重。边的权重越高则体现了两个人的关系越密切。

以任务二的输出，将统计出的共现次数结果，转换成临接表的形式表示，按照实验要求，每一行一个临接关系。根据每个人对应的不同的边，计算出每个人物与该行的节点人物共现的概率，通过归一化，确保定点的出边权重和为 1。

Map 部分我们将任务二输出的人物同现统计进行拆解，将原先的 key 中后一个人物的名字与共现次数作为新的 value 值，而原来的 key 值中的第一个人物名字作为当前 Map 所需要的 key 值。具体实现的代码如下：

```
public void map(Object key,Text value, Context context)
    throws IOException, InterruptedException{
    String[] words=value.toString().split("#|\\t");
    context.write(new Text(words[0]), new Text(words[1]+"#"+words[2]));
}
```

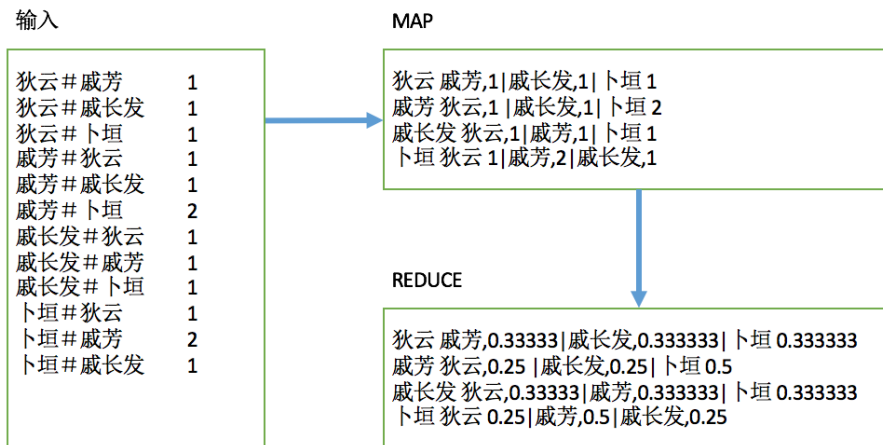
而在与 Reduce 部分我们将 Map 的零散 value 进行整合，将几个 value 值整合为同一个 key 下的 value 属性，并按照实验讲义进行计算，将值归一化。将 value 值中的几个共现人物次数求和并作为分母一一相除，得到共现概率并置为 value 值。

```

public void reduce(Text key,Iterable<Text> values,Context context)
    throws IOException, InterruptedException{
    List<String> sList=new ArrayList<>();
    List<Integer> iList=new ArrayList<>();
    StringBuilder result=new StringBuilder();
    int sum=0;
    for (Text value:values){
        String[] words=value.toString().split("#");
        sList.add(words[0]);
        int temp=Integer.parseInt(words[1]);
        iList.add(temp);
        sum+=temp;
    }
    for (int i=0;i<sList.size();i++){
        double p=(double)iList.get(i)/(double)sum;
        result.append("|"+sList.get(i)+","+p);
    }
    context.write(key, new Text(result.toString()));
}

```

具体的过程例子如下所示：



3.5 JOB4—数据分析：基于人物关系图的Pagerank值计算

本任务中，Map 阶段输出键值对为<人名，人名#部分 PageRank 值>以及<人名，邻接表>，Reduce 阶段输出键值对<人名#PageRank 值，邻接表 >。

我们在任务四中，我们使用了 PageRank 的随机浏览模型来完成。这个任务我们主要是用来计算 PageRank 值，来确定几部小说中的主角。使用迭代的方法来计算 PageRank 值，直到满足运算结束条件，我们在这边设计最大迭代次数为 15 次，迭代停止条件为当两次迭代的结果中，PageRank 的最大差值小于 0.05 时，即不再进行下一轮的迭代。设置条件如下所示：

```

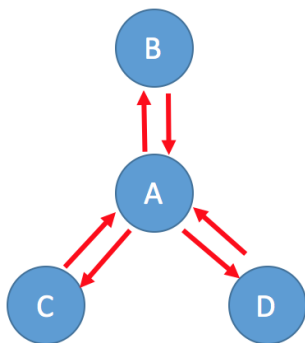
public static boolean judge(String filePath1,String filePath2)
    throws Exception{
    List<Double> list1,list2;
    list1=getList(filePath1);
    list2=getList(filePath2);
    int size=list1.size();
    double max=Double.NEGATIVE_INFINITY;
    for (int i=0;i<size;i++){
        double diff=Math.abs(list1.get(i)-list2.get(i));
        if (diff>max) max=diff;
    }
    if (max<0.05) return true;
    else return false;
}

```

在 PageRank 简化模型中，会出现排名泄漏，排名下沉的问题，所以我们选择了 PageRank 的随机浏览模型，这种模型更加接近于真实。在原型中，为了处理那些没有任何出度连接的页面，引入了阻尼系数 d 来表示拥护到达某页面后继续向后浏览的概率。我们在实验中将阻尼系数 d 设置为 0.85，PageRank 的随机浏览模型公式如下：

$$PR(u) = 1 - d + d \sum_{v \in B_u} \frac{PR(v)}{L(v)}$$

其中 B_u 是所有连接到人物 u 的人物集合，人物 v 是属于集合 B_u 的一个人物。 $L(v)$ 则是人物 v 对外连接数（即出度）。



A B,0.2|C,0.3|D,0.5
 B A,0.2|.....
 C A,0.6|.....
 D A,0.7|.....

当要计算A的PageRank值时：

将原来的邻接图转化为：
 <B,A 0.2>,<C,A 0.3>,<D,A 0.5>
 <A,B 0.2>,<A,C 0.6>,<A,D 0.7>
 的键值对传递给Reduce

Reduce将得到的值透过公式：
 $PR(A)=1-d+d(0.2PR(B)+0.6PR(C)+0.7PR(D))$

在 Map 部分中，根据 PageRank 的原理，我们需要计算出当前节点的出边连接到的所有节点中的出度以及其 PageRank 值。所以在 Map 部分，我们拆开任务三中归一化的输出中的 value 中的值，将几个人名全部拆开，并把 value 中的人名做为 key 值，并将原先的 key 值与权值整合为 value 值传递给 Reduce 节点，这样做以便于 Reduce 节点可以直接做求和的操作来完成当前点的 PageRank 的计算求值。具体计算方法如上图所示，实现方法如下所示：

```

public static class PageRankMapper extends Mapper<Object,Text,Text,Text>{
    public void map(Object key,Text value, Context context)
        throws IOException, InterruptedException{
        String[] tuple=value.toString().split("\t");
        String[] page=tuple[0].split("#");
        String pageKey=page[0];
        double pr=1;
        if (page.length==2)
            pr=Double.parseDouble(page[1]);
        String[] linkPages=tuple[1].split("\\|");
        for (String linkPage:linkPages){
            if (linkPage.length()>0){
                String[] infos=linkPage.split(",");
                String linkPageKey=infos[0];
                String prValue=pageKey+"#"+String.valueOf(pr*Double.parseDouble(infos[1]));
                context.write(new Text(linkPageKey), new Text(prValue));
            }
        }
        context.write(new Text(pageKey), new Text(tuple[1]));
    }
}
}

```

Reduce 部分负责收集零散的几个 PageRank 和 Value 值并进行计算，按照上述公式求和并得到当前节点的 PageRank 值。将当前的信息继续传递给下一轮的迭代计算，直到满足迭代条件才会结束该迭代工作。具体实现代码如下：

```

private static double d=0.85;
public void reduce(Text key,Iterable<Text> values,Context context)
    throws IOException, InterruptedException{
    String links="";
    double pagerank=0;
    for (Text value:values){
        String tmp=value.toString();
        if (tmp.startsWith("|")){
            links=tmp;
            continue;
        }
        String[] tuple=tmp.split("#");
        if (tuple.length>1)
            pagerank+=Double.parseDouble(tuple[1]);
    }
    pagerank=1.0-d+d*pagerank;
    context.write(new Text(key.toString()+"#"+String.valueOf(pagerank)),new Text(links));
}
}

```

在迭代计算完成后，我们使用了 PageRankViewr 来得到最终结果的排名，按照 PageRank 的值从大到小进行输出。这里我们主要使用到 Map 节点来完成这项任务，而不需要使用到 Reduce。我们将得到的 PageRank 值进行拆分，并将 Key 和 Value 倒过来，即 Key 为原来的 Value 而 Value 为原来的 Key，交换 Key 和 Value 即可交给节点自动排序。

其中 Map 的实现如下：

```

public void map(Object key,Text value, Context context)
    throws IOException, InterruptedException{
    String[] infos=value.toString().split("\t")[0].split("#");
    String page=infos[0];
    double pr=Double.parseDouble(infos[1]);
    context.write( new DoubleWritable(pr),new Text(page));
}
}

```

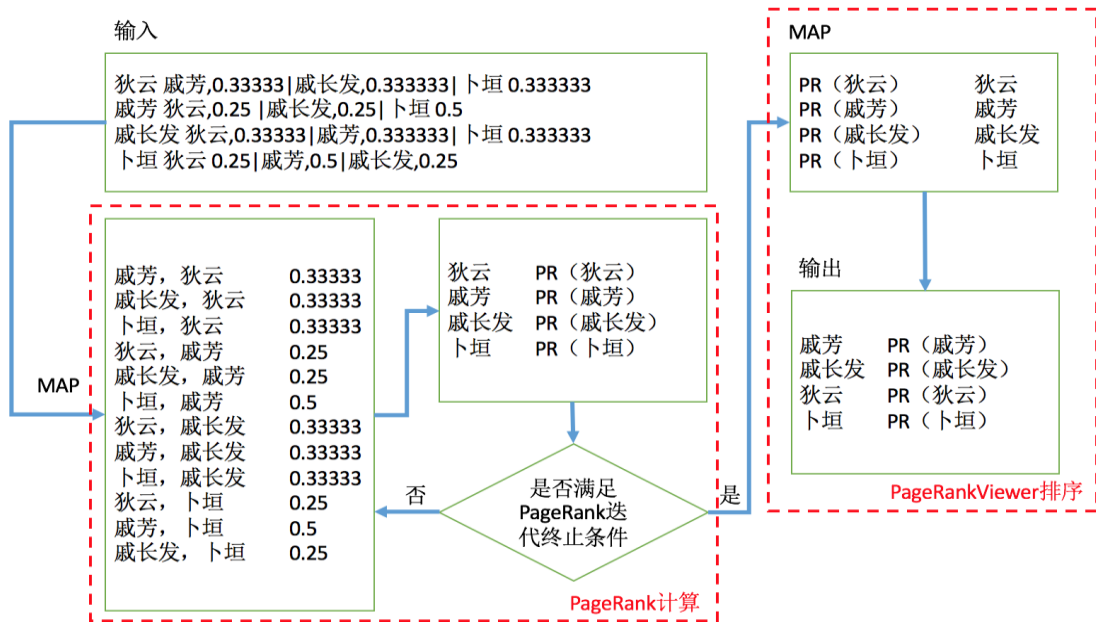
这里我们还对排序算法进行了定义，在 main 函数中加入了如下语句来完成这项任务的 PageRank 值的降序排序。


```

job.setOutputValueClass(Text.class);
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
job.setSortComparatorClass(DoubleWritableDecreasingComparator.class);
//job.waitForCompletion(true);
System.exit(job.waitForCompletion(true)?0:1);

```

任务四的具体过程如下所示：



3.6 JOB5—数据分析：在人物关系图上的标签传播

本任务中，Map 阶段输出键值对为<人名 1，人名 2#权重（人名 2→人名 1）#label（人名 2）>以及<人名 1，人名 2#权重（人名 1→人名 2）>以及<人名，邻接表>，Reduce 阶段输出键值对<人名#label，邻接表>。

标签传播算法是不重叠社区发现的经典算法，其基本思想是：将一个节点的邻居节点的标签中数量最多的标签作为该节点自身的标签。给每个节点添加标签以代表它所属的社区，并通过标签的“传播”形成同一标签的“社区”结构。给每个节点添加标签以代表它所属的社区，并通过标签的“传播”形成同一标签的“社区”结构。一个节点的标签取决于它邻居节点的标签：假设节点z的邻居节点有z1至zk，那么哪个社区包含z的邻居节点最多z就属于那个社区（或者说z的邻居中包含哪个社区的标签最多，z就属于哪个社区）。优点是收敛周期短，无需任何先验参数(不需事先指定社区个数和大小)，算法执行过程中不需要计算任何社区指标。具体的过程则是按照如下进行的：

- 1)初始时，给每个节点一个唯一的标签
- 2)每个节点使用其邻居节点的标签中最多的标签来更新自身的标签

3)反复执行步骤2)，直到每个节点的标签都不再发生变化为止

一次迭代过程中一个节点标签的更新可以分为同步和异步两种。所谓同步更新，即节点 z 在第 t 次迭代的 label 依据于它的邻居节点在第 $t-1$ 次迭代时所得的 label；异步更新，即节点 z 在第 t 次迭代的 label 依据于第 t 次迭代已经更新过 label 的节点和第 t 次迭代未更新过 label 的节点在第 $t-1$ 次迭代时的 label。

由于 Hadoop 的系统不支持使用当前正在计算中的结果，所以我们选择了异步更新的方法来实现标签传播。另外在算法上我们进行了改进，为了使结果更贴近于真实的情况，我们在计算的过程中，使用到了出边边的权重，每次判断都是以相同 Label 的边的权重进行求和，再判断哪个 Label 的和比较大，来判断当前点的 Label 值。这一部分中，我们将迭代终止条件设置为执行 15 次迭代或者当在迭代的过程中，两次的结果标签不变时，则终止迭代过程。

与 PageRank 算法中的 Map 过程相同，需要将任务三输出的原格式进行拆分，将当前行中 Key 和 Value 进行拆分，同样需要将 Value 中的几个人名进行拆分做为 Key 值，将原本的 Key 值和权值和 Label 值作为 Value 值传递给 Reduce 节点进行计算。在第一次进行计算，即还没有 Label 值时，我们默认将其 Label 值设定为其本身。实现的代码如下所示：

```
public void map(Object key, Text value, Context context)
    throws IOException, InterruptedException{
    String[] tuple=value.toString().split("\t");
    String[] page=tuple[0].split("#");
    String pageKey=page[0];
    String label=page[0];
    if (page.length==2)
        label=page[1];
    String[] linkPages=tuple[1].split("\\|");
    for (String linkPage:linkPages){
        if (linkPage.length()>0){
            String[] infos=linkPage.split(",");
            String linkPageKey=infos[0];
            String weight=infos[1];
            context.write(new Text(pageKey), new Text(linkPageKey+"#"+weight));
            context.write(new Text(linkPageKey), new Text(pageKey+"#"+weight+"#"+label));
        }
    }
    context.write(new Text(pageKey), new Text(tuple[1]));
}
```

在 Reduce 部分中，其节点负责用于整合各部分的 Map 节点的信息，并进行计算，将当前点的出边权值进行求和计算。当 Reduce 节点中当前点得到的权值和的差值小于 0.00000001 时，我们默认为这些权值相同，则使用随机的办法获取任意一个相邻节点的 Label 值作为其 Label 值。实现的代码如下：

```

public void reduce(Text key,Iterable<Text> values,Context context)
    throws IOException, InterruptedException{
    Map<String,Double> map=new HashMap<>();
    Map<String,String> labelMap=new HashMap<>();
    List<String> two=new ArrayList<>();
    String links="";
    for (Text value:values){
        String tmp=value.toString();
        if (tmp.startsWith("#")){
            links=tmp;
            continue;
        }
        String[] tuple=tmp.split("#");
        if (tuple.length>2) {
            labelMap.put(tuple[0],tuple[2]);
        }
        else {
            two.add(tmp);
        }
    }
    for (String str:two){
        String[] tuple=str.split("#");
        String pageKey=tuple[0];
        double weight=Double.parseDouble(tuple[1]);
        String label=labelMap.get(pageKey);
        if (map.containsKey(label)){
            double temp=map.get(label);
            temp+=weight;
            map.put(label,temp);
        }
        else
            map.put(label, weight);
    }
    Map<String,Double> mapSort=new TreeMap<>(map);
    List<Map.Entry<String,Double>> list = new ArrayList<Map.Entry<String,Double>>(mapSort.entrySet());
    Collections.sort(list,new Comparator<Map.Entry<String,Double>>() {
        public int compare(Entry<String, Double> o1,
            Entry<String, Double> o2) {
            return o2.getValue().compareTo(o1.getValue());
        }
    });
    int m=1;
    while (m<list.size()){
        if (Math.abs(list.get(m).getValue()-list.get(0).getValue())<0.00000001){
            m++;
        }
        else break;
    }
    Random random=new Random();
    int index=random.nextInt(m);
    String label=list.get(index).getKey();
    context.write(new Text(key.toString()+"#"+label),new Text(links));
}

```

在迭代完成后我们与 PageRank 算法相同，还是使用了一个排序算法，按照标签进行排序，将标签相同的聚集在一起进行输出，具体算法如下：

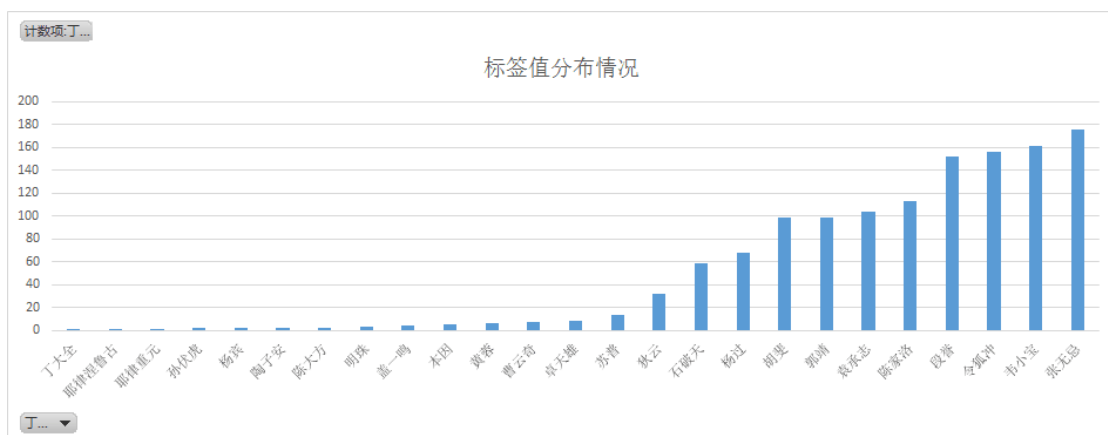
```

public static class LPAViewerMapper extends Mapper<Object,Text,Text,Text>{
    public void map(Object key,Text value, Context context)
        throws IOException, InterruptedException{
        String[] infos=value.toString().split("\t")[0].split("#");
        context.write(new Text(infos[1]),new Text(infos[0]));
    }
}

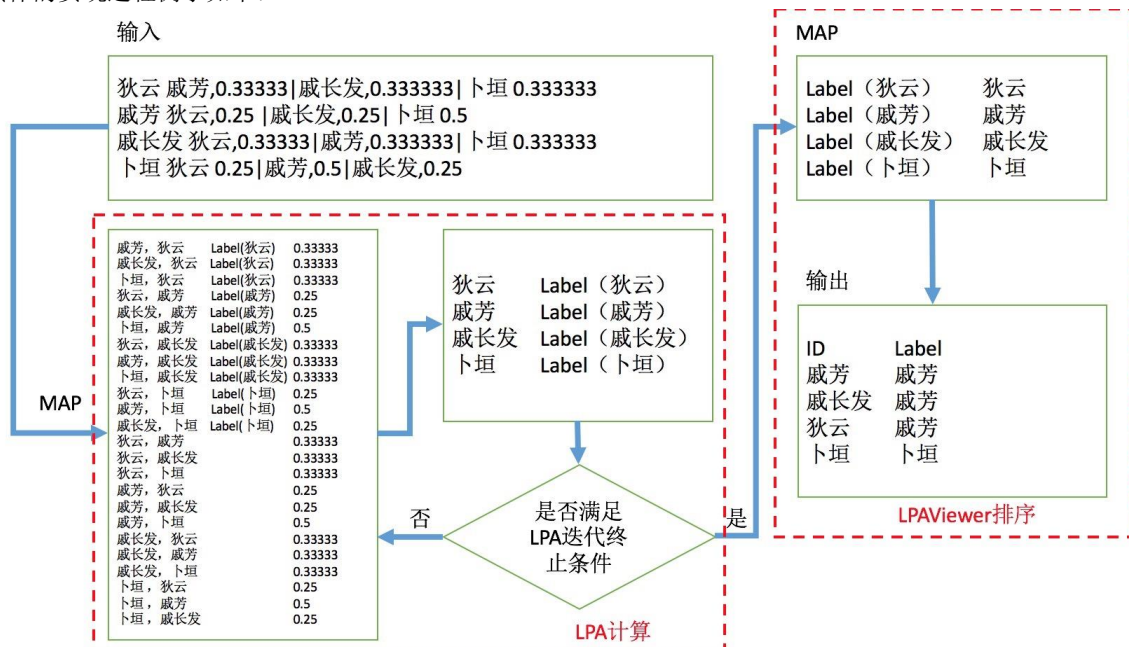
public static class LPAViewerReducer extends Reducer<Text,Text,Text,Text>{
    public void reduce(Text key,Iterable<Text> values,Context context)
        throws IOException, InterruptedException{
        for (Text value:values)
            context.write(key,value);
    }
}

```

由于在我们的实验过程中，我们使用的是同步的算法，存在二分图会造成结果的震荡，所以我们所设立第二种迭代终止条件并不能停止，最终我们在迭代 15 次后得到了一个最重的标签传播结果，其标签纸分布大致情况如下：



具体的实现过程例子如下：



4 实验中的主要优化

① Pr 值 pagerank 的迭代次数的确定以减少算法执行时间

算法思想:

对于迭代次数的确定，我们就是在程序中设定一个 `flag` 值，这个 `flag` 值是每两次邻近的

pagerank 迭代之后最大的 pagerank 值差值的阈值。我们设定这个值为 0.05，具体来说就是当某两次迭代的时候后一次的 pagerank 值与前一次不一样的 pagerank 值的差值最大值也不超过 0.05 分时候，算法就停止迭代，我们确认迭代结果已经收敛，从而得到 pagerank 值。

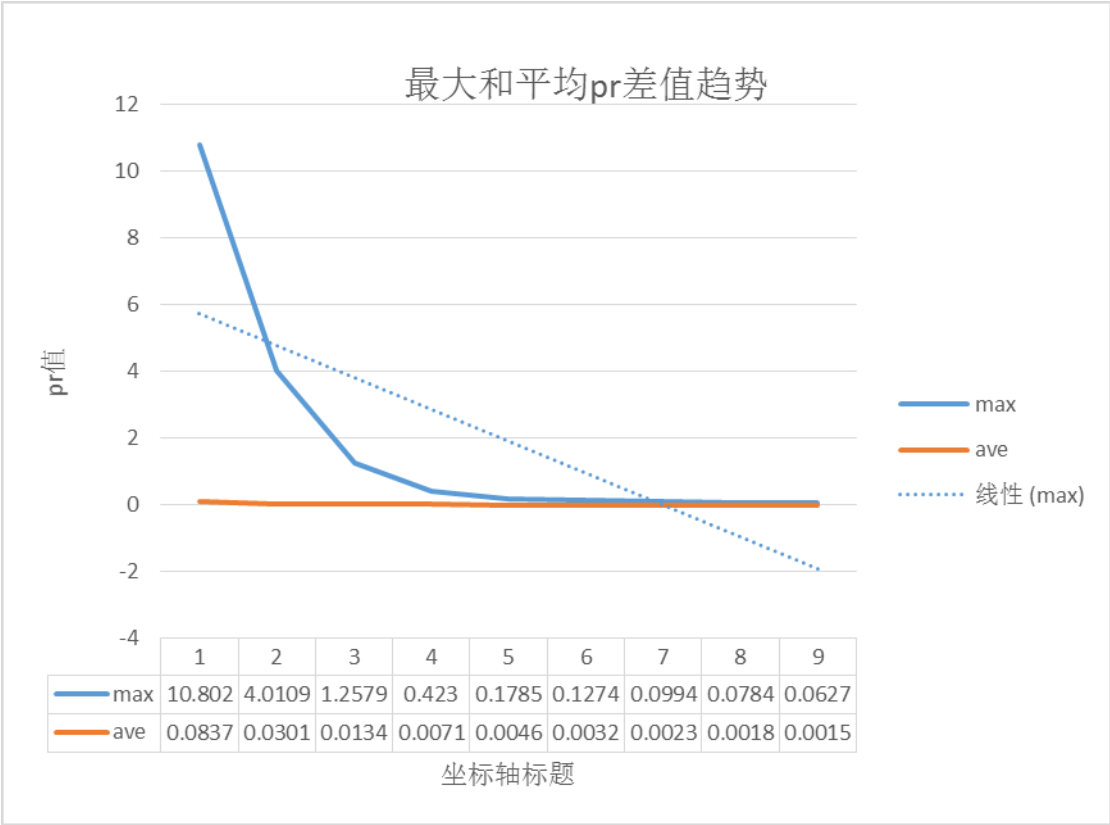
算法的具体实现：

```
public static boolean judge(String filePath1,String filePath2)
    throws Exception{
    List<Double> list1,list2;
    list1=getList(filePath1);
    list2=getList(filePath2);
    int size=list1.size();
    double max=Double.NEGATIVE_INFINITY;
    for (int i=0;i<size;i++){
        double diff=Math.abs(list1.get(i)-list2.get(i));
        if (diff>max) max=diff;
    }
    if (max<0.05) return true;//判断pagerank停止
    else return false;
}
```

在 pagerankdriver 之中设定阈值 0.05 以判断算法是否停止。

收敛性的展示：

我们根据每次迭代后 pr 值的最大差值已经平均差值的变化做出了变化图像如下：
的



图中 max 值为每次迭代后的 pr 值最大差值，ave 是 pr 值平均差值，图像显示为明显的收敛趋势，证明了迭代次数的控制恰当。

② Job2 之中 combiner 的优化操作

算法思想：

Job1 输出的 key 值每一段中出现的已经分开的人名列表，value 值就是一个#号，job2 读取 job1 的中间结果进行操作，经过 map 阶段的操作 key 值变为同现的两个人名，value 值为 1，在进行 reduce 的统计之前，先用一个 combiner 对同一个 map 节点的相同 key 值进行合并，以对运算速度和带宽进行优化，减少不必要的操作。

算法实现：

```

    public static class Job2Combiner extends Reducer<Text,IntWritable,Text,IntWritable>{
        public void reduce(Text key,Iterable<IntWritable> values,Context context)
            throws IOException, InterruptedException{
            int sum=0;
            for (IntWritable value:values){
                sum+=value.get();//combiner提前合并
            }
            context.write(key, new IntWritable(sum));
        }
    }

    public static class Job2Reducer extends Reducer<Text,IntWritable,Text,IntWritable>{
        public void reduce(Text key,Iterable<IntWritable> values,Context context)
            throws IOException, InterruptedException{
            int sum=0;
            for (IntWritable value:values){
                sum+=value.get();
            }
            context.write(key, new IntWritable(sum));
        }
    }
}

```

如图就是 combiner 的优化。优点就是在做统计的时候，大量具有相同的主键的键值对数据如果之间传送到 Reduce 节点会引起较大的网络带宽开销。可以对每一个 Map 节点处理完成的中间键值对做一个合并压缩，把那些主键相同的键值对归并为一个键名下的一组数值。这样做不仅可以减轻网络的压力，也可以大幅度提高程序的运行效率。

③ 标签传播算法迭代次数的确定

算法思想：

对每次标签传播算法过后相邻的结果进行比较，确定不一样的标签的个数，当不一样的标签的个数很少并且趋于稳定的时候我们就判断标签传播算法结果收敛。

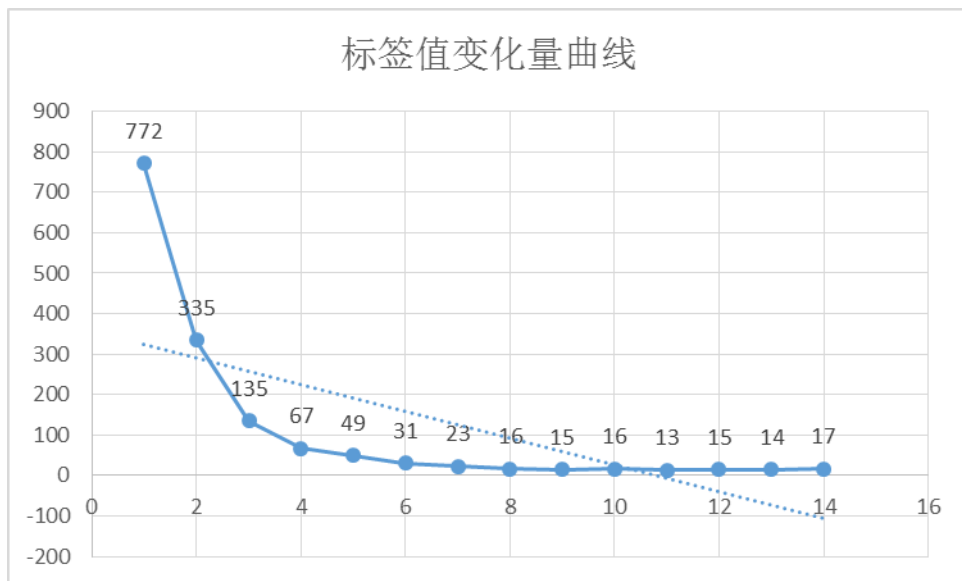
算法实现以及结果显示：

```

public static class LPAReducer extends Reducer<Text,Text,Text,Text>{
    public void reduce(Text key,Iterable<Text> values,Context context)
        throws IOException, InterruptedException{
        Map<String,Double> map=new HashMap<>();
        Map<String,String> labelMap=new HashMap<>();
        List<String> two=new ArrayList<>();
        String links="";
        for (Text value:values){
            String tmp=value.toString();
            if (tmp.startsWith("|")){
                links=tmp;
                continue;
            }
            String[] tuple=tmp.split("#");
            if (tuple.length>2) {
                labelMap.put(tuple[0],tuple[2]);
            }
            else {
                two.add(tmp);
            }
        }
        for (String str:two){
            String[] tuple=str.split("#");
            String pageKey=tuple[0];
            double weight=Double.parseDouble(tuple[1]);
            String label=labelMap.get(pageKey);
            if (map.containsKey(label)){
                double temp=map.get(label);
                temp+=weight;
                map.put(label,temp);
            }
            else
                map.put(label, weight);
        }
        Map<String,Double> mapSort=new TreeMap<>(map);

```

对于输出的结果我们画出了他的趋势图，如下所示：



如图所示，不一样的标签个数一开始非常多，经过几次迭代之后，数量明显减少，一直到15,16次附近之后，出现的个数在15个左右稳定，进行左右震荡，表明标签传播算法已经收敛，再进行迭代已经没有意义，在此处停止就好。

5 集群运行状况分析展示以及复杂度分析

5.1 集群运行记录截图

JOB1, JOB2, JOB3 整体:

application_1409060011991_0078	2016r03	Job3	MAPREDUCE	root.default	Thu Jul 21 10:37:39 +0800 2016	Thu Jul 21 10:38:00 +0800 2016	FINISHED	SUCCEEDED	History
application_1409060011991_0077	2016r03	Job2	MAPREDUCE	root.default	Thu Jul 21 10:37:16 +0800 2016	Thu Jul 21 10:37:37 +0800 2016	FINISHED	SUCCEEDED	History
application_1409060011991_0076	2016r03	Job1	MAPREDUCE	root.default	Thu Jul 21 10:36:38 +0800 2016	Thu Jul 21 10:37:13 +0800 2016	FINISHED	SUCCEEDED	History

JOB1 具体:

Counter Group	Name	Map	Reduce	Total
File System Counters	FILE: Number of bytes read	0	1,730,498	1,730,498
	FILE: Number of bytes written	5,476,707	0	5,476,707
	FILE: Number of large read operations	0	0	0
	FILE: Number of read operations	0	0	0
	FILE: Number of write operations	0	0	0
	HDFS: Number of bytes read	25,730,180	0	25,730,180
	HDFS: Number of bytes written	0	1,649,067	1,649,067
	HDFS: Number of large read operations	0	0	0
Job Counters	HDFS: Number of read operations	48	3	51
	HDFS: Number of write operations	0	2	2
	Map-Local map tasks	0	0	0
	Launched map tasks	0	0	0
	Launched reduce tasks	0	0	0
	Spilled-Map records	0	0	0
	Total mapreduce-records taken by all map tasks	0	0	0
	Total mapreduce-records taken by all reduce tasks	0	0	0
	Total time spent by all map tasks (ms)	0	0	0
	Total time spent by all map tasks in sorted splits (ms)	0	0	0
Map-Reduce Framework	Total time spent by all reduce tasks (ms)	0	0	0
	Total time spent by all reduce tasks in sorted splits (ms)	0	0	0
	Total record-records taken by all map tasks	0	0	0
	Total record-records taken by all reduce tasks	0	0	0
	Combine input records	0	0	0
	Combine output records	0	0	0
	Copy time spent (ms)	551,710	3,820	555,530
	Failed Shuffles	0	0	0
	GC time elapsed (ms)	63,363	49	63,412
	Input split bytes	1,914	0	1,914
	Map input records	47,488	0	47,488
	Map output bytes	1,651,604	0	1,651,604
	Map output materialized bytes	1,730,582	0	1,730,582
	Map output records	38,156	0	38,156
Shuffle Errors	Merged Map outputs	0	15	15
	Physical memory (bytes) snapshot	4,694,069,504	173,922,944	4,867,992,448
	Reduce input groups	0	20,996	20,996
	Reduce input records	0	38,156	38,156
	Reduce output records	0	38,156	38,156
	Reduce shuffle bytes	0	1,730,582	1,730,582
	Shuffled Map	0	15	15
	Spilled Records	38,156	38,156	76,312
	Total committed heap memory (bytes)	2,996,305,820	201,328,582	3,197,634,402
	Virtual memory (bytes) snapshot	24,734,625,792	1,658,044,416	26,392,670,208
	Map ID	0	0	0
	COMBINATION	0	0	0
File Input Format Counters	Bytes Read	25,730,266	0	25,730,266
	Bytes Written	0	1,649,067	1,649,067

JOB2 具体:









Counter Group	Counters			
	Name	Map	Reduce	Total
File System Counters	FILE: Number of bytes read	0	837,456	837,456
	FILE: Number of bytes written	963,161	963,096	1,926,247
	FILE: Number of large read operations	0	0	0
	FILE: Number of read operations	0	0	0
	FILE: Number of write operations	0	0	0
	HDFS: Number of bytes read	1,649,187	0	1,649,187
	HDFS: Number of bytes written	0	705,366	705,366
	HDFS: Number of large read operations	0	0	0
	HDFS: Number of read operations	3	3	6
	HDFS: Number of write operations	0	2	2
Job Counters	Launched map tasks	0	0	1
	Launched reduce tasks	0	0	1
	Red-Local map tasks	0	0	1
	Total executor-seconds taken by all map tasks	0	0	4,902,912
	Total executor-seconds taken by all reduce tasks	0	0	4,886,528
	Total time spent by all map tasks (ms)	0	0	4,788
	Total time spent by all maps in assigned slots (ms)	0	0	4,788
	Total time spent by all reduce tasks (ms)	0	0	4,772
	Total time spent by all reducers in assigned slots (ms)	0	0	4,772
	Total user-seconds taken by all map tasks	0	0	4,788
Map-Reduce Framework	Combine input records	215,066	0	215,066
	Combine output records	34,272	0	34,272
	CPU time spent (ms)	3,640	3,290	6,930
	Failed Shuffles	0	0	0
	GC time elapsed (ms)	76	42	118
	Input split bytes	120	0	120
	Map input records	36,156	0	36,156
	Map output bytes	4,900,546	0	4,900,546
	Map output materialized bytes	837,456	0	837,456
	Map output records	215,066	0	215,066
Map-Reduce Framework	Mapred Map outputs	0	1	1
	Physical memory (bytes) snapshot	273,145,956	180,371,456	453,517,312
	Reduce input groups	0	34,272	34,272
	Reduce input records	0	34,272	34,272
	Reduce output records	0	34,272	34,272
	Reduce shuffle bytes	0	837,456	837,456
	Shuffled Maps	0	1	1
	Spilled Records	34,272	34,272	68,544
	Total committed heap usage (bytes)	199,763,728	201,326,592	401,090,320
	Virtual memory (bytes) snapshot	1,649,939,008	1,695,950,144	3,304,497,152
Shuffle Errors	BAD_ID	0	0	0
	CONNECTION	0	0	0
	IO_EXCEPTION	0	0	0
	WRONG_LENGTH	0	0	0
	WRONG_MAP	0	0	0
	WRONG_REDUCE	0	0	0
File Input Format Counters	Bytes Read	1,649,067	0	1,649,067
File Output Format Counters	Bytes Written	0	705,366	705,366

JOB3 具体：

Counter Group	Counters			
	Name	Map	Reduce	Total
File System Counters	FILE: Number of bytes read	0	773,936	773,936
	FILE: Number of bytes written	889,465	889,410	1,778,875
	FILE: Number of large read operations	0	0	0
	FILE: Number of read operations	0	0	0
	FILE: Number of write operations	0	0	0
	HDFS: Number of bytes read	0	0	0
	HDFS: Number of bytes written	0	1,010,384	1,010,384
	HDFS: Number of large read operations	0	0	0
	HDFS: Number of read operations	3	3	6
	HDFS: Number of write operations	0	2	2
Job Counters	Launched map tasks	0	0	1
	Launched reduce tasks	0	0	1
	Red-Local map tasks	0	0	1
	Total executor-seconds taken by all map tasks	0	0	5,885,952
	Total executor-seconds taken by all reduce tasks	0	0	6,026,240
	Total time spent by all map tasks (ms)	0	1	5,748
	Total time spent by all maps in assigned slots (ms)	0	0	5,748
	Total time spent by all reduce tasks (ms)	0	0	5,885
	Total time spent by all reducers in assigned slots (ms)	0	0	5,885
	Total user-seconds taken by all map tasks	0	0	5,748
Map-Reduce Framework	Combine input records	0	0	0
	Combine output records	0	0	0
	CPU time spent (ms)	5,080	5,000	5,080
	Failed Shuffles	0	0	0
	GC time elapsed (ms)	63	46	99
	Input split bytes	120	0	120
	Map input records	34,272	0	34,272
	Map output bytes	705,366	0	705,366
	Map output materialized bytes	0	773,936	773,936
	Map output records	0	34,272	34,272
Map-Reduce Framework	Mapred Map outputs	0	1	1
	Physical memory (bytes) snapshot	268,681,216	169,705,472	438,386,688
	Reduce input groups	0	1,279	1,279
	Reduce input records	0	34,272	34,272
	Reduce output records	0	1,279	1,279
	Reduce shuffle bytes	0	773,936	773,936
	Shuffled Maps	0	1	1
	Spilled Records	34,272	34,272	68,544
	Total committed heap usage (bytes)	201,326,592	201,326,592	402,653,184
	Virtual memory (bytes) snapshot	1,646,922,624	1,695,562,240	3,304,484,864
Shuffle Errors	BAD_ID	0	0	0
	CONNECTION	0	0	0
	IO_EXCEPTION	0	0	0
	WRONG_LENGTH	0	0	0
	WRONG_MAP	0	0	0
	WRONG_REDUCE	0	0	0
File Input Format Counters	Bytes Read	705,366	0	705,366
File Output Format Counters	Bytes Written	0	1,010,384	1,010,384

Pagerank 部分运行截图（由于迭代次数较多就不全截图了）

整体：

application_1469066011991_0092	2016-03	PageRank	MAPREDUCE	root.default	Thu Jul 21 10:43:02 +0800 2016	Thu Jul 21 10:43:34 +0800 2016	FINISHED	SUCCEEDED	 History
application_1469066011991_0091	2016-03	PageRank	MAPREDUCE	root.default	Thu Jul 21 10:42:33 +0800 2016	Thu Jul 21 10:42:59 +0800 2016	FINISHED	SUCCEEDED	 History
application_1469066011991_0090	2016-03	PageRank	MAPREDUCE	root.default	Thu Jul 21 10:41:57 +0800 2016	Thu Jul 21 10:42:25 +0800 2016	FINISHED	SUCCEEDED	 History
application_1469066011991_0089	2016-03	PageRank	MAPREDUCE	root.default	Thu Jul 21 10:41:22 +0800 2016	Thu Jul 21 10:41:49 +0800 2016	FINISHED	SUCCEEDED	 History
application_1469066011991_0088	2016-03	PageRank	MAPREDUCE	root.default	Thu Jul 21 10:40:54 +0800 2016	Thu Jul 21 10:41:19 +0800 2016	FINISHED	SUCCEEDED	 History
application_1469066011991_0087	2016-03	PageRank	MAPREDUCE	root.default	Thu Jul 21 10:40:14 +0800 2016	Thu Jul 21 10:40:51 +0800 2016	FINISHED	SUCCEEDED	 History
application_1469066011991_0086	2016-03	PageRank	MAPREDUCE	root.default	Thu Jul 21 10:39:47 +0800 2016	Thu Jul 21 10:40:11 +0800 2016	FINISHED	SUCCEEDED	 History
application_1469066011991_0083	2016-03	PageRank	MAPREDUCE	root.default	Thu Jul 21 10:39:01 +0800 2016	Thu Jul 21 10:39:44 +0800 2016	FINISHED	SUCCEEDED	 History

具体部分截图：

Counter Group	Name	Map	Reduce	Total
File System Counters	FILE: Number of bytes read	0	2,396,427	2,396,427
	FILE: Number of bytes written	2,613,902	2,613,902	5,027,809
	FILE: Number of large read operations	0	0	0
	FILE: Number of read operations	0	0	0
	FILE: Number of write operations	0	0	0
	HDFS: Number of bytes read	1,010,504	0	1,010,504
	HDFS: Number of bytes written	0	1,034,753	1,034,753
	HDFS: Number of large read operations	0	0	0
	HDFS: Number of read operations	3	3	6
	HDFS: Number of write operations	0	2	2
Job Counters	Launched map tasks	0	0	1
	Launched reduce tasks	0	0	1
	Red-Local-map tasks	0	0	1
	Total executor-seconds taken by all map tasks	0	0	4,847,616
	Total executor-seconds taken by all reduce tasks	0	0	6,230,016
	Total time spent by all map tasks (ms)	0	0	4,734
	Total time spent by all maps in occupied slots (ms)	0	0	4,734
	Total time spent by all reduce tasks (ms)	0	0	6,004
	Total time spent by all reduces in occupied slots (ms)	0	0	6,004
	Total map-records taken by all map tasks	0	0	4,734
	Total reduce-records taken by all reduce tasks	0	0	6,004
Map-Reduce Framework	Combine input records	0	0	0
	Combine output records	0	0	0
	CTR time spent (ms)	2,360	4,290	6,650
	Failed Shuffles	0	0	0
	GC time elapsed (ms)	60	67	147
	Input split bytes	120	0	120
	Map input records	1,279	0	1,279
	Map output bytes	2,325,309	0	2,325,309
	Map output materialized bytes	2,396,427	0	2,396,427
	Map output records	36,651	0	36,651
	Mapred Map outputs	0	1	1
	Physical memory (bytes) memcache	269,997,840	185,376,768	495,364,608
	Reduce input records	0	1,279	1,279
	Reduce input records	0	36,651	36,651
	Reduce output records	0	1,279	1,279
	Reduce shuffle bytes	0	2,396,427	2,396,427
	Shuffled Map	0	1	1
	Spilled Records	36,651	36,651	71,102
	Total committed heap memory (bytes)	190,180,064	190,705,152	396,886,016
	Virtual memory (bytes) memcache	1,649,922,624	1,695,606,336	3,304,489,960
Shuffle Errors	BAD_ID	0	0	0
	CORRUPTIO	0	0	0
	IO_ERROR	0	0	0
	NR_FINDING	0	0	0
	WRONG_MAP	0	0	0
	WRONG_REDUCE	0	0	0
File Input Format Counters	Bytes Read	1,010,384	0	1,010,384
File Output Format Counters	Bytes Written	0	1,034,753	1,034,753

pagerankviewer:

2016.07.22 16:45:08 CST	2016.07.22 16:45:13 CST	2016.07.22 16:45:26 CST	job_1469066011991_0843	PageRankViewer	2016-07-23	root.default	SUCCEEDED
2016.07.22 16:44:46 CST	2016.07.22 16:44:51 CST	2016.07.22 16:45:06 CST	job_1469066011991_0842	PageRank	2016-07-23	root.default	SUCCEEDED
2016.07.22 16:44:24 CST	2016.07.22 16:44:29 CST	2016.07.22 16:44:43 CST	job_1469066011991_0841	PageRank	2016-07-23	root.default	SUCCEEDED
2016.07.22 16:44:00 CST	2016.07.22 16:44:06 CST	2016.07.22 16:44:21 CST	job_1469066011991_0840	PageRank	2016-07-23	root.default	SUCCEEDED

Counter Group	Counters			
	Name	Map	Reduce	Total
File System Counters	FILE_Member_of_bytes_read	0	24,693	24,693
	FILE_Member_of_bytes_written	140,704	140,649	281,353
	FILE_Member_of_large_read_operations	0	0	0
	FILE_Member_of_large_read_operations	0	0	0
	FILE_Member_of_large_read_operations	0	0	0
	FILE_Member_of_large_read_operations	0	0	0
	FILE_Member_of_large_read_operations	0	0	0
	FILE_Member_of_large_read_operations	0	0	0
	FILE_Member_of_large_read_operations	0	0	0
Job Counters	Map-Local-map-tasks	0	0	1
	Map-Local-map-tasks	0	0	1
	Map-Local-map-tasks	0	0	1
	Map-Local-map-tasks	0	0	1
	Map-Local-map-tasks	0	0	1
	Map-Local-map-tasks	0	0	1
	Map-Local-map-tasks	0	0	1
	Map-Local-map-tasks	0	0	1
	Map-Local-map-tasks	0	0	1
Map-Reduce Framework	Combine input records	0	0	0
	Combine output records	0	0	0
	Combine output records	0	0	0
	Combine output records	0	0	0
	Combine output records	0	0	0
	Combine output records	0	0	0
	Combine output records	0	0	0
	Combine output records	0	0	0
	Combine output records	0	0	0
Shuffle Errors	Bad ID	0	0	0
	Bad ID	0	0	0
	Bad ID	0	0	0
	Bad ID	0	0	0
	Bad ID	0	0	0
	Bad ID	0	0	0
	Bad ID	0	0	0
	Bad ID	0	0	0
	Bad ID	0	0	0
File Input Format Counters	Bytes Read	1,034,815	0	1,034,815
File Output Format Counters	Bytes Written	0	36,528	36,528

LPA 部分运行结果截图：

整体：

application 1469066011991_0132	2016st03	LPAViewer	MAPREDUCE	root.default	Thu Jul 21 11:00:06 +0800 2016	Thu Jul 21 11:00:35 +0800 2016	FINISHED	SUCCEEDED		History
application 1469066011991_0131	2016st25	PageRankSort	MAPREDUCE	root.default	Thu Jul 21 10:59:25 +0800 2016	Thu Jul 21 11:00:04 +0800 2016	FINISHED	SUCCEEDED		History
application 1469066011991_0130	2016st03	LPA	MAPREDUCE	root.default	Thu Jul 21 10:59:18 +0800 2016	Thu Jul 21 10:59:59 +0800 2016	FINISHED	SUCCEEDED		History
application 1469066011991_0129	2016st25	PageRank	MAPREDUCE	root.default	Thu Jul 21 10:58:54 +0800 2016	Thu Jul 21 10:59:21 +0800 2016	FINISHED	SUCCEEDED		History
application 1469066011991_0128	2016st03	LPA	MAPREDUCE	root.default	Thu Jul 21 10:58:35 +0800 2016	Thu Jul 21 10:59:11 +0800 2016	FINISHED	SUCCEEDED		History
application 1469066011991_0127	2016st03	LPA	MAPREDUCE	root.default	Thu Jul 21 10:58:03 +0800 2016	Thu Jul 21 10:58:31 +0800 2016	FINISHED	SUCCEEDED		History
application 1469066011991_0126	2016st25	PageRank	MAPREDUCE	root.default	Thu Jul 21 10:57:57 +0800 2016	Thu Jul 21 10:58:44 +0800 2016	FINISHED	SUCCEEDED		History
application 1469066011991_0125	2016st03	LPA	MAPREDUCE	root.default	Thu Jul 21 10:57:18 +0800 2016	Thu Jul 21 10:57:59 +0800 2016	FINISHED	SUCCEEDED		History
application 1469066011991_0124	2016st25	PageRank	MAPREDUCE	root.default	Thu Jul 21 10:57:00 +0800 2016	Thu Jul 21 10:57:53 +0800 2016	FINISHED	SUCCEEDED		History
application 1469066011991_0123	2016st03	LPA	MAPREDUCE	root.default	Thu Jul 21 10:56:47 +0800 2016	Thu Jul 21 10:57:15 +0800 2016	FINISHED	SUCCEEDED		History
application 1469066011991_0122	2016st32	Driver	MAPREDUCE	root.default	Thu Jul 21 10:56:46 +0800 2016	N/A	RUNNING	UNDEFINED		ApplicationMaster
application 1469066011991_0121	2016st25	PageRank	MAPREDUCE	root.default	Thu Jul 21 10:56:21 +0800 2016	Thu Jul 21 10:56:51 +0800 2016	FINISHED	SUCCEEDED		History
application 1469066011991_0120	2016st03	LPA	MAPREDUCE	root.default	Thu Jul 21 10:56:09 +0800 2016	Thu Jul 21 10:56:44 +0800 2016	FINISHED	SUCCEEDED		History

具体：

Counter Group		Counters			
	Name	Map	Reduce		Total
File System Counters	FILE_Member_of_bytes_read	0	4,095,797	4,095,797	
	FILE_Member_of_bytes_written	4,211,327	4,211,372	4,422,599	
	FILE_Member_of_large_read_operations	0	0	0	
	FILE_Member_of_read_operations	0	0	0	
	FILE_Member_of_write_operations	0	0	0	
	HDFS_Member_of_bytes_read	1,010,504	0	1,010,504	
	HDFS_Member_of_bytes_written	0	1,022,256	1,022,256	
	HDFS_Member_of_large_read_operations	0	0	0	
	HDFS_Member_of_read_operations	3	3	6	
	HDFS_Member_of_write_operations	0	2	2	
Job Counters	Launched map tasks	0	0	1	
	Launched reduce tasks	0	0	1	
	Successful map tasks	0	0	1	
	Total mapwriteseconds taken by all map tasks	0	0	5,873,664	
	Total write spent by all map tasks (ms)	0	0	5,738	
	Total time spent by all maps in assigned slots (ms)	0	0	5,736	
	Total time spent by all reduce tasks (ms)	0	0	5,167	
	Total time spent by all reduces in assigned slots (ms)	0	0	5,167	
	Total readseconds taken by all map tasks	0	0	5,736	
Map-Reduce Framework	Total readseconds taken by all reduce tasks	0	0	5,167	
	Combine input records	0	0	0	
	Combine output records	0	0	0	
	CP time spent (ms)	2,160	5,900	2,760	
	Failed shuffles	0	0	0	
	GC time elapsed (ms)	70	64	142	
	Input split bytes	120	0	120	
	Map input records	1,279	0	1,279	
	Map output bytes	3,964,134	0	3,964,134	
Map-Reduce Framework	Map output materialized bytes	4,095,797	0	4,095,797	
	Map output records	69,823	0	69,823	
	Physical memory (bytes) spilled	0	1	1	
	Physical memory (bytes) spewshot	284,622,948	191,311,872	475,934,720	
	Reduce input groups	0	1,279	1,279	
	Reduce input records	0	69,823	69,823	
	Reduce output records	0	1,279	1,279	
	Reduce shuffle bytes	0	4,095,797	4,095,797	
	Shuffled Map	0	1	1	
Shuffle Errors	Spilled Records	69,823	0	139,646	
	Total committed heap usage (bytes)	198,180,864	198,705,182	396,886,016	
	Virtual memory (bytes) spewshot	1,648,922,624	1,656,913,920	3,305,836,544	
	Map				Total
	Map				
	Map				
	Map				
	Map				
	Map				
File Input Format Counters	Bytes Read	1,010,384	0	1,010,384	
	Bytes Written	0	1,022,256	1,022,256	
File Output Format Counters	Bytes Read	1,010,384	0	1,010,384	
	Bytes Written	0	1,022,256	1,022,256	

LPViewer:

整体:

2016.07.22 16:53:20 CST	2016.07.22 16:53:25 CST	2016.07.22 16:53:39 CST	job_1469068011991_0873	LPViewer	2016-t03	root.default	SUCCEEDED
2016.07.22 16:52:57 CST	2016.07.22 16:53:02 CST	2016.07.22 16:53:16 CST	job_1469068011991_0872	LFA	2016-t03	root.default	SUCCEEDED
2016.07.22 16:52:34 CST	2016.07.22 16:52:39 CST	2016.07.22 16:52:54 CST	job_1469068011991_0871	LFA	2016-t03	root.default	SUCCEEDED
2016.07.22 16:52:09 CST	2016.07.22 16:52:15 CST	2016.07.22 16:52:30 CST	job_1469068011991_0870	LFA	2016-t03	root.default	SUCCEEDED

具体:

Counter Group	Name	Map	Reduce	Total
File System Counters	FILE: Number of bytes read	0	26,023	26,023
	FILE: Number of bytes written	141,595	141,540	283,135
	FILE: Number of large read operations	0	0	0
	FILE: Number of read operations	0	0	0
	FILE: Number of write operations	0	0	0
	HDFS: Number of bytes read	1,021,873	0	1,021,873
	HDFS: Number of bytes written	0	23,459	23,459
	HDFS: Number of large read operations	0	0	0
	HDFS: Number of read operations	3	3	6
	HDFS: Number of write operations	0	2	2
Job Counters	Task-local map tasks	0	0	1
	Launched map tasks	0	0	1
	Launched reduce tasks	0	0	1
	Total mapslots-seconds taken by all map tasks	0	0	4,240,804
	Total mapslots-seconds taken by all reduce tasks	0	0	4,648,940
	Total time spent by all map tasks (ms)	0	0	4,161
	Total time spent by all map in assigned slots (ms)	0	0	4,161
	Total time spent by all reduce tasks (ms)	0	0	4,560
	Total time spent by all reduce in assigned slots (ms)	0	0	4,560
	Total vcores-seconds taken by all map tasks	0	0	4,161
Map-Reduce Framework	Combine input records	0	0	0
	Combine output records	0	0	0
	CPU time spent (ms)	700	1,740	2,440
	Failed Shuffles	0	0	0
	C time elapsed (ms)	77	40	67
	Input split bytes	127	0	127
	Map input records	1,279	0	1,279
	Map output bytes	23,459	0	23,459
	Map output materialized bytes	26,023	0	26,023
	Map output records	1,279	0	1,279
Mapred Map outputs	Physical memory (bytes) snapshot	0	1	1
	Reduce input records	260,687,008	168,255,488	428,942,496
	Reduce input records	0	22	22
	Reduce input records	0	1,279	1,279
	Reduce output records	0	1,279	1,279
	Reduce shuffle bytes	0	26,023	26,023
	Shuffled Map	0	1	1
	Shuffled Records	1,279	1,279	2,558
	Total committed heap usage (bytes)	201,326,592	201,326,592	402,653,184
	Virtual memory (bytes) snapshot	1,645,922,024	1,695,410,088	3,341,332,112
Shuffle Errors	MAP_XX	0	0	0
	COMMITTER	0	0	0
	IO_ERROR	0	0	0
	WRITE_FAILURE	0	0	0
	WRITE_NULL	0	0	0
File Input Format Counters	Bytes Read	1,021,740	0	1,021,740
File Output Format Counters	Bytes Written	0	23,459	23,459

5.2 复杂度分析展示

时间复杂度展示：
整体运行时间：953s（12 次 pr 迭代，15 次 LPA 迭代）

各部分运行时间：

JOB1	34s
JOB2	20s
JOB3	20s
Pagerank	367s
Pagerankviewer	25s
LPA	458s
LPAviewer	29s
TOTAL	953s

Pagerank 各步迭代时间： 平均 30s
LPA 各步迭代时间： 平均 30s

6 结果处理以及分析

对于最后构建的金庸的江湖的人物社交网络，我们在输出的结果的同时，主要利用的 **gephi** 作为绘图软件对结果进行数据可视化的处理，对 **gephi** 做一个简要介绍。**Gephi** 是一款开源免费跨平台基于 **JVM** 的复杂网络分析软件，其主要用于各种网络和复杂系统，动态和分层图的交互可视化与探测开源工具，他对于数据集群聚类涂色以及发现数据的内在规律和特点十分的方便，有利于我们对结果进行令人信服的分析。
输出的数据结果，同时作为 gephi 的输入：

三个数据文件：

- 1、一个节点以及 pr 值的数据文件。

部分截图如下：

35.02727362532683	韦小宝
20.335327076140903	张无忌
20.102750943423	令狐冲
15.273027029369551	郭靖
14.312007742639077	袁承志
14.06896607596625	杨过
13.266900381929842	汉子
12.984034800846567	胡斐
12.938242643774503	段誉
12.627204246703375	黄蓉
9.368241771542161	陈家洛
8.17158722668364	石破天
7.980265061272337	吴三桂
7.558959908298476	岳不群
6.5920596488246	谢逊
6.515383136560769	赵敏
6.050830462100421	小龙女
5.907752810339623	虚竹
5.901789431574308	狄云
5.49621002296143	文泰来
5.471853337784832	徐天宏
5.4508972427245945	王语嫣

- 2、一个具有源节点和目标节点和权值的边表数据。

部分截图如下：

source	target	weight
一灯大师	完颜萍	0.005038
一灯大师	小龙女	0.017632
一灯大师	尹克西	0.002519
一灯大师	尼摩星	0.007557
一灯大师	张无忌	0.005038
一灯大师	无色	0.002519
一灯大师	明月	0.002519
一灯大师	朱九真	0.005038
一灯大师	朱子柳	0.032746
一灯大师	朱长龄	0.002519
一灯大师	李莫愁	0.012594
一灯大师	杨康	0.007557
一灯大师	丘处机	0.012594
一灯大师	杨过	0.09068
一灯大师	柯镇恶	0.005038
一灯大师	梅超风	0.002519
一灯大师	樵子	0.017632
一灯大师	欧阳克	0.007557
一灯大师	武三娘	0.005038
一灯大师	武三通	0.030227
一灯大师	武修文	0.007557
一灯大师	武敦儒	0.005038
一灯大师	武青婴	0.005038
一灯大师	汉子	0.005038
一灯大师	洪七公	0.020151
一灯大师	渔人	0.030227
一灯大师	潇湘子	0.002519
一灯大师	点苍渔隐	0.007557
一灯大师	王处一	0.005038

- 3、一个具有 id 和标签的数据文件。
部分截图如下：

丁大全	小王将军
丁大全	陈大方
令狐冲	桃枝仙
令狐冲	令狐冲
令狐冲	桃干仙
令狐冲	李大元
令狐冲	桃实仙
令狐冲	夏老拳师
令狐冲	齐堂主
令狐冲	鲁连荣
令狐冲	聋哑婆婆
令狐冲	木高峰
令狐冲	曲非烟
令狐冲	天乙道人
令狐冲	桃叶仙
令狐冲	天松道人
令狐冲	冲虚
令狐冲	郑镖头
令狐冲	曲洋
令狐冲	郑萼
令狐冲	易师爷
令狐冲	天门道人
令狐冲	风清扬
令狐冲	易堂主
令狐冲	上官云
令狐冲	丹青生
令狐冲	冲虚道长
令狐冲	任我行
令狐冲	不戒和尚
令狐冲	秃笔翁
令狐冲	上官
令狐冲	刘正风

输入 gephi 之后最终的数据集成形态部分如下图：
边数据：

Source	Target	类型	Id	Label	Interval	Weight
一灯大师	完颜泽	有向的	0			0.005038
一灯大师	小龙女	有向的	1			0.017632
一灯大师	尹克西	有向的	2			0.002519
一灯大师	尼摩星	有向的	3			0.007557
一灯大师	张无忌	有向的	4			0.005038
一灯大师	无色	有向的	5			0.002519
一灯大师	明月	有向的	6			0.002519
一灯大师	朱九真	有向的	7			0.005038
一灯大师	朱子柳	有向的	8			0.032746
一灯大师	朱长龄	有向的	9			0.002519
一灯大师	李莫愁	有向的	10			0.012594
一灯大师	杨康	有向的	11			0.007557
一灯大师	丘处机	有向的	12			0.012594
一灯大师	杨过	有向的	13			0.09068
一灯大师	柯镇恶	有向的	14			0.005038
一灯大师	梅超风	有向的	15			0.002519
一灯大师	樵子	有向的	16			0.017632
一灯大师	欧阳克	有向的	17			0.007557
一灯大师	武三娘	有向的	18			0.005038
一灯大师	武三通	有向的	19			0.030227
一灯大师	武修文	有向的	20			0.007557
一灯大师	武敦儒	有向的	21			0.005038
一灯大师	武青婴	有向的	22			0.005038
一灯大师	汉子	有向的	23			0.005038

节点数据：

一灯大师	郭靖		1.395365
完颜泽	杨过		0.804243
小龙女	杨过		6.034715
尹克西	杨过		1.357899
尼摩星	杨过		1.186722
张无忌	张无忌		20.325623
无色	杨过		1.310463
明月	张无忌		0.926775
朱九真	张无忌		0.856526
朱子柳	杨过		1.32781
朱长龄	张无忌		0.85091
李莫愁	杨过		3.247109
杨康	郭靖		2.27868
丘处机	郭靖		4.007579
杨过	杨过		14.035649
柯镇恶	郭靖		2.789343
梅超风	郭靖		2.149266
樵子	郭靖		0.537167
欧阳克	郭靖		1.920503
武三娘	杨过		0.416763
武三通	杨过		1.206059
武修文	杨过		1.197938
武敦儒	杨过		0.890485
武青婴	张无忌		0.717028
汉子	令狐冲		13.273516
洪七公	郭靖		3.189304
渔人	郭靖		0.609959
潇湘子	杨过		1.593651
点苍渔隐	杨过		0.248522
王处一	郭靖		1.50712
琴儿	胡斐		0.351121
穆念慈	郭靖		1.203124

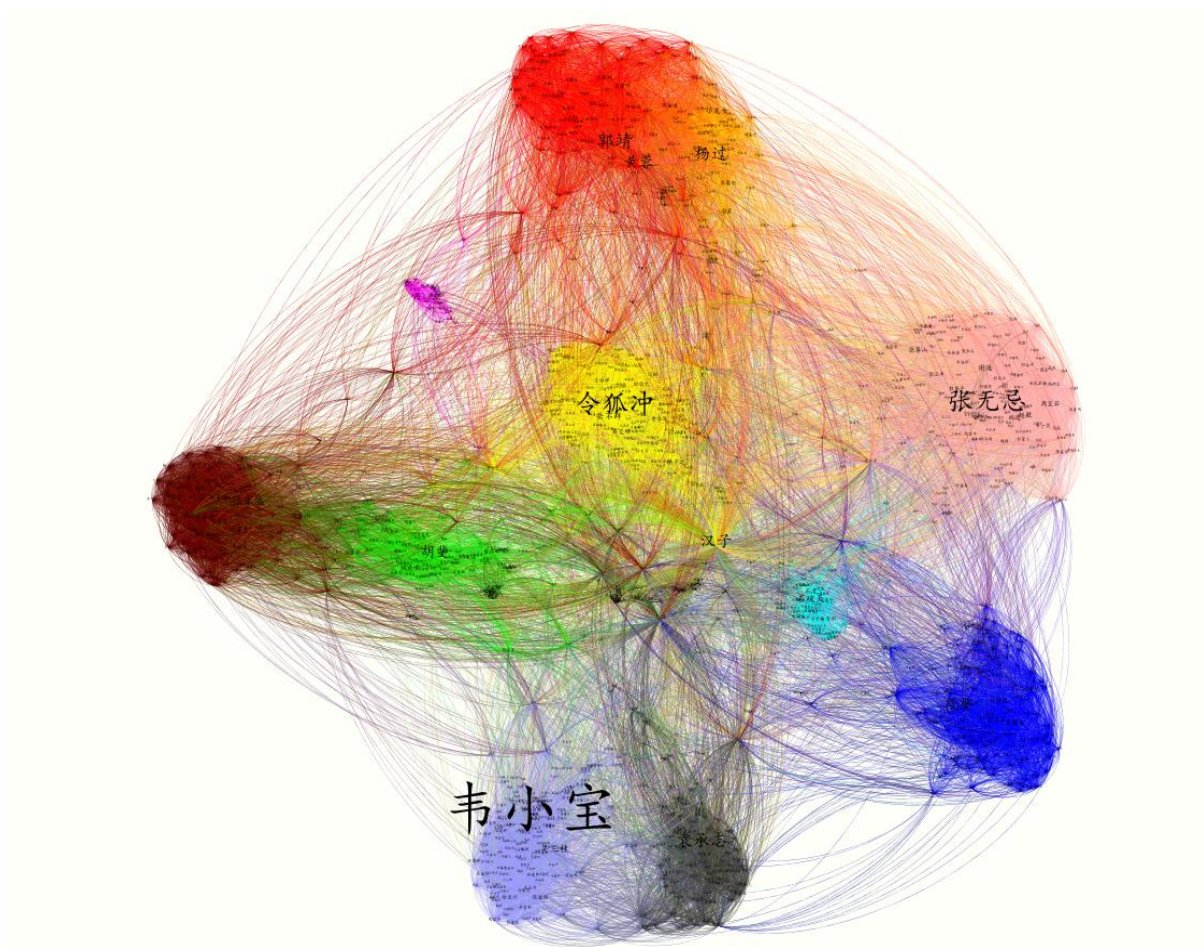
根据数据进行可视化处理。

可视化处理：

采用 forceatlas2 算法对图进行初步布局，然后根据标签对每一个节点进行涂色处理以显示出聚类的效果，同时节点和标签大小根据 pagerank 进行正相关处理，以凸显出哪些人物是金庸小说之中的主人公。

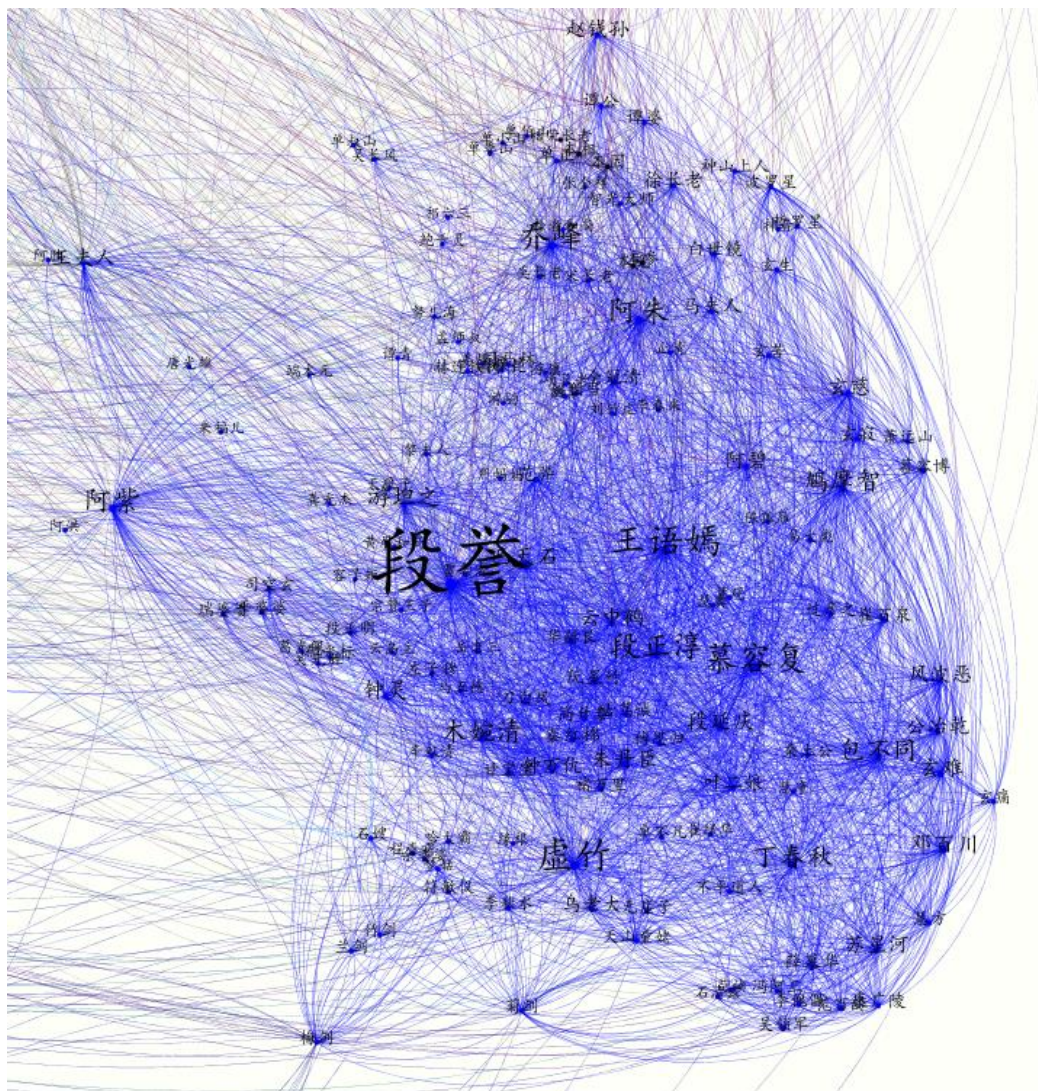
数据可视化展示：

整体如下图效果：



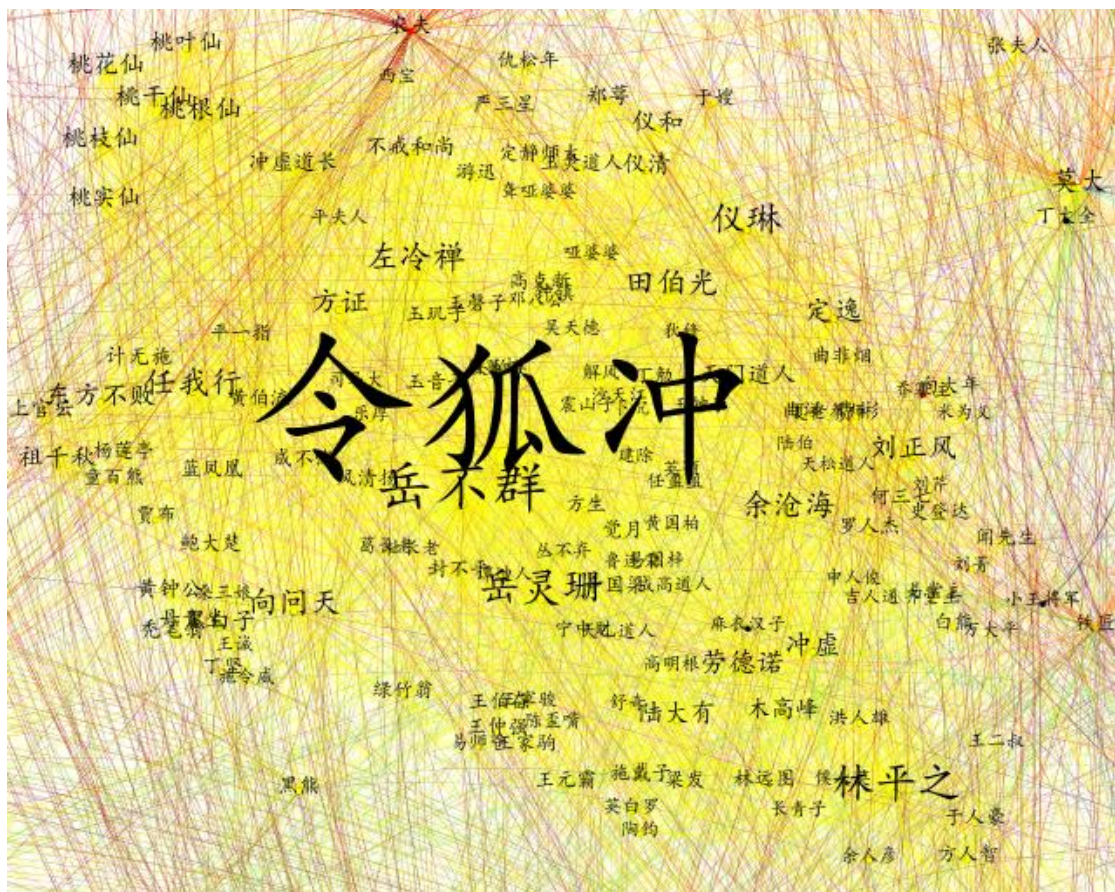
从整体图中可以看出根据标签的涂色的聚类效果良好，根据 pagerank 值的大小可以看出，排名前三名的韦小宝令狐冲以及张无忌的效果显示很好，下面对各个区域（大致为同一部作品）的内容进行展示以及对部分结果的检验。以蓝色和黄色区域为例：

蓝色：



如图所示三兄弟主人公乔峰，段誉以及虚竹的标签很明显，表明其 pagerank 值较大，从聚类情况来看，段誉与王语嫣，乔峰和阿朱，虚竹和无崖子这样关系紧密的组合聚在一起，显示出聚类情况良好。

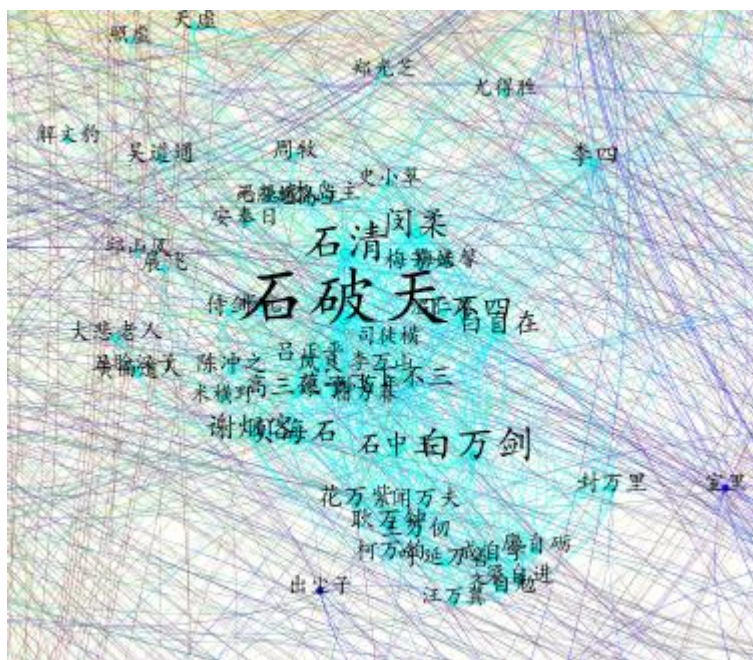
黄色：



从图中可以看出令狐冲是笑傲江湖当之无愧的主角，重点十分突出，同时岳不群和岳灵珊，东方不败和任我行，令狐冲和任盈盈等等组合的聚类效果良好，显示出关系的紧密性。

一些结果的思考，规律的发现以及创造性的认识：

- ① 一些小团体的聚类效果:

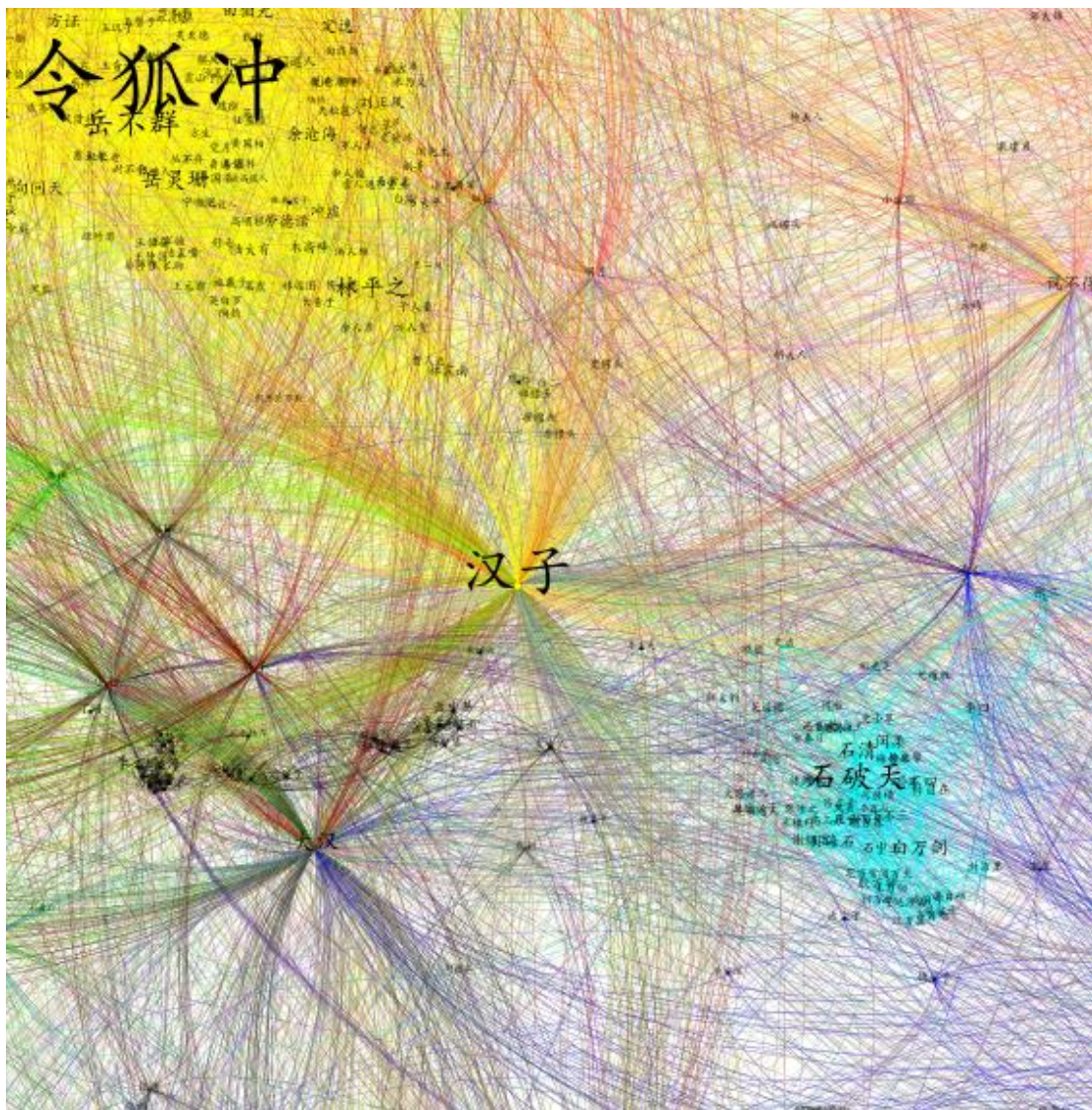


以上就是部分小团体聚类效应的体现，其他还有很多就不一一截图展示了。

一些有趣的现象：

部分特殊名字：

汉子这个节点的可视化情况如下：



他的 **pagerank** 值高居第七名，标签是令狐冲的标签。

Pagerank 值的情况我考虑是在进行数据归一化的时候，因为金庸的小说里面汉子出现的频率很高，所以他在某一个节点的邻接点群中的权值很高，所以在 **pagerank** 值的迭代计算过程中，他的 **pagerank** 值会很好，他的标签是令狐冲的原因是因为在选择标签的时候，我们利用的是他所有 **label** 之中出度权值最高的那一个标签，令狐冲的出度权值最高，所以导致是令狐冲这个标签来体现汉子。

对汉子在 **pagerank** 值总结表中进行查找证实了这一结果。

对于这个现象我们也可以对其进行合理的解释，汉子和那么多的人物之间连接，正好符合金庸江湖的特征，总所周知，金庸的文风是大开大合的类型，讲的更多的是江湖大义，是英雄豪杰的男儿豪气，所以汉子这个词在金庸的作品之中有着举足轻重的位置和出现率，从图中也可以看出，汉子几乎位于图的中心连接了许多部作品，所有对于寻找作品的主人公造成了一定的困扰，但是也可以有其合理性和作用。

同样具有这样的情况还有大汉这个词，比汉子 **pagerank** 值略低，但也有类似的现象和解释性。

跨作品人物的探究:

在图中存在一些跨作品人物的情况，我们可以对这些人物进行探究和观察了解一些我们算法的一些特征性。

李自成:

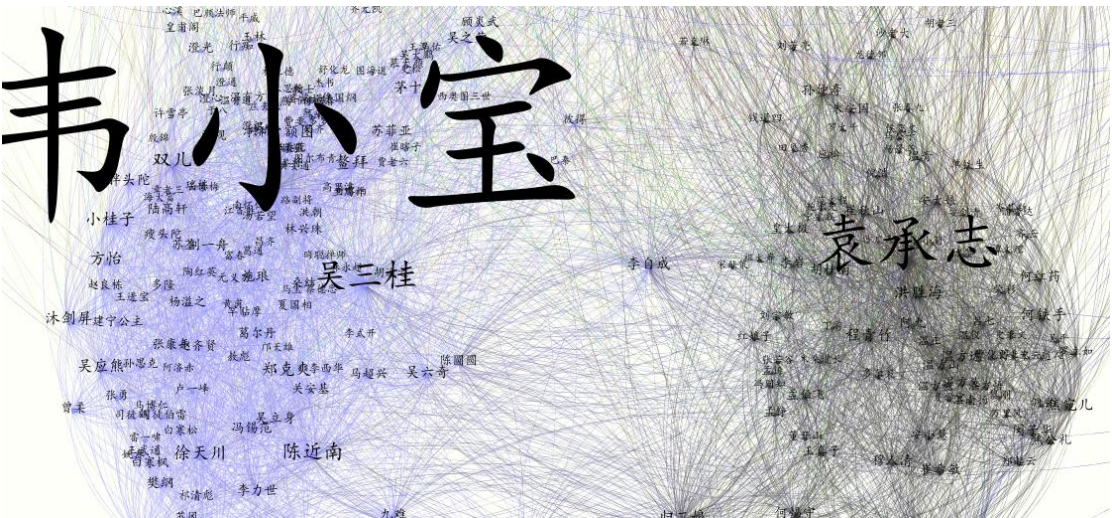
李自成这个人物主要跨碧血剑和鹿鼎记两部作品。

他的节点属性如下:

Label 是韦小宝，pagerank 值是 1.985.

韦小宝 - 属性	
尺寸	12. 098736
位置(x)	-321. 19904
位置(y)	-8860. 665
位置(z)	0. 0
颜色	<div><div></div> [153,153,255]</div> <div>+++</div>
Label Size	3. 4722154
Label Color	<div><div></div> [153,153,255]</div> <div>+++</div>
Label Visible	<input checked="" type="checkbox"/>
韦小宝 - 属性	
Id	李自成 <div>+++</div>
Label	韦小宝 <div>+++</div>
Interval	<空值> <div>+++</div>
pg	1. 985037168

在图中的可视化显示如下:



在图中李自成介于鹿鼎记和碧血剑两部作品之间，没有特定的聚类效果，是韦小宝这个标

签的原因还是因为出度权重的大小韦小宝更大使得李自成的标签是韦小宝，同时两部小说中李自成在鹿鼎记中确实有着更重的作用，所以他的标签是韦小宝也无可厚非，这样的可视化效果也良好。

周伯通：

周伯通这个角色主要在射雕英雄传和神雕侠侣里面出现，显而易见是在射雕英雄传中更重要。

他的节点性质如下：

编辑	
郭靖 - 属性	
尺寸	13.734564
位置(x)	615.53296
位置(y)	9049.173
位置(z)	0.0
颜色	<div><div></div>[255,0,0]</div>
Label Size	3.840277
Label Color	<div><div></div>[255,0,0]</div>
Label Visible	<input checked="" type="checkbox"/>
郭靖 - 属性	
Id	周伯通
Label	郭靖
Interval	<空值>
pg	3.412124249

他的 label 是郭靖，pg 值是 3.1421

在图中的可视化情况如下：



周伯通在图中介于射雕英雄传和神雕侠侣之间，他的标签是郭靖，符合我们的预知，周伯通主要出现在射雕英雄传之中，他的标签是郭靖十分的正常也很显而易见。还有很多这样的跨小说的人物，他们是金庸的江湖重要的组成部分，让一些有历史沿革以及人物交集的作品能够在可视化数据的操作中清楚的展现在我们眼前。

最终的结果分析

最终的结果我们得出很有用的一些规律。

首先是输出结果的 **pagerank** 值可以让我们很好的了解金庸小说之中的主人公到底有哪些，然后输出的标签可以告诉我们那些主角配角的“社交圈”和相互之间的关系的状况。

然后就是图中的各部分聚类很好的给出了金庸的小说的分布以及各部分的主人公，可视化的结果比文字和图表更加的直观以及形象。

图中的一些奇怪的现象比如“汉子问题”可以给我们自己思考总结解释的空间，同样也能让我们更加去优化的自己的代码。

进行的这样的数据收集处理以及可视化的展示能让我们不需要查阅很多的资料对金庸先生的 14 簿著名小说的人物进行一个感性的一目了然的认识，让我们感觉到了大数据的力量，以及这样的领域的实际应用以及未来如何处理类似的问题，给了我们非常多的经验。

总的来说，这样的结果给了我们发现很多个人观点和创造性规律的机会，结果的显示总的来说良好，符合我们做实验之前的预期，给了我们很多启示。

7 实验过程的反思和总结

- ① 第一点要反思和总结的就是觉得我们的动手能力和理论的结合度还很不足，再进行代码量比较大的 `mapreduce` 的实验的时候，就感觉我们的实际动手能力还有待提高，面对一个实际的项目或者问题，不能够在最短的时间里找到最好的方法，还好有任务发布的时候老师给的一步一步的步骤，能够让我们有规可循，一步步按部就班的做下去，会有一些小地方的卡壳，但是不会没有方向什么的，这样我们的实验还可以稳定的继续下去，但是 `mapreduce` 编程经过一个学期的训练显然还是不够的，还需要进一步的加强，才能适应以后更多更复杂更广泛的课题，通过这个实验能够认识到这点还是对我们很有利的。下面的部分就要谈谈一些细节的部分的总结和反思。
- ② 有关 `pagerank` 的迭代次数的优化问题和程序编写，一开始我们就是简单的把次数设定为一个我们认定为会收敛的数字，虽然确实在图像上显示确实是接近于收敛了，但是并没有事实的证据证明这一点，于是我们思考如何能在程序内部自己判断迭代的结束，但是由于 `mapreduce` 的静态变量设置的不熟悉，导致一开始做不好，后来经过自主学习和查资料，在对运行机制进行了进一步的思考，在 `driver` 程序中设置了一个 `flag` 值决定迭代的停止，这样就能自主判断迭代的停止，虽然和一开始随便设置的次数相差无几，但是这样的有事实依据，由代码来判断终止条件的方式显然更加优秀和自主化，这对我们是个挑战也是很好的编程经验。
- ③ 在标签传播算法的迭代次数的优化部分，一开始我们也打算沿袭 `pagerank` 迭代次数判断的方法，判断到所有 `id` 的标签都不变为止，但是因为有二分图的问题，很难达到这一点，因为相邻两次的结果总是有所振动，所以这样的判断临界没有办法让程序终止。另外我们又想要采取异步判断一些相关值来判断程序的停止，但是由于 `mapreduce` 的特性和他的运行机制，我们只能进行同步的操作，所以没有想到合适的办法决定算法的停止。
于是我们采取输出每次的结果并作出趋势图的方式，根据每一次不同的标签的个数进行判断，当个数基本稳定而且数值很小的时候，我们有理由相信算法已经收敛，我们的标签传播算法已经达到了我们所需要的功能，能够得到我们可以探究的结果。
- ④ 有关算法的效率的部分的优化，一开始就是根据任务的要求一步步的写下去，用的很普通的写法一步步继续，等基本的结果和功能都完成之后，我们开始思考一下算法的效率问题，比如在 `reduce` 之前写一些 `combiner` 更加有效的利用网络的带宽，设置静态变量减少 `pagerank` 的迭代次数，设置一定的值确定标签传播的迭代次数以及一些其他的尝试，这些尝试有的成功，有的用处不大，但是给我们的经验就是无论如何要追求更高的算法效率，在现在一个程序的效率非常的重要，决定了一个项目的成败，不是简单的得到最后的结果而已，精益求精是一件很重要的事，我们现在的水平，完成基本的成果可以，优化能力还是有些不足。

8 结束语

通过这次的实验，我们通过 `Mapreduce` 挖掘出金庸的小说集中的角色的所谓“社交圈”，同时进行了数据的可视化处理，使得结果明显，同时在其中得出了很多有趣的规律和特点，进一步的增强了我们的 `Mapreduce` 的编程能力以及处理实际问题的能力，希望以后能多做一点类似的实验，获益良多，基本完成了开题的时候的预期和要求。
