摘要 - 我们研究了有效识别网络社区结构的问题，该网络的元素及其各自的关系通过流表现出来。这个问题最近引起了人们的极大兴趣，因为它出现在新兴的计算环境中，并涉及各种领域的关键应用，包括社交计算，网络分析，物联网和生物学。尽管已经在检测网络中的社区方面已经进行了大量研究，但现实世界网络现在所达到的前所未有的数量，使得揭示社区结构的任务非常繁重。这种网络的庞大规模经常使得它们在主存储器中的表示变得不可能。因此，处理开发图表以提取底层社区仍然是一个开放的挑战。

在本文中，我们提出了一种图形流社区检测算法，该算法将节点的种子集扩展到社区。我们考虑边流并且旨在处理它们以形成社区而不维护整个图形结构。

相反，我们维护关于节点和我们寻求的社区的非常有限的信息。除了我们的新颖流媒体方法，我们也开发了一种技术，可以大大提高算法的准确性，并提出一种新的聚类算法，允许自动推导出我们想要检测的社区的大小。我们使用地面实况社区对各种大型实词网络进行的实验评估表明，我们提出的方法确实实现了与现有技术的非流媒体社区检测算法相当甚至更好的准确性。更重要的是，在执行时间和存储空间要求方面取得的进步是显着的。

Introduction

越来越流行的大规模图处理方法是考虑数据流模型，其中流包含图的边缘[20]。这是社区检测领域的一个新方向，据我们所知，没有先前的方法考虑过这样的设置，而没有对边缘可用的顺序施加限制[11]，[27]。在本文中，我们提出了一种新的社区检测算法COEUS，它完全适用于图流。图1描绘了这样的图形流，其边缘没有特定的顺序。COEUS用定义不同社区的节点的种子集初始化，例如用图1的圆圈描绘的三个集合。当边缘到达时，我们可以处理它们但我们不能将它们全部保存在内存中。因此，COEUS保持关于每个边缘的相邻节点及其参与所讨论的社区的相当有限的信息。使用概率数据结构保存此信息，以进一步降低算法的内存需求。除了我们在图形流中进行社区检测的最初想法。

算法两个参数

第一个参数：

K个社区种子集{K1,K2,…,KS}

对于每个社区种子集Ki = {k1,k2,…,kl}，表示每个社区中有l个节点。

第二个人参数S = {e1,e2,…,em}，表示边的集合。COEUS处理图流的边缘，以将K0中的每个种子集扩展到社区。

由于网络的规模太大，维持整个网络是不太可能的，所以我们聚焦每个节点的度，以及这个节点在每个社区的度，以及每个社区是由哪些节点构成的。

对于每个我们想扩展的社区，我们检测每一个邻居节点是否是社区的一个member。

对于每一个社区，设置一个窗口期，在窗口期内，新的边来了，社区可以自由的进行扩张，当窗口关闭，对每一个社区进行修剪，只保留对社区贡献度最大的节点。

COEUS修剪社区大小为100，因为相关研究表明质量社区不超过100个节点。

此外，COEUS使用了一个10,000个边的窗口，这个值通过广泛的探索性测试得出，并且一直运行良好。

COEUS可以随意停止，因为每个社区的成员节点随时都可用。在算法1的伪代码中，我们考虑有限流，并且当流的所有元素都已经处理过之后，COEUS终止。但是，COEUS也可以处理无限流。此外，每个社区的所有节点都与一个COEUS

社区参与值（社区贡献度）相关联。该值越高，我们就越确定相应的节点是社区的一部分。

算法4详细介绍了dropTail，这是一个识别与COEUS形成的社区无关的节点并删除它们的过程。在这方面，dropTail利用了社区中包含的节点的社区参与值，并允许完全自动、按需删除不相关的节点。更具体地说，不相关节点与实际社区的关系较弱，因此，与社区中包含的其他节点的值相比，它们各自的社区参与值是不重要的。

这在图3中很明显，图3展示了一个真实图的社区中包含的节点的社区参与值，如COEUS所示。我们观察到，根据节点的社区参与值对节点进行排序会得到一个清晰可见的尾部。社区参与价值的分布是不同的，这取决于网络和所寻找的社区。因此，设置一个恒定的阈值并丢弃显示较低的社区参与值的节点来删除这样的尾部不是一个选项。相反，我们需要对每个特定的社区进行调整，并通过集群隔离属于尾部的节点，为此，dropT ail根据相关的社区参与值计算两个连续节点之间的平均距离(第59行)。然后，从最后一个节点开始，dropT ail迭代地检查这个排名中两个节点的值距离。当发现这个距离大于节点的平均距离时，dropT ail停止，因为它发现两个连续节点的值之间有一个显著的间隙(第11-18行)。

图3还显示了我们示例的这个过程的结果。连续节点之间的平均距离w.r.t.按社区参与值排序为0:00043。

从末端开始的第一个节点比它的前任节点的差距更大，排名第35位。因此，dropT ail认为不相关节点的尾部从第35个节点开始(用红叉表示)，实际的群落由第34个节点组成(用绿点表示)。

说这么多 就是为了寻找图中的拐点，也就是斜率突变的地方。从最后一个节点开始找，

If distance(n,n-1) > distance(n-1,n-2)，那么节点（n-1）就是拐点，拐点前的节点保留，后面的舍弃。Distance（i , j）= 社区参与度相减。节点I的社区参与度减去节点j的社区参与度。

种子节点有不少是在尾部，所以dropTail不考虑种子节点，只考虑那些在扩展过程中添加到社区中的节点。如果考虑种子节点，则会把种子节点也给删除了。

实验：ground-true 只选取了5000个最高质量的社区。

自动确定社区容量

1.算法2：设置窗口期，窗口期内，随意扩展，长口气结束，对社区进行修剪，只保留对社区贡献度最大的节点。COEUS修剪社区大小为100。

2.算法4：对于剩下的100个节点，按照拐点法进行删除尾部，按照节点对社区贡献度的大小进行排列，在斜率突变的地方删除后面的节点。

对比算法:LEMEN

结果：小的数据集上比lemen稍微差一点，打的数据集上比lemen好很多，特大数据集上，lemen无法处理，但是我能处理而且效果还很不错。时间问题上很快。

Yun et al[27]考虑网络规模如此之大的设置，以至于维持相应的图形是禁止的。因此，他们研究了在流设置中将图的节点聚类到社区的问题，其中图的邻接矩阵的行被顺序地显示。他们提出了一种空间复杂度随着网络的大小呈线性增长的在线算法。我们的流设置不会假设邻接矩阵的行完全透露给我们。相反，我们认为涉及图的任何节点的边可能在任何时刻到达。而且，我们不知道图形的大小，它随着时间的推移而增长。Zakrzewska和Bader [28]提出了一种用于社区检测的动态种子集扩展算法。特别地，他们认为边缘可以动态地插入到图形中或从图形中移除，并且通过递增地调整图形的变化来检测种子集的局部社区。与每次更新后需要重新计算的算法相比，后者允许更快的执行，代价是社区质量略差。我们的方法是不同的，因为我们假设我们不能将整个图保持在内存中，而[28]执行的增量调整确实强加了这样的要求。此外，我们建议在每一步都对局部社区进行更具成本效益的重新计算。

Hollocou等[11]考虑边缘流设置并将图的所有节点分配给非重叠社区，每个节点仅使用两个整数，其中包含：i）节点的度，以及ii）分配给节点的当前社区索引。

他们的工作很大程度上基于以下观察结果：如果我们随机均匀地选择图的边缘，则此边缘更可能链接同一社区的节点，而不是来自不同社区的节点。预计这是正确的，因为节点往往在社区内比跨社区更多地连接，因此，如果我们以随机顺序处理边缘，我们期望许多社区内边缘在社区间边缘之前到达。然而，这要求我们已经完整地保持图形，并且我们能够随机地一个接一个地选择其边缘。我们对更实用的假设进行操作，即图的边缘没有特定的顺序。