分类号 _____ 密 级 _____

U D C _____ 编 号 10486

武漢大學

硕 士 专 业 学 位 论 文

属性图中密集社区搜索算法研究

研 究 生 姓 名:

学 号:

指导教师姓名、职称:

专 业 类 别 (领 域): 计算机技术

二〇一九年五月

Research on Cohesive Community Search Algorithm in Attributed Graphs

Candidate:

Student Number:

Supervisor:

Major: Computer Technology

School of Computer Science

Wuhan University

May, 2019

论 文 原 创 性 声 明

本人郑重声明: 所呈交的学位论文, 是本人在导师指导下, 独立进行研究工作所取得的研究成果. 除文中已经标明引用的内容外, 本论文不包含任何其他个人或集体已经发表或撰写过的研究成果. 对本文的研究做出贡献的个人和集体, 均已在文中以明确方式标明. 本声明的法律结果由本人承担.

学位论文作者 (签名):

年　　月　　日

摘　　要

社区搜索是众所周知的社区发现问题的一种在线依赖查询的变种。社区搜索可以个性化社区，它也是图查询和分析中的一项重要的研究任务（如社交网络，生物网络，协作网络和通信网络），旨在找出包含给定查询节点的社区。由于许多图与信息紧密相关，具有属性的网络图社区搜索近年来吸引了许多研究 [1] [2]。

在本文中，给定查询节点，目标是在属性图上找到一个密集的社区。同时，确保返回社区中的节点具有同构属性。在社区中，节点共享的属性越多，社区中节点之间的关系将更紧密。因此，本文想要找到的社区中的节点非常紧密关联，即同时满足结构密集和关键字密集。此外，共享的属性是查询节点的属性关键字集合的子集。本文就属性图中密集社区搜索问题，设计了多种算法去解决这个问题。基于 k-truss 结构模型使得社区中的节点在结构上密集，使用超图 k-truss 等价索引、节点索引和属性关键字索引加快了社区搜索的速度，社区中的节点共享最多的属性关键字，使得属性语义上密集。本文并在多个现实数据集上进行了广泛的实验，通过实验结果分析可以看出本文设计的算法的有效性和高效性。

关键词: **社区搜索；属性图；**k-truss**；属性关键字**

ABSTRACT

As an online, query-dependent variant of the well-known community detection problem, community search enables personalized community discovery that has found Community search is an important study task in graph query and analyze(like social, biological, collaboration, and communication networks) which aims to extract the community containing a given query node. Since many graphs associated with information, community search with attributes appeal many kinds of studies [1] [2] recently.

In this paper, we aim to find densely communities with one query node over attributed graphs. Besides, we want to ensure that the nodes in the returned communities have homogeneous attributes. In a community, the more common attributes are shared by nodes, the tighter relation in the community will be. Furthermore, the attributes are contained in the query node. Hence, the nodes in a community we want to find are closely related with each other which satisfies structure cohesiveness and keyword cohesiveness. In this paper, a variety of algorithms are designed to solve this problem in the dense community search problem in attributed graphs. Based on the k-truss structure model, the nodes in the community are structurally dense. Using the k-truss equivalent index, node index and attribute keyword index of the hypergraph accelerates the speed of community search. The nodes in the community share the most attribute keywords, making the attributes semantically dense. This paper has carried out extensive experiments on a number of realistic data sets. The experimental results show that the algorithm designed in this paper is effective and efficient.

Key words: Community Search；Attributed Graph；k-truss；Attribute Keyword

目　　录

1 绪论

图是计算机科学中常用的一类抽象数据结构，以描述事物之间的复杂关系。现实世界中的很多信息都是以网络的形式存在, 比如人际交往关系网、生物学中的蛋白质网、论文著作中的合作关系网等等。近年来，图结构已广泛应用于多种领域，如万维网、公路网、社交网络、知识图谱、蛋白质交互网络、化学分子结构、图像处理中的属性图、生态系统中的食物链等。随着以上应用领域的发展，图数据在不断的快速产生和积累，如何对其进行有效的管理、查询和挖掘等已经成为目前学术界和工业界共同关注的研究热点。

1.1 研究背景和研究意义

图是表示物体之间的复杂结构关系的强大模型，已被广泛应用各种实际生活中。在大多数应用中，社区这个概念常常会被提及到，社区中的顶点之间密切关联。社区结构是复杂网络中的一个重要特征, 它可以揭示社会网络的隐藏规律和行为特征。社区发现，即在网路中找到所有社区，[3] [4] 已经广泛研究了数十年。最近，依赖于社区发现的变体，社区搜索越来越引起研究人员的关注，旨在寻找查询节点包含的社区，为用户提供个性化社区的发现 [5] [6] [7]。和社区结构密切相关的社区发现问题被提出后就成为国内外学者的研究热点, 和社区发现相关但不相同的社区搜索问题是本文所研究的问题。社区搜索就是输入一个图和一个查询顶点, 然后找到包含这个查询节点的一个密集子图。

在许多真实世界的图数据中，如社交网络、协作网络和生物网络，丰富的文本信息与顶点相互关联，即顶点对象具有自己特有的属性，这样的图数据称为属性图。例如，图 1.1描绘了一张写作网络图，顶点表示作者，与顶点相关联的属性 (例如，DB，ML) 代表他们的专业领域。

事实上，在现实世界许多的图数据中，如社交网络，协作网络和生物网络。实体节点往往包含丰富的文本信息，挖掘文本中关键词作为实体节点的特征属性信息，这

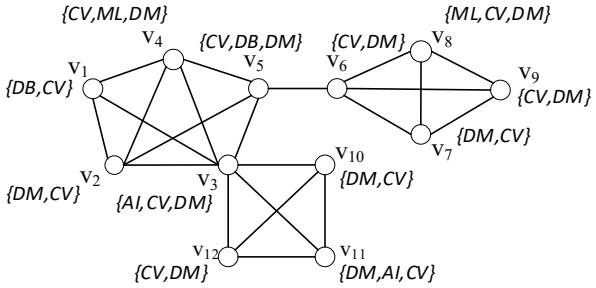


图 1.1: 属性网络图

样的图称为属性图。属性图更能展现现实的世界，属性图的社区搜索变得更加有研究意义。如何综合结构和属性信息进行属性图的社区搜索是一个新的挑战。

1.2 国内外研究现状

经过数十年的研究发展，国内外已经提出了各种社区搜索模型和方法，如 clique 和 quasi-clique [5] [8] ，k-core [6] [9] [10]，k-truss [7] [11] [12] 等。随着研究的深入以及相应技术的广泛应用，越来越多的研究者提出的方法在图数据结构上研究取得非常大的突破。相关研究大体可以分为四类：图上的关键词搜索、基于 k-truss 的社区搜索社区发现（community detection）和社区搜索（community search）。每一种相关研究中都有其代表性的经典算法，也各有其优缺点。

1.2.1 图上的关键词搜索

在相关文献中，图上的关键词搜索使用查找相应树数据结构的方法，通常使用 Q-SUBTREE 数据结构，生成树的方法通常基于斯坦纳结构或相异根结构的逻辑。生成最小斯坦纳树（也就是基于斯坦纳结构获得的 (top-1)-Q-SUBTREE）是 NP 完全问题，[2] 通过向后搜索的方式提出了近似启发式算法 Bank 的解决方法。对于生成若干最优斯坦纳树，DPBF [29] 算法通过动态规划解决这个问题，已经被广泛的研究，其主要目的是从大图中查找与给定关键词集合最相关的子结构 [1]。关键词搜索的结果通常是连通的树（tree）或图结构，在算法 [30] 中考虑多项式的延迟。之后，[31] 的 STAR 方法实现在伪多项式运行时间内以 O(log(n))- approximation 的近似比率获得最优斯坦纳树。近年来，动态规划算法 PrunedDP [32] 通过树的分解和合成解决该问题。BANKS-II [3] 和 BLINKS [4] 基于相异根结构逻辑，也可以找到 Q-SUBTREE，其中树的权重被定义为根到每个包含关键词的节点的最短距离之和。BANKS-II 从候选

的根节点开始向前搜索，而 BLINKS 通过分割图建立双向索引实现算法。

每个连接树只提供查询关键词之间的一部分关系，为了避免这个问题，连通图结构 r-radius [5]、社区 [6]、r-clique [7] 被随之提出。EASE [5] 算法提出建立反向索引，同时考虑结构和属性的排序方法获得 r-radius 子图，子图中包含关键词的节点之间的半径不大于 r。获得社区的多项式延迟算法 [6] 生成一系列有序社区，每个社区是一个多中心的子图，其中每个包含关键词的节点距离中心点的距离小于阈值。获得 r-clique 的多项式延迟算法是一个近似算法，生成一系列 r-clique 子图，每个子图中包含关键词的节点之间的距离小于 r。还有一些解决方法 [9] [10] [11] 研究考虑多样性的关键词检索，其生成的结果也是连通树结构或子图结构。

1.2.2 基于 k-truss 的社区搜索

基于 k-truss 的社区搜索致力于获得包含给定关键词且 truss 值最高的社区。文献 [17] 建立 TCP 索引有效得获得所有包含关键词的 k-truss 社区。文献 [19] 提出更为简洁的索引 EquiTruss 来加速计算。但是社区可能会受到 free-rider 的影响，为了避免该影响，文献 [18] 提出搜索具有最大 truss 值和最小直径的社区，并提出一个近似算法来解决这个问题。一些近似的工作 [16] [26] 研究带有属性的社区搜索，查询内容为节点和节点的一系列属性。

1.2.3 社区发现 (community detection)

在没有属性的图中社区发现，旨在整个网络中识别所有社区。如文献 [37] [4] 所述，这个问题已经被广泛研究。近年来，各种各样密集子图上的模型以分解方式也对社区发现进行了研究，如 core decomposition [35] [38] [21]，truss decomposition [22] [23] [24] [25] 和 k-edge connectivity components decomposition [34] [27]。

在属性图的社区发现中，属性社区发现就是发现全部具有同质性的密集连通社区 [28] [41]。[28] 考虑了计算顶点的连接和关键词之间的两两相似性，然后将顶点聚集在一起形成的图来获得社区。[41] 提出了一种名为 CODICIL 的方法，它基于内容相似性创建新的边缘，然后使用一个有效的图形采样来提高聚类效率。[43] 基于概率推理对属性图进行聚类。在 [33] 调查中对属性图进行了聚类。在在线社区搜索问题上，上面讨论的社区发现算法通常是低效的。

1.2.4 社区搜索 (community search)

在没有属性的图中进行社区搜索，目的是查找包含给定的一组查询节点的社区。目前已经提出了各种各样的模型来衡量社区的紧密性，基于诸如 random-walk [28] [39]，query biased edge density [15]，clique and quasi-clique [36] [40]（即以一种简单的方式建立的密集子图模型，在 quasi-cliaque 中，每个相邻的顶点的度数至少是对方的度数的 $\lambda \in (0,1]$ 倍），k-core [12] [13] [15] [14]（图中每个顶点的度数至少为 k，k-core 中可能包含多个 components），k- truss [17] [18] [19]，components [20] [21]（在连通分量子图中，任何两个顶点之间通过路径连接）。[36] 在重叠网络中进行在线搜索，基于 k-cliques 建立重叠社区搜索模型（OCS），查询顶点所在的社区。[40] 建立了一个密集子图模型 quasi-clique，并提出两种迭代求最大值的算法 DIM（Deterministic Iterative Maximization）和 SUM （Stochastically Updated Maximization）分别以确定和随机的方式找到包含查询点集合的最大 λ-quasi-clique。[12] 提出一个局部搜索策略，利用 k-core 和查询点的最大 core 值缩小查询范围，并在查询点的邻居中搜索最大的社区。[13] 首先提出一种基于最小度和距离约束的贪心算法，然后在此基础上进行了修改并设计了两种启发式算法找到查询点所在的社区。[15] 克服以往基于最小度的社区搜索的缺点，利用 core decomposition 查询出图中的 k-core，然后在 k-core 上进行社区搜索，这样保证了得到的子图的密集性。[14] 基于 k-core 建立了一种新奇的 k-influential 社区模型，提出了一个线性时间的搜索算法，并设计了一个线性空间的索引数据结构加快了社区搜索的速度。[20] 建立基于 MST 数据结构的索引，以斯坦纳树寻找 SMCC(Steiner Maximum-Connected Components)。[21] 讨论了 SMCS(Steiner Maximum-Connected Subgraph) 问题，面对 APX 难的斯坦纳树问题计算最小的 SMCS，提出了一个高效的 Expand-Refine 算法计算最小的 SMCS。[17] 构造了 TCP-Index 索引高效搜索包含给定查询节点的所有 k-truss 社区。[19] 提出了一个更简洁的索引 EquiTruss 加快了 k-truss 社区的搜索。为了避免 free rider 的影响，[18] 提出了一种近似方法来寻找最大 truss 值和最小直径。

在属性图中进行社区搜索，目的是查找包含给定的一组查询节点的社区，并且社区中的节点共享尽可能多的属性关键字。[1] [2] 提出了两种属性社区模型，[1] 模型是基于 k-core 的属性社区，其中社区节点的度数至少为 k，并且共享尽可能多共同的属性。[2] 是基于 k-truss 找到最大属性得分的社区。k-truss 是一个每个边都至少包含

(k-2) 个三角形的子图。与 k-core 相比，它是基于更高级的三角形图形架构，而不是原始的顶点或者边。一个连接 k-truss 子图也是一个 (k-1)-core 和一个 (k-1)-edge 的子图，即所有节点的度数至少为 k-1，移除少于 k-1 条边后并保持连接。在这个模型 [2] 中，除了 k 之外，还给出一个阈值 d，以使得查询节点和社区中的节点之间的距离不大于 d。

1.3 本文的组织结构

本文主要是对属性图中密集社区搜索算法的研究，基于属性图社区搜索问题设计了解决的算法，同时进行了广泛的实验进行对比分析，最终实验结果说明了本文设计的算法的有效性和高效性，本文的具体结构安排如下：

第一章绪论。首先介绍论文研究背景和意义，其次介绍了属性图中密集社区搜索算法研究的国内外现状，最后介绍本文的研究内容和结构。

第二章相关概念和相关工作介绍。首先介绍了本文的研究问题和相关工作介绍。其次介绍了本文中使用到的概念定义。然后介绍了本文中算法中使用的索引结构。最后对本文的算法模型进行了简单介绍。在这里，我们详细阐述了与本文相关的基于 truss 的社区搜索，其目标是找到最大 truss 值和包含一组给定的查询节点的社区。第三章基础算法。首先介绍了属性关键字的反单调性和领域约束两个性质，接着设计了一个基本的解决方案 ATCBasic 算法解决属性图中密集社区搜索问题。使用频繁挖掘算法 FP-Growth 计算出查询顶点和邻居顶点之间的频繁属性子集项，大大减少了候选属性关键字集合的数量。

第四章基于索引的算法。首先介绍说明了超图即 truss 等价索引和节点索引及相关概念定义，然后基于混合索引，提出了先计算出结构上密集的社区的 ATCIndex 算法，最后设计了一种改进的 ATCImprove 算法。

第五章基于候选属性关键字的算法。首先介绍了属性关键字索引，然后基于关键字索引，设计了从查询顶点的属性关键字集合出发的 KIndexInc 和 KIndexDec 算法。

第六章实验分析与结论。首先简单介绍了本文实验中使用的数据集相关信息和实验环境配置，其次介绍了对大量实验结果进行了分析和算法评估。

第七章总结与展望。对本文进行了概括总结，并对未来的属性图中密集社区搜索算法研究的展望。

2 相关概念及相关工作介绍

本章介绍了属性图中密集社区搜索算法研究问题，并介绍了其相关的工作。然后阐述了本文在其研究上的创新点。最后介绍在研究大型属性图中社区搜索过程中涉及到的概念、定义以及相关的名词解释，之后会详细介绍相关算法的定义与细节。为了让研究顺利的进行，本章也讲述与研究实验有关的前期准备工作。

2.1 问题陈述

将生活中的一些实体作为节点，每个实体都有自己的属性关键词信息，实体之间的关系形成边，以图数据结构 G 表现出一个真实世界。数字化为给定一个属性集合为 Σ，一个简单的无向属性图表示为 $G=(V,E,A)$，其中 V 是顶点集合，$E \subseteq V \times V$ 是边的集合，A 是一个属性函数，为每个节点分配一组属性 $A(v) \subset \Sigma$。我们使用 $V(G)$ 和 $E(G)$ 来分别表示图 G 的顶点集合和边集合，使用 $|V(G)|$ 和 $|E(G)|$ 表示顶点的数量和边的数量。对于一个顶点 $v \in V$，我们通过 $N(v)=\{u \in V|(u,v) \in E\}$ 表示它的另据节点，并且使用 $d(v)=|N(v)|$ 表示它的度。图 G 中的一个三角形子结构表示为 $\{(u,v),(v,w),(u,w)\} \in E$。

给定一个查询顶点 v_q 和对应的一组关键字集合 $S \subseteq A(v_q)$，本文的目标是找到包含 v_q 的 k-truss 社区。社区中的边与边之间是三角形连接的，并且顶点与顶点之间共同拥有 S 中最大的属性子集。三角形连通性确保了结构的密集性，即结构上社区中的两条边属于同一个三角形，或者通过一系列相邻的可达到的三角形。如果两个三角形共享一条边，则认为它们相邻的。社区搜索的社区中的顶点与顶点之间共享关键字集合 S 中最多属性关键字，这样就确保了属性语义上的密集性。

例 2.1.1 假设图 1.1中查询点为 v_4，其属性关键字集合为 $\{DM,CV\}$，本文的模型将会返回子图 H_3，其包含顶点 $\{v_2,v_3,v_4,v_5\}$。注意顶点 v_{10},v_{11},v_{12} 将不会包含在这个社区中，因为它们和 H_3 中其他边不是三角形连通的，即无法保证结构上的密集。

2.2 相关工作

在属性图的社区搜索中，属性社区搜索近年来开始引起研究人员越来越多的关注。[16] 旨在获得 k-core 社区，其中节点尽可能地共享公共属性，并且在社区内，节点的度至少为 k。Huang 等人在 [26] 提出基于最大属性分数发现 k-truss 社区的方法，并利用一种贪婪算法，给出了这个 NP 难题的近似解决方案。

[1] [2] 提出了两个属性社区模型。[1] 模型基于 k-core 的属性社区，其中社区节点的度数至少为 k，并且共享尽可能多共同的属性。但是，这种基于 k-core 的模型不能社区的连接性，即社区中存在某个点是一个孤立点，导致整个社区不是连接的。[2] 基于 k-truss 找到最大属性得分的社区。k-truss 是一个每个边都至少包含（k-2）个三角形的子图。与 k-core 相比，它是基于更高级的三角形图形架构，而不是原始的顶点或者边。一个连接 k-truss 子图也是一个（k-1-core 和一个（k-1-edge）的子图，即所有节点的度数至少为 k -1，移除少于 k-1 条边后并保持连接。在这个模型 [2] 中，除了 k 之外，还给出一个阈值 d，以使得查询节点和社区中的节点之间的距离不大于 d。但是，不相关的节点也可能会包含在此模型中，因此三角形连接确保结构上密集就被忽略了。本文使用以下例子来说明了两个模型的局限性，也阐明这两种方法与在本文研究的算法模型有本质的不同。

例 2.2.1 图 1.1是一个协作网络的例子，假如给定的查询点为 v_4，属性关键字集合 $S = \{DM, CV\}$。如果设置 truss 值 k 为 3，那么，[1] 返回的子图为 H_1，包含顶点 $\{v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9\}$，子图 H_1 中共享两个属性 $\{DM, CV\}$。如果我们删除边 (v_5, v_6)，我们可以看到 H_1 很容易断开，社区将会分成两个区域，这也就意味着它是弱连接。[2] 是基于 truss 的社区模型中，如果设置 $k = 4$ 和 $d = 2$，那么将返回最大属性分数子图 H_2，包含顶点 $\{v_2, v_3, v_4, v_5, v_{10}, v_{11}, v_{12}\}$ 。显然，节点 v_{10}, v_{11}, v_{12} 不应该包括在结果中，因为它们与查询节点 v_4 没有密切关系。

2.3 研究创新点

社区搜索作为社区发现的变体，越来越引起研究人员的关注。经过数十年的研究发展，国内外已经提出了各种社区搜索模型和方法，如 clique 和 quasi-clique [5] [8] ，k-core [6] [9] [10]，k-truss [7] [11] [12] 等。但是，他们大多数只考虑社区结构的密集性，

而长期忽略了社区另一个重要的衡量标准——属性密集性，用于评估社区中标签的共享程度。本文研究的社区搜索问题以找到综合结构和属性上密集社区，社区中的边是强连接的，并且节点之间尽可能多的共享相同属性。

在大型属性图中进行属性社区发现很具有挑战性，因为我们需要探索很多关键词子集来找到具有最多共同关键词数量的社区。考虑到国内外研究的局限性，本文提出了一个新颖的属性图社区搜索模型方法，以找到结构上和属性上都密集的社区。一个简单的解决方案是枚举所有的关键词并进行组合，返回共享关键词最多的 k-truss 社区。但是，枚举的子集数量可能非常大，这与关键词的大小 l 成指数关系，即我们需要验证 (2l-1) 个关键词组合来查找 k-truss 社区，以在线的方式在大型属性图上进行 k-truss 社区发现是不切实际的。虽然 [16] 已经设计出一个基于 k-core 的高效的索引进行属性社区搜索，但是它不能被扩展来查询 k-truss 社区，因为 k-core 与 k-truss 的内部结构存在着一定的差异。同时，[26] 中提出的方法也不能支持我的社区搜索，因为他们搜索的 k-truss 社区没有三角形连通性约束，并且查询的社区密集性层次不齐。因此，本文通过这三个方面来努力解决这个问题，也是本文中三个创新点。

搜索的社区满足结构和语义上最密集，即社区中的边与边之间是强连接的，并且节点尽可能多的共享相同属性。从最大频繁子集进行搜索，当找到其社区后就停止向下一个量级的频繁子集进行搜索，这样保证了属性密集性。然后使用 k-truss 社区结构来保证结构上的密集性。

多次进行全图搜索固然会花费大量的计算时间，影响社区搜索效率。如果能预先进行一些预处理，这部分的时间开销是不会算在搜索时间上的，这也是在应用中是允许的。因此，本文参考多篇文献提出了一个关于 k-truss 等价的一个索引。这个索引将原始图数据浓缩成一个简图，称为超图 (super graph)。超图中的节点称为超点，超点是将连通的具有相同 truss 值的边的集合，将这些边聚集在一起就变成了一个超节点。超点和超点之间的节点如果存在三角形连通，那么这两个超点就会有一条边进行连接，这条边称为超边。对整个原始图数据进行这样的预处理，就会形成一个超图。超图相对原始图点和边的数量减少了很多，在超图上进行社区搜索就相对来说就快了很多。

首先使用已有的关联算法 FP-Growth 发现属性关键词频繁项集，减少查询节点要探索的关键词子集的数量，避免了一些不必要的计算。其次建立了一个保留属性关键字信息和 truss 等价信息的混合索引，并基于这些索引设计了多种新的算法，加快

社区搜索的计算速度，准确高效搜索出满足条件的社区。最后，将本文设计的算法与现已有的算法进行大量实验数据比较，使用数据来说明本文设计的算法的有效性和高效性。

2.4 概念与定义

定义 2.4.1 **(边的支持度)** 图 G 中一条边 $e=(u,v)$ 的支持度是这条边 e 所出现在多少个三角形中，定义为 $sup_G(e)=|\{\triangle_{uvw}|w\in V(G)\}|$。

定义 2.4.2 **(k-truss)** 给定图 G 和一个整数 k，一个 k-truss 是一个子图 $H\subseteq G$，使得 $\forall e\in E(H)$，$sup_H(e)\geq k-2$，即子图 H 中每一条边的支持度都大于 k-2。

子图 $H\subseteq G$ 的 truss 值是子图 H 所有边中最小的支持度加上 2，定义为 $\tau(H)=\min_{e\in E(H)} sup_H(e)+2$，一条边 e 的 truss 值是包含边 e 的子图中最大 truss 值，即 $\tau(e)=\max_{H\subseteq G\wedge e\in E(H)}\tau(H)$。如果一个 k-truss 子图 $H\subseteq G$ 不存在 H 的子图 $H'(H\subseteq H'\subseteq G)$，即 $\tau(H')=\tau(H)$，那么我们称 H 为最大 k-truss。但是，一个 k-truss 可以断开，这不足以建模社区密集而紧密地联系在一起。因此，进一步利用三角形连通性来模拟现实社区 [7] [12]。

定义 2.4.3 **(三角形邻接)** 图 G 中给出两个三角形 $\triangle_1$ 和 $\triangle_2$，如果 $\triangle_1$ 和 $\triangle_2$ 共享一条边，那么它们就是邻接的，用 $\triangle_1\cap\triangle_2\neq\emptyset$ 表示。

定义 2.4.4 **(三角形连接性)** 图 G 中给出两个三角形 $\triangle_s$ 和 $\triangle_t$，如果它们之间能通过一系列三角形邻接，即 $\triangle_1$,...,$\triangle_n$ $(n\geq 2)$，其中 $\triangle_s=\triangle_1$，$\triangle_t=\triangle_n$，并且 $1\leq i<n$, $\triangle_i\cap\triangle_{i+1}\neq\emptyset$。那么就认为这两个三角形是连接的，表示为 $\triangle_s\leftrightarrow\triangle_t$。

类似地，图中的两条边 $e,e'\in E(G)$ 被认为是三角形连接时有两种情况。一种是两条边属于同一个三角形中，另一种情况就是两条边分别存在两个具有三角形连接的三角形中，即 $e\in\triangle_s, e'\in\triangle_t$ ，其中 $\triangle_s\leftrightarrow\triangle_t$。

定义 2.4.5 **(k-truss 社区)** 一个 k-truss 社区是图 G 中的一个子图，这个子图拥有最大的 k-truss 值，并且子图中的任意两条边是三角形连接的，即 $\forall e,e'\in E(H)$, $e\leftrightarrow e'$。

定义 2.4.6 **(属性 truss 社区 (ATC) 搜索算法)** 给定一个属性图 $G = (V, E, A)$，一个整数 $k \geq 3$，以及一个查询顶点 $v_q \in V$ 和对应的关键字集合 $S \subseteq A(q)$，属性 truss 社区搜索系统返回一个子图集合，集合中的每一个子图 $\forall H \in \mathcal{H}$ 都满足以下条件：

1. $\mathcal{H}$ 是包含 q 的 k-truss 社区;
2. $\mathcal{H}$ 中所有顶点共享属性关键字最多。

显然，ATC 模型可以保证搜索返回的社区满足结构上密集性和关键词密集性，即结构上社区是一个 k-truss 社区，内部的边是三角形连接的，属性关键字上社区中顶点共享最多的属性关键字。例如，基于我们的 ATC 模型，一个查询点 v_4，其对应的属性 $S = \{DM, CV\}$，经过搜索返回一个子图 H_5 包含 $\{v_2, v_3, v_4, v_5\}$，其中 $\tau(H_5) = 4$ 和 $L(H_5, S) = 2$。相对于另一个子图 H_4 包含 $\{v_1, v_2, v_3, v_4, v_5\}$ 等节点，它也是一个 4-truss 社区，但是 $L(H_5, S) = 1$，即子图中节点共享的属性关键字数量为 1，比子图 H_5 少，所以 ATC 返回的结果是子图 H_5。

定义 2.4.7 **(community member frequency(CMF))** 一个查询顶点 q 的关键词集 W(q) 的关键词 x，如果 x 出现在社区 C_i 中大多数顶点里，那么认为 C_i 是密集程度高。CMF 使用 C_i 中查询点 q 中关键字的出现频率确定密集程度。设 $f_{i,h}$ 为社区 C_i 中顶点关键字集合中包含 W(q) 中第 h 个关键字的数量。然后，$f_{i,h}$ 除以社区 C_i 中的顶点数量就是 C_i 中此关键字的相对出现频率。CMF 是所有社区中 $W(q)$ 中所有关键字的平均值：

$$CMF(C(q)) = \frac{1}{\zeta \cdot |W(q)|} \sum_{\zeta}^{i=1} \sum_{|W(q)|}^{h=1} \frac{f_{i,h}}{|C_i|} \tag{2.1}$$

定义 2.4.8 **(community pair-wise Faccard(CPJ))** 这是基于社区 Ci 的任何一对顶点的关键词集之间的相似性。我们采用 Jaccard 相似性，用于比较有限样本集之间的相似性与差异性。Jaccard 系数值越大，样本相似度越高。设 Ci，j 为 Ci 的第 j 个顶点。CPJ 然后是 Ci 的所有顶点对的平均相似度，以及所有顶点 C（q）社区：

$$CPJ(C(q)) = \frac{1}{\zeta} \sum_{\zeta}^{i=1} \left[\frac{1}{|C_i|^2} \sum_{|C_i|}^{j=1} \sum_{|C_i|}^{k=1} \frac{|W(C_{i,j}) \cap W(C_{i,k})|}{|W(C_{i,j}) \cup W(C_{i,k})|} \right] \tag{2.2}$$

2.5 本章小结

本章介绍了在研究大型属性图中社区搜索过程中涉及到的概念、定义以及相关的名词解释。为后面的研究顺利进行提供一些基础概念，对理解算法的细节有很大的帮助。首先介绍了本文研究的问题描述，阐述本文所要追求的目的。其次介绍了相关工作，主要阐述了两位研究者在研究上的不足，本文的算法弥补了不足并在此基础更加高效。然后介绍了本文的创新点，通过建立索引和减少查询关键字集合数量来加快查询速度。最后介绍了本文中使用到的一些定义，有一些基础定义和后续实验中所用的公式定义。

3 基础算法

属性网络图中密集社区搜索算法研究，首先从基本的解决方法去解决问题。然后在此基础上进行优化改进。本章给出了基础算法步骤，为后续优化改进的算法作一个铺垫。使用了关联算法 FP-Growth 减少了一部分候选属性关键字集合的数量，提出了自上而下的 ATCBasic 算法。后面将会循序渐进的对此方法改进来提高社区搜索效率。

3.1 简单的解决方案

对于属性 truss 社区搜索问题，一个简单的解决方案就是枚举所有属性关键字组合，然后检查 k-truss 社区是否存在。如果存在则返回具有最多共有属性的社区。具体来说，给定一个查询顶点 v_q 和对应的属性关键字集合 S，我们首先枚举它的所有非空子集 $S, S_1, S_2, \ldots, S_{2^l-1}$。对于每个子集 S_i $(1 \leq i \leq 2^l - 1)$，我们检查是否存在包含查询点 v_q 的 k-truss 社区，并且社区中的每个顶点的属性都包含一个关键字集合 S'。返回具有最多共同属性关键字的 k-truss 社区作为属性 truss 社区的查询结果。

在上述方法中，在大型属性图中进行在线搜索是不切实际的，因为我们需要重复 k-truss 社区搜索 $2^l - 1$ 次。因此，本文提出了一种可以大大减少检查的关键字属性子集数量的新方法。在阐述算法的具体细节之前，首先给出关键字的反单调性性质设置如下。

性质 3.1.1 **(反单调性)** 给定图 G，查询顶点 $v_q \in G$ 和属性关键字集合 S，如果存在 k-truss 社区 $C_k[S]$，其中节点包含集合 S 的所有关键字。那么对于任意一个子集 $S' \in S$ 都存在一个 k-truss 社区 $C_k[S']$.

证明反单调性这个性质并不难。因为一个 k-truss 社区中的所有顶点如果包含集合 S 的所有关键字，那么其任何子集也会在所有顶点之中。基于反单调性性质，当我们发现 $C_k[S']$ 不存在时，我们就可以停止检查 S' 的所有超集，即不能再向 S' 集合中添加任何属性关键字。然后我们可以得到如下基于 apriori 算法的解决方案。首先，我们将 S 的属性关键字子集分为 l 组，即 $\Psi_1, \Psi_2, \ldots, \Psi_l$，每个 Ψ_i 集合包含 i 个关键字

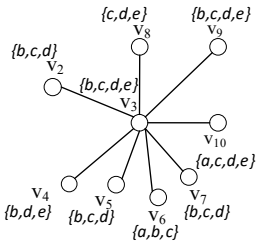
属性。我们开始检查每个只包含一个关键字集合，即 $S' \in \Psi_1$，查看是否存在 k-truss 社区 $C_k[S']$，以此慢慢地增加属性关键字数量。当我们检查了 Ψ_i 中候选的所有关键字之后，即有我们检查了 i 个关键字集合是否存在这样的 k-truss 社区，如果存在则可以继续增加到 i+1 个关键字集合，如果不存在则就不用继续检查 i+1 个属性关键字子集。但是，在这样的过程中，我们仍然需要检查很多关键字集合的 k-truss 社区是否存在。然后，自然会出现一个问题：我们可不可以在做 k-truss 之前设置过滤没有希望的关键字进行社区搜索？本文通过利用下面的一个 k-truss 性质来回答这个问题。

性质 3.1.2 **(邻域约束)** 对于属性关键字集合 S'，如果存在一个 k-truss 社区 $C_k[S']$，那么查询点 v_q 至少有 $k-1$ 个含有 S' 属性关键字集合的邻居节点。

这个性质可以很容易地得出，因为一个 k-truss 社区也是一个 (k-1)-core。因此在社区 $C_k[S']$ 中的每个顶点的度数至少为 k - 1。基于此性质，我们可以从中生成 v_q 和邻居节点的候选关键字子集，并且过滤掉不满足这个条件的关键字子集。具体来说，对于每个顶点 $u \in N(v_q) \cup \{v_q\}$，我们只选择了包含的关键字 S 的顶点，并获取新的关键字集合 $W'(u)$。然后我们应用频繁模式挖掘算法 FP-Growth，它可以找到支持度至少为 k 的频繁关键字集合。因此，我们只需要检查候选关键字集合，而不是所有关键字集合，进而加快搜索速度。

社区搜索的结果节点之间尽可能多共享相同属性，所以在普通枚举中对查询点 v_q 的属性集合 S 求出所有的子集集合，然后从最大的子集集合开始搜索，使得搜索的社区满足属性上的密集性。但是这样做却带来一些多余的计算开销，数据挖掘中关联算法 FP-Growth 算法发现查询点 v_q 和邻居点属性之间的频繁项集，减少属性子集集合的数量。FP-Growth 算法不同于 Apriori 算法的“试探”策略，算法只需扫描原始数据两遍，通过 FP-tree 数据结构对原始数据进行压缩，效率较高。原先的频繁子集是查询顶点 v_q 属性的所有子集，而频繁子集先将查询点 v_q 与它的邻居属性进行频繁集挖掘。也就是对查询顶点 v_q 的所有属性子集集合进行了一次筛选，即频繁项集数量一定是小于等于所有子集集合的数量 (2^l-1)。这一步也会带来一些时间上的开销，但是它带来的收益和这开销还是有必要做这步，实验效果也不错。

例 3.1.1 在图 3.1中，假设一个查询点是 v_3，并且它的关键字集合为 $S=\{b,c,d,e\}$。可以看出 v_3 有 8 个邻居。我们通过 FP-growth 算法生成 9 个候选关键字集在表中，很明显它小于枚举全部关键字集的数量 $2^4-1=15$。



Set	Keyword sets
ψ_1	$\{b\}$,$\{c\}$,$\{d\}$,$\{e\}$
ψ_2	$\{b,c\}$,$\{b,d\}$,$\{c,d\}$,$\{d,e\}$
ψ_3	$\{b,c,d\}$

图 3.1: 候选属性关键字集合

3.2 ATCBasic 算法

使用频繁挖掘算法 FP-Growth 算法大大减少了候选属性关键字集合的数量，对于这些集合，盲目的遍历查找显然效率是低下的。可以先对所有集合先按照集合中属性关键字数量进行排序，然后采用自上而下或者自下而上的方式进行遍历可以有效的加快搜索速度。但是，在上面的自下而上算法中，我们需要搜索大量关键字集，直到找到存在 k-truss 的最大的关键字集合。另一种搜索方法是按属性关键字数量以自上而下的方式，即开始检查最大的关键字集合 $S' \in \Psi_h$ 是否存在 k-truss 社区 $C_k[S']$。如果我们检查了所有关键字集后，在 Ψ_h 找不到 k-truss 社区，然后我们以递减的方式检查较小的候选关键字集合 Ψ_{h-1}。这种自上而下的算法比自下而上的方法更快，因为 k-truss 社区的验证可以在大型关键字集上轻松完成，它可以提前结束这种检查工作，即自上而下的算法不需要检查完所有属性关键字集合。因为当搜索到存在 k-truss 社区的时候就会停止查找，此时就是社区共享最多数量的属性关键字，这样的社区是我们所需要的。具体来说，对于任何两个关键字集 $S'' \subset S'$，包含 S' 顶点的数量不大于包含 S'' 的数量，也就意味着在 S' 的验证比在 S'' 更加有效率。

通过分析属性 truss 社区的反单调性和邻居约束，设计了一个自上而下的算法来进行属性图社区搜索，在算法1中显示了基于上述的自上而下方式的基础算法。输入参数包含属性图 G、查询顶点 v_q、查询顶点的属性关键字集合 S 和 truss 值 k，输出社区搜索的结果包含查询点 v_q 的所有属性 truss 社区。首先使用频繁挖掘算法 FP-growth 从查询点的关键字集合 S 和邻居 $N(v_q)$ 生成支持度为 k 的候选属性关键字集合，并对它们进行分组，即 $\Psi_1, \Psi_2, \ldots, \Psi_h$(第 1 行)。然后我们初始化属性 truss 社区的结果集合 $\mathcal{R}$ 为空集 $\emptyset$，并且把属性关键字集合的大小设置为 h。对于每个候选属性关键字集合 $S' \in \Psi_i$，我们首先从属性图中找到包含 S' 的子图 $G[S']$，然后从子图 $G[S']$ 找到包含 v_q 的 k-truss 社区 (第 5-6 行)。如果返回的 k-truss 社区集合不为空，我们添加它

们到结果集 $\mathcal{R}$(第 7-8 行)。如果我们检查 Ψ_i 中的所有属性关键字集合之后发现 $\mathcal{R}$ 仍然是空的，我们就减少 i 值开始下一个循环，否则我们会停止搜索过程 (第 9-12 行)。最后返回结果为查询的社区集 $\mathcal{R}$。

算法 1: ATCBasic(G, v_q, S, k)

输入: G, a vertex v_q, a keyword set S, and an integer k

输出: All the attributed truss communities containing v_q

```
1  generate Ψ_1, Ψ_2, ..., Ψ_h using S and N(v_q);
2  R ← ∅; i ← h;
3  while i ≥ 1 do
4      for each S' ∈ Ψ_i do
5          find subgraph G[S'] from G;
6          H ← findKTrussCom(G[S'], v_q, k);
7          if H ≠ ∅ then
8              R ← R ∪ H;
9      if R = ∅ then
10         i ← i − 1;
11     else
12         break;
13 Output the communities in R;
```

在算法2中，函数 findKTrussCom 是查找一个包含 v_q 的 k-truss 社区算法。输入参数包含属性图 G、查询顶点 v_q 和 truss 值 k，输出包含查询顶点 v_q 的所有 k-truss 社区。首先使用 truss 分解算法求出 G' 中所有边的 truss 值 (第 1 行)。然后使用二进制变量 visited 来存储每条边的访问状态，如果被访问就是 TRUE，未访问就是 FALSE，所以对所有边初始化为 FALSE(第 2 行)。对于一条未访问的边 (u, v_q)，如果 $\tau(u, v_q) \geq k$，即它的 truss 值大于参数 k，那么就创建一个新的 k-truss 社区 H_m，并将这条边加入队列 Q 中 (第 5-8 行)。然后使用经典的宽度搜索算法 BFS 由查询顶点 v_q 的邻接边向外一层一层的遍历，在队列中，将边 (u, v_q) 加入社区 H_m 中。然后检查顶点 u 和 v_q 的所有共同邻居点 w，即所有三角形 $\triangle_{wuv_q}$，(u, w) 和 (v_q, w) 分别是它们形成的邻接边。如果它们的邻接边 (u, w) 和 (v_q, w) 的 truss 值都大于等于 k，那么就将未访问的边加入队列中，并将边的访问状态写为 TRUE(第 9-17 行)。当队列 Q 为空时，这就

意味着 v_q 的一个邻居点查询完成，新的 truss 社区 H_m 查询结束，将 truss 社区 H_m 添加到结果集 $\mathcal{H}$ 中，接下来将循环继续 v_q 的下一个邻居点。最后返回结果包含查询顶点 v_q 的所有 k-truss 社区。

算法 2: findKTrussCom(G', v_q, k)

输入: A graph G', a vertex v_q, and an integer k

输出: All the k-truss communities containing q

```
1  compute τ(e) of each edge e ∈ V(G') by truss decomposition;
2  e.visited ← FALSE, ∀e ∈ E(G');
3  m ← 0; H ← ∅;
4  for u ∈ N(v_q) do
5      if τ(u, v_q) ≥ k and (u, v_q).visited = FALSE then
6          (u, v_q).visited ← TRUE;
7          m ← m + 1; H_m ← ∅;
8          Q ← ∅; Q.enqueue((u, v_q));
9          while Q ≠ ∅ do
10             (u, v_q) ← Q.dequeue();
11             H_m ← H_m ∪ {(u, v_q}};
12             for each w ∈ N(u) ∩ N(v_q) do
13                 if τ(u, w) ≥ k and τ(v_q, w) ≥ k then
14                     if (u, w).visited = FALSE then
15                         (u, w).visited = TRUE;
16                         Q.enqueue((u, w));
17                     if (v, w).visited = FALSE then
18                         (v_q, w).visited = TRUE;
19                         Q.enqueue((v_q, w));
20         H ← H ∪ {H_m};
21 return H;
```

算法2中 findKTrussCom 函数的时间复杂度为 $O(|E(G')|^{1.5})$。因此，算法1中 ATCBasic 函数的总体时间复杂度是 $O(n_s|E(G'_{max})|^{1.5})$，其中 n_s 是候选关键字集合的数量，G'_{max} 是包含候选关键字集合的最大子图。

3.3 本章小结

本章主要就属性网络图中密集社区搜索算法研究问题提出了一个基本的解决方案，并介绍了属性关键字反单调性和领域约束两个性质。然后使用了关联算法 FP-Growth 减少候选属性关键字集合的数量，并结合两个性质提出了自上而下的 ATCBasic 算法。对于每个候选属性关键字集合，每次都要从原属性网络图去搜索包含当前属性关键字集合的子图，其总体的时间复杂度是 $O(n_s|E(G'_{max})|^{1.5})$，这也是 ATCBasic 算法最耗时的关键部分。

4 基于索引的算法

在算法 ATCBasic 中，对于每个候选属性关键字集合，我们需要反复去检查是否存在 k-truss 社区，这是非常耗时的，其时间复杂度为 $O(|E(G)|^{1.5})$。在本节中，为了加速 k-truss 社区搜索的计算，本节提出了一种保存 truss 等价信息和节点超点信息的新型混合索引，并基于这个混合索引设计了新的 ATCIndex 算法，可以更加高效地搜索 k-truss 社区。

4.1 超图

在介绍和深入了解新型混合索引结构的细节之前，首先介绍有关 k-三角形连通性、k-truss 等价 [12] 等概念定义，其表现了一个 truss 社区中边与边之间的一种强连接关系。

定义 4.1.1 **(k-三角形)** 有这样一个三角形 $\triangle_{uvw}$，如果每个构成边的 truss 值不小于 k，则这个三角形就称为 k-三角形，即 $\min\{\tau(u,v),\tau(v,w),\tau(u,w)\} \geq k$。

定义 4.1.2 **(k-三角形连通性)** 如果两个 k-三角形 $\triangle_s$ 和 $\triangle_t$，它们之间存在一系列三角形 $\triangle_1$,...,$\triangle_n$ $(n \geq 2)$，其中 $\triangle_s = \triangle_1$, $\triangle_t = \triangle_n$，并且 $1 \leq i \leq n$, $\triangle_i \cap \triangle_{i+1} = \{e|e \in E_G\}$ and $\tau(e) = k$，即这一系列三角形邻接的三角形之间的公共边的 truss 值等于 k，那么我们就称这两个 k-三角形时 k-三角形连通的，表示为 $\triangle_s \overset{k}{\leftrightarrows} \triangle_t$。

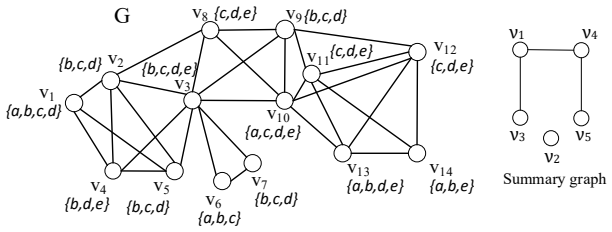
类似地，两个边 $e, e' \in E(G)$ 如果是 k-三角形连接，表示为 $e' \overset{k}{\leftrightarrows} e$，这样有两种情况，一种是 e 和 e 属于同一个 k-三角形，另一种是它们分别在两个 k-三角形连通的三角形中，即 $e \in \triangle_s$, $e' \in \triangle_t$ such that $\triangle_s \overset{k}{\leftrightarrows} \triangle_t$。

定义 4.1.3 **(k-truss 等价)** 给定任意两条边 $e, e' \in E_G$，如果它们的 truss 都等于 k $(k \geq 3)$，即 $\tau(e) = \tau(e') = k$，并且 $e \overset{k}{\leftrightarrows} e'$，那么它们是 k-truss 等价的，表示为 $e \overset{k}{=} e'$。

对于具有 truss 值等于 k 的边 $e \in E(G)$，我们定义它的等价类为 $\mathcal{C}_e = \{e' | e' \overset{k}{=} e, e' \in E(G)\}$。所有等价类的集合形成是相互排斥的，将整个图的边进行了详细的划分。特别地，任何一个等价类 Ce 由具有相同 truss 值的边组成，并且也是 k-三角形连接，所以每个等价类 C_e 是一个 k-truss 社区 (可能不是最大的)。

定义 4.1.4 **(超图)** 基于 k-truss 等价，我们可以构造一种称为超图 $\mathcal{G}(\mathcal{V}, \mathcal{E})$ 的新型结构索引，其中 $\mathcal{V}$ 是超节点集，$\mathcal{E}$ 是超边集。$\nu \in \mathcal{V}$ 表示不同的等价类，两个超节点 $\nu, \mu \in \mathcal{V}$ 之间的边称为超边 (ν, μ)，表明两个等价类之间是可以三角形连接，即对于 $\exists e \in \nu$ 和 $\exists e' \in \mu$ 都有 $e \leftrightarrow e'$。

构建这种超图的主要步骤如下：首先，我们使用 truss 分解算法计算出属性图 G 中每条边的 truss 值，并按照边的 truss 值对边进行分组，将相同 truss 值得边放在同一个小组中。然后从最小的 truss 值的小组中的一条边开始，使用 BFS 遍历寻找拥有相同的 truss 值的边并且要求边与边之间是三角形连接的，将这些边合在一起形成一个超点。所以一个超点是一个社区，里面存储的拥有相等 truss 值的边，是一个边的集合，并且每个超点拥有属于自己独有的 ID。对于每条超边，我们在 BFS 遍历时会记录以前探索过的超点 ID，当两个超点内的有一对边是三角形连接的，那么这两个超点之间会形成一条超边。所以说明两个超点是可以三角形连通，也就告知了每条超边的两个端点的超点的 truss 一定不相等，如果相等，那么它们将会形成一个超点。详细信息可以参考 [12]。



Super-node	Included edges
$\nu_1(k=3)$	$(2,8)$
$\nu_2(k=3)$	$(3,6),(3,7),(6,7)$
$\nu_3(k=4)$	$(1,2),(1,4),(1,5),(2,3),(2,4),$ $(2,5),(3,4),(3,5),(4,5)$
$\nu_4(k=4)$	$(3,8),(3,9),(3,10),(8,9),(8,10),$ $(9,10),(9,11),(9,12)$
$\nu_5(k=5)$	$(10,11),(10,12),(10,13),(10,14),(11,12),$ $(11,13),(11,14),(12,13),(12,14),(13,14)$

图 4.1: 属性网络图和超图

例 4.1.1 在图4.1中，其最左边是原属性图 G，包含 14 个顶点，30 条边。中间则是超图 $\mathcal{G}$，它是基于 k-truss 等价类建立的，包含 5 个超点，3 条超边。每个超点中详细信息如其右表所示。其中，边 $(2,8)$ 和边 $(3,6)$,$(3,7)$,$(6,7)$ 虽然拥有相同的 truss 值，但它们属于两个不同的超点 ν_1，ν_2，这是因为它们不是三角形连接的。超点 v_2 的边和

其他超点里的边都不能三角形连接，所以超点 v_2 是一个独立的超点。超点 $_3$ 代表一个 4-truss 子图，它包含 9 条边，每条边的 truss 值为 4 且是三角形连接的。超点 v_4 也是一个 4-truss 子图，和超点 v_3 中间夹着一个超点 truss 值为 3 的 v_1，所以不是一个 4-三角形连接，因此它们之间也没有超边。

4.2 一种新的混合索引

为了更加有效地进行属性 truss 社区搜索，本文构建了一个新型混合索引，包括以下两个部分：truss 等价索引和节点索引。

- truss 等价索引。truss 等价索引是一个基于 k-truss 等价构建的超图 $\mathcal{G}(\mathcal{V},\mathcal{E})$ 的索引。超图把原先庞大的属性图缩小，搜索起来更加方便快捷。在每个超点中，我们存储原属性图 G 中拥有相同的 truss 值的边集合，并且边与边之间是三角形连接的。
- 在节点索引中，对于每个节点，构建一张哈希表，用于存包含此节点的超点 ID 集合，即对于每个 v_i，我们存储包含 v_i 的超点的集合。这样在社区搜索的时候，可以很快的找到包含查询顶点 v_i 的 k-truss 社区，此时基本保证了结构上的密集性。

例 4.2.1 在图4.1中，中间的概略图即超图就是原属性网络图的 truss 等价索引，每一个超点里面存储的都是拥有相等 truss 值的边，将原属性网络图中的边进行了分类分组。对于顶点 v_3，构建一张哈希表存储包含 v_3 的超点集合则有 $\{\nu_2,\nu_3,\nu_4\}$，其中 ν_2 是一个 3-truss 超点，ν_2 ν_3 都是 4-truss 超点。当查询顶点是 v_3 的时候，通过节点索引就可以快速找到包含 v_3 的 k-truss 社区，省去了求 k-truss 社区的繁琐计算，大大增加了查询效率。

4.2.1 基于混合索引的 ATCIndex 算法

基于新型的混合索引首先提出了一个 ATCIndex 算法，其伪代码如算法3所示。输入参数包含属性图 G、查询顶点 v_q、查询顶点的属性关键字集合 S 和 truss 值 k，输出包含查询顶点 v_q 的所有 k-truss 社区。和 ATCBasic 算法相比，大大缩减了社区搜索时间，并且搜索的社区在结构和属性语义上密集成度更高。

与 ATCBasic 算法的主要区别在于算法3中第 2 行和第 6-7 行。首先对原属性网络

图 G 应用 truss 等价性质建立超图 $\mathcal{G}$，即 truss 等价索引。在算法3第 2 行中，将超图 $\mathcal{G}$truss 等价索引传入算法4中，并且基于节点索引快速地找到所有包含 q 点的 k-truss 社区 $H_1, \ldots, H_l$，使得结构上相对密集，筛选掉了很多不满足的顶点。在第 7 行中，并不像算法1中在整个原属性网络图 G 中搜索满足属性关键字集合的子图 $G[S']$。而是对于每个属性关键字集合 $S' \in \Psi_i$，只需要验证在每个包含查询顶点 v_q 的 k-truss 社区 H_j 中是否存在子图 $G[S']$(第 6-7 行)，这样可以省去很多属性匹配的计算，从而大大提高社区搜索效率。

算法 3: ATCIndex(G, v_q, S, k)

输入: A graph G, a vertex q, and an integer k

输出: All the attributed truss communities containing q;

```
1  generate Ψ_1, Ψ_2, ..., Ψ_h using S and N(q);
2  {H_1°, ..., H_l°} ← kTruss-Summary (G, v_q, k);
3  R ← ∅; i ← h;
4  while i ≥ 1 do
5      for each S' ∈ Ψ_i do
6          for j = 1 to l do
7              find subgraph G[S'] containing v_q from H_j°;
8              H ← findKTrussCom(G[S'], v_q, k);
9              if H ≠ emptyset then
10                 R ← R ∪ H;
11     if R = ∅ then
12         i ← i − 1;
13     else
14         break;
15 Output the communities in R;
```

算法4是在超图 $\mathcal{G}$truss 等价索引中找到所有包含 v_q 的 k-truss 社区。输入参数包含原属性图的超图结构索引 $\mathcal{G}(\mathcal{V}, \mathcal{E})$、查询顶点 v_q 和 truss 值 k。输出包含查询顶点 v_q 的所有 k-truss 社区。首先，为了避免重复访问超点，使用一个二进制变量集 *visited* 保存每个超点的访问状态，已访问表示为 TRUE，未访问表示为 FALSE，所以初始化为 FALSE(第 1-2 行)。其次通过节点索引快速找到包含 v_q 的超点集合 $\mathcal{V}'$(第 3 行)。然

后对于每个超点 $\nu \in \mathcal{V}'$，通过 BFS 搜索方式开始添加 truss 值大于等于 k 的超点进入队列 Q，直到队列为空。队列中都是满足条件的超点，这些超点之间并存在超边，从而形成一个新的 k-truss 社区 (第 7-16 行)。继续下一个超点，直到找到所有包含 v_q 的 k-truss 社区，并作为结果返回。

节点索引找到的包含 v_q 的 k-truss 社区和算法4返回的 k-truss 社区不一定相同，算法4对 v_q 所在的 k-truss 社区进行了扩展。并且如果两个包含 v_q 的超点都满足条件且超点之间有超边，那么这两个超点还会合并成一个 k-truss 社区。

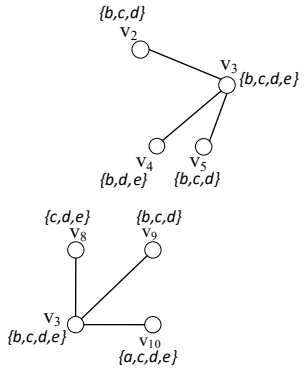
算法 4: kTruss-Summary $(\mathcal{G}, v_q, k)$

输入: A summary graph $\mathcal{G}$, a vertex v_q, and an integer k

输出: All the k-truss communities containing v_q;

```
1  for each ν ∈ V do
2      ν.visited ← FALSE;
3  find the set of super-nodes containing q, V';
4  l ← 0;
5  for each ν ∈ V' do
6      if τ(ν) ≥ k and ν.visited = FALSE then
7          ν.visited ← TRUE;
8          l ← l + 1; H_l° ← ∅;
9          Q ← ∅; Q.enqueue(ν);
10         while Q ≠ ∅ do
11             ν ← Q.dequeue();
12             H_l° ← H_l° ∪ {e | e ∈ ν};
13             for each (ν, μ) ∈ E do
14                 if τ(μ) ≥ k and μ.visited = FALSE then
15                     μ.visited ← TRUE;
16                     Q.enqueue(μ);
17 return {H_1°, ..., H_l°};
```

例 4.2.2 在图4.1中，一个属性图 G 及其超图 $\mathcal{G}$。对于一个查询顶点 v_3 和属性关键字集合 $S = \{b, c, d, e\}$，基于超图 $\mathcal{G}$，我们可以找到社区 H_1°(包含超点 ν_3) 和 H_2°(包含超点 ν_4 和 ν_5)。分别在社区 H_1° 和 H_2° 中就查询顶点 v_3 和其社区中的邻居顶点生



Set	Keyword sets
ψ_1	$\{b\}$,$\{d\}$
ψ_2	$\{b, d\}$

Set	Keyword sets
ψ_1	$\{c\}$,$\{d\}$
ψ_2	$\{c, d\}$

图 4.2: *Two subgraph of candidate generation*

成的候选属性关键字集合，其结果如图4.2所示。然后从候选关键字集合中找到共享最多共有关键词的社区，得到的结果有两个 truss 社区共享最多共有关键词，即社区 H_1 包含节点 $\{v_1,v_2,v_3,v_4,v_5\}$，共有属性关键字集合为 $\{b, d\}$。另一个是社区 H_2 包含节点 $\{v_3,v_8,v_9,v_{10}\}$，共有属性关键字集合为 $\{c, d\}$。

4.3 改进的 ATCImprove 算法

为了进一步提高效率，我们提出了一种新的策略来过滤掉没有希望的关键字集，并设计出一种改进的 ATCImprove 算法来加速计算 k-truss 社区搜索的过程，发现目标属性 truss 社区。

4.3.1 减少候选关键字集

通过观察来自图4.1和图4.2中的例子受到以下两个启发，想到了一个减少候选属性关键字集合数量的办法。

观察 1：考虑图中查询顶点 v_3 和其属性关键字为 $\{b, c, d, e\}$，设置 k = 4。很明显，节点 v_6 和 v_8 虽然在属性关键字上满足条件，但它们永远不会包含在 ATC 的最终 k-truss 社区结果中。因为边 (v_3, v_6) 和 (v_3, v_8) 的 truss 值都小于 4，从而不满足结构上密集的条件。因此，在生成候选关键字集之前，我们应该过滤掉 v_3 的 truss 值小于 k 的邻接边。

观察 2：考虑图中查询顶点 v_3 和其属性关键字为 $\{b, c, d, e\}$。共享最多关键字的 k-truss 社区是 H_1° 或者 H_2°，但不能将它们合在一起，因为它们之间没有超边，即不能

三角形连接。从而，我们在生成候选关键字集时，我们只需要考虑来自同一个 k-truss 社区的邻居。

根据这些观察，本文得出以下一条引理。

引理 4.3.1 如果有两个 truss 值等于 k 的超点 ν_1 和 ν_2，它们都包含顶点 $_q$，且它们之间没有超边连接。那么它们将属于不同的 k-truss 社区。

通过 k-truss 等值的定义可以很容易地得出这一点。如果它们属于相同的 k-truss 社区，那么它们之间必有超边连接，因为超边是三角形连接形成的。没有超边说明它们之间无法通过一系列三角形进行连通。基于以上定理，我们可以生成频繁的候选集通过划分邻居来设定。

例 4.3.1 在图4.1中一个属性图 G,如果查询点是 v_3,其属性关键字为 $S = \{b, c, d, e\}$，设置 k = 4。首先我们找到包含查询点 v_3 的两个 k-truss 社区 H_1° 和 H_2°。在 H_1° 中 v_3 的邻居点有 $\{v_2, v_4, v_5\}$，这 4 个点生成 3 个频繁子集，如图4.2中表所示。在 H_2° 中 v_3 的邻居点有 $\{v_8, v_9, v_{10}\}$，这 4 个节点生成 3 个频繁关键字集。相对于前一个例子中的 9 个频繁子集数量大大减少了。

4.3.2 ATCImprove 算法

基于以上观察和定理，本文设计了一个改进的 ATCImprove 算法如算法5所示。输入参数包含属性图 G、查询顶点 v_q、查询顶点的属性关键字集合 S 和 truss 值 k，输出包含查询顶点 v_q 的所有 k-truss 社区。首先从 truss 等价索引和节点索引新型混合索引中查找到包含查询顶点 v_q 的所在的所有超点，即所有 k-truss 社区 $H_1, H_2, ...H_l$(第 1 行)。对于每个 k-truss 社区 H_i，利用查询顶点 v_q 和它在当前 k-truss 社区中的邻居顶点之间通过关联算法 FP-Growth 生成候选属性关键字集合 $\Psi_{h_i}^i$(第 3-4 行)。这时就可以知道了查询顶点的最大候选属性关键字集合的大小，并将最大值赋值为变量 j(第 5 行)。然后在每个候选 k-truss 社区 H_i 中，根据包含关键字的数量 h_i 计算出每个候选属性关键字集合 $\Psi_{h_i}^i$ 的候选节点集，即在 k-truss 社区 H_i 中找出包含 i 个关键字的节点集合子图 $G[S']$(第 6-10 行)。接着通过算法2在子图 $G[S']$ 中检查是否存在 k-truss 社区 $G_k[S']$，如果存在就将这些社区 $G_k[S']$ 放入结果集合 $\mathcal{R}$ 中。当 while 循环一次后，判断结果集合是否为空，如果为空，将共享的最大属性关键字集合大小 l 减 1，继

续下一个循环，直到候选属性关键字集合遍历完毕。如果结果集合不为空，那么停止遍历，因为已经找到我们需要的属性 truss 社区。此时的属性 truss 社区是共享最多属性关键字 (属性语义密集) 和顶点之间结构紧密 (结构密集) 的社区。

算法 5: ATCImprove(G, v_q, S, k)

输入: G, a vertex v_q, a keyword set S, and an integer k

输出: All the attributed truss communities containing v_q

```
1  find all the k-truss communities {H_1, ..., H_l} containing v_q;
2  R ← ∅;
3  for i = 1 to l do
4      generate Ψ^i_1, Ψ^i_2, ..., Ψ^i_{h_i} using S and N_{H_i}(v_q);
5  j ← max_{1≤i≤l} h_i;
6  while j ≥ 1 do
7      for i = 1 to l do
8          if h_i ≥ j then
9              for each S' ∈ Ψ^i_j do
10                 find subgraph G[S'] from H_i;
11                 find k-truss community G_k[S'] from G[S'];
12                 if G_k[S'] exists then
13                     R ← R ∪ {G_k[S']};
14     if R = ∅ then
15         j ← j − 1;
16     else
17         break;
18 Output the communities in R;
```

和 ATCIndex 算法相比，ATCImprove 算法主要在一个细节上进行了优化，那就是属性关键字集合的频繁子集挖掘部分。在 ATCIndex 算法中只进行了一次 FP-Growth 算法计算，即在一开始就计算查询顶点 v_q 和其在原属性图中邻居点之间的频繁子集项。但 ATCImprove 算法仅仅在包含查询顶点 v_q 的 k-truss 社区中进行多次频繁子集项挖掘。虽然调用 FP-Growth 算法次数增多了，但是这样得到的候选属性关键字频繁子集数量减少了。很明显，在 FP-Growth 算法计算中时间可以几乎不计，减少候选属

性关键字集合数量会在整个社区搜索中减少大量的计算，从而减少搜索时间。

4.4 本章小结

本章考虑到了 ATCBasic 算法的不足之处，提出了两种新颖的索引结构：超图 truss 等价索引和节点索引。并介绍了 k-三角形、k-三角形连通性、k-truss 等价和超图等概念定义，说明了超图 truss 等价索引的有效性和优越性。接着设计了基于两种新颖的索引的 ATCIndex 算法，算法利用混合索引快速定位到包含查询顶点的 k-truss 社区，并在 k-truss 社区中去搜索共享最多属性关键字的 k-truss 社区。最后，在 ATCIndex 算法的基础上进行了改进，并提出了改进的 ATCImprove 算法，该算法进一步减少了候选属性关键字的数量，从而提高了查询效率。

5 基于候选属性关键字的算法

在前一章的基于混合索引的两种算法中，它们都是从结构上密集性先去筛选出子图。考虑到本文的目标是在属性图上进行社区搜索，搜索的社区同时满足结构上密集性和属性语义上密集性。在算法3和算法5中都是从前一个目标开始筛选出子图，缩小搜索范围，然后再在此子图上找到共享最多的属性关键字的 k-truss 社区。那如果先从属性语义上密集筛选出子图，再在子图上进行社区搜索到结构上密集的 k-truss 社区的方式是否效果更好呢？

因此，在本章中就是从查询顶点的属性关键字集合中出发，先找到属性语义上相对密集的子图社区，然后再在此子图上进行社区搜索 k-truss 社区。在阐述详细的思路之前，本章先讲解算法中使用到另一种索引——节点索引。

5.1 属性关键字索引

在属性关键字索引中，首先存储一个反向属性关键字列表，用于保存包含某个关键字属性 w_i 的节点集合 V_i，并对集合中的顶点应用 k-truss 等价算法求出超图，并对每个超点包含的信息存储到文件中。这样便可以在社区搜索的时候，按照查询点的属性集合中关键字快速筛选出包含相应的顶点，并且快速找到满足 truss 值要求的顶点。这样可以筛选出很多顶点，大大缩小了搜索范围。

例 5.1.1 在图4.1中，对于属性关键字 c，在原属性网络图 G 中包含此属性关键字的顶点列表有 $\{v_1, v_2, v_3, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}, v_{12}\}$，现在把这些顶点当作一个子图，然后对它求出超图 truss 等价。很显然，边 $\{(v_1, v_5), (v_1, v_2), (v_2, v_8), (v_2, v_5), (v_3, v_5)\}$ 的边只包含一个三角形，即支持度为 1，所以 truss 值都是 3，它们组成了一个 3-truss 的超点。剩下的所有边包含两个及以上三角形，所以组合成了一个 4-truss 超点。两个超点之间能进行三角形连通，所以它们之间有一条超边。当一个查询顶点包含此属性关键字 c 的时候，通过属性关键字索引便可快速找到包含此属性关键字的顶点集合，并且可以进一步找到满足相应查询参数 k 值的顶点，这样大大的缩小了搜索范围。

5.2 KIndexInc 算法

考虑到一个查询顶点的属性关键字数量相对来说比较小，所以基于查询顶点的属性关键字出发并设计一个自下而上的 KIndexInc 算法。基于属性关键字索引首先找到属性语义上相对密集的子图社区，然后再在此子图上进行社区搜索目标 k-truss 社区。

大致思路是：首先求出查询顶点和其邻居顶点之间的属性关键字频繁子集，然后对子集中存在的属性关键字进行组合增加。采用一个倒二叉树的形式，根节点是所有属性关键字集合，然后向上一层是单独的每个属性关键字，在此之前对属性关键字基于其索引所在的社区的大小进行了排序，首先处理较小的属性关键字。接下来每一层的子节点都是不包含祖先节点的属性关键字，采用深度搜索的遍历方式进行候选属性关键字组合，快速筛选出属性语义密集的子图社区。最后在子图社区上进行 k-truss 社区搜索。

算法 6: KIndexInc(G, v_q, S, k, $attrIndexFile$)

输入: G, a vertex v_q, a keyword set S, an integer k, and $attrIndexFile$

输出: All the attributed truss communities containing v_q

```
1  N_(v_q) ← trussness[nbr] ≥ k;
2  if |N_(v_q)| == 0 then
3      return ∅;
4  generate Ψ_1, Ψ_2, ..., Ψ_h using S and N_(v_q);
5  for i = 1 to h do
6      for each S' ∈ Ψ_i do
7          str ← sortS' by ascending;
8          isExistPsi[str] = TRUE;
9          S' insert S;
10 load attribute index file into variable attr_com;
11 sort S' ∈ S by k-truss commnunity size;
12 R ←KindexIncDFS(ifExistPsi, 1, FALSE, k, q, S);
13 Output the communities in R;
```

KIndexInc 算法如算法6所示。输入参数包含属性图 G、查询顶点 v_q、查询顶点的属性关键字集合 S、truss 值 k 和属性关键字索引文件 $attrIndexFile$，输出包含查询

顶点 v_q 的所有 k-truss 社区。首先进行一个预处理，将查询顶点 v_q 的邻接边 (v_q, w) 的 truss 值大于 k 的邻居点加入邻居点集合 $N_{(v_q)}$ 中。如果集合为空，即没有满足 truss 值条件的邻居节点，那么此时就返回空集，即没有搜索到社区 (第 1-3 行)。这样就避免了程序后面繁琐的计算，可以提早的结束社区搜索。接着将查询顶点 v_q 的属性关键字和邻居点集合中的节点的属性关键字使用关联算法 FP-Growth 生成候选属性关键字频繁子集项 Ψ_i(第 4 行)。对于候选属性关键字集合 Ψ_i，将里面的属性关键字排序后合并成一个字符串，并使用一个二进制变量存储这个属性关键字组合的存在状态 (第 5-9 行)。这里主要是为了后面以自下而上的方式深度遍历的时候可以将不存在的属性关键字组合给筛选掉。其次加载属性关键字索引文件，将查询顶点 v_q 的每个属性关键字的索引加载到内存中，即将包含属性关键字的顶点 k-truss 社区存储到变量 $attr\_com$ 中 (第 10 行)。对于候选属性关键字集合 Ψ_i，首先对集合中的属性关键字按照属性的 k-truss 社区中边的数量进行从小到大排序 (第 11 行)，然后从拥有最小的 k-truss 社区的属性关键字开始，使用深度遍历 DFS 和其他属性关键字进行组合，即所有候选属性关键字中的属性的所有子集，通过调用函数 KindexIncDFS 进行社区搜索并返回 k-truss 社区结果给 $\mathcal{R}$。

函数 KindexIncDFS 如算法7所示。使用深度遍历的方式，所以第一次是 k-truss 社区最小的属性关键字，将这个属性关键字的社区导入子图结构中 (第 2 行)。然后和当前候选属性关键字集合中的其他关键字的社区进行求节点交集，即从子图中删除没有包含属性关键字 S' 的节点 (第 3-5 行)，以此类推，直到最后的子图的每个顶点都包含候选属性关键字集合 $combineS$。其次可以对子图进行剪枝优化，将顶点度数小于 k-1 的顶点从子图中删除，因为一个 k-truss 社区也是一个 (k-1)-core 社区。从查询顶点 v_q 以 BFS 方式向外一层一层的遍历，删除不连通的部分，因为经过上面筛选后子图可能不是一个连通块。接着就通过算法2检查子图 $G[S']$ 是否存在 k-truss 社区 $G_k[combineS]$(第 7 行)，如果存在就将 k-truss 社区 $G_k[combineS]$ 放入结果集合 $\mathcal{R}$ 中，并且判断当前的候选属性关键字的大小是否超过了最大值，如果超过就更新最大值变量 $maxMatchPsi$(第 8-11 行)。遍历递归下一次的时候，便会从这一层开始，跳过其他层的计算。因为这是当前最大共享属性关键字数量，只需检查是否存在比这更大共享属性关键字的社区。接着递归下一个候选属性关键字，选择或者不选择由变量 $ifSlect$ 决定，直到搜索到所有属性关键字组合。

算法 7: KindexIncDFS($ifExistPsi$, $psiLevel$, $ifSelect$, k, q, S)

输入: $ifExistPsi$, $psiLevel$, $ifSelect$, an integer k, a vertex v_q and a keyword set S

输出: All the attributed truss communities containing v_q

1 **if** $psiLevel \geq maxMatchPsi$ *and* $ifSelect$ **then**
2 $subG[S''] \leftarrow \min|attr\_com[S' \in combineS]|$;
3 **for** *other* $S' \in combineS$ **do**
4 **if** $node \in sub[S'']and node \notin attr\_com[S']$ **then**
5 $subG[S'']$ delete node;
6 subgraph $G[combineS] \leftarrow subG[S'']$;
7 find k-truss community $G_k[combineS]$ from $G[combineS]$;
8 **if** $G_k[combineS]$ *exists* **then**
9 **if** $psiLevel > maxMatchPsi$ **then**
10 $maxMatchPsi \leftarrow psiLevel$;
11 clear R;
12 $\mathcal{R} \leftarrow \mathcal{R} \cup \{G_k[combineS]\}$;
13 KindexIncDFS(ifExistPsi, psiLevel + 1, TRUE, k, q, S);
14 KindexIncDFS(ifExistPsi, psiLevel, FALSE, k, q, S);
15 Output the communities containing v_q;

5.3 KIndexDec 算法

KIndexInc 算法采用从一个属性关键字自下而上开始匹配，自然而然会想到可以从最大的候选频繁属性关键字集合开始，采用自上而下的方式进行属性匹配。因此，本节设计的 KIndexDec 算法便是从最大候选属性关键字集合开始。与 KIndexInc 算法相比最大的优势是，当前的候选属性关键字集合找不到 k-truss 社区的时候就结束搜索，并不需要检查完所有候选属性关键字集合。

KIndexDec 算法如算法8所示。输入参数包含属性图 G、查询顶点 v_q、查询顶点的属性关键字集合 S、truss 值 k 和属性关键字索引文件 $attrIndexFile$，输出包含查询顶点 v_q 的所有 k-truss 社区。算法 1-5 行和算法6一样，接着将候选频繁子集项集合的最大值赋值给变量 i，即从频繁子集集合中自上而下的遍历集合 (第 6 行)。对于候选属性关键字集合 Ψ_i，首先对集合中的属性关键字按照属性的 k-truss 社区的大小进

行从小到大排序，将 k-truss 社区最小的属性关键字的社区导入子图 $subG[S'']$ 结构中(第 7-9 行)。然后与第二小的属性关键字的社区进行求节点交集，即从子图中删除没

算法 8: KIndexDec(G, v_q, S, k, $attrIndexFile$)

输入: G, a vertex v_q, a keyword set S, an integer k, and $attrIndexFile$

输出: All the attributed truss communities containing v_q

```
1  N_(v_q) ← trussness[nbr] ≥ k;
2  if |N_(v_q)| == 0 then
3      return ∅;
4  generate Ψ_1, Ψ_2, ..., Ψ_h using S and N_(v_q);
5  load attribute index file into variable attr_com;
6  R ← ∅; i ← h;
7  while i ≥ 1 do
8      sort S' ∈ Ψ_i by k-truss commnunity size;
9      subG[S''] ← min |attr_com[S'] ∈ Ψ_i|;
10     for other S' ∈ Ψ_i do
11         if node ∈ sub[S'']andnode ∉ attr_com[S'] then
12             subG[S''] delete node;
13         G[S'] ← subG[S''];
14         find k-truss community G_k[S'] from G[S'];
15         if G_k[S'] exists then
16             R ← R ∪ {G_k[S']};
17     if R = ∅ then
18         i ← i − 1;
19     else
20         break;
21 Output the communities in R;
```

有包含属性关键字 S' 的节点 node(第 10-12 行)，以此类推，直到最后的子图的每个顶点都包含候选属性关键字集合。其次可以对子图进行剪枝优化，将顶点度数小于 k-1 的顶点从子图中删除，因为一个 k-truss 社区也是一个 (k-1)-core 社区。从查询顶点 v_q 以 BFS 方式向外一层一层的遍历，删除不连通的部分，因为经过上面筛选后子图可能

不是一个连通块。接着就通过算法2检查子图 $G[S']$ 是否存在 k-truss 社区 $G_k[S']$，如果存在就将这些社区 $G_k[S']$ 放入结果集合 $\mathcal{R}$ 中。当 while 循环一次后，判断结果集合 $\mathcal{R}$ 是否为空，如果为空，将共享的最大候选属性关键字集合大小 i 减 1，继续下一个循环，直到候选属性关键字集合遍历完毕。如果结果集合不为空，那么停止遍历，因为已经找到我们需要的属性 truss 社区。此时的属性 truss 社区是共享最多属性关键字 (属性语义密集) 和顶点之间结构紧密 (结构密集) 的社区。

5.4 本章小结

本文主要介绍两个 KIndexInc 算法和 KIndexDec 算法，两者均基于属性关键字索引，从查询顶点的属性关键字集合中出发，先找到属性语义上相对密集的子图社区，然后再在此子图上进行社区搜索 k-truss 社区。相比前面的 ATCIndex 和 ATCImprove 算法，这两者并没有检查完所有的候选属性关键字集合，减少一部分的候选属性关键字集合的计算。KIndexInc 算法自下而上的方式匹配属性关键字，KIndexDec 算法则是自上而下的方式从最大的候选属性关键字集合出发。

6 实验分析与结论

本章会根据前三章所讲述的算法和模型，在多个真实数据集上进行广泛的实验研究。首先会讲述实验中需要的数据集及其相关信息。接着会讲述实验进行所需的基础环境配置。之后讲述本实验所要进行的多种实验来说明本文中设计的算法的有效性和高效性。首先是本文算法的有效性，包含两个实验。第一个实验是算法返回的社区在属性关键字上的密集程度比较实验，本文通过两种计算方式来计算社区中顶点之间的属性关键字密集性程度。第二个实验是算法返回的社区在结构上的密集程度对比实验，利用社区中每个顶点和每条边的 truss 值的平均值来说明结构上的密集性程度。然后是本文算法的高效性，包含多个实验。最后是本章的小结，实验结果表明本文设计的算法在属性图上社区搜索的有效性和高效性，有效地解决了属性语义密集和结构密集并存的问题。

6.1 数据集

在实验中，本文涉及到了四个真实数据集 DBLP、DBpedia、YAGO 和 Tencent。对于 DBLP，顶点表示作者，边是两位作者之间的合作关系。对于每个作者，我们使用他出版文章的标题中最频繁的前 20 个关键词作为他的关键词集合。对于 DBpedia，每个节点代表一个实体，并属于 272 种类型中一种类型 (例如，animal，architectures，famous places 等)，具有一组属性 (例如，jaguar，Ford 等)。对于 YAGO，它也是一个知识图谱，但它比 DBLP 和 DBpedia 更稀疏。请注意本文没有测试另一个广泛使用的数据集 IMDB，因为它是一个三元图，其中子图的最大 truss 值为 3。Tencent 数据集，由 2012 年 KDD 比赛中提供下载的，一个节点表示一个人，一个组织或者一个微博团队。每条边表示两个用户之间的朋友关系，每个节点的属性关键字集合则是从每个用户的个人资料中提取的。对于 Tencent 数据集，由于用于比赛，涉及个人隐私，所以数据集的属性关键字做了数据脱敏，每个属性关键字由数字表示。数据集的相关信息如表6.1所示。

表 6.1: 实验数据集

Dataset	Vertices	Edges	k_{max}	$\widehat{d}$	$\widehat{l}$
DBLP	2,000,979	9,925,613	287	9.92	14.02
YAGO	2,637,144	5,226,311	28	3.96	9.58
DBpedia	5,897,742	17,642,447	18	5.98	4.26
Tencent	2,320,895	50,133,369	405	43.2	6.96

从在表6.1中，第一列是数据集名称，第二列是数据集中的顶点数量，第三列代表的是数据集中边的数量，第四列则是数据集中最大的 truss 值。最后的两列中，$\widehat{d}$ 表示的是数据集中顶点的平均度数，$\widehat{l}$ 表示的是数据集中顶点拥有的平均属性关键字数量。从表中可以看出，DBLP 和 Tencent 数据集的属性图结构相对来说比较密集，顶点与顶点之间连接紧密。相对于于 DBpedia 数据集，它拥有最多的顶点数量，但是最大 truss 值只有 18，说明 DBpedia 数据集属性图相对来说比较稀疏，顶点与顶点之间连接不够紧密。并且 DBpedia 数据集中顶点的平均属性关键字数量仅有 4.26，这表明了在 DBpedia 数据集上进行社区搜索很难搜索到目标社区。

6.2 实验环境

对于每一个数据集，实验中随机选择 300 个 truss 值为 6 或更大的查询顶点。这样做是为了尽可能确保每个查询顶点都至少有一个 k-truss 社区结果。对于这 300 个查询的结果，实验中也将删除掉最终没有找到 k-truss 社区的数据，因为最终没有找到结果的查询点，它使用的时间几乎是为 0。如果将它计算在内，会整体拉低平均社区搜索时间，这也不是本文算法研究的初衷。因此，所有算法的结果最终会取有搜索结果的平均社区搜索时间。

本文进行了广泛的实验测试来评估本文设计的算法，进行了 ATCBasic，ATCIndex，ATCImprove，KIndexInc 和 KIndexDec 等算法相关的大量实验。本文主要使用的编程语言是 C ++，以此实现所有算法编写，并采用 CentOS 系统的 Intel E5-2640 处理器和 128GB 内存的机器进行编译运行。其中，C/C++ 编译环境版本分别是 Make: GNU Make 4.1、Cmake: cmake version 3.7.2、Gcc: gcc 6.3.0、G++: g++6.3.0，在这样的一台机器上完成了所有的实验。

6.3 实验结果

通过控制不同参数值进行大量的实验，并实验中获得数据制作成相应的实验结果图，并对每个实验结果图进行了分析。

6.3.1 算法的有效性

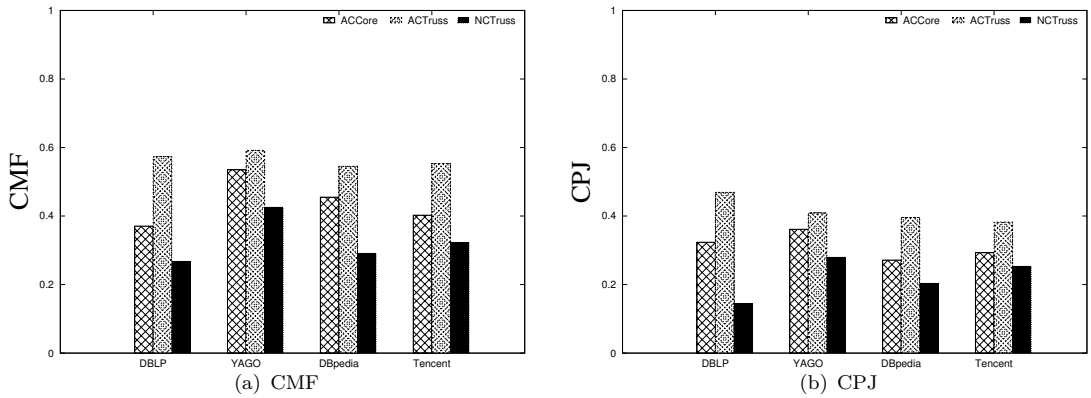
在本小节中进行了两个实验来验证本文设计的算法的有效性。第一个实验是算法返回的社区在属性关键字上的密集程度比较实验，我们通过两种计算方式来计算社区中顶点之间的属性关键字密集性程度。第二个实验是算法返回的社区在结构上的密集程度对比实验，我们会利用社区中每个顶点和每条边的 truss 值的平均值来说明结构上的密集性程度。

6.3.1.1 属性关键字密集程度

为了实验的完整性，本文实现了基于 k-core 的属性社区搜索算法，命名为 ACCore（Attributed Community Core）。然后将本文中设计的最高效的算法 KIndexDec 命名为 ACTruss（Attributed Community Truss）。也实现了在无属性的图中进行社区搜索算法，命名为 NCTruss（NoAttributed Community Truss）。这样，NCTruss 算法代表曾经在无属性网络图上的相关研究，他们只考虑了结构上的密集性，忽略了属性关键字的重要性，使得属性语义密集程度不高。ACCore 算法则说明了在属性图上基于 k-truss 结构比基于 k-core 结构在结构上更加密集，返回的社区中顶点与顶点之间结构更加密集。两个对比的算法来说明本文中所追求的属性图社区搜索的结果社区满足在结构上密集程度高和在属性语义上密集程度高。

返回社区结果需要评估方法来对结果进行判断，本文通过两个评估社区的有效性措施方法 CMF（community member frequency）和 CPJ（community pair-wise Faccard）去评估社区搜索返回的社区中属性的密集程度，如 [1] [2] 中所述。CMF 和 CPJ 的值范围在 0-1 之间，值越大说明社区的密集程度越高。实验表明本文设计的算法可以找到一个具有较高 CMF 值和 CPJ 值的社区。

属性关键字上的密集程度实验并不需要实验所有的数据集，只需要对其中一个数据集进行实验就能说明问题。本文选取了数据集 DBLP 进行本小节的实验测试，因为数



(a) CMF (b) CPJ

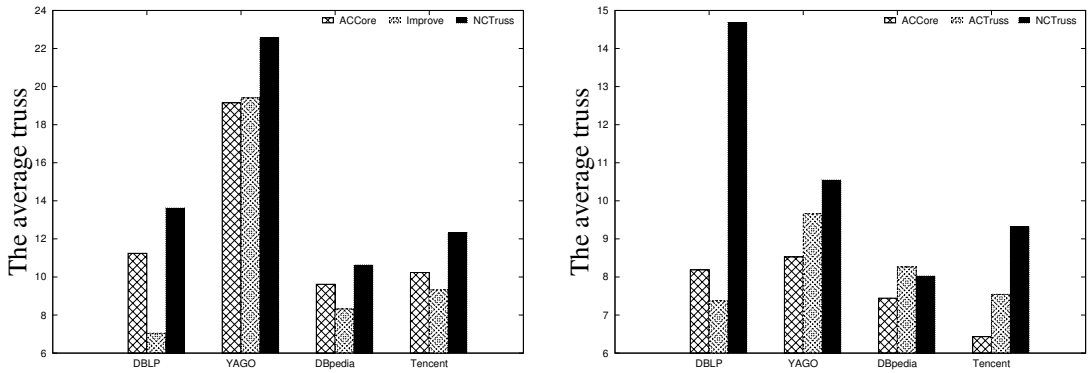
图 6.1: 属性关键字密集性

据集 DBLP 相对于其它三个数据集来说具有代表性，它其中的结构中规中矩，表6.1中可以看出 DBLP 数据集图里的顶点平均度数为 9.92，顶点包含的平均属性关键字数量为 14.02。这表明 DBLP 数据集本身就是一个较为密集的图，每个顶点包含的属性较多，社区搜索起来会更加具有挑战性。

算法 ACCore，ACTruss 和 NCTruss 在 DBLP 数据集上的 CMF 值和 CPJ 值实验结果直方图如图6.1所示。在 CMF 值中，ACTruss 算法结果在四个数据集整体上比另外两种算法好。NCTruss 在三者之中是最差的，这也在意料之中。因为 NCTruss 算法社区搜索出来的结果社区是 truss 结构，即社区中边与边之间是三角形连接的。它只考虑结构上密集性，未曾考虑节点上带有的属性信息。因为 CMF 求的是属性关键字的密集性，NCTruss 算法社区搜索的社区的 CMF 值几乎为 0。再看看算法 ACCore 和 ACTruss，它们的 CMF 值算比较相近。它们之间的区别在于 ACCore 在结构上使用的是经典的 k-core 结构，而本文设计的 ACTruss 算法采用 k-truss 结构。相对来说，三角形架构使顶点与顶点之间的关系更加密切。所以说实验结果也表明 k-truss 结构能使社区更加密集。

6.3.1.2 社区结构密集程度

检验社区结构密集程度，本文使用的方法是分别计算社区中每个顶点的平均 truss 值和每条边的平均 truss 值。因为在前一部分已经说明，k-truss 结构比 k-core 结构使社区中结构更加密集，所以此次实验是计算点和边的 truss 值来评估社区结构密集程度。算法 ACCore，ACTruss 和 NCTruss 在 DBLP 数据集上社区搜索的实验结果直



(a) 顶点的平均 truss 值　　(b) 边的平均 truss 值

图 6.2: 社区结构密集性

方图如图6.2所示。从图中可以看出，它们求出的社区在结构上的密集性并不像属性关键字密集性那么一致。特别地，在顶点的平均 truss 值中，NCTruss 算法所求出的社区结果比本文的 ACTruss 结果更好一些。原因在于 NCTruss 算法完全只考虑社区的结构上的密集性，没有考虑属性关键字密集性，导致它求出的社区有可能是一个大的 k-truss 社区，但是这个社区中的顶点关系并不是很密切。本文设计的 ACTruss 则是综合考虑结构密集性和属性密集性两点，所以求出来的结果可能是一个小一点的 k-truss 社区，但是社区中的顶点在属性关键字上也是最密集的。

例 6.3.1 在图4.1中，此时我们只考虑超点 ν_5 这个 5-truss 社区，社区中包含五个顶点 $\{v_{10}, v_{11}, v_{12}, v_{13}, v_{14}\}$。当查询顶点是 v_{12} 的时候，它的属性集合为 $\{c, d, e\}$，并且此时的 k 值为 3。那么，算法 ACTruss 社区搜索的结果是一个包含三个顶点的 3-truss 社区 $H_1^{\circ}\{v_{10}, v_{11}, v_{12}\}$，而算法 NCTruss 社区搜索的结果是一个包含五个顶点的 5-truss 社区 $H_2^{\circ}\{v_{10}, v_{11}, v_{12}, v_{13}, v_{14}\}$。原因是，$H_1^{\circ}$ 社区中三个顶点共享查询顶点 v_{12} 的全部属性 $\{c, d, e\}$，但是 H_2° 社区中顶点仅共享了 $\{d, e\}$ 两个属性。

6.3.2 算法的效率性能

在本小节中进行了五个实验来评估本文设计的算法的效率性能，分别是索引伸缩性的效率、属性关键字的效率、k 值的效率、属性关键字的伸缩性以及顶点的伸缩性。通过改变查询参数值评估本文的所有算法在数据集中的运行效率，每个实验改变其中一个参数值，其余参数值不变。默认设置查询顶点的属性关键字数量 $|S| = 5$，truss 值 $k = 6$。

表 6.2: 超图索引信息

Dataset	Vertices	Edges	SGVertices	SGEdges	Construct Time
DBLP	2,000,979	9,925,613	994,517	1,531,908	527.81s
YAGO	2,637,144	5,226,311	278,504	340,092	91.87s
DBpedia	5,897,742	17,642,447	2,037,982	1,916,019	426.25s
Tencent	2,320,895	50,133,369	16,293,407	69,302,288	69,101.90s

6.3.2.1 索引伸缩性的效率

本文中的 truss 等价索引是为原属性网络图构造一个概略图——超图，超图相比原图来说对其边进行分组，使得在社区搜索过程中很快速的找到目标结构。实验中，四个数据集的原属性网络图和其超图的相关信息如表6.2所示。

在表6.2中看到，前三个数据集的超图在超点和超边的数量方面都相对比原属性网络图的顶点和边的数量少了许多。而在 Tencent 数据集中，超图中的超点和超边数量明显增加了许多，并且索引建立时间也很长。这是因为一个超点是一个社区，是原属性网络图中边的集合，所以原属性网络图中边的数量是 5000 万之多，所以超点数量达到 1600 万也不足为奇。虽然超图比原属性网络图整体变得更大了些，但是在社区搜索上得效果还是很显著的，相关的实验会在后续中阐述。

本文中使用了三种索引：truss 等价索引，属性关键字索引和节点索引。其中节点索引和 truss 等价索引可以在一起建立，所以它们的索引建立时间合并在一起计算。因此，在建立索引的时间计算上就仅考虑 truss 等价索引 (EquiTruss) 时间和属性关键字索引 (AttrIndex) 时间。

对于每个数据集，随机选择 20%、40%、60%、80% 和 100% 规格比例的顶点集组成新图，测试索引在不同大小的属性网络图的建立时间对比，看看索引建立的伸缩性效率。其索引伸缩性实验结果绘如图 6.3所示。在图中可以看出，truss 等价索引 (EquiTruss) 和属性关键字索引 (AttrIndex) 建立的伸缩性还算理想，增长趋势大致是一条 x^2 曲线。但是在 Tencent 数据集中却表现出异常的现象，这也很好理解。Tencent 数据集的顶点平均属性关键字数量仅有 6.96，在进行属性关键字索引 (AttrIndex) 建立的时候，属性包含的顶点并不是很多，边的数量也减少了很多。但也可以看出比其他数据集的建立时间长。

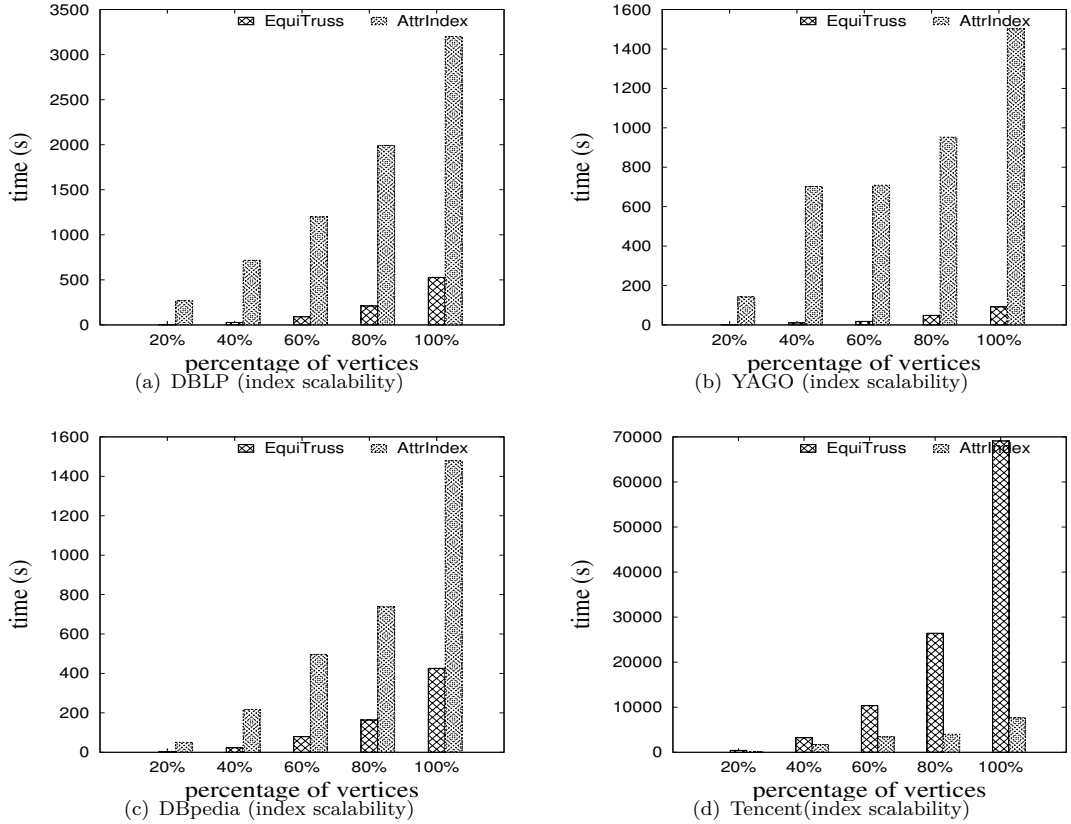


图 6.3: 索引建立的效率性能

6.3.2.2 属性关键字的效率

本小节实验是测试算法在查询顶点的不同属性关键字数量的查询效率。对于一个查询顶点，它所拥有的属性关键字数量是不可控制，所以测试算法在查询顶点不同关键字数量的社区搜索效率实验也是必不可少的。

对于每个查询顶点，随机选择查询顶点的属性关键字集合中的 1、3、5、7、9 个属性关键字数量形成候选查询关键字集。对于查询顶点达不到条件要求的属性关键字数量时，舍弃这个查询顶点，保证每次查询的时候一定是要求的属性关键字数量。其实验结果如图6.4所示。图中横轴表示的是选取查询顶点属性关键字集合中关键字的数量，纵轴表示的是 300 个查询顶点社区搜索的平均时间。实验图整体上表明了所有算法随属性关键字数量的成本增加而随之增加。并且可以很清楚地看出算法 KIndexDec 的效果是最好的，但是在 DBLP 和 DBpedia 数据集中，算法 KIndexDec 和算法 KIndexInc 的查询效率相当。原因是在 DBLP 数据集中顶点的平均属性关键字达到了 14.02，或许搜索到的社区共享的属性关键字数量可能是查询顶点的折中。而在 DBpedia 数据集

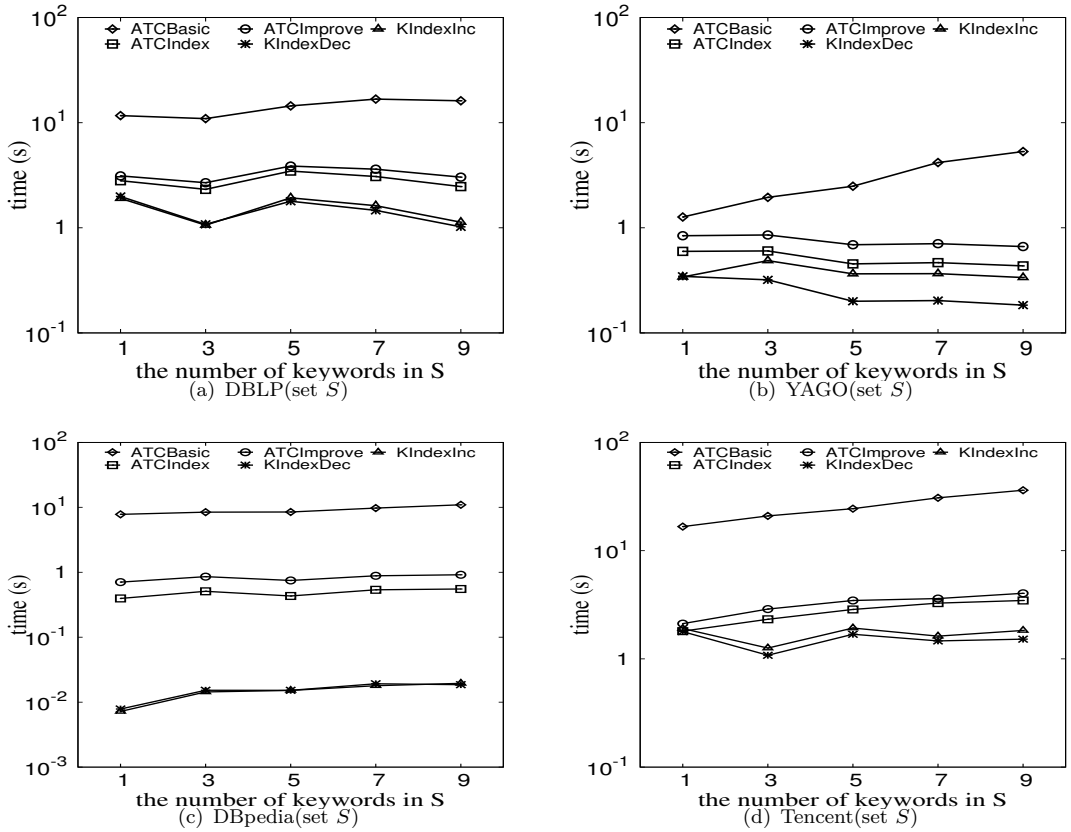


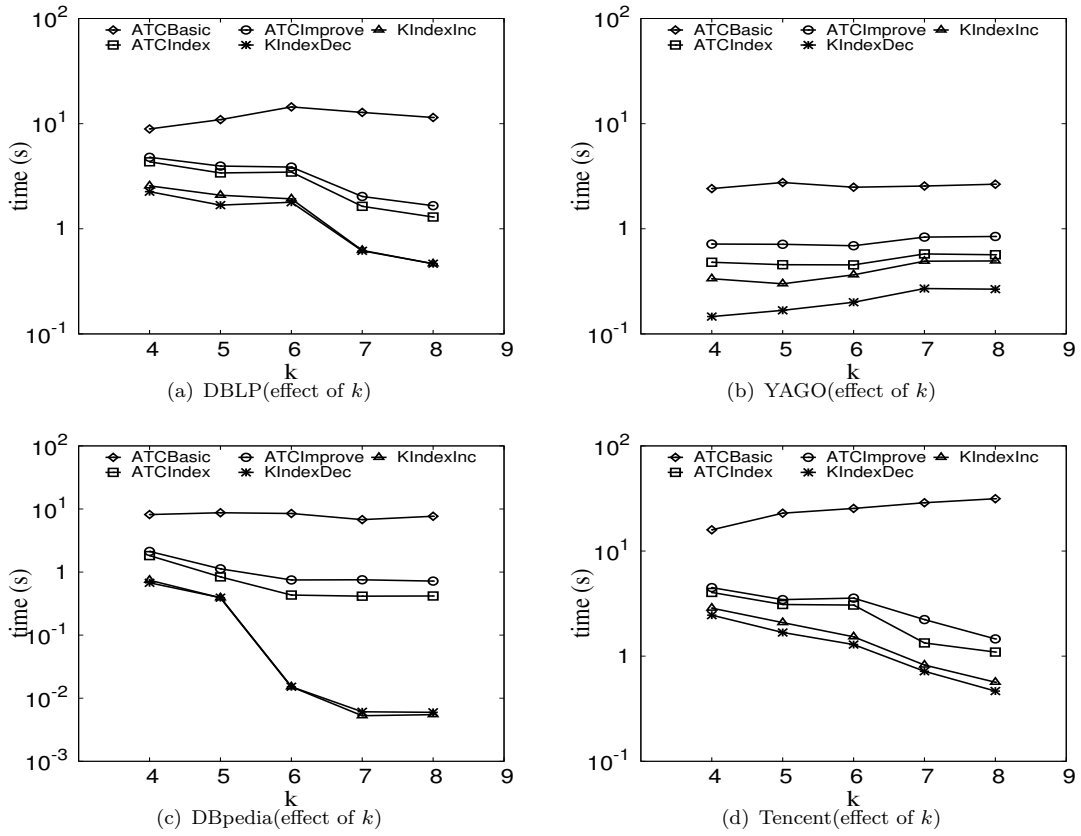
图 6.4: 属性关键字的效率

中，由于顶点的平均属性关键字数量仅为 4.26，所以可能社区在频繁属性关键字挖掘的候选频繁子集的大小为 1，导致了两种算法在社区搜索上的效果是差不多的。而在另外两个数据集就很好的说明了算法 KIndexDec 在属性关键字上的效率是最好的。

6.3.2.3 k 值的效率

对于可变查询参数 truss 值 k，当有不同的查询要求 k 值时，社区搜索的时间也会有所变化。一个好的算法对于不同的请求应该表现出稳定的效率，所以本小节实验就是对不同的查询参数 k 值，测试本文的所有算法在四个数据集上的社区搜索效率。对于每个查询顶点 v_q，实验中 k 值取值为 4、5、6、7、8，并将查询顶点的属性关键字数量设置为 5 个进行实验比较，并将实验数据结果绘制成折线图。

图6.5展示了不同 k 的取值的算法社区搜索效率。图中横轴表示的是查询参数 k 的不同取值，纵轴表示的是 300 个查询顶点社区搜索的平均时间。从图中看出，随着查询参数 k 值的增加整体趋势并不是很好把握，有上升也有下降。随着 k 值的增加，社



(a) DBLP(effect of k)　(b) YAGO(effect of k)

(c) DBpedia(effect of k)　(d) Tencent(effect of k)

图 6.5: k 值的效率

区搜索的时间应该是会随之减少。因为查询的门槛高了就会筛选掉很多的不满足条件的顶点和边，最终社区搜索的社区也将会是一个相对来说更小的一个社区。这可以从属性关键字的反单调性性质得出，对于同一个查询顶点 v_q，社区搜索的 k-truss 社区一定被包含在 (k-1)-truss 社区中。但整体来说实验结果表明的算法 KIndexDec 的效果更好一些，社区搜索中所花费的时间是最少的。

6.3.2.4 属性关键字的伸缩性

本次实验是在查询顶点的属性关键字集合的不同比例大小进行评估，和前面的属性关键字效率实验不同的是，本次实验的取值是按照占比进行实验。对于每个顶点，随机选择查询顶点属性关键字集合中 20%、40%、60%、80% 和 100% 的属性关键字数量进行实验，并设置 k 值为 6，其在各数据集社区搜索的平均时间如图6.6所示。

在图 fig:ascala 中，横轴表示的是查询顶点属性关键字数量的不同比例，纵轴表示的是 300 个查询顶点社区搜索的平均时间。从实验结果可以看出，整体上算法 KIndexDec

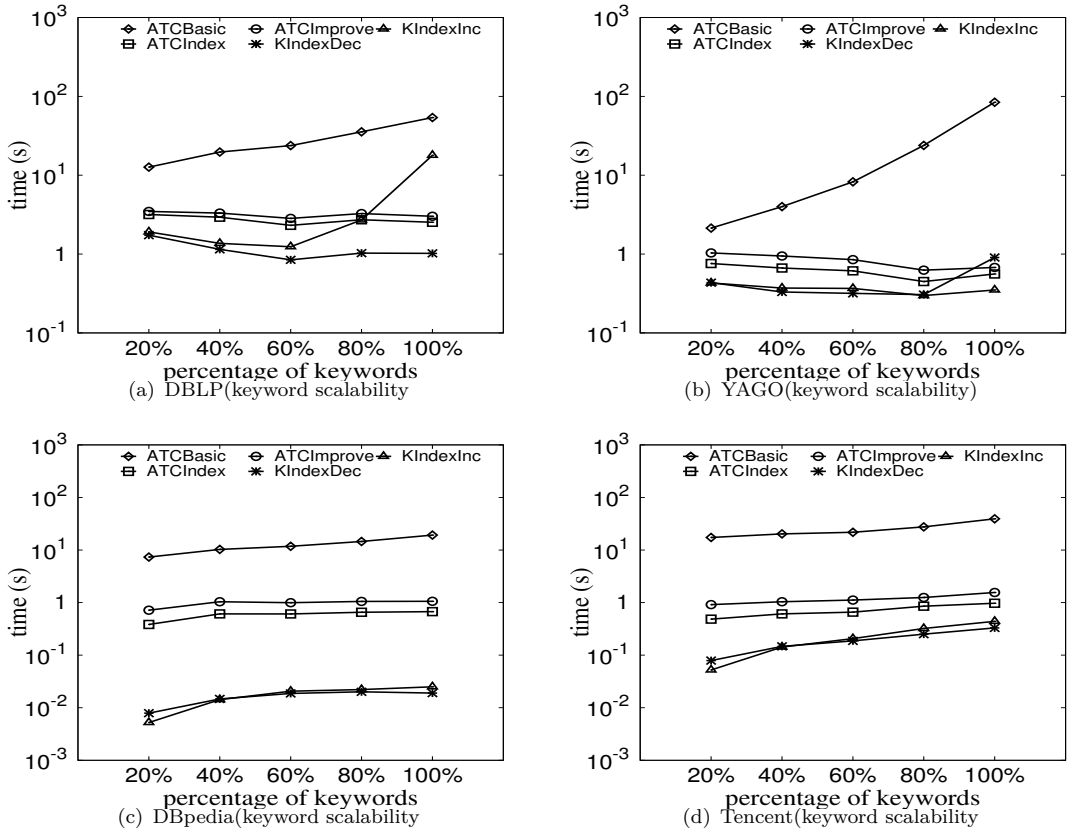
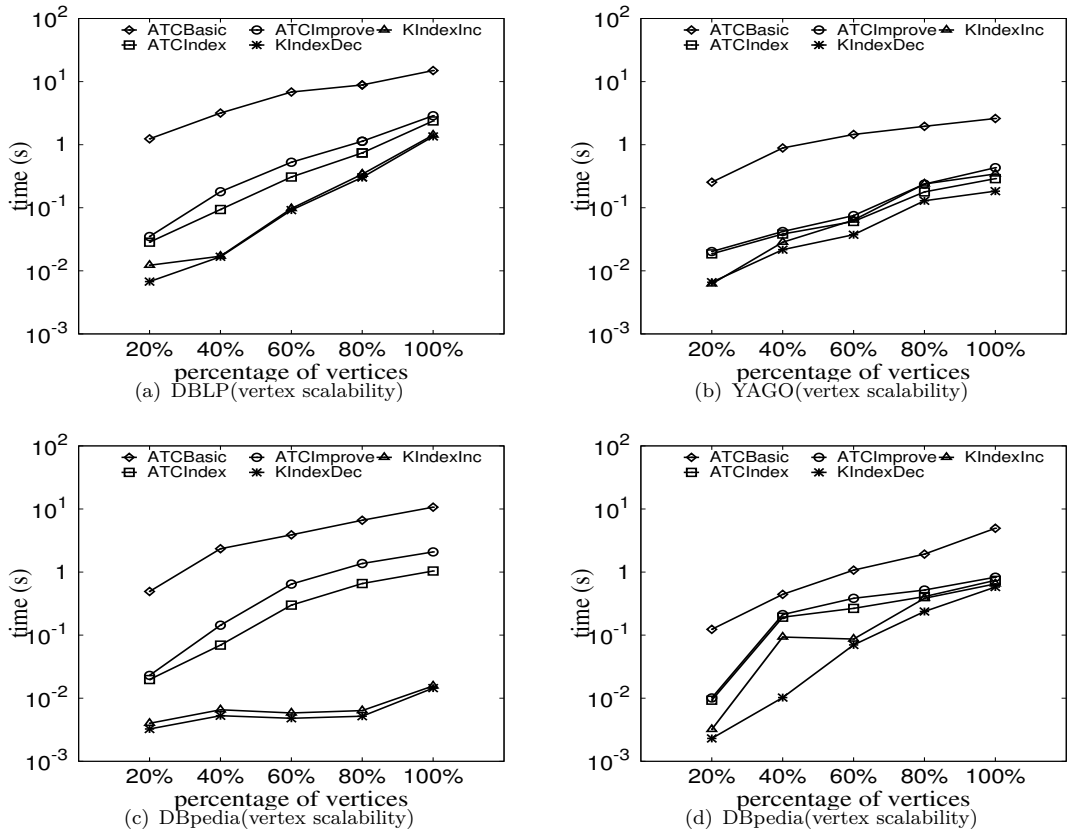


图 6.6: 属性关键字的伸缩性

表现出了巨大的优势。随着比例的增加，社区搜索的时间也会随之增加，增加了社区搜索中顶点与顶点之间属性关键字的匹配计算。算法 KIndexDec 和算法 KIndexInc 在 DBpeida 数据集结果很接近的原因在前面的一个实验中已经说明，对于 DBLP 算法 KIndexInc 曲线在 100% 比例的时候社区搜索时间有一个跳跃，导致了实验效果还没有 ATCIndex 和 ATCImprove 算法好。其中的原因可能是 DBLP 数据集中顶点的平均社区搜索的结果社区共享了几乎是查询顶点的所有属性关键字，其中算法 KIndexInc 采用的是自下而上的组合方式进行属性关键字匹配，而 ATCIndex、ATCImprove 和 KIndexDec 算法均采用自上而下的方式进行匹配，导致了出现了这种结果。这也说明了算法 KIndexInc 效果和 KIndexDec 算法很相当，但是不是很稳定。

6.3.2.5 顶点的伸缩性

最后测试的就是在不同大小的属性网络图中算法的运行效率，即对每个数据集按不同比例选取子图，然后在子图上进行社区搜索。对于每个数据集，随机选择原属性



(a) DBLP(vertex scalability) (b) YAGO(vertex scalability)

(c) DBpedia(vertex scalability) (d) DBpedia(vertex scalability)

图 6.7: 顶点的伸缩性

网络图中 20%、40%、60%、80% 和 100% 规格比例的顶点组成的新的子图进行实验对比，并且设置 k 值为 6，每个查询顶点的属性关键字数量为 5，其各数据集社区搜索的平均时间如图6.7所示。

在图6.7中，整体的趋势是随着比例的增加，社区搜索的平均时间也会随之增加。在所有算法中，算法 KIndexDec 表现出了最好的实验效果。ATCBasic 算法依然是最差的，它几乎没有经过方法去处理社区搜索的频繁计算。算法 KIndexDec 和 KIndexInc 虽然几乎是重叠的，但还是能看出算法 KIndexDec 稍逊一筹，并且在前一个实验中可以得知算法 KIndexDec 比算法 KIndexInc 更加稳定。

6.4 本章小结

本文设计的所有算法进行了大量的实验对比，从实验结果分析可以看出，KIndexDec 算法社区搜索效率更好，搜索的结果社区在结构上和属性语义上都是最密集的。传统的 ATCBasic 算法效率是最低效的，基于混合索引的 ATCIndex 和 ATCImprove 算

法相比 ATCBasic 算法有了明显的提高。但是在进一步的缩减候选关键字集合的计算，并且先从查询顶点的属性关键字集合出发的 KIndexInc 和 KIndexDec 算法表现出了更好的效果。由于 KIndexInc 算法存在不稳定的弊端，KIndexDec 算法稳定且高效。所以 KIndexDec 算法在属性图中密集社区搜索中表现出了其有效性和高效性。

7 总结与展望

7.1 本文总结

在本文中，主要介绍了在属性网络图中密集社区搜索的算法研究，基于属性网络图社区搜索问题，社区搜索的结果社区在结构上和属性语义上都是高度密集的子图社区。本文基于 k-truss 模型结构保证了社区在结构上的密集，并设计了 ATCBasic、ATCIndex、ATCImprove、KIndexInc 和 KIndexDec 等算法来解决属性网络图密集社区搜索问题。解决基于给定的查询顶点的密集社区搜索问题，将其主要分为两个子任务。一是找到包含查询顶点的最大 k-truss 社区，另一个是 k-truss 社区中的顶点与顶点之间共享最多查询顶点的属性关键字。最大 k-truss 结构满足了结构上密集的条件，共享最大属性关键字则是满足了属性语义上的密集条件。为了有效地处理大型属性网络图，本文首先设计了基于 truss 等价索引和节点索引的 ATCIndex 算法，使用关联算法 FP-Growth 发现属性关键字的频繁项集，减少了查询顶点要探索的关键字子集的数量，避免了一些不必要的计算，无需重复的访问原属性网络图。为了进一步提高搜索效率，并基于 k-truss 的反单调性，设计了一个改进的 ATCImprove 算法，进一步减少的候选属性关键字集合的匹配计算。接着想出了先从查询顶点的属性关键字集合计算出属性语义上相对密集的社区结构，并建立属性关键字索引，设计了基于关键字索引的 KIndexInc 和 KIndexDec 算法，两者先完成子任务的另一个子任务，并且过滤了一些不满足条件候选属性关键字子集，从而提高了密集社区搜索的效率。最后并对本文中的所有算法在 4 个真实世界的数据集中进行了广泛的实验，从实验结果分析中得出本文设计的 KIndexDec 算法在属性图中密集社区搜索问题上的有效性和高效性。

7.2 未来展望

本文的属性图中密集社区搜索算法研究，设计的算法仅仅是在静态的属性网络图中应用，建立的索引需要对静态的数据进行预处理，算法基于这些索引进行密集社区

搜索。属性图中密集社区搜索算法研究不仅仅局限在静态属性网络图，而且是可以在动态变化的属性网络图进行研究。在大数据爆发的时代，属性网络图数据资料常常是动态式进行变化的。动态属性图中进行密集社区搜索算法研究是本文研究的下一个计划，本文仅仅是为动态属性图中密集社区搜索算法研究提供了基础。

在动态属性图中进行密集社区搜索算法研究，比本文所要研究的问题更加具有挑战性，目前有一些研究者已经在初探动态网络图的研究，但在动态属性图上的研究微乎其微。其难点主要表现在以下几点。当一个顶点加入图中，其原先的图结构固然发生变化。在属性网络图，新加的顶点也有可能会带来新的属性关键字。这些都是不确定因素，对于同一个查询顶点和相同的其他查询参数，搜索到的社区结果也将有可能是不同的。另外一点就是社区搜索的社区如何去验证这就是最密集的，即社区搜索结果的正确性。

如果本文设计的索引也能随动态的属性图实时的变化，那么就可以应用本文设计算法进行密集社区搜索，应该也会有另一种更好的方法去解决这个问题。本文的研究也为了未来的研究奠定了一些基础，属性图中密集社区搜索算法研究还有很漫长的路要走。

参考文献

[1] Yu J X, Qin L, Chang L. Keyword search in relational databases: A survey[J]. IEEE Data Eng. Bull., 2010, 33(1): 67-78.

[2] Huang X, Lakshmanan L V S. Attribute-driven community search[J]. Proceedings of the VLDB Endowment, 2017, 10(9): 949-960.

[3] Fortunato S. Community detection in graphs[J]. Physics reports, 2010, 486(3-5): 75-174.

[4] Xie J, Kelley S, Szymanski B K. Overlapping community detection in networks: The state-of-the-art and comparative study[J]. Acm computing surveys (csur), 2013, 45(4): 43.

[5] Lee P, Lakshmanan L V S. Query-driven maximum quasi-clique search[C]//Proceedings of the 2016 SIAM International Conference on Data Mining. Society for Industrial and Applied Mathematics, 2016: 522-530.

[6] Barbieri N, Bonchi F, Galimberti E, et al. Efficient and effective community search[J]. Data mining and knowledge discovery, 2015, 29(5): 1406-1433.

[7] Huang X, Cheng H, Qin L, et al. Querying k-truss community in large and dynamic graphs[C]//Proceedings of the 2014 ACM SIGMOD international conference on Management of data. ACM, 2014: 1311-1322.

[8] Cui W, Xiao Y, Wang H, et al. Online search of overlapping communities[C]//Proceedings of the 2013 ACM SIGMOD international conference on Management of data. ACM, 2013: 277-288.

[9] Cui W, Xiao Y, Wang H, et al. Local search of communities in large graphs[C]//Proceedings of the 2014 ACM SIGMOD international conference on Management of data. ACM, 2014: 991-1002.

[10] Sozio M, Gionis A. The community-search problem and how to plan a successful cocktail party[C]//Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2010: 939-948.

[11] Huang X, Lakshmanan L V S, Yu J X, et al. Approximate closest community search in networks[J]. Proceedings of the VLDB Endowment, 2015, 9(4): 276-287.

[12] Akbas E, Zhao P. Truss-based community search: a truss-equivalence based indexing approach[J]. Proceedings of the VLDB Endowment, 2017, 10(11): 1298-1309.

[13] Akbas E, Zhao P. Truss-based community search: a truss-equivalence based indexing approach[J]. Proceedings of the VLDB Endowment, 2017, 10(11): 1298-1309.

[14] Koren Y, North S C, Volinsky C. Measuring and extracting proximity graphs in networks[J]. ACM Transactions on Knowledge Discovery from Data (TKDD), 2007, 1(3): 12.

[15] Wu Y, Jin R, Li J, et al. Robust local community detection: on free rider effect and its elimination[J]. Proceedings of the VLDB Endowment, 2015, 8(7): 798-809.

[16] Li R H, Qin L, Yu J X, et al. Influential community search in large networks[J]. Proceedings of the VLDB Endowment, 2015, 8(5): 509-520.

[17] Chang L, Lin X, Qin L, et al. Index-based optimal algorithms for computing Steiner components with maximum connectivity[C]//Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. ACM, 2015: 459-474.

[18] Hu J, Wu X, Cheng R, et al. Querying minimal steiner maximum-connected subgraphs in large graphs[C]//Proceedings of the 25th ACM International on Conference on Information and Knowledge Management. ACM, 2016: 1241-1250.

[19] Cheng J, Ke Y, Chu S, et al. Efficient core decomposition in massive networks[C]//2011 IEEE 27th International Conference on Data Engineering. IEEE, 2011: 51-62.

[20] Khaouid W, Barsky M, Srinivasan V, et al. K-core decomposition of large networks on a single PC[J]. Proceedings of the VLDB Endowment, 2015, 9(1): 13-23.

[21] Wen D, Qin L, Zhang Y, et al. I/O efficient core graph decomposition at web scale[C]//2016 IEEE 32nd International Conference on Data Engineering (ICDE). IEEE, 2016: 133-144.

[22] Cohen J. Trusses: Cohesive subgraphs for social network analysis[J]. National security agency technical report, 2008, 16: 3.1.

[23] Wang J, Cheng J. Truss decomposition in massive networks[J]. Proceedings of the VLDB Endowment, 2012, 5(9): 812-823.

[24] Zhang Y, Parthasarathy S. Extracting analyzing and visualizing triangle k-core motifs within networks[C]//2012 IEEE 28th International Conference on Data Engineering. IEEE, 2012: 1049-1060.

[25] Sariyüce A E, Pinar A. Fast hierarchy construction for dense subgraphs[J]. Proceedings of the VLDB Endowment, 2016, 10(3): 97-108.

[26] Chang L, Yu J X, Qin L, et al. Efficiently computing k-edge connected components via graph decomposition[C]//Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. ACM, 2013: 205-216.

[27] Yuan L, Qin L, Lin X, et al. I/O efficient ECC graph decomposition via graph reduction[J].

Proceedings of the VLDB Endowment, 2016, 9(7): 516-527.

[28] Zhou Y, Cheng H, Yu J X. Graph clustering based on structural/attribute similarities[J]. Proceedings of the VLDB Endowment, 2009, 2(1): 718-729.

[29] Ruan Y, Fuhry D, Parthasarathy S. Efficient community detection in large networks using content and links[C]//Proceedings of the 22nd international conference on World Wide Web. ACM, 2013: 1089-1098.

[30] Xu Z, Ke Y, Wang Y, et al. A model-based approach to attributed graph clustering[C]//Proceedings of the 2012 ACM SIGMOD international conference on management of data. ACM, 2012: 505-516.

[31] Bothorel C, Cruz J D, Magnani M, et al. Clustering attributed graphs: models, measures and methods[J]. Network Science, 2015, 3(3): 408-444.

[32] Yu J X, Qin L, Chang L. Keyword search in relational databases: A survey[J]. IEEE Data Eng. Bull., 2010, 33(1): 67-78.

[33] Bhalotia G, Hulgeri A, Nakhe C, et al. Keyword searching and browsing in databases using BANKS[C]//Proceedings 18th International Conference on Data Engineering. IEEE, 2002: 431-440.

[34] Kacholia V, Pandit S, Chakrabarti S, et al. Bidirectional expansion for keyword search on graph databases[C]//Proceedings of the 31st international conference on Very large data bases. VLDB Endowment, 2005: 505-516.

[35] He H, Wang H, Yang J, et al. BLINKS: ranked keyword searches on graphs[C]//Proceedings of the 2007 ACM SIGMOD international conference on Management of data. ACM, 2007: 305-316.

[36] Li G, Ooi B C, Feng J, et al. EASE: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data[C]//Proceedings of the 2008 ACM SIGMOD international conference on Management of data. ACM, 2008: 903-914.

[37] Qin L, Yu J X, Chang L, et al. Querying communities in relational databases[C]//2009 IEEE 25th International Conference on Data Engineering. IEEE, 2009: 724-735.

[38] Keyword search in graphs: Finding r-clique

[39] Zhao F, Zhang X, Tung A K H, et al. Broad: Diversified keyword search in databases[R]. 2011.

[40] Kargar M, An A, Yu X. Efficient duplication free and minimal keyword search in graphs[J]. IEEE transactions on knowledge and data engineering, 2014, 26(7): 1657-1669.

[41] Wu Y, Yang S, Srivatsa M, et al. Summarizing answer graphs induced by keyword queries[J].

Proceedings of the VLDB Endowment, 2013, 6(14): 1774-1785.

[42] Tong H , Faloutsos C . Center-piece subgraphs: problem definition and fast solutions[C]// Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Philadelphia, PA, USA, August 20-23, 2006. ACM, 2006.

[43] Xu Z, Ke Y, Wang Y, et al. A model-based approach to attributed graph clustering[C]//Proceedings of the 2012 ACM SIGMOD international conference on management of data. ACM, 2012: 505-516.

[44] Yang J, McAuley J, Leskovec J. Community detection in networks with node attributes[C]//2013 IEEE 13th International Conference on Data Mining. IEEE, 2013: 1151-1156.

[45] Zhang Y, Yu J X, Zhang Y, et al. A fast order-based approach for core maintenance[C]//2017 IEEE 33rd International Conference on Data Engineering (ICDE). IEEE, 2017: 337-348.

致　　谢

两年前，我通过自己的努力考上了心中的武汉大学。怀着对美好未来的憧憬，载着家人的期望走进了武大。两年时光如白驹过隙，一晃而过。那一幕幕仿佛发生在昨天，清晨坐在校园的大循环班车去上课学习，课后便坐在了实验室的座位上，让人充满了美好而难忘的回忆。经历了找工作的喧嚣与坎坷，我深深体会到了写作论文时的那份宁静与思考。有人说，最浪漫的告白，就是把你写进我的致谢里，可惜你却没有出现。回首两年的求学历程，值此论文完成之际，谨向所有曾给予我帮助和指导的老师、同学、朋友们和父母致以衷心的感谢！

首先要衷心的感谢我的研究生导师 X 老师，也许我不是老师心中最出色的学生，但 X 老师是我所最尊敬的老师。感谢 X 老师两年来的敦敦教导，在导师的教导下，我受益良多，唯一的遗憾是我不够主动，错失了许多交流的机会。

其次感谢学校对我的栽培，两年来为我提供了良好的学习和生活环境，使我能在这里快乐无忧的学习和生活。感谢所有在大学期间传授我知识的老师，您们的悉心教导让使我掌握了良好的专业课知识。

还要感谢两年来和我一起在大学校园里学习交流的同学，感谢我的室友和小伙伴们，谢谢你们为我提供的帮助和宝贵的意见。

最后要深深地感谢我的父母和家人，感谢您们两年来默默的关心和支持。

感谢这几年坚强的自己。