图的可视化分析

——对哔哩哔哩弹幕视频网 2019 年“百大 UP 主”的关系分析

1. 选题说明

十多年前还是小众的亚文化社区的哔哩哔哩弹幕视频网（以下简称 B 站），自从 2018 在纳斯达克上市后发展越来越壮大，用户基数和日活跃用户数等数据不断上涨，许多例如央视新闻这样的官方平台也开始入驻。B 站优质内容的主要生产者是上传原创视频的“UP 主”，从 2018 年开始 B 站都会评选出当年最具有影响力的 100 位 UP 为当年的“百大 UP 主”（以下简称百大）。作为 B 站的资深用户，本人也关注了百大中的许多优质 UP 主，对于探究各个优秀 UP 主的关系十分感兴趣，所以选择了这个题目。另外探究百大的关系或许可以增进对 B 站社区文化的了解，挖掘出平常注意不到的信息。

2. 数据收集

2.1 收集思路

由于没有现成的数据，需要用爬虫来抓取一手数据。[1]

爬虫使用 python 编写（见文件 main.py），利用 b 站提供的 api 获取信息。Api 是从 github 项目“bilibili-API-collect”中获取的（项目地址：https://github.com/SocialSisterYi/bilibili-API-collect）。

2.2 收集方法

$prepare$: $api, Stack\ or\ Queue$,百大uid

$Stack/Queue$ ← 百大uid.

$While\ Stack/Queue\ not\ empty$:.

    $up \leftarrow Stack.top/Queue.front$.

    $get\ follow\_list\ \ of\ up$

    储存下up即其关注者信息.

    $for\ follow\ in\ follow\_list$:.

        $If\ follow\ not\ visited$:.

            $Stack/Queue \leftarrow follow$.

获取百大 uid 的方式是首先注册一个 b 站账号，关注所有 2019 年百大 up 主，通过 api 读取这个账号关注的所有人，就可以得到百大的 uid 了。这个爬取过程本身是一个图构建的过程：每个 uid 对应一个 up 主，可以作为结点，利用邻接表的方式储存边（即关注）的信息，最终会得到一个有向图，也就是说图的结构在爬虫阶段就能自动构成。

2.3 收集的局限性

上述过程在实践中出现了问题，在实践中，获取了 10000 个 up 主的信息后，队列/栈

[1] 注意，爬取数据的时间是五月初，到现在可能关注信息有变，但影响不会太大

中还有 70000 以上的 uid，且队列和栈的大小还在不断增加，没有收敛的迹象，说明我最后至少可以得到 80000 个 up 主的信息，但这些人中有许多并不是知名的 up 主（粉丝数少，影响力低），若在爬虫的过程中就根据粉丝数进行筛选会使效率较低，由于频繁访问我的请求也被服务器拒绝了。

在实验的时候我使用了广搜和深搜两种方法，但鉴于让这个获取过程收敛是很难的，所以决定不使用深度搜索的思路，只使用广度优先搜索，优先搜索与百大 up 主关系较近的，当搜索到一定人数时停止。

2.4 数据获取的结果

最终得到了 15000 位 UP 主的名称，分区于关注者信息用于分析。在文件 up_meta.json 中。

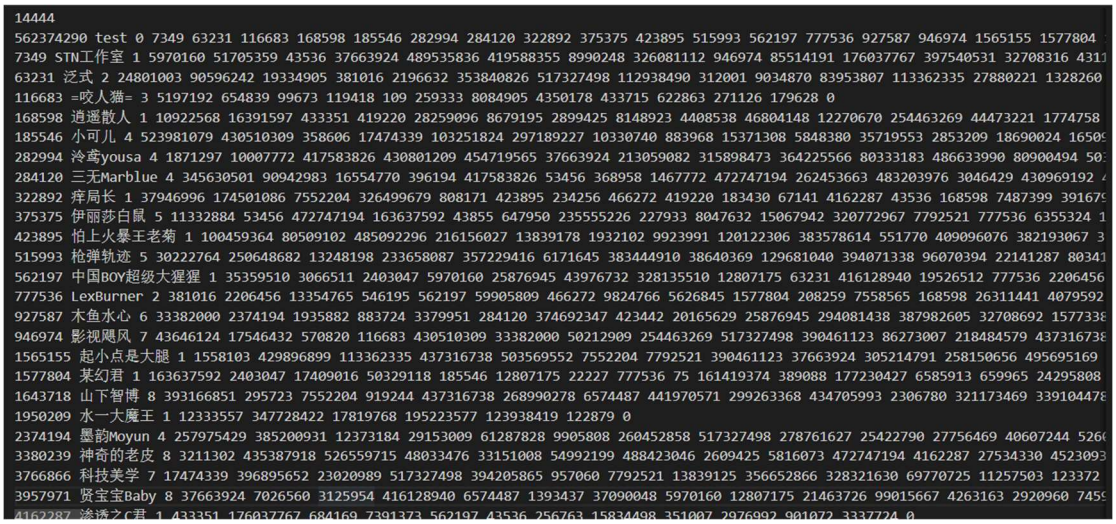
3 数据预处理与图构建

3.1 数据格式处理

进行图的算法编写时我更倾向于使用 C++,所以先将得到的原始数据转化为方便读取的格式。这一步还要：顺便筛选掉一部分没有关注任何人的 UP,因为没有关注任何人的话在图构建中是用不上的；把每个 UP 对应的分区转化成数字编号，方便分析。

把符合条件的 UP 主信息存入一个文本文件中，格式为：第一行是总人数，从第二行开始，每行为该行 up 的 uid+空格+up 名称+空格+以空格分隔的关注者 uid+0。当读到 0 时就结束一位 UP 主信息的读取。（数据处理代码在 convert_data.ipynb 中，得到的数据为 up_info_utf8.txt,由于 C++处理 utf-8 比较困难，所以把编码调整后得到 up_info.txt）

最终用于图构建的数据如下：



3.2 图构建

在 2.2 中说到在爬虫过程中就可以得到一个图结构，但是这个图结构肯定不是我们想要的，因为从现实来看，仅有每个 up 主关注的人并不足以判断他们是否认识还是单方面关注。但如果知道两个人相互关注，那我们基本可以判定他们是认识的。所以最终图构建的方法是，遍历每个 up 关注的人，若他关注的人也关注了他，就认为他们之间有边。如果一个 UP 和任何人之间都没有边，就把他去除。最终图中还留有 4560 个节点。

仅知道两个节点间有边是不够的，还需要知道权重。比如我要知道怎样最快的从一个 UP 主开始联系上一些 UP 主来参与活动，就需要得到最小支撑树。那么我应当设定权重越小时

两人关系更密切。最近 B 站有推出联合投稿功能，最好的方式是用联合投稿的次数来衡量关系密切程度，但是我并没有爬取联合投稿次数的方法，因此只能退而求其次，认为两个人共同关注的 UP 数量多就说明他们关系密切。这样的假设也是有合理性的，比如有几个 UP 主都是好朋友，他们之间都互相关注（他们几个在一个完全子图中），那么他们共同关注的数量肯定是大的。所以我们把共同关注的人数取倒数当作边权（ 边权范围$(0,1]$），若两个人没有共同关注就把边权置为 2。

4 图算法

4.1 最短路径：

对于 UP 主的最短路径，其现实意义是当一个 UP 主要与另一个合作时最短通过几个人可以找到他。在这个算法中我不打算考虑他们之间的权重，实现了一个不带权重的 Dijkstra 算法。算法的实现在函数 Node* find_way(int up1,int up2)中，使用时调用 void query_way(),在控制台输入想查询的两个 UP 主的 uid,会输出他们之间的路径，一次查询结果如下：
查找 UP 主中国 BOY 和半佛仙人间的路径:

```
输入两个up的uid，查找路径:562197 37663924
 中国BOY超级大猩猩 特效小哥studio 陈睿 硬核的半佛仙人
```

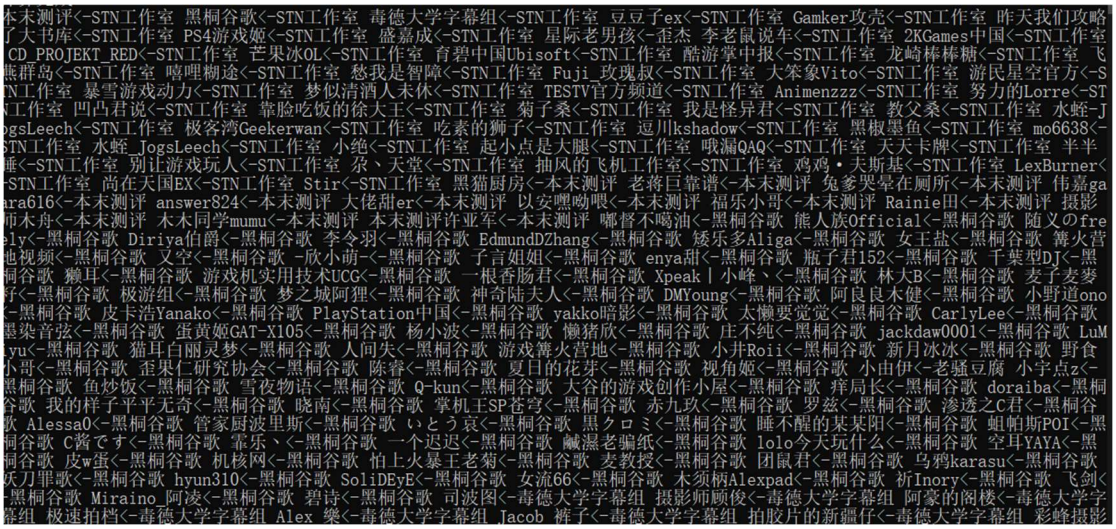
4.2 连通分量:

这个算法的实现是比较简单的，实现方法就是用广搜或者深搜中的任意一种方法搜索这个图到收敛，然后再遍历所有结点看有没有未被访问的，再从这个未被访问的开始搜索…每搜索一轮，都把节点储存在一个子图中(结构体 Graph),这样我们可以通过读取每个 Graph 来展示一个连通分支的结构，或者求这个连通分支的最小支撑树。求连通分量的算法实现在函数 get_branch()中，所有分支都会被存在 vector<Graph*>branch 中。

从我的数据中求出了 42 个连通分支，最大的连通分支有 4466 个节点，占总数的 97.9%。其他的连通分支有 2-5 个节点不等。

4.3 最小生成树

利用 4.2 中计算得到的各个分量和 3.2 中的权重，我们可以计算出每个连通分量的最小生成树，对一个 Graph 调用了 prim(Graph* G)后，可以用 Graph 的方法 show_tree()在控制台来展示最小生成树的结构，最大的生成树结构如下:



这个显示方式比较难看，之后可视化部分会有可视化的显示。

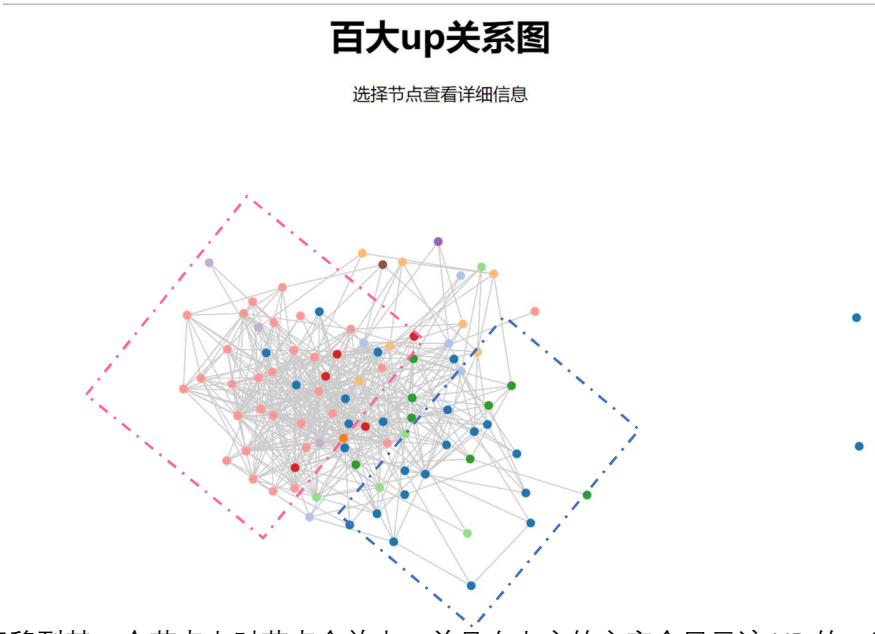
4.4 中心度

实现了计算了紧密中心度，方法比较粗暴，直接计算一个节点到其它节点的最短路径之和，就可以得到紧密中心度。在函数 cal_closeness()中。

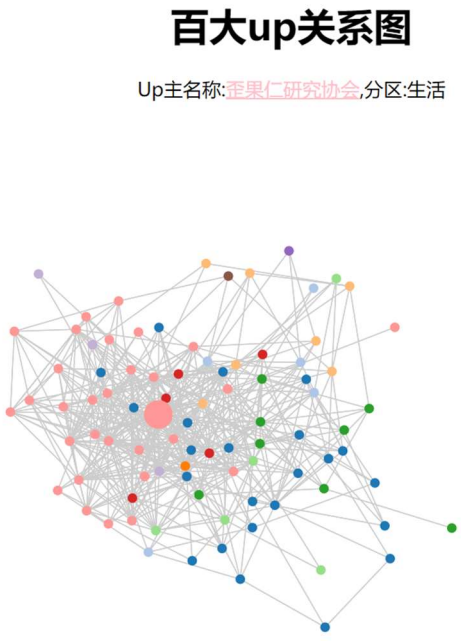
5 可视分析

5.1 百大 UP 关系图

选题初衷就是探究百大 UP 的关系，之所以除了百大 UP 之外搜集一些别的相关的 UP 主是为了获取更多的信息用于分析，比如给出权重或者进行路径查询等等。把 4560 个节点全部都进行可视化是很难的，所以我先仅可视化百大 UP 的关系图，使用 d3.js,可视化结果在 graph_big_100.html 中：



在鼠标移到某一个节点上时节点会放大，并且在上方的文字会显示该 UP 的一些信息，可以点击链接进入该 UP 主的 B 站空间查看详情，如下图：



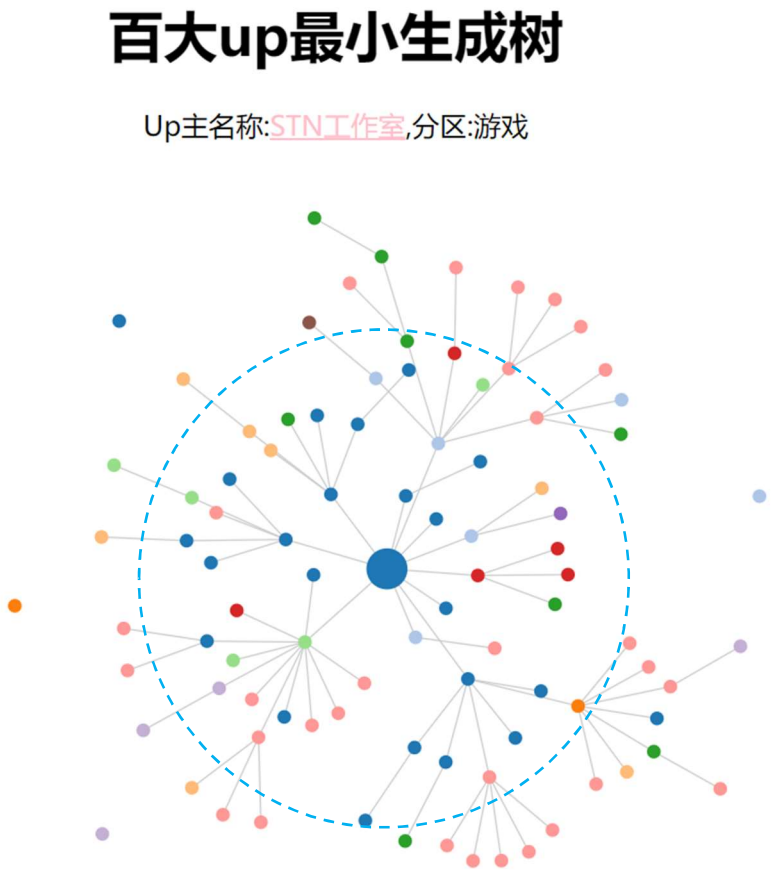
为了了解各个分区之间UP主的关系，根据不同的分区给UP主对应的节点染上了颜色，如果几个节点之间互相连接的边较多，那么这些节点很容易聚集到一起，他们间的连边也会显得很密集。如图所示，粉色，蓝色这两种数量最多的颜色有明显的聚集，粉色为生活区，蓝色为游戏区，说明这两个分区的UP主内部关系是比较紧密的，实际上他们确实会经常合作发布视频或者参加活动。

也可以发现其实粉色和蓝色两种颜色的节点之间的联系也很密切，有些蓝色节点几乎打入了粉色节点的“内部“，有可能是这两个分区之间互动频繁。可以查看几乎进入粉色内部的蓝色节点代表的UP主有“中国BOY”和”小潮院长“等，查看他们的投稿，中国BOY和小潮院长的生活区投稿都占到了总投稿的20%以上，他们也与主攻生活区的UP主有着合作。

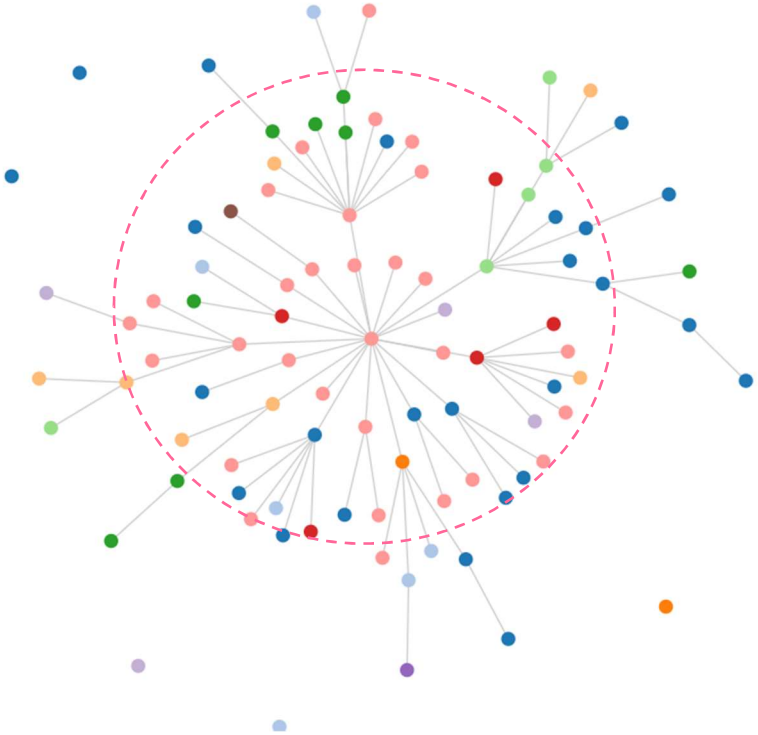
通过以上分析，可视化的图结构确实可以帮助发掘UP主的信息，经过验证这些分析都是正确的。如果将更多的节点加入图中进行可视化，可能可以发掘出更多的信息。

5.2 最小生成树

同样我可视化了百大间的最小生成树，我进行了两次prim算法，一次是以一位游戏区



UP主开始进行，另一次是以一位生活区UP主开始进行，会得到两棵不同的最小成生成树如下



最小生成树可视化界面的交互方式与 5.1 种图的界面相同。

本身最小生成树就不一定是唯一的，因为最小生成树算法只保证了边权之和是最小的，由于我的边权计算方式是共同关注数，所以出现很多边边权相同也是正常的。

可以看到若以游戏区（蓝色）UP 主为起点开始生成，蓝色节点都会占据中心的位置，以粉色节点为起点那么粉色节点就会占据中心位置，且可以发现蓝色节点“围绕”着粉色节点，也可以看出游戏区与生活区两个分区之间的关系密切。

5.3 紧密中心度

可视化之后发现紧密中心度的意义并不怎么大，在计算最短路径的时候我也发现了取两个百大 UP 主之间的路径，长度不会有特别大的区别，一般来说路径长为 5 左右就是比较长的了，说明百大 UP 主之间的联系是比较紧密的，紧密中心度的差别并没有特别大，可视化出来这些圆圈的大小也区别不大。

6 该项目的贡献与待改进的地方

编写了可以获取 2019 百大 UP 主即其相关 UP 主信息的爬虫，编写了处理爬取得到的原始数据的脚本。对处理后的数据进行整理和图的构建，最短路径算法可以找出两个 UP 主之间最短的“联络路线”，连通分支算法把整个非连通的图分成数个连通的子图，对于每个子图都可以调用最小生成树的算法来求的最小生成树。将百大 UP 主的节点单独导出进行了可视化，可以从可视化的图中发掘信息，如发现某些分区间的合作关系等。

当然这个项目有很多不完善的地方，由于能力有限没有办法把所有节点都可视化，导致有大量的数据没有用于分析，除了百大的节点还有约 4500 个节点可以发掘更多信息；没有进一步实现提高算法等等。如果再给我一次机会我一定会找人组队，考虑到疫情线上交流不便就打算一个人做，但一个人的力量终究是有限的，有很多想法比如 Community detection 并没有成功实现（写出了无法解决的 bug）。

7.对于环境与文件的说明

使用的 python 版本为 3.7.4，C++的集成开发环境为 VS2019。

bin 文件夹内为网页的文件，index.html 包含了三个可视化的图/

src 文件夹内 graphAnalysis.sln 是 C++的项目文件，graphAnalysis 文件夹内有所有用到的数据，生成的数据，和生成数据的脚本，spider 文件夹内是爬虫的脚本与爬虫得到的原始数据。

8.参考资料

爬虫：https://github.com/SocialSisterYi/bilibili-API-collect

D3: https://wiki.jikexueyuan.com/project/d3wiki/

https://observablehq.com/@d3/gallery