复杂网络大师赛第四名技术分享（篇一：思路与分析）

2017-12-28 宾狗 学术 复杂网络

0x00 背景

复杂网络一直是个很有意思的领域，虽然它现在没有深度学习那么火爆，但这么多年来一直在稳步的发展，更是与人们生活的方方面面都息息相关，计算机科学自然不用多说，物理化学、生物医药甚至传播学都有复杂网络的应用场景。而且，复杂网络的论文大多都发表在物理相关的期刊上，这也侧面说明了复杂网络和物理学的紧密联系。

回到正题，此次复杂网络大师赛是由数据城堡举办，主题是寻找复杂网络中的关键节点，并按照节点重要性程度从高到低排序，给出一个节点重要性序列。比赛链接在此，提供了8个规模不一的复杂网络，下载链接点这里。

0x01 一般思路

寻找复杂网络中的关键节点是一个由来以久的问题，前人做过的研究多如牛毛。想快速了解的话可以看看吕琳媛老师和她学生的综述文章，非常全面的从各个方面阐述了重要节点排序的各类方法。

在我的理解中，对节点重要性进行排序本质上是一个最优化的问题：只要给定了评价指标，我们穷举出所有可能的序列，依次进行评估就能够找到最优解。当然这种方法也只是理论上可行，即使网络中只有100个节点，穷举的空间也是爆炸的，更不用说本次比赛提供的复杂网络，规模最小的也有40万个节点。

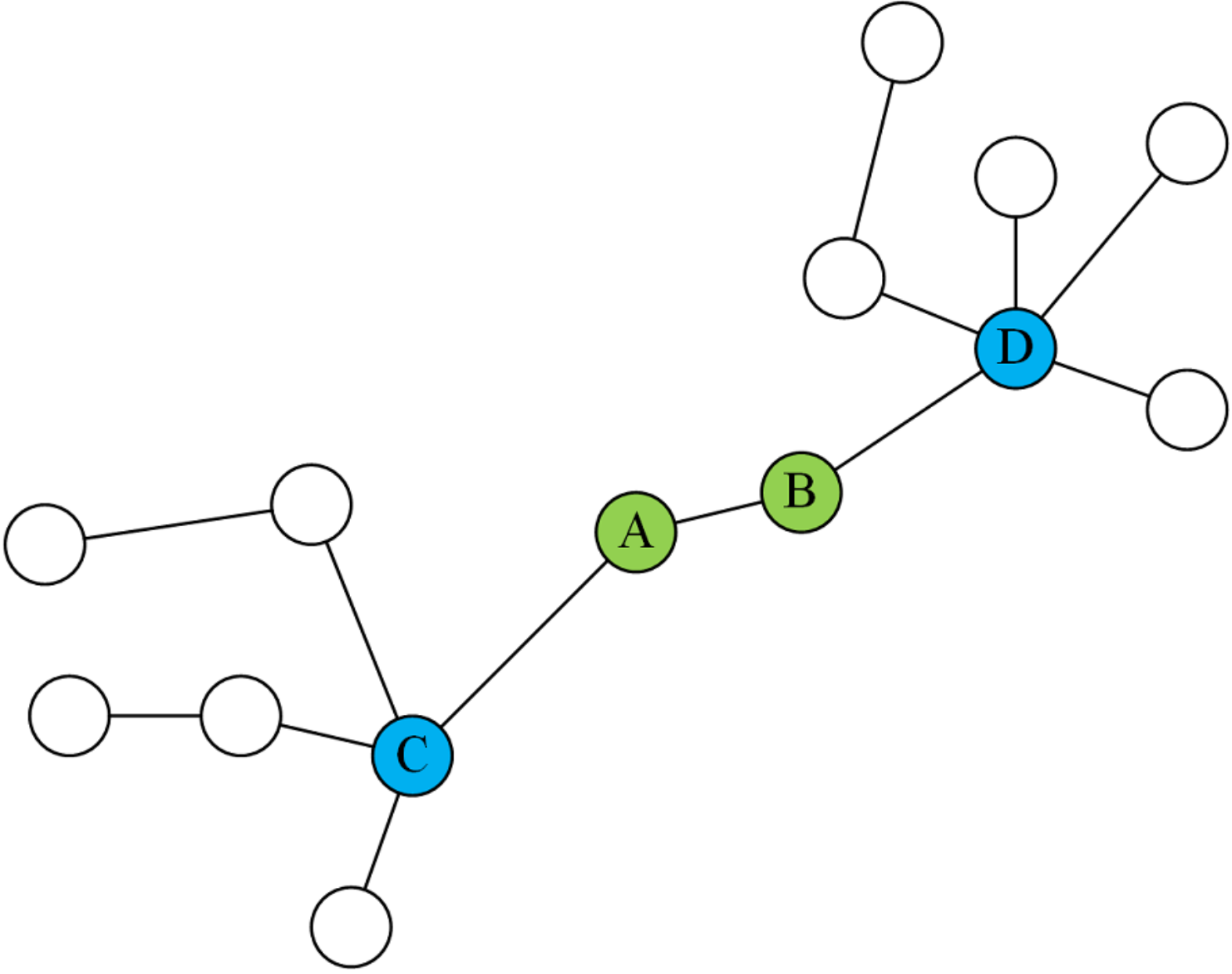
在刚才提到的综述文章中，介绍了许多比较常见的算法，有基于度的，基于`k-shell`分解的，基于介数的，还有大名鼎鼎的`PageRank`算法。这些方法虽然在真实场景中各有各的局限性，但其算法思想是非常简洁非常美的，无论大家是否有志于做复杂网络领域的研究，我觉得都有必要好好了解一下。

除了综述文章中的提到的方法，我也找了很多近几年的论文，看了不少老外做的研究。但是经过实验验证，除了`PageRank`算法和`LeaderRank`算法(吕琳媛老师提出的`PageRank`算法的改进)，其他方法都是槽点满满。最为普遍的一个问题就是效率问题，很多人折腾出一个看起来很高大上，碾压别人结果的算法，结果一看实验部分，所测试的网络规模都是几百到几千个结点左右的……那基本上不用多说了，几十万节点的网络肯定没法跑。

除此之外，我们还需要重点关注一下节点重要性排序的评价指标，这部分在刚才提到的综述文章中也有涉及，一般来说有两种评价方法：第一种是用网络鲁棒性和脆弱性评价排序算法，第二种是用传播动力学评价排序算法。第一种方法的思路比较直接，如果一个节点越重要，那么移除它以后对网络的破坏性就越大，那么我们通过不断移除节点，观察网络中最大连通集团规模的变化情况，就可以“客观”的评价出某个节点的重要性程度。本次大师赛采用的就是这种评价指标，后面我们会详细讲讲该算法的实现。那么第二种评价指标主要是在研究传染病模型、社交网络中的信息传播过程中使用较多，包括SIS模型和SIR模型，这里就不继续深入了。

0x02 PageRank算法+贪心策略

上一小节说到，除了`PageRank`算法和`LeaderRank`算法，其他算法都很难在评价指标和时间效率上实现双赢。但是事实证明，在此次比赛的评价指标下，直接使用`PageRank`算法的效果并不好。为什么呢？我们来看如下情形：



在上图中，毫无疑问`PageRank`值最高的两个节点为A和B，但是如果我们按照评价指标逐个删除时就会发现，当删除掉第一个节点A之后，第二个节点B的重要性似乎并没有那么高了。反而是删掉节点C或D能够使复杂网络的最大连通子图规模下降的更快。这仅仅是简单的示例，在真实网络中必然也存在类似情况：即删除某个节点后，导致整个网络中大量节点的`PageRank`值发生剧烈变化，使得原节点重要性序列不再可靠。

那么怎么办？可以采用贪心策略，最暴力的做法是每删除一个节点，重新计算一次`PageRank`值。但是很遗憾，虽然`PageRank`算法的效率非常高，但是重复计算几十万到几百万次也不是我普通PC能够承受的。因此，我们把删除的节点数放宽到500，即每删除500个节点，重新计算一下剩余网络中节点的`PageRank`值，然后再重复这个过程即可。

对于普通玩家来说，能够做到这一步，结果已经很不错了，大概能排到30名左右。但是如果要继续向前推进，那还需要从其他角度去思考问题。

0x03 从评价算法的实现谈起

本次比赛提供了评价算法的思路和源码是一个非常重要的`hint`，如果没有认真看将是一个非常大的损失，大赛评价算法的页面点这里。

简单概述一下算法的基本思想，本次比赛将网络的鲁棒性(Robustness)作为评价指标，给定的一个节点重要性序列(重要性从高到低排列)，逐个从中删除节点，每删除一个节点，计算当前网络最大连通集团规模。

如果按照字面上的理解，直接去实现这样的一个算法当然是可以的，求最大连通分量大多使用深度优先搜索算法，虽然时间复杂度不算高(一般为$O(N+M)$，其中N为节点数量，M为边数量)，但是在百万级别的网络规模下，仍然是一笔不小的时间开销，何况每删除一个节点就需要计算一次。因此，这里采用了一种“逆向思维”的方式：不从已有网络中“删节点”，而是从零开始构建一个网络，每一步变为“添节点”。这两种方式是完全等价的，最后算出来的结果也是一致的。

我根据官方的代码和描述，写了一个不那么规范的伪代码(重在领会思想)，如下所示：

算法 **基于构建网络的鲁棒值算法**

输入： 网络G，节点数量n，节点重要性(由高到低)序列V

输出： 鲁棒值R

步骤：

1. $R \leftarrow 0$
2. $maxclustersize \leftarrow 0$
3. **for** $i \leftarrow n-1$ **to** 0 **do**
4. $v_i \leftarrow V[i]$

   /*从网络G中获取v_i的所有邻居节点*/
5. $neighbors \leftarrow getNeighbors(v_i, G)$
6. $clusterArray \leftarrow$ empty array

   /*对每个邻居节点判断其是否在某集团中，若是则将该集团放入$clusterArray$*/
7. **for each** vertex $v_j \in neighbors$ **do**
8. **if** v_j in one cluster **then**
9. $c_k \leftarrow getCluster(v_j)$
10. $clusterArray \leftarrow appendToArray(c_k, clusterArray)$
11. **end if**
12. **end for**

    /*若$clusterArray$为空，新建一个空集团；否则，合并其中所有集团*/
13. **if** $clusterArray$ is empty **then**
14. $c_l \leftarrow$ new empty cluster
15. **else**
16. $c_l \leftarrow mergeAllCluster(clusterArray)$
17. **end if**

    /*将节点v_i加入集团c_l，获取集团c_l的规模$csize$，并更新最大集团规模*/
18. $c_l \leftarrow addToCluster(v_i, c_l)$
19. $csize \leftarrow getClusterSize(c_l)$
20. **if** $csize > maxclustersize$ **then**
21. $maxclustersize \leftarrow csize$
22. **end if**
23. $R \leftarrow R + maxclustersize$
24. **end for**
26. $R \leftarrow (R-n)/n^2$

官方提供的版本是用`groovy`写的，实现中有一些细节差异。运行效率上差强人意，在普通PC大概耗时30秒左右可完成8个网络的鲁棒值的计算。

0x04 将贪心策略进行到底

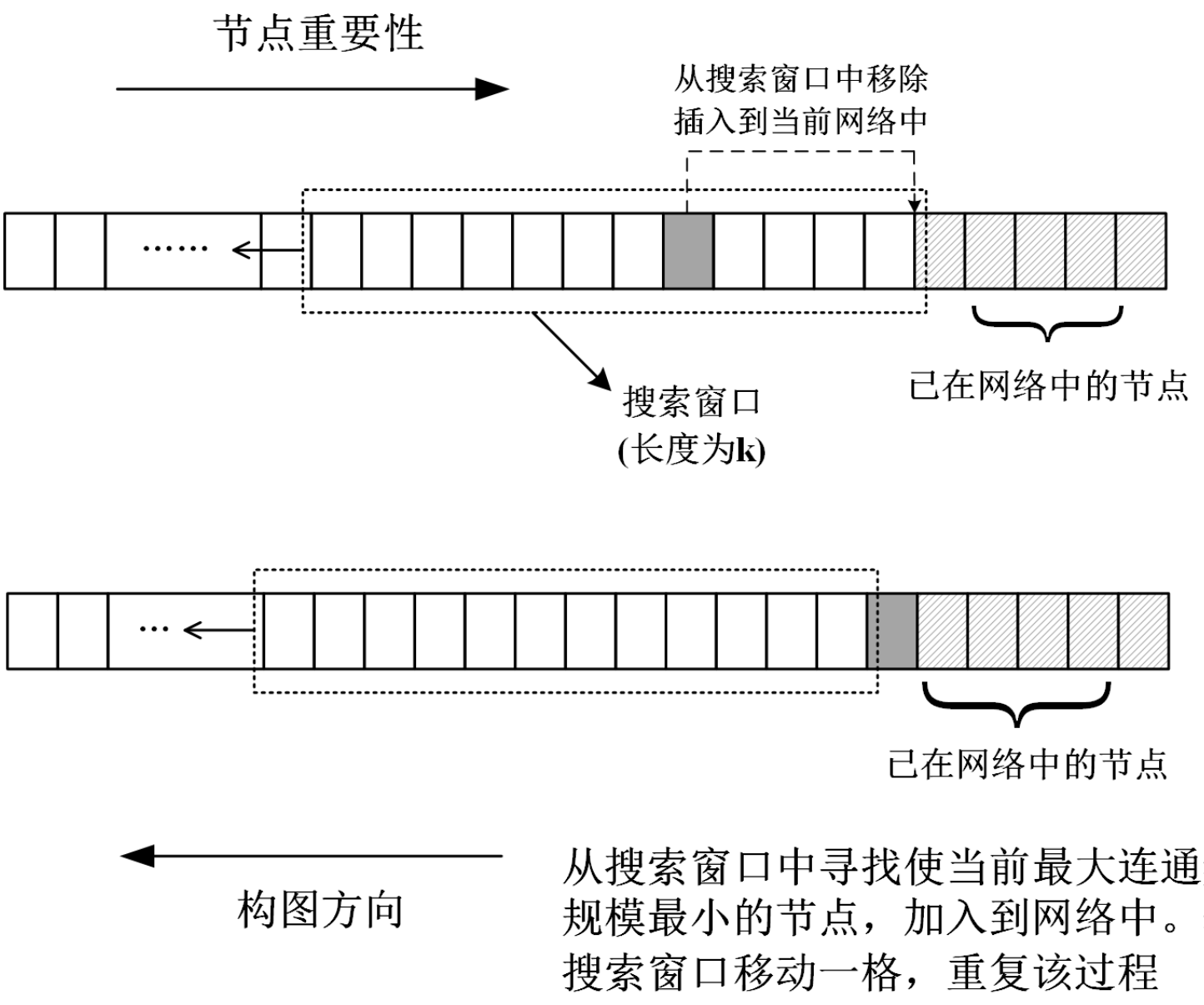
前面我们利用`PageRank`算法和贪心策略得到了一个初步的节点重要性序列。但是它并不完美，原因是多方面的：包括我们每删除500个节点才重新计算一次`PageRank`，也包括评价指标本身并非完全科学(或者说评价指标所期望的和`PageRank`的最终目的并不一致，`PageRank`算法是建立在一个静态网络的基础之上，而评价指标在不断的删除节点，本质是一个动态变化的网络)。总而言之，我们第一步得到的节点重要性序列虽然算不上完美，但离最优解“并不遥远”，我们可以在它的基础上做一些微调优化，尽可能逼近最优解。

其实看完刚才评价算法的介绍以后，稍微敏锐一点的同学立马就能想到：既然能用“构图”的方式来计算鲁棒值，那是不是也可以用“构图”的方式来得到一个更优的序列？是的，完全可以，这正是我所使用方法，且同样是一种基于贪心策略的方法。

这个方法出发点非常直观：从重要性最低的节点开始构图，我们希望每一步所添加进来的节点**对当前网络最大集团规模的影响最小**。也就是说，在符合算法规则的基础上，这个节点要么自己新建一个集团，要么加入到其他较

小的集团中，这样对当前最大集团规模不构成影响。如果一定要加入到当前最大集团中，那么也应该尽量使集团规模只增加1或其它较小的数值。

那么如何寻找这样的一个节点呢？直接在所有节点中寻找当然是可以的，但寻找范围过于庞大，有时很多计算是完全无必要的。例如整个网络中`PageRank`值最高的节点，显然是不可能在早期构图阶段成为候选节点并加入到网络中的。因此，我设置了一个搜索窗口，如下图所示：



其中搜索窗口为一个动态链表，其长度为k，我们每一步都从搜索窗口中寻找一个对当前连通集团规模影响最小的节点，从链表中删除并加入到当前网络中。然后搜索窗口沿着前进方向移动一格(实际上就是从节点序列中读取一个新节点，插入到链表头部)，重复该过程即可。

当然，在这个过程中，k的取值至关重要，k较小时算法运行快但效果差，k越大效果越好但所需时间较长。不过k也并非越大越好，这个我们后续再详细说。

0x05 小结

以上就是我在大师赛中的思路与方法，与那些高大上的方法相比是不是显得有些low？其实我本人也觉得这个方法有些“笨”，显得不是那么的“学术”。总的来看，我只做了两件事：

- 用`PageRank`算法和贪心策略得到了一个看起来还不错的节点重要性序列
- 受大赛官方评价算法启发，采用逆向思维，在构建图的过程中继续使用贪心策略，对第一步得到的节点重要性序列进行优化

但是，真正让我觉得自己的工作略有价值的部分不在于思路部分，而是在工程实现上。正如前文所提到的，官方提供的评价算法是用`groovy`实现的(实际就是`Java`)，效率并不算高。可问题在于，我们后续的算法基本就是建立在评价算法的基础之上，需要频繁的调用并计算。

所以，如果直接在`groovy`版本的代码上改造并实现我们的算法，除非使用非常小的k值，否则最终的时间成本几乎是不可接受的，但使用较小k值又势必会影响我们最终的结果。

这样似乎只剩下一种解决途径：用更高效的语言从头实现一遍评价算法。这也是我们下一篇文章要谈的，请继续关注本系列的第二篇文章《复杂网络大师赛第四名技术分享（篇二：工程技巧的胜利）》

参考资料

- 网络重要节点排序方法综述