哈尔滨工业大学计算机科学与技术学院

《算法设计与分析》

课程报告

|  |  |
| --- | --- |
| 学号 | 1190201215 |
| 姓名 | 冯开来 |
| 报告日期 | 2021年1月12日 |

# 论文题目

# Finding Temporal Influential Users over Evolving Social Networks

2019 IEEE 35th International Conference on Data Engineering (ICDE)

Shixun Huang †, Zhifeng Bao †, J.Shane Culpepper †, Bang Zhang ‡ †RMIT University, Melbourne, Australia ‡Damo Academy Alibaba, Hangzhou, China †{shixun.huang, zhifeng.bao, shane.culpepper}@rmit.edu.au, ‡zhangbang.zb@alibaba-inc.com

# 论文阅读报告

## 解决的问题

* 背景：

社交网络中人们的影响力问题一直是非常重要的研究课题。目前研究的内容主要是如何在各种扩散模型下找到网络图中影响力最大的目标用户的最小集合（称为种子集）。

在社交网络影响最大化问题中，网络中的结点一般只有两种状态:激活状态(受影响)和未激活状态(未受影响)。结点间的影响概率遵循-一定的规则，称这些规则为影响传播模型。独立级联(IC, Independent Cascase)模型和线性阈值(LT, Linear Threshold)模型分别被Goldenberg和Granoertr提出，因为他们能够很好地反应现实网络中结点间的影响，而且非常简单，成为最受关注的两种影响传播模型。

独立级联模型。独立级联模型是被研究者广泛使用的一种影响传播模型，该模型中每条边都包含一个影响概率，表示出边结点激活入边结点的概率，设参数puv表示结点u对结点v的激活概率。当结点u去尝试影响v时，无论v是否被影响，u只能一次尝试去影响v;而且各个结点之间的影响是独立的，即对v的激活不会受其他节点的影响。种子集合S传播的过程类似于图的广度优先遍历。

线性阈值模型。线性阈值模型考虑到用户的坚持度，在独立级联模型的基础上又引入一个参数0。,表示结点v的坚持度。对于结点v，如果其所有已激活入边邻居u(u∈inNeighbor.)的激活概率之和EPuv大于该结点的坚持度0。，则激活该结点v。对于给定的种子集合S,该模型的激活结果是确定的。结点的激活过程也可以理解一次图的广度优先遍历，

本篇论文是在独立级联模型下研究影响力的问题。

* 当前研究存在的问题Q：

随着时间推移种子集会发生变化。所以虽然是围绕着动态社交网络图进行研究，但是得到的种子集不能保证其在一段时间内都是最好的。

* 目前已有的解决问题Q的一些算法的缺点：
* 找到的种子集可能会重复影响社交网络图中的某些人而使真实的影响范围受到限制
* 随着时间变化更新种子集的频率太高不能最大化影响效率
* 算法占用大量的内存
* 算法不能提高最坏情况下的时间复杂度
* 本篇论文解决的问题和设计思路：

解决上述说明的目前已存在算法出现的各种问题；

主要设计思路是：

* 针对影响效率：找到一个固定的种子集，该种子集可以在某一段连续的时间内影响最多的人，既可以提高真实的影响力，又不会使种子集更新频率太高
* 针对内存高消耗：采用压缩图的方式存储各种数据，减少内存消耗

## 采用什么思想解决这个问题

总的来说，本篇论文采用贪心算法：每次寻找所有点中，平均可达性最高的点——能够活跃可到达节点最多的点，加入种子集中，即根据贪心策略，寻找当前最好的决策，一步一步贪下来，那么最后得到的固定种子集一定是最优解。

而在寻找平均可达性最高的点的过程中，是在独立级联(IC)模型——一种经典且广泛采用的信息扩散模型的变体下解决问题。将不断发展的社交网络建模为一系列进化阶段，如一组快照图的形式，优化基于子图的策略来解决DIM问题。

同时，文章中证明了使用IC模型分别模拟每个快照上种子集S的扩散过程后，得到的由种子集S激活的节点集上的分布，与通过蒙特卡洛模拟随机生成子图中从种子集可到达的节点集的分布相同。

因此，为了在随机扩散模型下捕获不同影响扩散的随机性，可以通过蒙特卡洛模拟从社交网络的快照中随机生成子图并通过平均它们在子图上的不同可达性来近似用户的影响差异。

接着，为了缓解生成子图引起的高内存使用问题，论文中提出了两种策略——基于水平压缩的策略(HCS)和基于垂直压缩的策略(VCS)，实现以不同的方式压缩生成的子图，然后在相应压缩图上识别目标用户，获得种子集从而解决DIM问题。

## 基于水平压缩的策略(HCS)和基于垂直压缩的策略(VCS)

**（算法或证明过程）**

* **水平压缩（HCS）**
* 在压缩图中，每个节点每条边均有一个包含位集(a containment bitset),用于存储该水平实例中哪个子图有这个节点或这条边；

假定我们有一个水平实例，初始化一个空的水平压缩图和一个大小为的空位集，即，接着我们扫描每一个生成子图，来迭代的构造水平压缩图；当扫描到子图时，如果中的一条边e不在中，那么就将边e加入，并设置，若存在于中，则直接设置。

* 由于生成子图的过程中，只涉及除去边，而不是节点，因此我们不妨认为每个子图中都包含所有的节点，只不过是有的节点与其余节点没有连接边。

所以，可以构造节点的包含位集，并且节点位集中所有位都是1，

* 在最终构造好的压缩图中，包含了水平实例中的所有节点和边，并且解决的节点或边的重复问题；
* 算法：

Algorithm 1:The Horizontal-Compression-Based Strategy (HCS)

Input : Snapshots ,k(需要找到的种子集的元素个数) and R(每个快照图生成了R个子图)；

Output : The seed set S(种子集)；

S，=∅； //初始化种子集为空集，水平压缩图集合为空集

For j=1 to R do{ //循环R次，构造R个水平压缩图

<- (=∅，=∅)；//初始化第j个水平实例的压缩图，顶点集和边集均为空

For i=1 to  do //循环次

Generate subgraph  to update ;//生成子图，迭代更新所在水平压缩图

Add  into  //将构造好的第j个水平压缩图加入压缩图集合中

}

While |S|<k do {

t<-;//利用HCA算法找到压缩图中平均不同可达性最大的节点

S<-S∪{t}; //加入要找的固定种子集中

Update(,t);//更新压缩图集合，即从压缩图中去掉加入种子集的节点

}

Return S;//返回得到的种子集

总的来说，共有四个步骤：

1. 构造压缩图；(对应于算法1中的第一个for循环)
2. 使用HCA算法计算压缩图中每个节点的平均不同可达性；
3. 迭代的将有最大平均不同可达性的节点加入种子集中；(体现贪心算法)
4. 每次选定一个节点放入种子集后更新压缩图；

* **垂直压缩（VCS）**

VCS与HCS设计思路相同，只有图形的压缩方式不同。由于篇幅原因，不再赘述。

**以下介绍两种算法的具体设计。**

## 2.3.1HORIZONTAL-COMPRESSION-BASED ALGORITHM (HCA)

**1.前提假定：**

* 有一组快照序列,以及他们每个快照都有R个生成子图。用表示快照的第j个生成子图，其中，



* 子图集合表示第j个水平压缩图实例，是每一个快照图中第j个生成子图的合成，因为影响范围的近似不同，我们可以在水平实例上来估计种子集的平均不同可达性：



说明：针对每一组水平实例，先找到每个子图中种子集可到达的节点集合，但由于相同的节点只算一次，因此将水平实例中找到的可达节点集合做并运算，这样便保证了重复节点只算一次；接着将所有水平实例的可达节点数做和，再平均在每个水平实例中，即可得种子集的平均不同可达性；

* 使用贪心算法，迭代的将每次有最大平均不同可达性的节点放入种子集中；

在提出的HCA算法中，我们采用了BFS广度优先遍历的方法来求水平实例中每个子图的每个节点的平均不同可达性；

**2.算法描述**

* **数据结构，**

记为第j个水平实例的压缩图；

* 遍历位集(Traversal bitset)

队列中的每个节点u都有一个遍历位集(a traversal bitset)，用来描述哪个子图可以继续从节点u开始遍历它的邻接节点；当且仅当全为0时，从u开始的任何遍历都不被允许；

* 本地包含位集(Local containment bitset)

每一个可达的结点u都有一个本地包含位集(a local containment bitset)，初始值为u的包含位集值；用来描述哪个子图有节点u且未被访问；如果子图成功访问节点u，那么需要设置来保证子图不会再经过节点u；

当且仅当以下三个进行AND位运算结果不是0时，才可以说能够从节点u遍历到邻接节点w：

* [三条遍历规则]

 ->节点u在子图中存在

 ->边u->w在子图中存在

 ->节点w在子图中存在且未被访问过

那么，结果b便存储了哪些子图可以成功从节点u遍历到节点w的信息；

同时，结果b也是节点w更新的遍历位集，即表示了在哪些子图中可以从节点w开始遍历它的邻接节点；

如果b非0，我们也将与b进行异或运算来更新本地包含位集(即将成功从u遍历到w的对应位更新为0，表示w节点已被访问过，保证w不会再被访问，为了满足重复节点只算一次的要求)，之后将节点u和更新后的w的遍历位集b分别进队列中存放；

* **伪代码**

Input ：R个压缩子图 ，以及开始的节点v

Output ：节点v的平均不同可达性

d<-0; //存储节点v可到达节点数，初始化为0

For j=1 to R do{ //循环R次，即在R个子图中都要找可达节点

If  in 0 then { //能够从节点v开始遍历邻接点的一个前提是节点v还存在于压缩图中

d<-d+1;//节点v本身也算可达节点，因此加一

<-a queue initialized with node v;//用来存放被访问过的节点

<-a queue initialized with  in ;//用来存放相应的遍历到的邻接点的遍历位集

While  is not empty do {

u<- dequeue from ;//利用队列，BFS广度优先遍历邻接点

<- dequeue from ;//取出其邻接节点更新的遍历位集

For each edge(u,x)∈ do{ //遍历压缩图中所有与节点u有关的边

If x is not visited in  then

<-  in ;//此处是将的含义分两部分(一是存在于压缩图中，二是未被访问过)来初始化。If语句中的条件保证了x未被访问过，那么便直接取决于是否存在于压缩图中，即

b<- &  in  & ;//做AND运算

If b ≠ 0 then {

If = in  then

d<-d+1;//如果x的本地位集全为1，那么说明节点x未被计数一次过，此时需要对d加一，即可达节点x

<-;//x的本地位集与结果b做异或运算，用来更新节点x的本地位集

<-enqueue the node x;//将访问过的结点x进队列，便于BFS

<-enqueue the bitset b;//将结果b，即节点x更新的遍历位集进队列，便于下次从节点x开始遍历邻接节点

}

}

}

}

}

Return d/R;//返回结点v的平均不同可达性

具体算法描述：

1. 初始化计数器d为0；
2. 在每一个水平压缩图中，如果节点v在压缩图中不存在，则直接循环到下一个压缩图；

如果节点v在压缩图中存在，就从初始节点v开始，因为v是起始节点，遍历的整个过程是从v开始的，因此只要它存在于压缩图中，就可以从v开始遍历它的邻接节点，所以可以初始化它的遍历位集为它的包含位集；

1. 计数器d的值加一(由于节点v本身也算其可达节点)，初始化队列有节点v，队列为节点v的包含位集；(使用BFS广度优先遍历)当队列不为空时，就取出队首节点赋给节点u，此时该节点是从v开始遍历的可达节点，且被访问过；接着取出队列中对应的位集作为该节点的遍历位集；
2. 对于与u有关的边(u,x)，如果节点x未被访问过(不在队列)，那么节点x的本地位集将只取决于节点x的包含位集，因此<-  in ;

接着按三条遍历规则做AND运算，结果赋给b，若b为0，即位集所有位均为0，说明该压缩图的水平实例中没有任何一个子图成功从节点u遍历到了节点x，那么节点x将不具备继续往下遍历的资格，因此不做任何位集的更新，也不做任何队列元素的更新，直接循环至下一个边；

若b不为0，则说明在该水平压缩图的水平实例中，有子图可以成功的从节点u遍历到节点x，因此将x的本地位集与b做异或运算来更新本地位集，即将所有能够成功遍历到x的子图对应的位改为0，表示在子图中，节点x已被访问过(这样一来循环多次AND运算后，总会出现b为0的情况，导致没有节点进队列，最终队列为空算法终止)；接着将节点x进队列，结果b(x的遍历位集)进队列；

另外需要判断此时与是否相等，由于全为1，那么，如果两者相等，说明节点x一次都未被访问过，而此时节点x可达，那么计数器需要加1；如果两者不相等，即中有0位，则说明之前遍历边的过程中，曾经访问过节点x且已算入可达节点中，为了满足重复节点只算一次的前提设定，此时计数器不再加1；

1. 重复步骤4直至遍历完所有与u有关的边；
2. 重复步骤2,3,4,5，直至找完所有R个压缩图；
3. 返回平均不同可达性d/R,算法终止；

可计算，其中cond指

在每一次迭代中，使用BFS，从包含位非0的开始节点v开始，初始化两个队列，Q1用于存储被访问过的节点，初始化为节点v，Q2用来存储邻接节点更新的遍历位集，初始化为节点v的包含位集，逐个处理队列Q1中的节点直至为空；每次处理都取Q1中取队首节点，从Q2中取相应的遍历位值，接着进行AND运算判断可否遍历其邻接节点，如果可以，就更新其邻接节点的本地包含位值，并使邻接节点及其遍历位值进队列，并将初始值为1的可达性d加1；最后返回平均不同可达性即可；

至于更新压缩图，采用和HCA 算法同样的遍历方法，不同在于每次更新的是包含位集；

1. **算法分析的结论**

* **正确性分析**

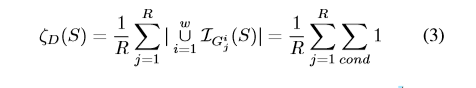
只需要证明公式(2)与(1)相等即可；

给定水平实例，和压缩图，

引理：如果在中节点u可达x，那么在也可以；

证明：被遍历的每一条边(a,b),更新的遍历位是三个遍历规则进行AND运算得到的，如果在，节点u可达v，那么在中一定有一条路径p，其上每个边e和每个节点z都有，由于位集是由初始化的，那么第一次遍历从u到v的路径p时的第i位定为1，因此中u可达v；

而在压缩图中种子集可达所有在水平实例的任何独立子图中可达的所有节点，同时由于对重复可达节点我们只计数一次，所以有



* **空间复杂度分析**

针对快照序列，我们可以进行快照的合成：，其中，有，那么水平实例的压缩图中最多共有的节点和边，假定每一个大小为的包含集使用了的空间，那么一个水平压缩图总共消耗了空间；

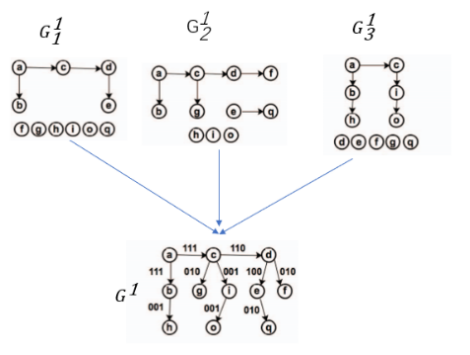
* **时间复杂度分析**

压缩图中每个独立的子图在遍历时至多遍历的边，而进行AND运算有固定的时间，因此，对于R个压缩图，HCA算法时间复杂度为，设为快照序列中的平均边数，那么算法一在构造R个压缩图时的时间复杂度为，调用HCA算法次，总共更新压缩图次，因此HCS策略的时间复杂度为

## 2.3.2 The Vertical-Compression-Based Strategy (VCS)

1. **垂直压缩图的方法**

利用蒙特卡罗模拟方法生成某一时刻的R个子图（R个社交网络图），将R个子图压缩到一个图中，记为，R个子图分别记为，举例如下（时刻）：



其中在图中省略了每个顶点的一个数组{1，1，1}，其表示该顶点出现在所有的子图中

则VCS算法是在上进行的算法

1. **算法基本描述**

注：w是压缩图的个数，R是每个压缩图包含的子图的个数

* **数据结构**：
* 压缩图其中
* 记录位集（recording bitset）（数组，长度为w）：

每个顶点x分配一个记为。如果顶点v在第i个图（）的至少一个子图中可以到达顶点x，则可认为v可以到达顶点x。位集记录了在图（）中顶点v是否访问过顶点x。

* 位集，，（都是长度为w的数组）：
* 遍历位集（Traversal bitset）：每个在队列的顶点u都有一个，表示在哪些子图中还可以从u出发访问其他顶点。若=1，则说明子图可以继续从u出发访问其他顶点。当且仅当任意i，（）都有=0时，从顶点u出发不可再访问其他顶点。
* 本地包含位集（Local containment bitset）：每一个可达的顶点都有一个位集，初始化为。表示哪些子图包含u但是还未访问过u。若图成功访问过u，则将设置为0，表示图不能再访问u
* 包含位集（containmentbitset）：包含位集，压缩图中的每一个顶点（边）都含有一个位集，记录该压缩图的哪些子图包含该顶点（边）
* 队列：
* ：存储将要被访问的顶点，初始化为顶点v（要计算可达度的顶点）
* ：存储要被访问的顶点的包含位集，初始化为
* :存储被访问的顶点，用来计算可达度。
* **伪代码**：

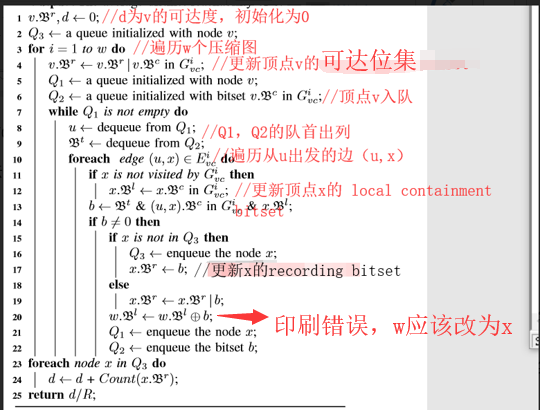
Input：

，……，，顶点v

Output：

V的平均可达度

伪代码如下：



算法描述如下：

该算法主要用于计算每个点的可达度。

解释上述伪代码中第4行，第12行，第13行和14-22行的设计思路和作用：

* 第4行：

更新顶点v的位集，由于每个顶点的包含位集的所有元素都是1，所以每次循环都初始化为全1的位集

* 第12行

只有在压缩图中没有被访问过的顶点x才能将厨初始化为包含位集，即全1的位集

* 第13行

更新b的取值，必须满足在3个位集都不全为0时，b的值就不为0

* 第14-22行

当b不为0时，说明此时该条边是可达的，此时如果x不在队列3中，则可以将x入队Q3，以便计算可达度。第17行和第19行更新位集以便标记x在子图中已经被访问过。第19行代表在垂直压缩图中可以到达相应的节点，但是在水平压缩图中无法到达，却又属于垂直压缩图的顶点。第20行做异或运算用来标记哪些子图访问了顶点x。由于标记了顶点被访问的次数，所以用于最后计算顶点的可达度。

* 第23-24行

代表通过节点v一共可以到达哪些顶点以及哪些顶点在每个相应的水平实例中可以到达。

1. **算法分析**

* **算法正确性证明**

公式5：给定一个垂直实例和其压缩图。如果顶点u在可以到达顶点x，则在同样可以。

证明：如果顶点u在中可以到达顶点x，则在压缩图一定存在一条路径p，使得路径p上的每一个顶点z和每一条边e都有，。因为在初始化为，在更新的的第i位在第一次从u遍历到x时一定总是1，所以当沿着路径p从u到x时更新的一定至少有一位是1。因此在上u可以到达x。

定理6：给定一系列压缩图和他们的R个子图，一个种子集S和一个存储从S可以遍历到的顶点的队列Q通过VCA策略计算的S的平均可达度为：



证明：

对于一个水平实例压缩图来说，他的子图属于w个不同的垂直压缩实例，所以被压缩到w个不同的垂直压缩实例中。基于定理5，在不同的垂直压缩图中推导出了S的可达度测试，队列Q存储所有的在种子集S中的顶点在不同水平实例的不同子图中可以到达的顶点。因为在任何垂直压缩图中S可以到达的点都含有一个位集，用来标记顶点被访问的次数。如果的第i位为1，它表明第i个子图水平实例中的某些子图在相应的垂直压缩图中已经访问过顶点x。该位集只有在当前的垂直压缩图的某些子图可以到达顶点但是不能在相应的水平压缩图中到达，却又属于之前的垂直压缩图。通过利用存储在队列Q中的节点的记录位集，我们可以知道总共达到了多少个不同的节点，以及哪些节点S可以在每个水平实例中到达。 因此，公式6是正确的。

* **算法的时间和空间复杂度**
* 空间复杂度
* 假设有压缩图其中，则所有的压缩图

最多有个顶点和边。如果每个containment bitset 空间复杂度为，所有的压缩图的空间复杂度为

设，则每一个压缩图空间复杂度为

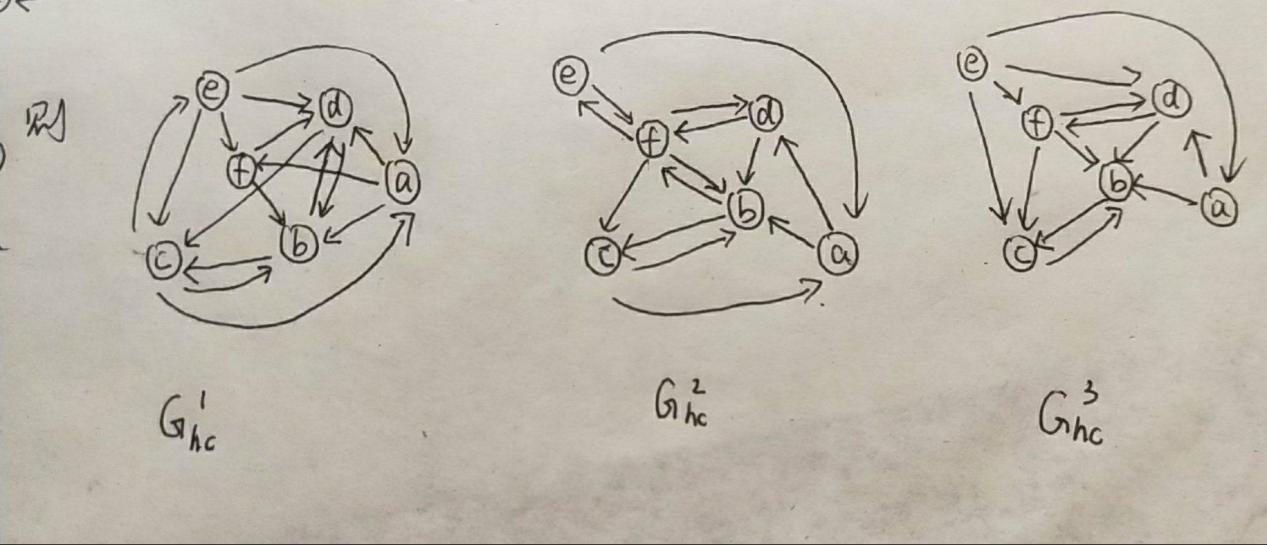
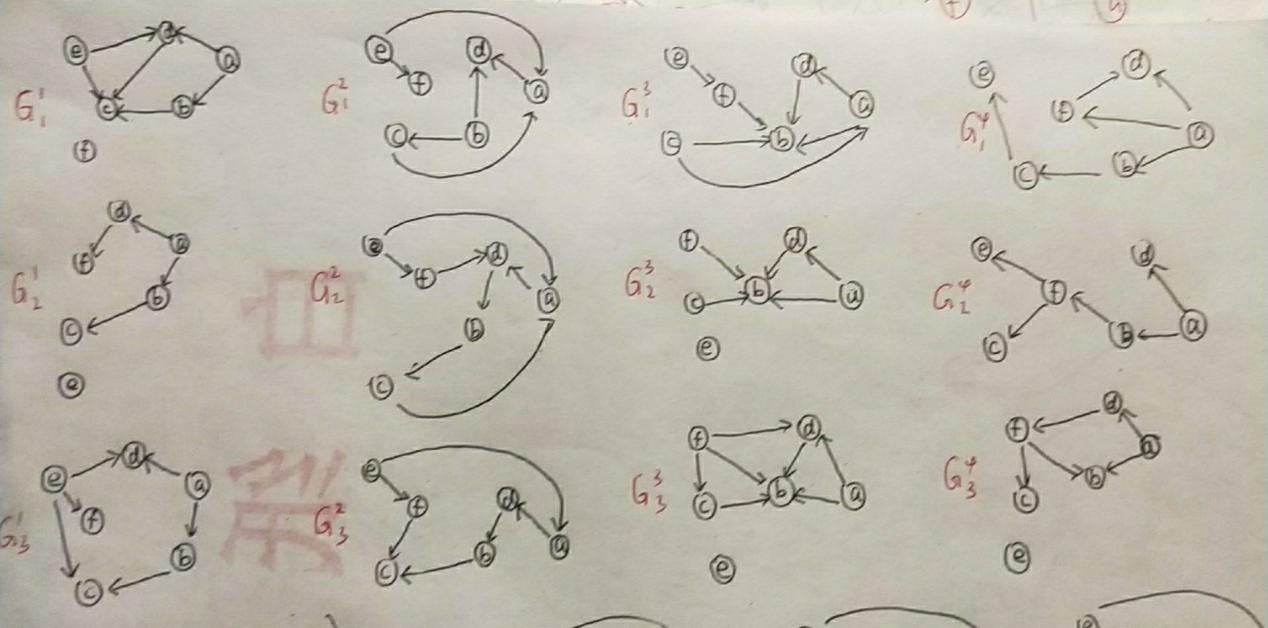
* 时间复杂度

假设有垂直压缩图其中。

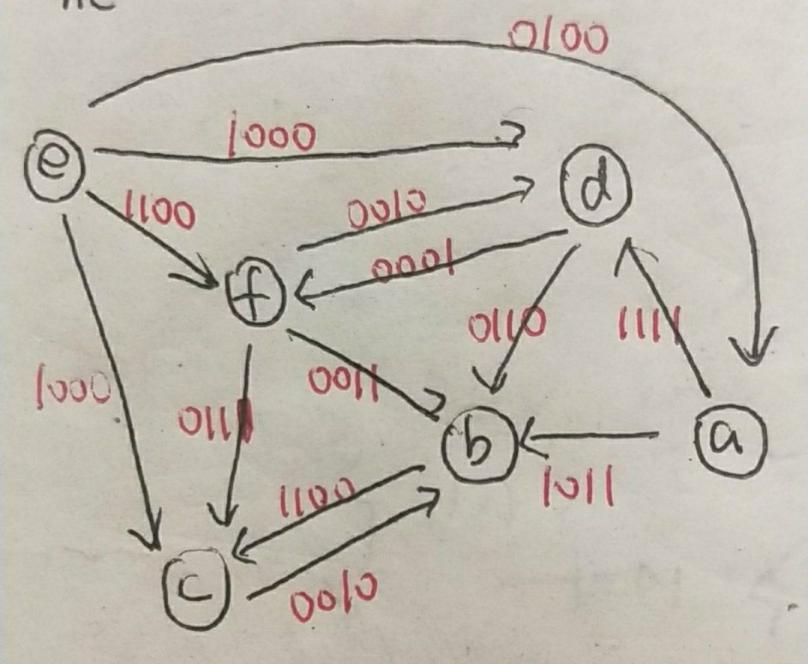
假设，，每一个垂直压缩图（）含有R个子图，对于每一个子图的可达性测试最多遍历条边。每次遍历涉及一个固定数量的位操作，其时间复杂度为。则上述算法的时间复杂度为。利用VCS策略的算法的时间复杂度为（其中构建压缩图的复杂度为，调用上述算法的次数为，更新压缩图的时间复杂度为）

## 实验结论及算法举例

**2.4.1 HCA算法举例**



我们选择进行HCA算法模拟，并选择初始节点b，则在中，边的包含位集如下图：



d初始化为0，=1111≠0；

∴ d = d+1 = 0+1 = 1;(点b本身也算可达点)

={b}，={1111}；

1. 非空，取出队首节点b，赋给u，赋值为队首元素1111，则此时两队列均为空；

遍历以b为起点的所有边：

对于边(b,c),点c未访问，则；

则 b =  &  &  = 1111 & 1100 & 1111 = 1100；

b≠0，且 = ，则d = d+1 = 1+1 = 2;

 =   b = 1111  1100 = 0011；

则 = {c}, = {1100};

1. 非空，取队首元素点c，赋值为队首元素1100，则，此时均为空；

遍历所有以c为起点的边；

对于边(c,b)，b访问过了，

则 b =  &  &  = 1100 & 0010 & 1111 = 0000；

b = 0,则不作任何操作；

③为空，则while循环终止，即在中，点b的可达点为2；

按照如上过程继续在其他水平压缩图中累计计算b的可达点，即可算出点b的平均可达性；

再对其他所有顶点重复以上过程，即可算出每个点的平均可达性，取出最大者，加入种子集；

重复以上所有过程，直至将种子集个数k找满；

**2.4.2 VCA算法举例**

以下面的4个不同时刻的垂直压缩图为例，其中R=3，计算顶点的平均可达度。

由于每个顶点的平均可达度的计算原理是一样的，所以仅以其中一个点为例。

如下是计算顶点e的可达度

* 首先初始化d为0，e的={1,0,0}

然后e进入队列Q3，

* 对于for（i from 1 to w）的循环，以i=1为例，寻找所有可达的点，并计算d

然后e进入队列Q1，e.={1，1，1}进入队列Q2，

因为Q1不为空，此时Q1中的e出队

赋值u=e

= e.={1，1，1}

* 此时对所有以e为起点的边进行11-22行伪代码的循环

从压缩图中可看出，此时所有的边为（e,c）(e,f)和(e,d)

* 从边(e,c)开始，此时c未被访问过，c.={1,1,1}

b=&(e,c).&c.={1,1,1}

因为b不为0，且c不在Q3中，所以c入队Q3，c.=b={1,1,1}

c.=c.异或b={0,0,0}

c进入队列Q1,b进入队列Q2

* 对另外两条边(e,f)和(e,d)也按上述步骤进行，可得

f.={1,1,1},b={0,0,1},顶点f入队Q3，f.={0,0,1} f.={1,1,0}

f进入队列Q1,b进入队列Q2

d.={1,1,1},b={1,0,1},顶点d入队Q3，d.={1,0,1} d.={0,1,0}

d进入队列Q1,b进入队列Q2

* 然后c出队，u=c，= b ={0，0，0}

此时没有以c为起点的边

* 然后f出队，u=f，= b ={0，0，1}

此时没有以f为起点的边

* 然后d出队，u=d，= b ={1，0，1}

此时以d为起点的边有（d，c）

此时c已经被访问过，则b=&(d,c).&c.={0，0，0}

因为b为0，所以此时无需进行伪代码14-22行

* 此时Q1为空，所以循环7-22行伪代码结束
* 对另外3个压缩图进行上述同样的操作（不再说明），最后对队列Q3中的每个点x，计算x.中1的个数，利用d=d+Count(x.)
* 然后用d处以R即可得点e的平均可达度，其中在本例子中R=3

# 领域综述

1. **Simulation-based Methods**

Leskovec等人，采用早期终止技术CELF算法减少了不必要的估计数量，同时达到与贪婪算法相同的精度，近似比能达到(1-1/e)。

CELF算法：算法在原有的贪心算法上进行优化，使算法的性能得到提升。在独立级联传播模型下，节点的边际影响力符合子模性，CELF算法正是利用这个性质来进行优化。当传统的贪心算法根据边际影响力将第一个节点A加入种子节点后，将第一次计算各个节点中的边际影响力次小的节点B再次计算边际影响力，如果这个节点B的新一轮边际影响力大于或等于上一轮又比自己次小的节点C的上一轮边际影响力，即可直接把结点B作为新的种子节点，不用再次计算后面结点的边际影响力。若结点B的影响力不大于或等于上一轮又比自己次小的节点C的上一轮边际影响力，就依次逐个算出各个节点边际影响力，排序选择最大的作为种子节点，放入种子集。

总的来说，CELF使用优先队列存储每个结点v的影响增量|R(v|S)|,每次选取种子结点时，判断优先队列顶部结点v的影响增量|R(v|S)|是否已被更新，如果更新则将该顶部结点o加入种子集合S中，否则重新更新顶部结点v的影响增量|R(o|S)|,由于影响力函数满足次模特性，每次结点选择过程(除第-次以外，因为第一次选取种子结点需要计算所有结点的影响力增量)中，需要更新影响增量的结点数量特别少，大幅度地减少种子结点选取过程中影响增量|R(v|S)| 的计算，从而提升了贪心算法的效率。其时间复杂度为O(kNRM)，比使用传统贪心算法速度快了近700倍。

但是，CELF算法并没有改善最坏情况下的时间复杂度，且运行效率很低。

Goyal等人提出针对CELF方法的改进版本CELF++,CELF++在上一次迭代计算中保存一定的计算结果，减少在下次迭代过程中的计算,实验表明CELF++比CELF方法大概快35-55%，但在实践中，新方法并不比CELF更有效。

1. **Subgraph-based Methods**

SGDU 重用了网络中使用蒙特卡罗模拟生成的少量子图来计算影响扩散，解决影响最大化的可扩展性 - 准确性困境，且大大减少了所需的模拟次数。

PMC 通过将子图转换为非循环图并引入额外的修剪技术，进一步提高了重用生成的子图的效率。 但是，由于存储子图的高内存成本，PMC的可扩展性仍然有限。

**3.Sketch-based Methods**

Borgs等人提出了一个具有理论保证的,且接近线性时间的方法RIS来计算顶点的影响力问题, RIS的计算主要分为以下两部分:

* 反向可达集合(Reverse Reachable Set, RRS)的计算。有放回地随机从节点集合V中选取一个节点v,然后让该节点在原始图的转置图中,传播影响，。在传播过程中,激活的结点集合称为反向可达集合,重复进行θ次,产生θ个反向可达集合。
* 种子结点的选择。使用最大覆盖方法选择种子集合,首先,选择反向可达集合最大覆盖结点v加入种子集合S,然后删除包含该结点的所有反向可达集合，重复此过程，直至选取k个种子结点。
* **优点：**

RIS的主要贡献是舍弃用蒙特卡罗模拟计算结点的影响力,而是用结点的反向可达集合近似结点的影响力，很大程度上提升了算法的效率。Tang 等人给出了结点反向可达集合的数量的合适计算，提出两步法TIM和TIM+方法,他们还利用节点反向可达集合的数量的合适计算，提出两步法TIM和TIM+方法,他们还对种子结点选择过程进行优化,提出基于鞅策略(Martingale)的方法IMM,混合计算反向可达集合和反向可达集合的数量。TIM/TIM+ 和IMM的贡献是在保证影响力具有理论保证的情况下，确定反向可达集合的具体数量(由参数e决定，∈反映到方法的理论界)，IMM方法确定反向可达集合数量时，同时也在生成反向可达集合，其效率高于方法TIM/TIM+.实验表明IMM的效率已经超过一些效果比较好的启发式方法(E设置比较大的值，理论的界比较低),比如IRIE和SIMPATH。TIM/TIM+ 和IMM方法的时间复杂度接近线性，具有一定的理论保证，而且适用于独立级联和线性阈值等更广泛的模型。

* **缺点：**

由于需要很多RR集来实现严格的理论界限，所以该算法很容易受到高内存成本的影响。

**4.Heuristic Methods.**

Score-estimation methods中诸如IMRank算法可以有效地解决独立级联模型下的影响最大化问题。从初始排名开始，例如，从有效启发式算法获得的初始排名，IMRank通过根据根据当前排名计算的基于排名的边际影响扩展迭代地重新排序节点来找到自洽排名。现在已经证明了IMRank绝对会从任何初始排名开始收敛到自我排名。此外，在此框架内，提出了最后一个分配策略和该策略的概括，以提高估计给定排名的基于排名的边际影响差异的效率。

* **优点：**

通过这种方式，IMRank同时利用贪婪算法和启发式算法的优势，实现了卓越的效率和高精度。正如大规模现实世界社交网络上的大量实验所证明的那样，IMRank始终可以实现与贪婪算法相媲美的高精度，计算成本显着降低，甚至可以达到10-100倍。比其他可扩展的启发式算法快一倍。

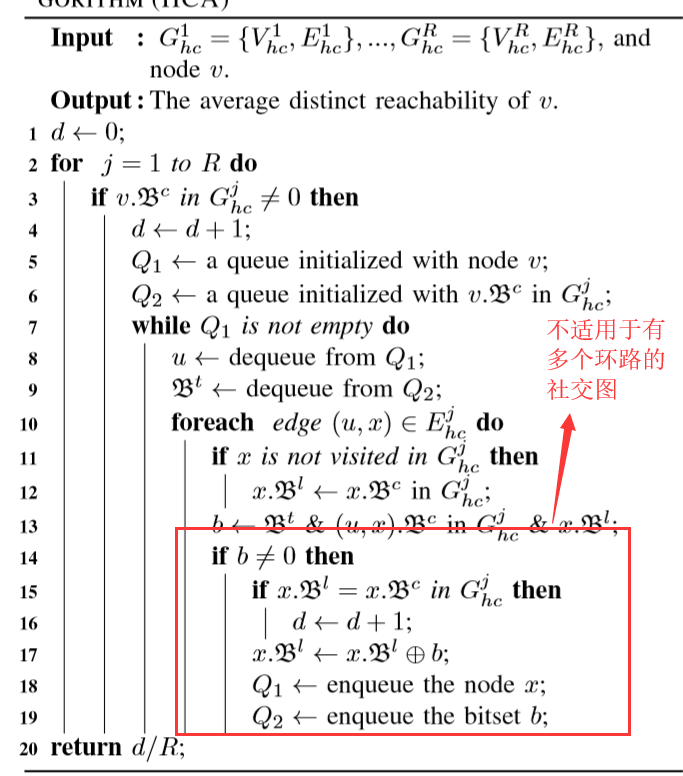
* **缺点：**

但是，这些启发式的算法的结果的质量通常相对较低。

# 方法不足与改进

## 方法不足

算法HCA中遍历图中的边时，如下截图所示，会出现效率较低的情况。



当出现多个环路时，同一个顶点有可能会多次进入队列Q1中，但是却不会增加

可达点的个数，即不会增加d的值，所以会有很多无效的遍历而使程序效率低下。

因此论文中的算法更适用于无环路或环路较少的图。

## 方法改进

对压缩图中的每一个顶点增加一个标志位flag，当flag为true时表示该顶点已完成其对可达点计算的作用，无需再进入队列进行无效遍历。