Hub-Accelerator: 快速和精确的最短路径大型社交网络中的计算

Ruoming Jin† Ning Ruan⎚ Bo You† Haixun Wang‡

†Kent State University ⎚ Google Inc ‡ Microsoft Research Asia

{jin,byou}@cs.kent.edu ningruan@google.com haixunw@microsoft.com

摘要

最短路径计算是管理和分析大型社交网络的最基本操作之一。虽然现有的技术对于在大型但稀疏的道路网络上找到最短路径非常有效，但社交图具有完全不同的特征：它们通常是非空间的，非加权的，无尺度的，并且它们除了巨大的尺寸之外还表现出小世界的属性。特别是，Hubs（即具有大量连接的顶点）的存在会破坏搜索空间，从而使最短路径计算变得异常具有挑战性。在本文中，我们介绍了一组以集线器为中心的新技术，统称为Hub-Accelerator框架，以计算*k*度最短路径（如果两个顶点的距离在k范围内，则在两个顶点之间找到最短路径）。这些技术使我们能够通过大大限制Hub的扩展范围（使用新的*省距集线器网络*概念）或完全修剪在线搜索中的集线器（使用Hub²*标记方法*）来显著减少搜索空间。HubAccelerator方法用于最短路径计算比BFS和最先进的近似最短路径方法Sketch快两个数量级以上。HubNetwork方法不会引入额外的索引成本和轻度的预计算成本;Hub2-Labeling的索引大小和索引构建成本也适中，优于或可与近似索引 Sketch 方法相媲美。

1. 简介

社交网络正在变得无处不在，其数据量正在急剧增加。流行的在线社交网络网站，如Facebook，Twitter和LinkedIn，如今都拥有数亿活跃用户。谷歌的新社交网络Google+吸引了2500万独立用户，并在推出后的第一个月以每天约100万访问者的速度增长。实现这些海量图形的在线和交互式查询处理，特别是快速捕获和发现实体之间的关系，正在成为从社会科学到广告和营销研究再到国土安全的新兴应用不可或缺的组成部分。

最短路径计算是管理和查询社交网络最基本但最关键的问题之一。社交网络网站LinkedIn开创了著名的最短路径服务"你如何连到A"，它在3个步骤内表示出了你和用户A之间友谊链的精确描述。微软的Renlifang（EntityCube）[37]记录了超过1000万个实体（人员，位置，组织）的十多亿个关系，允许用户在距离小于或等于6时检索两个实体之间的最短路径。新出现的在线应用程序"Six Degrees"[38]提供了一种互动方式来展示您如何与Facebook网络中的其他人建立联系。此外，最短路径计算在确定信任和发现网络游戏中的朋友方面也很有用[41，42]。

在本文中，我们研究了***k*度最短路径**查询（$k \le 6$），其正式描述为：给定大型（社交）网络中的*两个顶点（用户）s和t，如果它们的距离小于或等于k，则从s到t的最短路径是什么？*在所有这些新兴的社交网络应用程序中，（一）两个用户之间的最短路径通常需要计算，只有当他们的距离小于某个阈值（例如6）。这种关注直接与 在这些庞大的社交网络中观察到的小世界现象产生共鸣。例如，大量Facebook用户[38]的平均成对距离已被证明仅为5。73.此外，Twitter上大约一半的用户平均距离另一个用户4步，而几乎每个人都相距5步[39]。不仅大型社交网络中的大多数用户相隔不到6个步骤，社交网络中较长的连接或路径也不太有意义和/或有用。

在大型社交网络中计算k度最短路径是非常具有挑战性的，特别是当*k*相对较大时，例如*k*=6。单个BFS（广度优先搜索）可以在具有数百万个顶点的大型网络中以6个步骤轻松访问超过100万个顶点。虽然现有的技术[20，21，34，31，3，17，15，30，32，23，14，35，4]对于在大型但稀疏的道路网络上找到最短路径非常有效，但社交图具有完全不同的特征。社交网络不是空间的，边缘权重和低顶点度，通常是*非空间*的，*非加权的*，*无尺度*的（因此包含高度的中心节点），并且它们除了巨大的尺寸之外还表现出小世界的属性。事实上，由于在社交网络中很难找到最短路径，最近的研究[16，41，42]都集中在只发现近似路径（比真正的最短路径长）。此外，即使使用近似值，最快的方法，如 *Sketch* [16]、*TreeSketch* [16] 和 *RigelPaths* [42]，仍然需要数十或数百毫秒（10^{-3} 秒）来计算具有几百万个顶点的社交网络中的近似最短路径。

大规模社交网络中最短路径计算的核心问题来自Hub：那些具有大量连接的顶点。与总网络规模相比，Hub 点的数量可能很少，但是，它们出现在几乎任何顶点的近邻中。事实上，中心在小世界（社会）网络中发挥着关键作用。它们充当连接顶点之间最短路径的常见中介，就像航空公司航班小世界中的枢纽城市一样。事实上，理论分析表明，少量的Hub（由于幂律度分布）显着缩短了顶点之间的距离，并使网络"超小"[8]。然而，枢纽是导致搜索空间爆炸的关键因素。假设一个中心有5000个朋友，普通人有大约100个朋友，那么来自中心的两步BFS将访问≈50万个顶点；在Twitter网络中，一些顶点（名人）包含超过1000万粉丝，所以一个反向的一步BFS（从那个顶点到它的追随者）已经太昂贵了。因此，Hub 点是问题的核心：没有它们，最短的路径就较难计算，但

它们的发现比较困难。我们能解开最短路径和枢纽之间的相对关系吗？我们能否利用 Hub 点更友好地进行最短路径计算？

在本文中，我们针对海量社交图谱中最短路径计算的这些挑战性问题给出了肯定的答案。我们介绍了一系列以Hub 点为中心的新技术，统称为Hub-Accelerator框架。这些技术使我们能够通过大大限制Hub 点的扩展范围（使用新颖的*保持距离的集线器网络*概念）或完全修剪在线搜索中的集线器（使用*Hub2标记*方法）来显着减少搜索空间。Hub-Accelerator方法平均比BFS和最先进的近似最短路径方法快两个数量级以上，包括*Sketch* [16]，*TreeSketch* [16]和*RigelPaths* [42]。集线器网络方法不会引入额外的索引成本和轻度的预计算成本；Hub2-Labeling的索引大小和索引构建成本也适中，优于或可与近似索引 Sketch 方法相媲美。我们注意到，尽管最短路径计算已经过广泛研究，但大多数研究仅关注道路网络[20，21，34，31，3，17，15，30，32，23，14，35，4，2，1]或大规模社交网络上的近似最短路径（距离）计算[16，42]。据我们所知，这是第一部明确解决这些网络中确切最短路径计算的工作。Hub-Accelerator技术也很新颖，距离保持子图（Hub-network）发现问题本身对于图的挖掘和管理具有理论和实践的重要性。

2. 相关工作

在下文中，我们将回顾最短路径计算的现有方法，特别是与社交网络相关的方法。在整个讨论中，我们使用n和m分别表示图G中的节点数和边数。

线上网络最短路径计算：最短路径计算最着名的方法之一是Dijkstra算法[12]。它计算加权图中的单源最短路径，可以使用$O(m + \text{nlogn})$时间实现。如果图形未加权（与许多社交网络一样），则广度优先搜索（BFS）过程可以计算$O(m + n)$中的最短路径。但是，将这些方法应用于具有数百万个顶点的社交网络的成本高得令人望而却步，即使将搜索深度限制为6步也是如此。首先，社交网络的平均程度相对较高。例如，Facebook中的每个用户平均有大约130个朋友。一个简单的BFS可以在6个步骤内轻松扫描100万个顶点。一个简单的策略是采用双向搜索来减少搜索空间。其次，由于集线器的存在和小世界属性，可以在双向BFS中遍历大量集线器（即使在最短路径查询的开始 s 或结束 t 的三个步骤内）。例如，在由超过300万个顶点和2.2亿条边组成的Orkut图（一个常用的基准测试社交网络）中，双向BFS仍然需要访问每个查询近20万个顶点，而传统的BFS需要访问近600万个顶点。

道路网络上的最短路径计算：计算道路网络上的最短路径已被广泛研究[20，21，34，31，3，17，15，30，32，23，14，35，4，2，1]。在这里，我们只提供简短的评论。有关此主题的更详细评论可以在[11]中找到。一些早期的研究[20，21，34]，如 *HEPV* [20]和 *HiTi* [21]，利用拓扑图的分解来加速最短路径搜索。最近，各种技术[11]，如 A^* [15]，Arc-flag（将搜索指向目标）[4]，高速公路层次结构（构建快捷方式以减少搜索空间）[17，31]，公交节点路由（使用一小组顶点中继最短路径计算）[3]，以及利用空间数据结构积极压缩距离矩阵[30， [32]，已开发。然而，这些方法的有效性取决于道路网络的基本属性，例如几乎平面，低顶点度，加权，空间和分层结构的存在[16]。正如我们之前提到的，社交网络具有不同的属性，例如非空间，未加权，无缩放（存在中心）以及展示小世界属性。例如，那些利用空间属性（三角形不等式）来修剪搜索空间的技术在社交网络中立即变得不可行。此外，高顶点度（集线器）很容易导致搜索空间的爆炸式增长。

理论距离标记和地标：有几项关于估计大型（社会）网络中任何顶点之间距离的研究[26，9，16，41，42，29]。这些方法通常属于距离标记 [13]，它为每个顶点 u 分配 一个标签（例如，一组顶点以及从 u 到每个顶点的距离），然后使用分配的标签估计两个顶点之间的最短路径距离。由Thorup和Zwick撰写的开创性工作，称为距离预言机[36]，显示了一个（$2k - 1$）乘法距离标记方案（近似距离不超过精确距离的2k−1倍），对于每个整数$k \geq 1$，标签为$O(n^{1/k} \log 2\, n)$ 位。然而，正如Potamias *等人*。 [26]认为，出于实际目的，即使$k = 2$也是不可接受的（由于小世界现象）。最近，萨尔玛 *等人*。 [9]在真实的Web图上研究了Thorup和Zwick的距离预言机方法，他们发现这种方法可以提供相当准确的估计。

Cohen等人开创性的2跳距离方法。[7]在有向图上提供精确的距离标记（与2跳可达性索引非常相似）。具体来说，每个顶点 u 都记录了它可以到达的中间顶点 $Lout(u)$ 的列表以及它们的（最短）距离，以及一个可以到达它的中间顶点 $Lin(u)$ 的列表，这些顶点可以与其距离一起到达它。为了找到从 u 到 v 的距离，2-hop 方法只需检查 $Lout(u)$ 和 $Lin(v)$ 之间的所有常见中间顶点，并选择顶点 p，使得 $dist(u, p) + dist(p,$ 对于所有 $p \in Lout(u) \cap Lin(v)$ 最小化。然而，构建最佳2-hop标记的计算成本高得令人望而却步[33，18]。

一些作品使用 *地标* 来近似最短路径距离[28，22，26，41，42，29]。在这里，每个顶点预先计算到一组地标的最短距离，因此地标方法可以被视为 2跳和距离标记的特殊情况，其中每个顶点可以记录到不同顶点的距离。Potamias *等人*。 [26]研究最佳地标集的选择，以估计最短路径距离。乔 *等.* [29]观察到全局选择的地标集引入了太多的误差，特别是对于一些距离较小的顶点对，因此提出了一种查询加载感知地标选择方法。赵 *等.* [42]引入Rigel，它利用双曲空间嵌入在地标顶部以提高估计精度。

社交网络中的近似最短路径计算： 最近的一些研究旨在计算大型社交网络中最短路径。它们扩展了距离标记或地标方法，以近似最短路径。古比切夫 *等人* 提出 *Sketch*，它推广了距离预言机方法[36，9]，以发现大图[16]中最短路径（不仅是距离）。他们观察到路径长度足够小，可以被认为是几乎恒定的，因此除了距离标记之外，还存储了一组预先计算的最短路径。他们还提出了一些改进，例如 *循环消除* （SketchCE）和 *基于树的搜索* （TreeSketch），以提高最短路径估计精度。赵 *等.* [42]开发 *RigelPath* ，以在其距离估计方法Rigel之上近似社交网络中最短路径。他们的基本思想是使用距离估计来

帮助确定搜索方向并修剪搜索空间。Sketch 是最快的近似最短路径方法，尽管 RigelPath 和 TreeSketch 可以更准确。此外，RigelPath主要关注无向图，而Sketch可以同时处理有向图和无向图。

最短路径计算的其他最新进展：最近，数据库研究界对最短路径和距离计算进行了一些研究。在[40]中，Wei开发了一个树分解索引结构，以找到未加权无向图中最短的路径;在[5]中，开发了一种基于分层顶点覆盖的方法，用于单源磁盘上最短路径（距离）计算。在[6]中，Cheng *等人*。 引入 k-reach 问题，它为两个顶点是否通过 k 步连接提供了二进制答案 。此外，在 [6] 中开发的 k-reach 索引方法不可扩展，只能处理小图形（因为它试图在一定距离阈值内实现顶点对）。最后，金 *等人*。[19]提出了一种以高速公路为中心的标记（HCL）方案，以有效地计算稀疏图中的距离。利用高速公路结构，与最先进的 2-hop 标签相比，此距离标注提供了更紧凑的索引尺寸，并且还能够以有界精度提供精确和近似距离。然而，很难扩展到大型社交网络，因为真正的社交网络通常不是稀疏的，并可能导致昂贵的指数建设成本和大的指数规模。

3. HUB-ACCELERATOR 框架

在本节中，我们将概述用于最短路径计算的中心加速器（HA） 框架。在前面的讨论中，我们观察到最短路径和中心之间存在爱恨交织的关系：一方面，任何最短路径都可能包含一些中心，因此需要在最短路径搜索过程中访问;另一方面，为了提供最快的最短路径搜索，我们需要尽量避免集线器节点的完全扩展。我们注意到，一般来说，中心的符号是相当非正式的，尽管通常基于程度;在本文中，我们仅引用其度数最高（顶 *点*β 顶点数的顶点集;β 是一个常量，可以指定）。

Hub-Accelerator的设计旨在利用这些枢纽进行最短路径计算，而无需完全扩展其邻域。为此，需要回答以下研究问题：

1. 我们如何在最短路径搜索期间限制集线器的扩展？一个集线器可能有数千甚至数百万个连接（邻居）;在最短路径搜索中，哪些邻居应被视为必需的并给予高度优先权？为了解决这个问题，我们制定了 *集线器网络* 表示法，它捕获了这些集线器之间最短路径和拓扑的高级视图。集线器网络可以被认为是一种高速公路结构，由集线器锚定，用于在大型社交网络中路由最短路径。由于枢纽的重要性，非枢纽顶点对之间的最短路径可能需要通过这样的网络，即起始顶点到达一个枢纽（作为高速公路入口），然后行进到另一个枢纽（作为高速公路出口），最后离开高速公路到达目的地。换句话说，集线器网络可用于限制（或优先处理）集线器的邻居。集线器应仅在集线器网络内扩展。

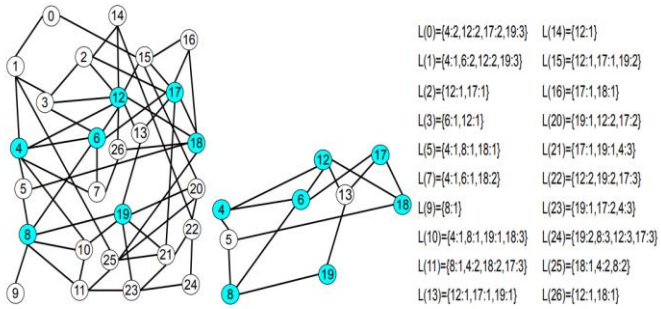
2. 我们如何有效和高效地利用集线器网络进行最短路径搜索？请注意，集线器网络捕获集线器之间的最短路径。但是，并非所有顶点之间的最短路径都需要通过集线器网络：它们可能不包含任何集线器，或者它们可能仅由一个集线器组成（在后一种情况下，集线器网络中可能不需要遍历）。因此，问题在于我们如何扩展典型的双向BFS以采用集线器网络来加速最短路径计算？

3. 我们能完全避免枢纽的扩张吗？这样，即使是集线器网络也变得不必要。但是，应该预先计算哪些基本信息呢？当集线器的数量不大时，比如10K，那么集线器之间的成对距离矩阵可能会被具体化。对于 10K 集线器，这仅花费大约 100MB = 10K × 10Kb（假设距离可以保持在 8 位），但可能需要额外的内存来恢复最短路径。鉴于此，双向搜索如何利用这样的矩阵，以及还需要预先计算哪些其他信息？

在这项工作中，通过研究和解决这些问题，我们能够有效地利用集线器来加速最短路径搜索，同时显着降低或避免扩展它们的成本。具体来说，我们做出了以下贡献： **Hub-Network Discovery（第4节）：** Hub-Network的概念是Hub-Accelerator框架的核心：给定集线器的集合， *距离保持子图* 试图从原始图形中提取最少数量的附加顶点和边缘，以便可以恢复集线器之间的距离（和最短路径）， 即，它们在集线器网络中的距离等效于它们在原始图形中的距离。如前所述，枢纽网络充当运输系统中的高速公路，以实现最短路径搜索的加速：任何枢纽都不会完全扩展（在原始图表中）;相反，只有它们在集线器网络中的邻居将被扩展。有趣的是，尽管距离保持子图（和集线器网络）的发现似乎相当直观，但这个问题的计算方面以前没有被研究过（尽管在理论图论中定义了类似的概念[10]）。在第4节中，我们展示了发现最小距离保持子图的NP硬度，并开发了一种快速贪婪的方法来提取集线器网络（和保持距离的子图）。我们的实验研究表明，集线器网络中的集线器程度明显低于原始图中的集线器程度;因此，集线器网络可以限制集线器的扩展，并实现更快的最短路径计算。

基于集线器网络的双向 BFS（第 5 节） 如上所述，将集线器网络合并到双向 BFS 中并非易事。通常，如果我们使用集线器网络并扩展网络内的集线器，那么当它们在公共顶点相遇时，不能简单地停止两个方向的搜索。这是因为集线器网络不会捕获仅由一个集线器组成的最短路径。

Hub2-Labeling（第 6 节）：在此技术中，我们通过完全避免扩展任何集线器来进一步推动最短路径计算的速度边界。为了实现这一目标，使用更昂贵但通常负担得起的预计算和内存成本来加快在线搜索速度。它由三个基本元素组成：1）首先，这种技术不是提取和搜索集线器网络，而是将这些集线器的距离矩阵（称为*Hub2*矩阵）进行匹配。如前所述，即使对于 10K 集线器，矩阵也可以很容易地实现。2） 引入了*集线器标记*，以便每个顶点将预先计算和物化少量集线器（称为核心集线器），这对于使用集线器和集线器矩阵恢复最短路径至关重要。 3） 给定 Hub2 距离矩阵和集线器标记， 可以执行更快的双向BFS以发现确切的*k*度最短路径。它首先使用距离矩阵和集线器标记来估计距离上限。在双向搜索期间，不需要扩展任何中心，即中心修剪双向 BFS。



(a) Original Graph (b) Hub-Network (c) Hub-Labeling

Figure 1: Running Example of Hub-Accelerator Framework

4. HUB-NETWORK 研究成果

在本节中，我们正式定义了 *集线器网络*（第 4.1 小节），并提出了一种发现它的有效方法（第 4.2 小节）。

为了便于我们的讨论，我们首先介绍以下表示法。设 $G =（V，E）$ 为图，其中 $V = \{1，2,...n\}$ 是顶点集，$E \subseteq V \times V$ 是边集。顶点 u 和 v 的边缘由 $（u，v）$ 表示，我们使用 $P（v0，vp）=（v0，_{v1},...,vp）$ 来表示 $v0$ 和 vp 之间的简单路径. 简单路径的长度是路径中的边数，用 $|P（v0，vp）|$.给定两个顶点 u 和 v，它们的最短路径 $SP（u，v）$ 是它们之间具有最小长度的路径。从顶点 u 到 v 的距离是 u 和 v 之间最短路径 $SP（u，v）$ 的长度，用 $d（u，v）$ 表示。请注意，对于有向图，边集可能包含 $（u，v）$、$（v，u）$ 或两者。对于无向图，边没有方向;换句话说，它可以被认为是双向的，因此边集包含两个边 $（u，v）$ 和 $（v，u）$ 或者两个边都不包含。在无向图中，从 u 到 v 的最短路径距离等效于从 v 到 u 的最短路径距离，即 $d（u，v）= d（v，u）$。本文讨论的技术可以应用于无向图和有向图;为简单起见，我们将重点介绍无向图，我们将简要提及每种技术如何自然地扩展以处理有向图。

1. 保持距离的子图和中心网络

直观地说，集线器网络是原始G的最小子图，因此可以在子图中恢复两个集线器之间的至少一条最短路径（保留距离）。为了正式定义集线器网络，我们首先介绍 *保持距离子图*的概念及其发现。

定义 1. **保持距离的子图** *给定图 $G =（V，E）$ 和一组顶点对 $D =（u，v）\subseteq V \times V$，保持距离的子图 $Gs =（_{Vs}，Es）$ 的 G（$Vs \subseteq V$ 和 $Es \subseteq E$）具有以下属性：对于任何 $（u，v）\in D$，$d（u，v|Gs）= d（u，v|G）$，其中 $d（u，v|Gs）$ 和 $d（u，v|G）$ 分别是子图 $G_{和}$原始图 G 中的距离。*

给定一组顶点对，其距离需要在子图中保留，子图发现问题旨在根据顶点（或边）的数量来识别*最小*子图。

定义 2. **最小距离保持子图（MDPS）问题** *给定图 $G =（V，E）$ 和一组顶点对 $D =（u，v）\subseteq V \times V$，最小距离保持子图（MDPS）问题旨在发现最小子图 $G^*_s =（_{Vs}*，_{Es}*）$ 具有最小数量的顶点，即 $G^*_s = \text{argmin}_{|Vs|} Gs$，其中 $Gs =（_{Vs}，Es）$ 是相对于 D 的保持距离的子图。*

一旦发现所有顶点 $Vs*$，G 的诱导子图 $G[Vs*]$ 就是候选的最小子图。请注意，其边缘集可能会进一步稀疏化。然而，关于顶点对集合的边缘稀疏化问题（相当于边数方面的最小距离保持子图问题）与MDPS问题一样困难（见下面的讨论）;在未加权的图形中，可以移除的边数通常很小。因此，我们不会在这项工作中探索进一步的边缘减少。

给定图 $G =（V，E）$ 和一组集集线器 $H \subseteq V$，让 Dk 包含距离不大于 k 的所有集线器对，则*集线器网络被定义为 G 中 Dk 的最小保持距离子图。*

EXAMPLE 4.1. *图 1（a） 显示了我们将用作运行示例的网络。图1（b）是当 $k = 4$ 时 $H = \{4，6，8，12，17，18，19\}$（度数≥ 5） 的相应集线器网络。由于这些集线器之间的成对距离都小于 4，因此 D4 包含所有集线器对，总共有 15 个顶点对。*

请注意，另一种方法是构建显式连接中心对的加权集线器网络：例如，如果任何其他集线器位于两个集线器之间的最短路径中，则可以添加一条边来直接链接它们。事实上，大多数现有研究都采用了类似的方法来建造和利用一些 *高速公路结构*（但它们主要针对道路网络，这些网络相当稀疏）。然而，这种方法在搜索大规模社交网络时可能会导致许多问题：1）这样的集线器网络将被加权并且可能是密集的（可能需要在集线器之间添加许多新的边缘），并且为了搜索它，必须使用Dijkstra的算法（或其变体）并且比BFS慢（因为使用优先级队列）。较高的边缘密度加剧了这种减速。2） 双向 BFS 通常用于搜索未加权的网络，并可用于搜索剩余的网络（不包括集线器网络）。但是，将双向BFS与Dijkstra的组合可能相当困难。3）可能需要大量内存来记录这样的集线器网络，因为它相当密集。此外，要恢复最短路径，必须为每个添加的新边记录其他信息。考虑到这些问题，我们利用保持距离的子图作为集线器网络，它不会产生额外的内存成本，并且可以自然地支持（双向）BFS。请注意，在第 5 节和第 6 节中，我们将研究如何使用更多内存来获得更高的查询性能（不涉及加权集线器网络的难度）。

为了在庞大的社交网络中发现集线器网络，我们需要一个快速的解决方案来解决最小距离保持子图（MDPS）问题。然而，找到确切的最佳解决方案是很困难的。

定理 1. *在图中查找顶点对集合 D 的最小保持距离子图是一个 NP 难题。*

Proof Sketch：我们将集合覆盖决策问题r简化为最小距离保持子图问题的决策版本。在集合覆盖决策问题中，设 U 为基集，C 记录所有候选集，其中对于任何候选集 $C \in$ C 和 $C \subseteq$ U。集合覆盖决策问题询问 C 中是否存在 K 个或更少的候选集合，使得 $\cup iCi =$ U。

现在，我们基于集合覆盖实例构造以下 MDPS 实例：考虑一个三方图 $G =（X \cup Y \cup Z，EXY \cup EYZ）$，其中 X 和 Z 中的

顶点与地面集 U 中的元素具有一对一的对应关系，而 Y 中的顶点具有一对应关系。

一对一对应于 C 中的候选集。为简单起见，设 $u \in U \leftrightarrow xu \in X$（$_{zu} \in Z$）（顶点 xu（zu） 对应于元素 u）;并让 $C \in C \leftrightarrow yC \in Y$（顶点 yC 对应于候选集） C）。然后，如果元素 u 属于候选 集 C，则边集 $_{EXY}$（$_{EYZ}$） 包含所有边 （xu，$_{yC}$） （(yC，zu)）。请注意，三方图可以被认为是对称的

($X \equiv Z$ and $E_{XY} \equiv E_{YZ}$).

我们声称集合覆盖决策问题是可满足的，当且仅当以下 MDPS 问题为真时：有一个子图 G 与 $2|U|+K$个顶点覆盖|的最短路径距离 U|顶点对 （$_{xu}$，zu）， $u \in U$.

这种说法的证据如下。假设集合覆盖问题是可以满足的，让$C1,\cdot Ck$（$k \leq K$） 是 覆盖基集的 k 个候选集，即 $\cup Ci = U$。让 YC 包含 Y 中对应于 $C1$ *的所有*顶点，$\cdot$，Ck.很容易观察到 $G[X \cup YC \cup Z]$的诱导子图可以恢复所有|U|pairs （$_{xu}$，zu）， $u \in U$.请注意，它们在原始图G和诱导子图$G[X \cup YC \cup Z]$中的距离都等于2。

从另一个方向，让 $_{Gs}$成为具有 2 的子图|U|+ K个顶点，用于恢复这些|的距离 U|对。由于对中的顶点必须包含在子图中（否则，无法显式恢复距离），因此附加*的* K 个顶点只能来自顶点集 Y（有 2 个|U|在 X 和 Z 的顶点对中）。请注意，原始图中 （xu，$_{zu}$） 的距离为 2，为了恢复该距离， Y 中的顶点 $_{yC}$ 必须出现在子图中，以便 （xu，$_{yC}$） 和 （yC，$_{zu}$） 在子图中（和原始图中）。这表示相应的候选集 C 覆盖元素 u。因为Y中最多有K个顶点 ，所以最多有K个候选者需要覆盖所有的地面集U。 ✷

基于类似的简化，我们还可以证明，根据边数找到最小保持距离的子图也是一个NP难题。由于简单起见，我们不会在本文中进一步探讨这种替代方案。

1. Hub-NETWORK DISCOVERY算法

在本节中，我们将讨论一种发现保持距离的子图和集线器网络的有效方法。为了简化我们的讨论，我们专注于提取集线器网络，尽管该方法直接适用于任何顶点对的集合（以及一般的保持距离的子图）。回想一下，在集线器网络发现问题中，给定一组集线器 H 和一组距离不超过 k 的中心对 的集合 D（对于 k 度最短路径搜索），则目标是使用最小（距离保存）子图恢复 D 中对的距离。

为了有效地处理集线器网络（和保持距离的子图），我们做了以下简单的观察。对于 D 中的任何顶点对 （x，y），如果有另一个中心 z，使得 d（x，y）= d（x，z）+ d（z，y），那么我们将顶点对 （x，y） 称为*复合对*;否则，它是一个*基本对*，即连接 x 和 y 的任何最短路径都不包含 H 中的集线器。设 $D_b \subseteq D$ 是基本对的集合。鉴于此，很容易看出，*如果子图可以恢复*Db*中的所有顶点对*，那么它就是D*（以及集线器网络）的距离保持子图*。这表明我们只需要关注基本对（Db），因为复合对的距离可以使用基本对之间的路径直接恢复。

考虑到这一点，在高层次上，中心网络发现的算法从每个中心h执行BFS类型的遍历，并完成两个任务：1）在BFS期间，所有基本对，包括h，即（h，v），$v \in H$，都应该被识别和收集;2）一旦一个基本对（h，v） 被识别后，算法将选择一个"好"的最短路径，该路径由最少数量的"新"顶点（尚未包含在集线器网络中）组成。换句话说，当我们从每个中心遍历图形时，我们逐渐用新的顶点来增强中心网络，以恢复新发现的基本对的距离（最短路径）。

识别基本对：为了在从集线器 h 开始的 BFS 过程中快速确定（h，v） 是否是一个基本对，我们利用以下观察结果：*让顶点 y 位于从集线器 h 到非中心顶点 v* 的最短路径上，*距离为 d*（h，v）− 1*（即，y 比中心 h，v）− 1 更近一跳 v 关于 h）。如果有一个集线器 h'* 出现在从 *h 到 y 的最短路径中（h' 和 y 可能不是不同的），则 h'* 肯定位于从 *h 到 v 的最短路径上，并且* （h，v） *是一个复合对（不是基本对）。* 基于这一观察结果，我们简单地维护一个二进制标志b（v）来表示在h和v之间的最短路径中是否存在另一个中心。具体来说，它的更新规则如下：*如果 v* 本身是集线器或 b（y） = 0（ y 是 *BFS* 中的父级，即 d（h，y）= d（h ，v）− 1 和 d（y，v））= 1）。因此，在BFS遍历期间，当我们访问顶点v时，如果它的标志 b（v）= 1（true）意味着没有其他集线器位于h和v之间的最短路径上，并且我们能够识别它是一个基本对。

在基本对之间选择"良好"的最短路径：要在基本对 h 和 v 之间选择一条最短路径 ，基本度量是需要添加到集线器网络的"新顶点"的数量。作为一个贪婪的标准，需要添加的越少，路径就越好。为了计算这一点，对于从起点 h 到 v 的任何最短路径，分数 f 记录集线器网络中已存在的顶点的最大数量。可以很容易地逐步维持这一措施。简单地说，它的更新规则如下：如果 v 本身在中心网络中，则 f（v）= $maxf$（u）+1，或者 f（v）= $maxf$（u），其中 u 是 v 在 BFS 中的父级（从 h 到 v 的最短路径通过 u 和 u 直接链接到 v）。顶点 v 记录 u，它具有 用于跟踪如此短路径的最大 f（在集线器网络中具有最大数量的顶点）。最后，我们注意到，只有对于 b（v）= 1 的顶点 v，即当 h 和 v 之间的最短路径不经过任何其他中心时，才需要保持分数 f。否则，v 及其后代将无法生成任何基本对。

总体算法：算法 1 中描述了此基于 BFS 的发现集线器网络过程的概述。此处 $H^\star$ 是记录集线器网络中顶点的集合。最初，$H^\star = H$，然后在处理过程中添加新的顶点。请注意，在BFS遍历的队列（第3行）中，我们总是访问那些b（u） = 0的顶点，即它们和它们的任何后代（在BFS遍历中）不会形成一个基本对，因此 不需要为它们维护分数f。一旦访问了一个集线器，并且它最初具有b（u）= 1，那么（h，u）是一个基本对（第5行）;我们将提取在集线器网络中具有最大顶点数的最短路径，并将新顶点添加到$H^\star$（第6行）.现在，由于这个中心的后代（在BFS遍历中）不会形成一个基本对，我们只需将其标志更改为false，即b（u） = 0（第7行）。此外，由于我们只对 k-hop 中最短路径感兴趣，因此我们不会将任何距离为 h 的顶点扩展为 k。（第9-11行）。在扩展 u 的邻居之前，我

们还需要根据 u 本身是否在集线器网络中来更新其 f 分数（第 12 行 − 14 行）。

Algorithm 1 BFSExtraction($G = (V, E), h, H, H^\star$)

```
1: Initialize b(u) ← 1; f(u) ← 0 for each vertex u;
2: level(h) ← 0; Q ← {h} {queue for BFS};
3: while Q ≠ ∅ {vertices with b(u) = 0 visited first at each level} do
4:    u ← Q.pop();
5:    if u ∈ H and level(u) ≥ 1 and b(u) = 1 {basic pair} then
6:       extract shortest path SP(h, u) with minimal f(u) and add to H*
7:       b(u) ← 0 {all later extension will become false}
8:    end if
9:    if level(u) = k {no expansion more than levelk for k-degree short-
      est path} then
10:      continue;
11:   end if
12:   if b(u) = 1 and u ∈ H* then
13:      f(u) ← f(u) + 1 {increase f}
14:   end if
15:   for all v ∈ neighbor(u) {(u, v) ∈ E; expanding u} do
16:      if v is not visited then
17:         add v to queue Q;
18:      else if level(v) = level(u) + 1 then
19:         if b(u) = 0 {update b} then
20:            b(v) ← 0;
21:         else if b(v) = 1 and f(u) > f(v) {update f} then
22:            f(v) ← f(u) and parent(v) ← u;
23:         end if
24:      end if
25:   end for
26: end while
```

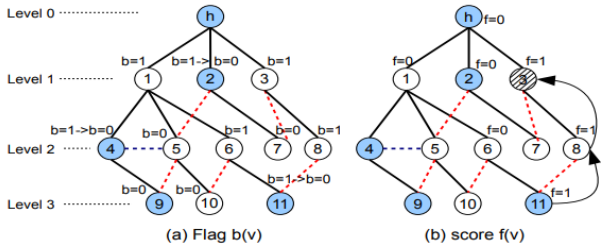


图 2：标志 b 和分数 f 的增量维护

顶点 u 的完全展开是从第 15 行到第 28 行。我们将访问它的每个邻居v。如果 v 尚未访问，我们将将其添加到队列中以供将来访问（第 16 行 − 18 行）。然后，我们执行标志 b（v）和分数 f（v） 的增量更新。如果 b（u） = 0（第 20 行 − 22），并且*如果* f（u） 大于 f（v），则标志 b（v） 将被关闭，即从 h 到 u 的最短路径在集线器网络中具有迄今为止最多的顶点数。顶点 v 将记录 u 作为父级（用于最短路径跟踪），并且 f（v） 将更新（第 24 行 − 第 26 行）。将为 H 中的每个中心调用此过程。

EXAMPLE 4.2. *图2 说明了 BFS 过程中的标志 b 和分数 f。此处的顶点 h、2、4、9 和 11 是中心。在图 2 （a）、（h，2）、（h，4） 和 （h，11） 中是基本对;标志 b 从原来的 b = 1 变为 b = 0（第5-7 行）。在标志 b 为 2、4 和 11 更改为 false （b = 0） 后，它们在 BFS 遍历中的所有后代都变为 false。例如，顶点 5 的标志 b 是假的，因为它也被认为是中心 2 的后代。在图 2（b） 中，阴影顶点 3 表示它已包含在集线器网络中（H⋆ ∈ 3）。因此，顶点 11 指向顶点 8（父（11）=8 和父（8）=3），因为它的 f 分数高于顶点 6 的分数。*

定理 2. *如果我们对每个 h ∈ H 调用算法 1，则诱导子图 G[H⋆] 是 H 相对于 k 度最短路径的集线器网络。*

简单证明：算法的正确性可以从以下两个观察值推导出来：1）对于任何距离不超过k的基本对（h，u），$G[H^\star]$中至少有一条最短路径，因为该算法显式提取最短路径并将其所有顶点添加到$H^\star$; 2）对于任何复合对（h，h'） 距离不超过 k，则它始终可以表示为基本对序列，其在 $G[H^\star]$ 中至少有一个最短路径。因此，对于任何距离不超过 k 的集线器对 （h，h'），它们的距离（至少一条最短路径）保留在诱导子图 $G[H^\star]$ 中。✷

算法 1 中描述的集线器网络发现的计算复杂性基本上等同于简单的 BFS 过程。整个过程采用O（Ph∈H（|Nk（h）|+|Ek（h）|））time，其中 H 是中心集，Nk（h） 和 $_{Ek}$（h） 分别是 u 的 k 度邻域中的顶点和边数。我们还注意到，此算法适用于无向图和有向图。有趣的是，我们注意到将算法 1 应用于无向图的以下属性。

引理 1. *设（u，v） 为无向图中的基本集线器对。考虑算法 1 首先从 u 执行 BFS ，它发现最短路径 SP（u，v）。当它从 v 执行 BFS 并发现对称基本对（v，u） 时，该算法不会添加任何其他新顶点。*

证明草图：分数 f 保证 f（v） = |SP（v，u）|=|SP（u，v）| 因此，将提取像SP（u，v）一样"良好"的最短路径，该路径不需要向H⋆添加任何新顶点。 ✷

此观察结果导致集线器网络的简单绑定约束（$H^\star$ 的最终大小），并且算法 1 的结果将与此类边界匹配。

引理 2. *设 $_{Dkb}$ ⊆ Dk ⊆ $H \times H$ 是距离不超过 k 的所有唯一基本集线器对的集合，则，*

$$|H^\star| \le \sum_{(u,v)\in D_k^b} (d(u,v)-1) + |H| \le \frac{|H|B}{2}(k-1) + |H|$$

where B is the average numnber of basic pairs per hub.

证明草图：术语 P$_{(u,\,v)\,\in Dkb}$（d（u，v） − 1） 对应于任何基本对只需要恢复一条最短路径的定义;这也对应于算法 1 中最坏的情况，对于任何基本对，沿新最短路径的所有非中心顶点都需要添加到 $H^\star$中。请注意，对于无向图，$_{Dkb}$ 将基本对（u，v） 和 （v，u） 视为单个对。这直接导致了术语|H|B/2（k−1）考虑了任何基本中心对之间的最大距离为k，对于对称的基本对（u，v）和（v，u）只需要恢复一条最短路径。算法 1 也认为（引理 1）。请注意，结果也适用于有向图，其中 B 是传入和传出边的总度数。 ✷

5. 基于 HUB 网络的搜索

在本节中，我们将介绍基于集线器网络的双向 BFS。这里的主要挑战是给定一个集线器网络，我们如何利用它来最大

限度地减少集线器的扩展，同时仍然保证发现正确的k度最短路径？回想一下，引入集线器网络的一个关键原因是使用它来约束集线器的扩展。因此，一个基本的搜索原则是，*任何集线器将只访问集线器网络中的邻居*。但是，集线器网络中的任何非集线器顶点 v（例如 $v \in H^{\star} \setminus H$）呢？是应该仅在集线器网络内扩展它们，还是应将其视为集线器网络外部的其余顶点？此外，在传统的双向BFS中，当两个搜索（向前和向后）首次相遇时，会发现最短的路径。不幸的是，如果集线器没有完全扩展，这并不一定成立，因此问题就变成了：正确的停止条件应该是什么？停止条件至关重要，因为它决定了搜索空间和发现确切最短路径的正确性。

在下文中，我们首先介绍基于集线器网络的双向BFS算法（小节5.1），然后证明其正确性并讨论其搜索成本（小节5.2）。

5.1 HN-BBFS 算法

基于集线器网络的双向BFS（HN-BBFS）算法由两个步骤过程组成：1）（**相遇步骤**）双向搜索将遍历集线器网络和其余图形，直到向前和向后搜索在第一个公共顶点相遇; 2）（**验证步骤**）接下来，搜索继续在其余图形（不是集线器网络）中进行，以验证第一步中发现的路径是否最短。如果没有，此步骤将发现另一个最短路径。

扩展规则：在会议步骤中，前进（后退）BFS遵循以下规则来扩展G中的顶点v：1）如果顶点是集线器，则它只扩展其在集线器网络中的邻居;2）如果顶点是常规顶点（不在集线器网络中），则它会扩展其所有邻居;3） 对于顶点是非中心顶点，但在集线器网络中，$H^{*}\backslash H$，如果 BFS 遍历首先通过集线器到达它，则它只扩展其在集线器网络中的邻居;否则，它被视为常规顶点（从开始（结束）顶点到它通过集线器的最短路径）。在验证步骤中，向前和向后 BFS 遍历都将继续，但它们不需要扩展任何集线器，并且集线器网络中的任何常规顶点和非中心顶点将扩展其在整个网络中的所有邻居。

停止条件：验证步骤中前进（后退）BFS的停止条件如下。设 *dist* 是迄今为止发现的最短路径距离;设 ^{dhs}（^{dht}） 是 s（h）与其最近的集线器 h 之间的距离;设 $_{levelf}$（$_{levelb}$） 是正向（后退）BFS 遍历的当前级别。然后，当满足以下条件时，向前（向后）BFS 将停止：

$$dist \geq level_f + d^h{}_s + 1 \ (dist \geq level_b + d^h{}_t + 1) \qquad (1)$$

总体算法：基于集线器网络的双向BFS（HNBBFS）在算法2中绘制。请注意，*向后搜索*本质上与"*反向搜索*"相同，为简单起见，省略了。最初，对于 k 度最短路径搜索，*dist* 设置为 k+1（表示 k 跳内没有路径），并且*满足*条件为 false（第 2 行）。

第一步（相遇步骤）由第一个 while 循环（第 3 行 – 第 6 行）执行，其中正向搜索和向后搜索以交替方式使用。在"*向前搜索*"（和"*向后搜索*"）中，将展开相应队列 *Qf*（*Qb*） 中的顶点。第 15 行中使用了前面描述的扩展规则。基本上，如果顶点是集线器或位于集线器网络 $H^{\star}\backslash H$ 中，但 BFS 遍历首先通过集线器到达它（通过集线器从 s 到 u 有一条最短的路径 ），则它被视为"集线器内网络"。否则，它是"出中心网络"。对于集线器内网络顶点，BFS 仅扩展其在集线器网络中的邻居。请注意，识别这些"集线器内网络"顶点非常简单，可以增量计算（类似于 在算法 1 中使用标志 b）。一旦向前（向后）搜索访问了向后（向前）搜索已访问的顶点，就会发现候选最短路径并将其*设置为* true。请注意，当 *V 序列化*为假时（在第一步中），将访问并展开每个顶点（集线器和非中心）。

Algorithm 2 HN-BBFS($G, G[H^{\star}], s, t$)

1: $Q_f \leftarrow \{s\}$; $Q_b \leftarrow \{t\}$; {Queues for forward and backward search}
2: $dist \leftarrow k+1$; $met \leftarrow false$;
3: **while** ($Q_f \neq \emptyset$ AND $Q_b \neq \emptyset$) AND NOT met AND $d(s, Q_f.top) + d(Q_b.top, t) < dist$ **do**
4: ForwardSearch($Q_f, false$); {not Verification Step}
5: BackwardSearch($Q_b, false$);
6: **end while**
7: $stop_f \leftarrow false$; $stop_b \leftarrow false$;
8: **while** (NOT (($Q_f = \emptyset$ OR $stop_f$) AND ($Q_b = \emptyset$ OR $stop_b$))) **do**
9: NOT $stop_f$: ForwardSearch($Q_f, true$); {true: Verification Step}
10: NOT $stop_b$: BackwardSearch($Q_b, true$)
11: **end while**
12: **return** $dist$ and shortest path;
Procedure ForwardSearch(Q_f,$Verification$)
13: $u \leftarrow Q_f.pop()$ {if $Verification$ is true, only out-hub-network vertices will be visited}
14: u is set to be visited by forward search;
15: **for all** $v \leftarrow neighbor(u)$ {if u is a hub or there is a shortest path from s to u via a hub, $neighbor(u)$ is within the hub-network} **do**
16: **if** v is visited by backward search {searches meet} **then**
17: **if** $d(s,u) + d(v,t) + 1 < dist$ **then**
18: update $dist$ and the shortest path correspondingly;
19: **if** NOT met {the first time meet} **then**
20: $met \leftarrow true$
21: **end if**
22: **end if**
23: **end if**
24: **if** v is not visited AND NOT ($Verification$ and $v \in H$) **then**
25: $Q_f.push\_back(v)$;
26: **end if**
27: **if** $Verification$ AND $dist \geq d(s,v) + d_t^h + 1$ **then**
28: $stop_f \leftarrow true$;
29: **end if**
30: **end for**

一旦*满足*即为真，第二步（验证步骤）由第二个 while 循环（第 8–11 行）执行。在满足前向停止条件（*stopf* 为 false）之前，前向搜索将继续。但是，将仅访问和扩展中心外网络顶点（13 号线和 24–26 号线）。此外，在扩展过程中，可以更新候选的最短路径（第 17–19 行）。最后，当满足停止条件时（第 26 行：d（s，v） 是当前正在扩展的 BFS 水平，因此 *level*），*stopf* 将变为 true，并且不会执行任何前向搜索（第 9 行）。请注意，在$_{BFS}$遍历期间可以轻松计算*dhs*（^{dht}）：第一次访问集线器时，其与*s的*距离被记录为^{dhs}。

5.2 正确性和搜索成本

我们现在讨论HN-BBFS（算法2）的正确性，然后讨论其搜索成本（特别是在新的停止条件F1方面）。为了证明HN-BBFS的正确性，我们将进行以下重要观察：

LEMMA 3. *对于任何集线器 $h \in H$，在第一步（会议步骤）中，使用正向 BFS 搜索计算的*距离 d（s，h）是 s *和 h 之间的*

确切最短路径距离。对于向后 BFS 遍历，d（h，t） 也是如此。

证明草图：如果 *s* 是集线器，那么根据集线器网络定义，这显然是成立的。如果 *s* 不是集线器，则以下两种情况之一必须成立：1） （*s*，*h*） 之间的所有最短路径都不包含除 h 之外的中心，因此正向 BFS 通过仅遍历原始图形中的非中心顶点来查找最短路径距离 d（s，h）;2） 之间有一条最短路径（*s*，*h*） 包含另一个集线器，因此始终存在 h'，使得 （*s*，h'）不包含任何集线器，并且可以在集线器网络中发现 （h'，*h*）。✷

Lemma 3 演示了集线器网络的强大功能，并表明 HN-BBFS 可以正确计算查询顶点到集线器之间（以及集线器之间）的最短路径（距离）。但是，尽管如此，在第一个会议顶点发现的候选最短路径可能不是确切的路径。以下引理对确切的最短路径进行分类，如果它们比在第一步（会议步骤）中发现的候选最短路径短。

莱格玛 4. *假设 u 是前向和后向搜索首先相遇的相遇顶点（*算法 2 中的第 22 − 26 行*），候选最短路径表示为SP（s，u，t），距离 d（s，u） + d（u，t）。如果存在较短的路径，则它必须包含集线器 h，以便确切的最短路径可以表示为两个段 SP（s，h） 和 SP（h，t）。此外，1） SP（s，h） 除 h 之外不包含距离为 d（s，h） ≥ d（s，u） 和 d（h，t） < d（u，t） 或 2 以外的*集线器 SP（h，t） *不包含集线器距离为 d（s，h） < d（u，t） 和 d（h，t） ≥ d（u，t） 的 h 除外。*

证明素描：我们通过矛盾的方式证明这一点。如果引理不成立，则以下两种类型的路径不能短于发现的候选最短路径：1）在确切的最短路径SP（*s*，*t*）中没有集线器，以及2）有两个集线器*hs*和$_{ht}$，使得最短路径有三个段：*SP*（*s*，*hs*），*SP*（$_{hs}$，*ht*） 和 *SP*（*ht*，*t*），其中 *d*（*s*，*hs*） < *d*（*s*，*u*） 和 *d*（*ht*，*t*） < *d*（*u*，*t*）。对于第一种情况，双向 BFS 应该能够更早地找到这样的路径（如果它们比候选 *SP*（*s*，*u*，*t*）短），因为它只涉及访问图中的非中心顶点。对于第二种情况，基于引理 3，算法 2 在两个 BFS 在 u 处相遇之前 计算出精确的 *d*（*s*，*hs*） 和 *d*（*ht*，*t*），并且集线器网络编码 *d*（*hs*，$_{ht}$） 之间的正确距离。因此，如果 *d*（*s*，*hs*） + *d*（*hs*，$_{ht}$） + *d*（*ht*，*t*） < *d*（*s*，*u*） +*d*（*u*，*t*），则应在第一步（会议步骤）中发现此最短路径（在集线器内网络顶点处满足）。由于这两种情况都是不可能的，因此引理成立。✷

定理 3. *基于集线器网络的双向BFS方法（HN-BBFS，算法2）保证了确切的k度最短路径*的发现。

证明草图：基本上，我们需要证明，当满足停止条件时，不存在更短的替代路径。根据引理 4，如果存在比候选最短路径 *SP*（*s*，*u*，*t*）更好的最短路径，则它必须遵循两种简单格式之一。这些格式表明，我们只需要扩展集线器网络外顶点，直到它们遇到已经从另一个方向访问的集线器（*d*（*s*，$_{hs}$） < *d*（*s*，*u*） 或 *d*（$_{ht}$，*t*） < *d*（*u*，*t*））。如果可以找到这样的路径，它必须比已经发现的距离 *dist* 短，即 *dist* > *levelf* + 1 + $_{dht}$（最佳情况是当最短路径从当前步长一步延伸到最接近查询顶点的中心时）。显然，如果这不成立，则此格式中的任何最短路径都不会小于 *dist*。✷

在经典的双向搜索中，一旦两个方向在一个共同的顶点相遇，就可以停止搜索并发现确切的最短路径。但是，在HN-BBFS中，为了减少集线器的扩展，必须采取一些额外的遍历（验证步骤）。显然，如果我们需要走*k* / 2额外的步骤，那么HN-BBFS的好处可能会受到很大的影响。

那么，HN-BBFS 在验证步骤中的典型（随机）查询需要执行的平均步骤数是多少？该数字接近 *子零*或最多为 1。为了说明这一点，首先将两个顶点对之间的距离视为6（因为大多数距离小于社交网络中的距离[39]），并假设*s*和*t*不是集线器（因为集线器很少），但它们中的每一个都有一个直接的hubneighbor ^{dhs} = 1（^{dht} = 1）。请注意，两个方向通常最多有三个步骤，即 *levelf* = *levelb* = 3。因此，在这种情况下，最多需要采取一个额外的步骤来使停止条件为真：*dist*−*levelf* −^{dht} −1 ≥ 0，其中 $_{levelf}$ = 4。同样，让我们将距离视为 4，并假设每个方向在 会议步骤中走了 2 步。在这种情况下，无需采取额外的步骤（假设 *s* 和 *t* 不是中心），我们可以立即识别出候选的最短路径确实是确切的路径。最后，我们注意到，当 *dist*−*levelf* −^{d} ^{ht} −1 = 1 时，即 BFS 验证的最后一步，不需要展开给定顶点的所有邻居。只有其直接的枢纽需要扩展和检查（引理4和定理3）。为了便于实现此目的，可以重新组织常规顶点的相邻要素，以便分别记录中心邻居和非中心邻居。

6. 用于最短路径计算的 HUB ^{2}-LABELING

In this section, we present a Hub²-labeling approach which aims to completely avoid visiting (and expanding) any hub. To achieve this, more expensive though often affordable pre-computation and memory cost are utilized for faster online querying processing. In Subsection 6.1, we will describe the Hub²-labeling framework and its index construction. In Subsection 6.2, we will discuss the faster bidirectional BFS.

6.1 Hub^{2}-Labeling 框架

Hub2-Labeling 将 Hub-Network 替换为 *Hub2* 距离矩阵和 *Hub Labeling*。

Hub²：集线器对（称为 Hub2）之间的距离矩阵是预先计算的，并存储在主内存中。实际上，对于 k 度最短路径，只需要计算距离不超过 *k* 的对的距离。正如我们之前所讨论的，如今，具有中等内存大小的台式计算机可以轻松地为10K（或更多）集线器保存这样的矩阵。

Hub Labeling: 为了有效地利用距离矩阵，图中的每个顶点 *v* 还记录了一小部分集线器（称为 *核心集线器*）以及距离。基本上，这些核心中心以及距离矩阵可以帮助快速估计查询顶点对之间的距离上限，并可用于限制双向BFS的搜索步骤。现在，我们正式定义了 *核心中心*。

定义 3. **（Core-Hubs）***给定图G* = （*V*，*E*）*和一个集线器的集合H，对于每个顶点v，我们说顶点h*∈*H是v的核心枢纽，*

如果没有其他中心$h' \in H$*使得*$d(v, h) = d(v, h') + d(h', h)$*.形式上，*$L(v) = \{h \in H : \nexists h' \in H, d(v, h) = d(v, h') + d(h', h)\}$。

简单地说，如果在 v 和 h 之间的任何最短路径中没有其他顶点 h' 出现，*则* h 是 v 的核心中心。请注意，一对 (v, h)（其中 $v \in L(v)$）类似于集线器网络中的基本对（第 4.2 小节）。原始的基本对定义仅引用中心对，但此处将其扩展到具有一个中心和一个非中心顶点的顶点对。

示例 6.1. *图1（c）说明了原始图形中每个非中心顶点的核心枢纽（以及距离）（图1（a））。此处的集线器为* 4、6、8、12、17、18 *和* 19*。例如，Vertex* 1 *只需要记录核心集线器* 4、6、12 *和*19，它可以通过*它们以最短的路径到达集线器* 8 *和*17。

使用核心枢纽 L 和距离矩阵 Hub2，我们可以按以下方式近似顶点对 (s, t) 的距离和最短路径：

$$d_H(s,t) = min_{x \in L(s) \wedge y \in L(t)}\{d(s,x) + d(x,y) + d(y,t)\} \quad (2)$$

Here, $d(x,y)$ is the exact distance recorded in the distance-matrix Hub².

距离矩阵 Hub2 的构造和核心集线器的标记也相当简单。算法 1 中的 BFS 过程可以很容易地采用：1） 每个 BFS 执行 k 个步骤，因此可以直接构造距离矩阵;2） 当顶点 v 从 BFS 遍历 h 时具有标志 $b = 1$（基本对），我们只需将 h 附加到 $L(v)$ 中。因此，预计算的总计算复杂度为$O(P_{h \in H}(Nk(h) + Ek(h)))$时间，其中$H$是中心集，$Nk(h)$和$Ek(h)$分别是 u 的 k 度邻域中的顶点和边的数量。我们注意到，对于有向图，我们将计算 $Lin(v)$ 和 $Lout(v)$，一个用于传入的核心集线器 (h, v)，另一个用于传出的核心集线器 (v, h)。要构造这样的标签，我们需要从每个中心执行向前和向后 BFS。

Hub2 标记的总体内存成本是距离矩阵 （Hub2） 的成本与每个顶点 （$L(v)$的核心中心标记的成本之和：$P_{v \in V} O(|L(v)|) + O(|H|^2)$。事实证明，这是相当实惠的。在实验研究中，我们发现对于大多数真实的社交网络，每个顶点v的核心中心 仅占总中心数的一小部分（在大多数情况下，小于或接近2%）。因此，Hub2 标签可以轻松处理具有 10K 个以上集线器的图形。此外，由于第二项（距离矩阵的大小）是稳定的，因此随着原始图形中顶点数量的增加，第一项将相对于$|V|$.

6.2 Hub²-Labeling 查询处理

为了计算顶点对（s，t）之间的k度最短路径，Hub2-Labeling中的在线查询过程包括两个步骤：**步骤1（距离估计）**：使用距离矩阵Hub2和核心枢纽标记$L(s)$和$L(t)$，距离$d_H(s, t)$是估计值（公式 2）。

步骤 2（集线器修剪双向 BFS （HP-BBFS））：执行来自 s 和 t 的双向 BFS，并且搜索步骤受 k（对于 k 度最短路径）和 $d_H(s, t)$ 之间的最小值的约束。特别是，在双向搜索期间，不需要扩展任何中心。在数学上，中心修剪双向BFS等效于在G的非中心诱导子图$G[V \setminus H]$ 上执行典型的双向*BFS*。

4. *两步* Hub2 *标记查询过程可以正确计算*图 G 中的 k *度最短路径。*

证明草图：我们观察到，任何距离不超过k的顶点对 都可以归类为：1）顶点对至少有一个最短路径穿过H中的至少一个集线器;2）最短路径从不通过任何集线器的顶点对。

对于任何距离不大于 k （$d(s, t) \le k$） 的顶点对 （s，t），如果存在一个满足 $d(s,t) = d(s, x') + d(x', t) \in H$的枢纽 $x' = d(s, x') + d(x', t)$，那么，我们可以始终找到 $x \in LH(s)$ 和 $y \in LH(t)$，使得 $d(s, t) = d(s, x) + d(x, y) + d(y, t)$。换句话说，使用距离矩阵 Hub2 和核心中心标记的步骤 1（距离估计）可以处理此类别。此外，步骤2将帮助确认最短的路径属于此类别（找不到较短的路径）。

如果在步骤 1 中计算的近似最短路径不是精确路径，则最短路径不涉及任何中心。因此，步骤2可以保证使用非集线器诱导子图 $G[V \setminus H]$中的双向搜索提取精确的最短路径。 ✷

一对s和t的在线查询处理的时间复杂度可以写成$O(|L(秒)||L(t)| + Nk/2|G[V \setminus H]) + Ek/2(s|G[V \setminus H]) + Nk'/2(t|G[V \setminus H]) + Ek'/2(t|G[V \setminus H]))$，其中$|L(秒)||L(t)|$是距离估计成本，其余术语是双向搜索的成本。 $Nk/2$（$Nk'/2$） 和 $Ek/2$（Ek' 2） 是非中心诱导子图 $G[V \setminus H]$ 的 $k/2$ 邻域（跟随传入边的反向邻域）中的顶点和边数].由于排除了中心，因此中心修剪双向 BFS 的成本明显低于原始图形上的成本。

但是，如果核心标签的数量很大，则距离估计可能很昂贵（在 $L(s)$ 和 $L(t)$ 上执行成对连接）。为了解决这个问题，$L(u)$ 中的核心集线器可以按级别组织，每个级别对应于它们与u的距离，例如$L1(u)$，$L2(u)$，$\cdot Lk(u)$.使用这种按级别的组织，我们可以执行更有效的距离估计：首先在$L1(s)$和 $L1(t)$ 之间执行成对连接;然后在（$L1(s)$，$L2(t)$），（L）上执行²、$L1(t)$）、（$L2(s)$、$L2(t)$）等。鉴于此，让我们将 d 表示为迄今为止通过成对连接获得的最短路径长度。假设我们当前正在处理 （$Lp(s)$，$Lq(t)$），如果 $d < p + q$，那么我们立即终止成对连接。这是因为 （$Lp'(s)$，$Lq'(t)$） 不可能产生更好的结果，因为$p'+q' \ge p+q > d$.这种基于逐级组织的提前终止策略，可以帮助我们有效修剪不必要的成对联接操作，提高查询效率。

7. 实验评估

在本节中，我们根据经验评估了算法在一系列大型真实社交网络上的性能。特别是，我们将比较Hub-Network方法（表示为 **HN**）和Hub2-Labeling方法（表示为 **HL**）与以下方法：1）基本的广度优先搜索（表示为 **BFS**）;2）双向广度优先搜索（表示为 **BiBFS**）;3）草图算法[9]（表示为 **S***），最先进的近似距离估计算法;4）树草图方法[16]（表示为 **TS***），该方法利用树来提高基于Sketch的最短路径计算的近似精度。这里的符号 $\star$ 也表示它是一种近似方法。

此外，我们还测试了两种最新的精确最短路径距离方法，包括基于树分解的最短路径计算[40]和基于作者提供的实现的以高速公路为中心的标记方法[19]。但是，它们都不能在本研究中使用的图形上工作。这是意料之中的，因为它们的索引成本非常高（树分解或集合覆盖方法），并且它们主要关注非常稀疏的图形。

我们还测试了RigelPath，这是社交网络中近似最短路径发现的另一种最新方法[42]。然而，它的查询性能比Sketch慢（在他们自己的研究中也证实了这一点[42]）。此外，它目前的实现只关注无向图，其中大多数真正的基准测试网络都是定向的。因此，我们在这里不报告RigelPath的实验结果。

我们在C++和标准模板库（STL）中实现了我们的算法。基于草图的方法（包括**S***和**TS***）的实施由作者[16]提供（也C++中实施）。所有实验均在具有2.48GHz AMD皓龙处理器和32 GB RAM的Linux服务器上运行。

在实验中，我们对两个重要的度量感兴趣：查询时间和预处理成本，它由预计算时间和索引大小组成。为了测量查询时间，我们随机生成 10，000 个顶点对，并获取每个查询的平均运行时间。对于索引大小，由于所有 Sketch 索引都以 RDF 格式存储，因此其索引大小将根据相应的 RDF 文件大小进行度量。如果预处理在48小时内无法完成，我们将停止它并在结果表中记录"-"。此外，我们注意到，所有基于 Sketch 的基准测试只能近似最短路径，其中近似精度受迭代采样过程的影响。指定参数 r 以确定采样迭代次数，从而生成 $2r \log|V|$每个顶点的草图。为了与精确的查询方案进行公平的比较，我们 按照[16]中的建议设置r = 2，这可以生成具有良好近似精度和高效查询处理的草图。此外，在这项研究中，我们专注于再次比较新方法的查询时间，尽管它们只能提供近似的解决方案，而我们的方法可以提供精确的解决方案。

基准数据集列于表 1 中。他们中的大多数是从在线社交网络收集的，顶点的数量从几万到超过 1000 万不等。其他人还表现出在社交网络中常见的某些属性，例如小直径和相对较高的平均顶点度。所有数据集均可从斯坦福大学大型网络数据集集合 [1]、马克斯普朗克研究所的在线社交网络研究中心 [2] 和亚利桑那州立大学的社会计算数据存储库下载 [3]。

在表1中，我们提出了所有真实数据集的重要特征，其中δ是平均顶点度（即$2|E|/|V|$） 和 $d0.9$为90百分之一有效直径[24]。最后，在实验研究中，我们专注于6度最短路径查询（$k = 6$），因为它们是最常用的，也是最具挑战性的。

1. 实验结果

在下文中，我们从不同角度报告了最短路径计算算法的有效性和效率： **随机查询的查询结果** 在本实验中，我们随机生成了10，000个具有不同距离的顶点对，并在这些查询上执行所有算法以研究其性能。在这里，我们选择 10，000 个顶点度最高的顶点作为中心。表 3 显示了所有方法上 10，000 个查询的平均查询时间，表 4 突出显示了距离不小于 4（较长路径）的顶点对的平均查询时间 ，因为这些顶点对更具挑战性（路径越长，扩展的中心就越多）。请注意，对于BFS和两种草图方法Sketch（S*）和Tree Sketch（TS*），我们使用*毫秒*（10-3）作为单位，因为它们通常具有更长的查询时间，对于BiBFS和我们的新方法，Hub-Network（HN）和Hub2Labeling（HL）方法，我们使用*微秒*（10-6）） 作为单位，因为它们要快得多。表 5 中报告了每个查询的相应平均搜索空间，其中"HP-BBFS"列记录了 HP-BBFS（集线器修剪双向 BFS）在 Hub2 标签 （HL） 中访问的平均顶点数，"Join"列记录了在核心集线器上成对联接的平均次数，这些点标记 L（s） 和 L（t） 在HL中。我们对查询时间和平均搜索空间进行以下观察：

1）Hub2-Labeling（HL）显然是所有算法中的赢家，平均比BFS快2000倍以上。在大多数社交网络中，如As-skitter和WikiTalk， Hub2-Labeling（HL）的平均查询时间仅为数十微秒（10-8秒），除了一个（Orkut），所有tham都小于1ms。比 Bi BFS 快一些。具体而言，我们观察到，与BiBFS相比，Hub-Pruning双向搜索（HP-BBFS）在搜索空间方面实现了显着改善，比BiBFS小约800倍（表5）。 它比BFS快约两个数量级，但比Hub2-Labeling方法慢约10倍。3）Sketch（S*）平均比BFS快约10倍，但它无法在少数数据集上运行。树草图（TS*）平均 比草图慢70倍。Hub-Network和Hub Labeling方法平均比最快的近似方法Sketch快两个数量级以上。4） 对于 d（u，v） ≥ 4 的长距离查询，最短路径方法需要更长的查询时间（表 4）。但是，集线器网络 （HN） 和 Hub2 标记 （HL） 的增长幅度小于 BFS 和 BiBFS。此外，有趣的是，观察到近似的最短路径方法不会显示性能下降，尽管它们仍然非常慢。

预处理成本：表 6 显示了基于 Sketch 的方法和 HL 的预处理成本，包括索引大小和预计算时间。第一列 S* 记录 Sketch 方法的索引大小 （MB）。第二列 HL_{total} 记录 Hub2 标签 （HL） 的总索引大小，这是标记成本和距离矩阵大小的核心枢纽的总和。列$|L(v)|$记录每个顶点存储的核心中心的平均数。值得注意的是，Hub2-Labeling（HL）中的核心-中心标记方案非常有效，因为每个顶点记录了非常小的一部分核心中心。在大多数网络中，每个顶点的平均核心枢纽数量不超过 总枢纽数的 2%。特别是，对于网络WikiTalk，只有2。 每个顶点平均存储 5 个核心中心，这可能会导致高效的查询应答。但是，对于LiveJournal来说，Hub2-Labeling太昂贵了，无法在主内存中实现。在预计算时间方面，Hub2-Labeling的构建速度比10个网络中的7个网络的Sketch更快。HubNetwork （HN） 的构建时间平均比 Hub2-Labeling （HL） 快三倍以上，并且不需要任何额外的内存成本。

中心编号的影响：在此实验中，我们研究了不同数量的中心对查询性能的影响。在这里，我们将中心集的大小从 5，000 更改为 15，000，并对具有不同距离的 10，000 个随机生成的查询进行实验。表 2 显示了使用不同数量的集线器的集线器

网络（HN）和集线器 [2] 标记方法的平均查询时间。在大多数这些网络中，当集线器数量介于 10K 和 15K 之间时，可实现最佳查询性能。尽管大量中心可能有助于减少 Hub2 标记（HL） 中双向搜索的搜索空间，但它也可能会增加与每个顶点关联的核心中心的大小。我们观察到，使用 10K 中心获得的查询性能与最佳中心相当）。请注意，由于空间限制，我们不会根据施工时间和索引大小（对于 Hub2 标签）报告详细的预计算成本。总体而言，随着集线器数量的增加，大多数大型网络在平均索引大小方面都呈现出增加的趋势。有趣的是，当中心集的大小增加时，在WikiTalk上观察到平均索引大小的显着减少。这部分原因在于其直径非常小。在预计算时间方面，随着更多的集线器被选择，集线器网络和集线器[2]标签的计算成本会变大，因为需要执行更多的BFS。实际上，相对于中心集的大小，预计算时间几乎呈线性增长。

Dataset	$\|V\|$	$\|E\|$	$\bar{\delta}$	$d_{0.9}$
Facebook	63731	1545686	48.51	8.2
Slashdot	82168	948464	23.09	4.7
BerkStan	685230	7600595	22.18	10
Youtube	1138499	4945382	8.69	7.14
As-skitter	1696415	11095298	13.08	5.9
Flickr	1715255	22613981	26.37	7.32
Flickr-growth	2302925	33140018	28.78	7.19
Wiki-talk	2394385	5021410	4.19	4
Orkut	3072441	223534301	145.51	5.7
LiveJournal	5204176	77402652	29.75	8.34
Twitter	11316811	85331845	15.08	24.97

Table 1: Network Statistics

Dataset	$\|H\| = 5000$		$\|H\| = 8000$		$\|H\| = 10000$		$\|H\| = 15000$	
	HN	HL	HN	HL	HN	HL	HN	HL
Facebook	0.043	0.018	0.044	0.017	0.042	0.017	0.040	0.019
Slashdot	0.023	0.002	0.021	0.001	0.022	0.001	0.022	0.002
BerkSta	0.011	0.005	0.005	0.009	0.010	0.004	0.014	0.002
Youtube	0.106	0.006	0.119	0.005	0.125	0.005	0.136	0.005
As-skitter	0.051	0.016	0.044	0.015	0.040	0.013	0.041	0.011
Flickr	1.600	0.112	1.671	0.073	1.739	0.067	1.888	0.061
Flickr-growth	0.998	0.138	1.130	0.113	1.193	0.100	1.236	0.136
Wiki-talk	0.014	0.002	0.016	0.002	0.014	0.002	0.014	0.001
Orkut	0.952	3.653	0.955	3.314	0.978	3.356	1.078	3.282
LiveJournal	0.466	-	0.526	-	0.513	-	0.577	-
Twitter	1.850	0.306	1.947	0.314	2.083	0.340	2.121	-

Table 2: Average Query Time with Different Hub Sizes (ms)

Dataset	BFS	S*	TS*	BiBFS	HN	HL
	ms			μs		
Facebook	1.7	0.5	20.4	55.2	41.9	17.4
Slashdot	1.4	0.7	34.5	31.6	22.2	1.3
BerkStan	0.3	4.7	559.1	33.9	10.2	3.5
Youtube	15.3	2	171.2	312.2	125.1	5.4
As-skitter	4.9	1.5	114.9	86.7	40.4	12.7
Flickr	42.6	2.7	288.7	2887.9	1738.8	67.3
Flickr-growth	71.8	5.1	305	1607.4	1193.3	100.3
Wiki-talk	18.8	-	-	56.4	14.1	1.5
Orkut	202.5	7.8	258.5	1338.7	978.1	3356.4
LiveJournal	131.4	-	-	749.6	513.1	-
Twitter	221.4	-	-	2311.8	2082.6	339.7

Table 3: Average Query Time on Random Query

Dataset	BFS	S*	TS*	BiBFS	HN	HL
	ms			μs		
Facebook	1.9	0.5	19.6	61.2	45.7	19.9
Slashdot	1.7	0.7	46.8	31.4	20.3	1.5
BerkStan	0.3	2.1	206.7	36.1	10.6	3.8
Youtube	16	1.2	95	325.8	130.7	5.6
As-skitter	5.4	1.2	84.2	94.7	46.3	14
Flickr	45.2	2.9	182.1	3060	1825.2	79.1
Flickr-growth	71.9	3.7	332.5	1616.6	1219.6	103.6
Wiki-talk	21.7	-	-	58.3	14.2	1.1
Orkut	225.8	3.4	268	1372.9	1111.1	4639.5
LiveJournal	127.7	-	-	699.3	524	-
Twitter	250.4	-	-	2384.3	2190.1	254.5

Table 4: Average Query Time on Random Query with Distance ≥ 4

Dataset	BFS	BiBFS	HN	HL	
				HP-BBFS	Join
Facebook	30589	1723	1867	208	466
Slashdot	41030	1380	1358	3	20
BerkStan	11099	1462	405	78	39
Youtube	505842	13941	6303	78	90
As-skitter	161878	3580	1551	292	265
Flickr	580315	36161	15494	1431	1330
Flickr-growth	777994	23738	12412	2382	1431
Wiki-Talk	1178526	4255	1111	1	7
Orkut	1522640	29341	21954	71331	5367
LiveJournal	1784211	14172	15554	-	-
Twitter	3275797	55558	54884	13866	10757

Table 5: Average Search Space on Random Query

Dataset	Indexing Cost			Preproc.Time(min)		
	S^*(MB)	HL_{all}(MB)	$\overline{\|L(v)\|}$	S^*	HN	HL
Facebook	10	955	8.2	3.2	2.2	3.8
Slashdot	26	496	11.1	6.5	1.3	4.3
BerkStan	193	291	21.6	64.3	0.3	1.7
Youtube	217	757	38.9	100.8	15.5	66
As-skitter	391	1229	101.9	109.9	7.1	31.7
Flickr	626	1536	232	163.8	43.4	202.5
Flickr-growth	1004	4403.2	315.9	242.8	71.8	363.5
WikiTalk	-	481	2.5	-	12.5	41.2
Orkut	8397	13517	749.3	773.2	412.5	1431.6
LiveJournal	-	-	-	-	334.2	-
Twitter	-	26931	464	-	233.9	390.2

Table 6: Preprocessing Cost on Random Query

Dataset	$\|H\| = 5000$			$\|H\| = 8000$			$\|H\| = 10000$			$\|H\| = 15000$		
	$\|H^*\|$	$d_1(H)$	$d_2(H)$	$\|H^*\|$	$d_1(H)$	$d_2(H)$	$\|H^*\|$	$d_1(H)$	$d_2(H)$	$\|H^*\|$	$d_1(H)$	$d_2(H)$
Facebook	20854	247.7	217.1	27364	202.7	184.5	30554	182.2	168.1	36188	146.6	137.5
Slashdot	23359	204.5	179.5	27581	150.1	135.6	29500	128.4	117.2	32665	95.2	88.0
BerkStan	8290	769.3	177.8	16563	574.3	152.8	24618	492.8	138.1	34342	364.6	110.3
Youtube	49516	587.5	299.9	69474	429.9	254.9	76894	369.4	231.1	100595	279.2	189.3
As-skitter	41371	958.9	211.0	56245	701.3	184.8	64785	601.4	171.3	82439	453.0	146.3
Flickr	19198	2539.3	1433.3	32972	2005.8	1364.7	42312	1776.7	1295.0	63774	1403.7	1128.9
Flickr-growth	22715	3175.3	1626.7	38819	2555.4	1615.5	49450	2284.0	1565.4	74569	1833.5	1407.7
Wiki-talk	24139	984.5	294.7	32435	669.2	220.3	36081	552.4	188.8	41567	385.9	139.9
Orkut	124607	3808.5	1720.9	189686	3022.9	1763.0	225678	2734.3	1763.4	319989	2305.0	1720.0
LiveJournal	151348	1172.3	702.1	229836	1004.5	673.4	278203	932.8	653.7	392423	808.8	611.0
Twitter	201521	9556.6	2877.8	346091	6762.9	2641.2	424853	5749.2	2463.3	564435	4267.5	2084.0

Table 7: Hub-Network Statistics

集线器网络统计信息：最后，我们报告发现的保持集线器网络的距离的基本统计信息。具体来说，我们在以下两个问题中介绍：1）给定一组集线器，集线器网络将有多大？|的大小是多少 H^*|？ 2） 原始网络中的集线器与集线器网络中的集线器之间的程度差异是什么？我们是否观察到显著程度的下降？为了回答这两个问题，在表7中，我们报告了|H^*|（集线器网络中的总顶点数），*d1*（H）原始图中集线器的平均度数，*以及d2*（H），提取的集线器网络中相对于5K，*8K*， 10K和15K的平均集线器度枢纽。我们观察到大多数图表的大小|H^*|比中心数量大几倍;但是，对于Orkut，LiveJournal和Twitter，

中心网络在10K和15K中心变得相当大。此外，一般来说，集线器网络中的集线器度已经降低，并且在几个图形上，平均度数降低小于原始平均度的1/3。我们还观察到，降低度的能力与搜索性能相关：中心度越低，我们从基于中心网络的双向BFS获得的查询性能改进就越好。

8. 结论

在本文中，我们介绍了一组以大型社交网络中k度最短路径计算中心为中心的新技术。集线器网络和集线器 [2] 标记算法可以帮助显著减少搜索空间。广泛的实验研究表明，这些方法可以处理具有数百万个顶点的非常大的网络，其查询处理速度比在线搜索算法和基于Sketch的方法（最先进的最短路径近似算法）快得多。据我们所知，这是第一个关于在大型社交网络上计算精确最短路径的实践研究。未来，我们将研究如何并行化索引构建和查询应答过程。我们还计划研究如何在动态网络上计算*k*度最短路径。

9. 引用

[1] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck. A hub-based l abeling algorithm for shortest paths in road networks. In *Proceedings of th e 10th international conference on Experimental algorithms*, 2011.

[2] I. Abraham, A. Fiat, A. V. Goldberg, and R. F. Werneck. Highway dimensi on, shortest paths, and provably efficient algorithms. In *SODA '10*, 2010.

[3] H. Bast, S. Funke, P. Sanders, and D. Schultes. Fast Routing in Road Net works with Transit Nodes. *Science*, 316:566–, April 2007.

[4] R. Bauer, D. Delling, P. Sanders, D. Schieferdecker, D. Schultes, and D. Wagner. Combining hierarchical and goal-directed speed-up techniques fo r dijkstra's algorithm. *J. Exp. Algorithmics*, 15, March 2010.

[5] James Cheng, Yiping Ke, Shumo Chu, and Carter Cheng. Efficient proces sing of distance queries in large graphs: a vertex cover approach. In *SIGM OD Conference*, pages 457–468, 2012.

[6] James Cheng, Zechao Shang, Hong Cheng, Haixun Wang, and Jeffrey Xu Yu. K-reach: Who is in your small world. *PVLDB*, 5(11):1292–1303, 201 2.

[7] Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. Reachability a nd distance queries via 2-hop labels. *SIAM J. Comput.*, 32(5):1338–1355, 2003.

[8] Reuven Cohen and Shlomo Havlin. Scale-free networks are ultrasmall. *Ph ys. Rev. Lett.*, 90, Feb 2003.

[9] A. Das Sarma, S. Gollapudi, and R. Najork, M.and Panigrahy. A sketch-ba sed distance oracle for web-scale graphs. In *WSDM '10*, 2010.

[10] D.Djokovic. Distance-preserving subgraphs of hypercubes. *Journal of Co mbinatorial Theory, Series B*, 14(3):263 – 267, 1973.

[11] D. Delling, P. Sanders, D. Schultes, and D. Wagner. Algorithmics of large and complex networks. chapter Engineering Route Planning Algorithms. 2 009.

[12] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numeri sche Mathematik*, 1(1):269–271, December 1959.

[13] Cyril Gavoille, David Peleg, Ste´phane Pe´rennes, and Ran Raz.

Distance labeling in graphs. *J. Algorithms*, 53(1):85–112, 2004.

[14] R. Geisberger, P. Sanders, D. Schultes, and D. Delling. Contraction hierarc hies: faster and simpler hierarchical routing in road networks. In *Proceedi ngs of the 7th international conference on Experimental algorithms*, 2008.

[15] A. V. Goldberg and C. Harrelson. Computing the shortest path: A search m eets graph theory. In *SODA '05*, 2005.

[16] Andrey Gubichev, Srikanta Bedathur, Stephan Seufert, and Gerhard Weik um. Fast and accurate estimation of shortest paths in large graphs. In *Proc eedings of the 19th ACM international conference on Information and kno wledge management*, CIKM '10, pages 499–508, 2010.

[17] R. J. Gutman. Reach-based routing: A new approach to shortest path algor ithms optimized for road networks. In *ALENEX/ANALC*, pages 100–111, 2004.

[18] R. Jin, Y. Xiang, N. Ruan, and D. Fuhry. 3-hop: a high-compression indexi ng scheme for reachability query. In *SIGMOD'09*, 2009.

[19] Ruoming Jin, Ning Ruan, Yang Xiang, and Victor E. Lee. A highway-centr ic labeling approach for answering distance queries on large sparse graph s. In *SIGMOD Conference*, pages 445–456, 2012.

[20] N. Jing, Y. Huang, and E. A. Rundensteiner. Hierarchical encoded path vie ws for path query processing: An optimal model and its performance evalu ation. *TKDE*, 10(3):409–432, 1998.

[21] S. Jung and S. Pramanik. An efficient path computation model for hierarc hically structured topographical road maps. *TKDE*, 14(5):1029–1046, 200 2.

[22] J. Kleinberg, A. Slivkins, and T. Wexler. Triangulation and embedding usi ng small sets of beacons. In *FOCS '04*, 2004.

[23] H. Kriegel, P. Kro¨ger, M. Renz, and T. Schmidt. Hierarchical graph embe dding for efficient query processing in very large traffic networks. In *SSD BM '08*, 2008.

[24] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *ACM KDD '05*, pages 177–187, 2005.

[25] Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahon ey. Community structure in large networks: Natural cluster sizes and the a bsence of large well-defined clusters. *CoRR*, abs/0810.1355, 2008.

[26] C. Castillo M. Potamias, F. Bonchi and A. Gionis. Fast shortest path distan ce estimation in large networks. In *CIKM '09*, 2009.

[27] Alan Mislove, Hema Swetha Koppula, Krishna P. Gummadi, Peter Drusch el, and Bobby Bhattacharjee. Growth of the flickr social network. In *Proce edings of the 1st ACM SIGCOMM Workshop on Social Networks (WOSN' 08)*, August 2008.

[28] T. S. Eugene Ng and H. Zhang. Predicting internet network distance with coordinates-based approaches. In *INFOCOM*, 2001.

[29] Miao Qiao, Hong Cheng, Lijun Chang, and Jeffrey Xu Yu. Approximate s hortest distance computing: A query-dependent local landmark scheme. In *ICDE*, 2012.

[30] H. Samet, J. Sankaranarayanan, and H. Alborzi. Scalable network distance browsing in spatial databases. In *SIGMOD'08*, 2008.

[31] P. Sanders and D. Schultes. Highway hierarchies hasten exact shortest pat h queries. In *17th Eur. Symp. Algorithms (ESA)*, 2005.

[32] J. Sankaranarayanan, H. Samet, and H. Alborzi. Path oracles for spatial ne tworks. *PVLDB*, 2, August 2009.

[33] R. Schenkel, A. Theobald, and G. Weikum. HOPI: An efficient connection index for complex XML document collections. In *EDBT*, 2004.

[34] S. Shekhar, A. Fetterer, and B. Goyal. Materialization trade-offs in hierarc hical shortest path algorithms. In *SSD '97*, 1997.

[35] Y. Tao, C. Sheng, and J. Pei. On *k*-skip shortest paths. In *SIGMOD'11*, 20 11.

[36] M. Thorup and U. Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, 2005.

[37] `http://entitycube.research.microsoft.com`.

[38] `http://www.6-degreeonline.com`.

[39] `http://www.sysomos.com/insidetwitter/sixdegrees/`.

[40] Fang Wei. Tedi: efficient shortest path query answering on graphs. In *SIG MOD Conference*, pages 99–110, 2010.

[41] Xiaohan Zhao, Alessandra Sala, Christo Wilson, Haitao Zheng, and Ben Y. Zhao. Orion: Shortest path estimation for large social graphs. In *Proceedi ngs of the 3rd Workshop on Online Social Networks (WOSN 2010)*, 2010.

[42] Xiaohan Zhao, Alessandra Sala, Haitao Zheng, and Ben Y. Zhao. Efficient shortest paths on massive social graphs. In *Proceedings of 7th Internation al Conference on Collaborative Computing:*

Networking, Applications and Worksharing (CollaborateCom), 2011.