# x. 程序实现的方法

这次的大作业的程序是我（尚嘉宣）全程写的。下面我来介绍一下最终程序的结构和设计的思路。编程语言是C++ 11标准，未用第三方库，编译、链接为MSVC（VS2017 Community自带），Windows SDK 10.0.17134.0，最终版本中启用了AVX2指令集。运行用机器为我自己的电脑，硬件CPU Intel(R) Core i5-5257 2.70GHz，8GB RAM。

## x.1 输入和初始化

首先是需求。因为要对大量数据进行分析，一个高效率的框架是很重要的。先不说最终结果，但是最终程序，跑9种可能的输出的算法，3种分析原因数据统计的算法，总共用时大约2分钟。其中文件读取花取的时间大约为4s，后面的数据分析大约要一分半钟，输出半分钟。最终我把IO还是分散在了前面的过程，从而又减少了一点时间（用附加线程IO减少等待）。

然后是数据结构的选择。因为数据相当大，速度就相当重要。因为要计算两个无关系的人之间的共同朋友的性质，时间复杂度的量级大约是o(n^3)。在最初的分析中，首先要完成对完整的图的信息无损地读入。因为数据集的大小约是40k，人数是5k，所以这是一个很稀疏的网络（0.04 / 25），可以考虑直接使用邻接矩阵表示有向边的关系。

邻接矩阵的实现上，矩阵元上用一个结构体表示这个有向边上的所有联系的统计顺序表（以及这个有向边对应的总时长、总次数）对应的。同时这个网络信息结构体也包含了这个网络上的基本统计（总共有向边数，无向边数等），以及人对应下标的翻译用数组和下标翻译到人的数组（index[person]给出人在网络中的下标，person[index]给出网络下标对应的人。无效的person[index]置-1，均按index按person值的大小排列）。

为此对网络的分量进行了基本分析，发现网络的person并不是连续的（有约20人不存在），并且网络相当稀疏（有147个分量），并且边绝大多数为单向的（3万无向边，近3万有向边，差值只有约2800多）。

为此为了减小实际计算时的复杂度（o(n^3)量级），将整个网络拆成了147个互相完全没有联系的分量分别运算，最后再进行综合。这样复杂度由（无向图）变化为，可以证明这个数值一定会变小。在实际操作中，直接读入的图时内存占用最高826.1mb，分解之后变为126mb左右。明显不考虑综合已经计算出的数据的时间（量级小于，对总的复杂度来说没有贡献），实际运行中使复杂度降低了几倍。

## x.2 运算处理的架构

在最初设计架构的时候，我的想法是我来负责整个程序的架构，提供接口给其他人。其他人只要写一个函数根据关系矩阵计算给定坐标在矩阵中，计算边被删除可能性权重，得到一个double的函数就可以了。

这样就需要对这个接口和接口后面的内容进行设计。因为可能需要有很多算法进行测试，需要对大量函数进行组织，需要有一个管理算法的模块。同时因为要做很多互相完全不相关的相似复杂操作，应该有一个线程管理的模块。（数据结构，数据结构的移动和拷贝的构造函数在前面已经解决，又是一个模块，但在这里就不讨论了。）

算法管理最终解决方案是这样的。用一个纯static 的抽象Algorithms类，把所有这些内容封装起来（代码会自动被inline）。不用namespace再手动加上inline的主要原因是1）有些函数不适合在外部调用，把它们作为private或者protected会比较合适。2）类是一个整体的，因为这些函数之间联系相当大，写作一个类可以帮助编译器自动作inline的优化，减少部分代码重用带来的效率降低。然后用一个数组保存最后外部调用的相同作用的函数的指针，需要用的时候只要用下标调用函数指针就可以了。（除了最终算法以外，还有一些分析数据输出的函数也是这样封装在一个constexpr的数组里面进行轮流调用的）

对于线程管理，也采用了类似的模式，并使用了模板让输入输出可以变化。因为每个算法都是对很多个网络分量的分别计算，主要的操作就是异步运算一个数组。最开始的程序是主动控制创建进程等到全部算完返回。主线程负责管理一个布尔型数组（标准库的vector不能在这里使用，因为不是布尔型的数组，从而无法传递&lock[j]作为布尔型的指针），在创建线程的时候对应置为true，子线程置为false完成是否完成运算的通信方式，主线程负责监视完成情况。后来发现这样效率很低下，又发现C++ 11自带异步运算的库<future>，就让管理函数只负责创建vector<future> 和 array<future>，需要的时候get回来整个数组就可以了。（分解整个网络的时候也用到了这个操作）

这样整个操作内容就全部都被封装在了几个类里面，而异步运行的顺序则在main函数里指出。因此IO除了初始化的读入写出以外都是在main函数里完成的，并且之间的关联基本没有，inline优化又因为异步没有意义，都放在了main函数前面，暂时没有整理，直接在main函数里用一个队列管理异步，减少对IO设备的等待。

## x.3 可能的优化和代码归档位置

1. 可以在双向边的计算的管理函数中少分配一半空间，把指针数组的值调整到合适位置即可

2. 减少主线程等待其他线程完成运算时间的计算顺序的优化

3. 初始读入的时候可以使用其他数据结构减少初始的巨大内存需求和浪费

代码可以在如下网址找到

<https://github.com/NetworksProject/NetworksPrediction>