**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。

# 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*≤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。

# 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* + 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*≥1，标签为*O*（*n1* */ k* log2*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* ∈ *Lout*（*u*） ∩ *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。

**Approximate Shortest Path Computation in Social Networks:** A few recent studies aim to compute the shortest path in large social networks. They extend the distance-labeling or the landmarking approach to approximate the shortest paths. Gubichev *et al.* propose *Sketch*, which generalizes the distance oracle method [36, 9] to discover the shortest path (not only the distance) in large graphs [16]. They observe that the path lengths are small enough to be considered as almost constant and therefore store a set of precomputed shortest path in addition to the distance labeling. They also propose several improvements, such as *cycle elimination* (SketchCE) and *tree-based search* (TreeSketch), to boost the shortest path estimation accuracy. Zhao *et al.* [42] develop *RigelPath* to approximate the shortest path in social networks on top of their distance estimation method, Rigel. Their basic idea is to use the distance estimation to help determine the search direction and prune search space. Sketch is the fastest approximate shortest path method, though RigelPath and TreeSketch can be more accurate. In addition, RigelPath mainly focuses on the undirected graph, while Sketch can handle both directed and undirected graphs.

**Other Recent Progress on Shortest Path Computation:** Very recently, there have been a few studies in the database research community on shortest path and distance computation. In [40], Wei develops a tree decomposition indexing structure to find the shortest paths in an unweighted undirected graph; In [5], a hierarchical vertex-cover based approach is developed for single-source on-disk shortest path (distance) computation. In [6], Cheng *et al.* introduce *k*-reach problem which provides binary answer to whether two vertices are connected by *k* steps. Also, the *k*-reach indexing approach developed in [6] is not scalable and can only handle small graphs (as it tries to materializes the vertex pairs within certain distance threshold). Finally, Jin *et al.* [19] propose a highway-centric labeling (HCL) scheme to efficiently compute distance in sparse graphs. Leveraging highway structure, this distance labeling offers a more compact index size compared to the state-of-the-art 2-hop labeling, and is also able to provide both exact and approximate distance with bounded accuracy. However, it is hard to scale to large social networks as real social networks are generally not sparse and potentially lead to expensive index construction cost and large index size.

# HUB-ACCELERATOR FRAMEWORK

In this section, we give an overview of the Hub-Accelerator (HA) framework for the shortest path computation. In the earlier discussion, we observe a love-hate relationship between shortest-path and hubs: on one hand, any shortest paths likely contain some hubs and thus need to be visited in the shortest path search process; on the other hand, in order to provide the fast shortest path search, we need to try to avoid a full expansion of hub nodes. We note that in general, the notation of hubs is rather informal though generally based on degree; in this paper, we simply refer to the set of vertices whose degree are the highest (top *β* number of vertices; *β* is a constant and can be specified).

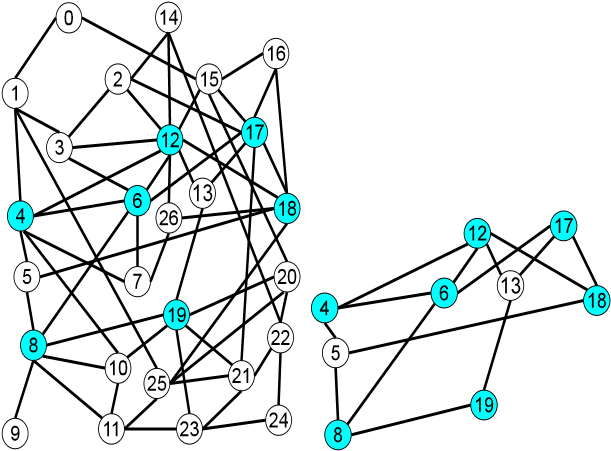
The design of Hub-Accelerator aims to utilize these hubs for shortest-path computation without fully expanding their neighborhoods. To achieve this, the following research questions need to answered:

1. How we can limit the expansion of hubs during the shortest path search? A hub may have thousands or even millions of connections (neighbors); what neighbors should be considered to be essential and given high priority in the shortest path search? To address this question, we formulate the *hub-network* notation, which captures a high-level view of the shortest path and topology between these hubs. The hub-network can be considered a highway structure anchored by hubs for routing the shortest paths in a massive social network. Due to the importance of hubs, most shortest paths between non-hub vertex pairs may need go through such a network, i.e., the starting vertex reaches a hub (as the highway entry), then travels to another hub (as the highway exit), and finally leaves the highway reaching the destination. In other words, the hub-network can be used to limit (or prioritize) the neighbors of hubs; a hub should only expand within the hub-network.
2. How we can effectively and efficiently utilize the hub-network for shortest path search? Note that the hub-network captures the shortest paths between hubs. However, not all shortest paths between vertices need to go through the hub-network: they may not contain any hub or they may consist of only one hub (in the later case, no traversal may be needed in the hub network). Thus, the problem is how we can extend the typical bidirectional BFS to adopt the hub-network for speeding up the shortest path computation?
3. Can we completely avoid the expansion of hubs? In this way, even the hub-network becomes unnecessary. But what essential information should be precomputed? When the number of hubs is not large, say 10*K*, then the pair-wise distance matrix between hubs may be materialized. For 10*K* hubs, this only costs about 100*MB* = 10*K* × 10*Kb* (assuming the distance can be held in 8 bits), but additional memory may be needed to recover the shortest path. Given this, how can bidirectional search take advantage of such a matrix and what other information may also need to be precomputed?

In this work, by investigating and solving these problems, we are able to utilize the hubs effectively to accelerate the shortest path search while significantly reducing or avoiding the cost of expanding them. Specifically, we make the following contributions: **Hub-Network Discovery (Section 4):** The concept of hub-network is at the heart of the Hub-Accelerator framework: given a collection of hubs, a *distance-preserving subgraph* seeks to extract a minimal number of additional vertices and edges from the original graphs so that the distance (and shortest path) between hubs can be recovered, i.e., their distances in the hub-network are equivalent to their distances in the original graph. As we mentioned before, the hubnetwork serves as the highway in the transportation system to enable the acceleration of the shortest path search: any hub will not be fully expanded (in the original graph); instead, only their neighbors in the hub networks will be expanded. Interestingly, though the discovery of a distance-preserving subgraph (and hub-network) seems rather intuitive, the computational aspect of the problem has not been studied before (despite similar notions being defined in theoretical graph theory [10]). In Section 4, we show the NP-hardness of discovering the minimal distance-preserving subgraph and we develop a fast greedy approach to extract the hub-network (and the distance-preserving subgraph). Our experimental study shows the degree of hubs in the hub-network is significantly lower than that in the original graph; thus the hub-network can limit the expansion of hubs and enables faster shortest path computation.

**Hub-Network based Bidirectional BFS (Section 5)** As we mentioned above, it is nontrivial to incorporate the hub-network into the bi-directional BFS. In general, if we use the hub-network and also expand the hubs within the network, then the searches in both directions cannot simply be stopped when they meet at a common vertex. This is because the hub-network does not capture those shortest paths consisting of only one hub.

**Hub**2**-Labeling (Section 6):** In this technique, we further push the speed boundary for shortest path computation by completely avoiding expanding any hub. To achieve this, a more expensive though often affordable precomputation and memory cost is used for faster online search. It consists of three basic elements: 1) First, instead of extracting and searching the hub-network, this technique mate-



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

Figure 1: Running Example of Hub-Accelerator Framework

rializes the distance matrix of those hubs, referred to as the *Hub*2 matrix. As we mentioned before, even for 10*K* hubs, the matrix can be rather easily materialized. 2) *Hub-Labeling* is introduced so that each vertex will precompute and materialize a small number of hubs (referred to as the core-hubs) which are essential for recovering the shortest path using hubs and hub-matrix. 3) Given the Hub2 distance matrix and hub-labeling, a faster bidirectional BFS can be performed to discover the exact *k*-degree shortest path. It first estimates a distance upper bound using the distance matrix and the hub labeling. No hub needs to be expanded during the bidirectional search, i.e., hub-pruning bidirectional BFS.

# HUB-NETWORK DISCOVERY

In this section, we formally define the *Hub-Network* (Subsection 4.1) and present an efficient approach to discover it (Subsection 4.2).

To facilitate our discussion, we first introduce the following notation. Let *G* = (*V,E*) be a graph where *V* = {1*,*2*,...n*} is the vertex set and *E* ⊆ *V* × *V* is the edge set. The edge from vertex *u* and *v* is denoted by (*u,v*), and we use *P*(*v*0*,vp*) = (*v*0*,v*1*,...,vp*) to denote a simple path between *v*0 and *vp*. The length of a simple path is the number of edges in the path, denoted by |*P*(*v*0*,vp*)|. Given two vertices *u* and *v*, their shortest path *SP*(*u,v*) is the path between them with the minimal length. The distance from vertex *u* to *v* is the length of shortest path *SP*(*u,v*) between *u* and *v*, denoted by *d*(*u,v*). Note that for a directed graph, the edge set may contain either (*u,v*), (*v,u*), or both. For an undirected graph, the edge has no direction; in other words, it can be considered bidirectional, so the edge set contains either both edges (*u,v*) and (*v,u*) or neither of them. In undirected graph, the shortest path distance from *u* to *v* is equivalent to the one from *v* to *u*, i.e., *d*(*u,v*) = *d*(*v,u*). The techniques discussed in the paper can be applied both undirected and directed graph; for simplicity, we will focus on the undirected graph and we will briefly mention how each technique can be naturally extended to handle directed graphs.

## Distance-Preserving Subgraph and HubNetwork

Intuitively, a hub-network is a minimal subgraph of the original *G*, such that at least one shortest path between two hubs can be recovered in the subgraph (the distance is preserved). To formally define the hub-network, we first introduce the concept of *distancepreserving subgraph* and its discovery.

DEFINITION 1. **Distance-Preserving Subgraph** *Given graph G* = (*V,E*) *and a set of vertex pairs D* = (*u,v*) ⊆ *V* × *V , a distance-preserving subgraph Gs* = (*Vs,Es*) *of G (Vs* ⊆ *V and Es* ⊆ *E) has the following property: for any* (*u,v*) ∈ *D, d*(*u,v*|*Gs*) = *d*(*u,v*|*G*)*, where d*(*u,v*|*Gs*) *and d*(*u,v*|*G*) *are the distances in subgraph Gs and original graph G, respectively.*

Given a collection of vertex pairs whose distance need to be preserved in the subgraph, the subgraph discovery problem aims to identify a *minimal* subgraph in terms of the number of vertices (or edges).

DEFINITION 2. **Minimal Distance-Preserving Subgraph (MDPS) Problem** *Given graph G* = (*V,E*) *and a set of vertex pairs D* = (*u,v*) ⊆ *V* × *V , the minimal distance-preserving subgraph (MDPS) problem aims to discover a minimal subgraph G⋆s* = (*Vs⋆,Es⋆*) *with the smallest number of vertices, i.e., G⋆s* = argmin|*Vs*| *Gs, where Gs* = (*Vs,Es*) *is a distance-preserving subgraph with respect to D.*

Once all the vertices *Vs⋆* are discovered, the induced subgraph *G*[*Vs⋆*] of *G* is a candidate minimal subgraph. Note that its edge set may be further sparsified. However, the edge sparsification problem with respect to a collection of vertex pairs (equivalent to the minimal distance-preserving subgraph problem in terms of the number of edges) is equally difficult as the MDPS problem (see discussion below); and the number of edges which can be removed are typically small in the unweighted graph. Thus, we will not explore the further edge reduction in this work.

Given graph *G* = (*V,E*) and a set of hubs *H* ⊆ *V* , let *Dk* contain all the hub pairs whose distance is no greater than *k*, then *the hub-network is defined as the minimal distance-preserving subgraph of Dk in G*.

EXAMPLE 4.1. *Figure 1(a) shows the network we will use as a running example. Figure 1(b) is the corresponding hub-network with H* = {4*,*6*,*8*,*12*,*17*,*18*,*19} *(degree* ≥ 5*) when k* = 4*. Since the pairwise distances between these hubs are all less than* 4*, D*4 *contains all the hub pairs with a total of* 15 *vertex pairs.*

Note that an alternative approach is to build the weighted hubnetwork which explicitly connects the hub pairs: for instance, if any other hub lies in a shortest paths between two hubs, an edge can be added to directly link them. Indeed, most of the existing studies have adopted a similar approach to build and utilize some *highway structure* (but they target mainly road networks, which are rather sparse). However, this approach can lead to a number of problems when searching a massive social network: 1) Such hub-network would be weighted and could be dense (many new edges may need to be added between hubs) and to search through it, Dijkstra’s algorithm (or its variant) must be utilized and would be slower than BFS (because of using the priority queue). Higher edge density exacerbates this slowdown. 2) Bidirectional BFS is typically used to search an unweighted network and could be adopted to search the remaining network (excluding the hub-network). However, combining bidirectional BFS with Dijkstra’s can be rather difficult; 3) Significant memory may be needed to record such a hub-network as it is rather dense. Moreover, to recover the shortest path, additional information has to be recorded for each added new edge. Considering these issues, we utilize the distance-preserving subgraph as the hub-network, which does not induce additional memory cost, and can naturally support (bidirectional) BFS. Note that in Sections 5 and 6, we will study how to use more memory for higher query performance (without involving the difficulty of weighted hub-network).

To discover the hub-network in a massive social network, we need a fast solution for the Minimal Distance-Preserving Subgraph (MDPS) problem. However, finding the exact optimal solution is hard.

THEOREM 1. *Finding the minimal distance-preserving subgraph of a collection D of vertex pairs in a graph is an NP-hard problem.*

**Proof Sketch:** We reduce the set-cover decision problem r to the decision version of the minimal distance-preserving subgraph problem. In the set-cover decision problem, let U be the ground set and C records all the candidate sets, where for any candidate set *C* ∈ C and *C* ⊆ U. The set-cover decision problem asks whether there are *K* or fewer candidate sets in C, such that ∪*iCi* = U.

Now we construct the following MDPS instance based on a set cover instance: consider a tripartite graph *G* = (*X*∪*Y* ∪*Z,EXY* ∪ *EYZ*) where the vertices in *X* and *Z* have one-to-one correspondence to the elements in the ground set U, and the vertices in *Y*

one-to-one correspond to the candidate sets in C. For simplicity, let *u* ∈ U ↔ *xu* ∈ *X*(*zu* ∈ *Z*) (vertex *xu* (*zu*) corresponds to element *u*); and let *C* ∈ C ↔ *yC* ∈ *Y* (vertex *yC* corresponds to candidate set *C*). Then, the edge set *EXY* (*EYZ*) contains all the edges (*xu,yC*) ((*yC,zu*)) if element *u* belongs to the candidate set *C*. Note that the tripartite graph can be considered symmetric

(*X* ≡ *Z* and *EXY* ≡ *EYZ*).

We claim that the set-cover decision problem is satisfiable if and only if the following MDPS problem is true: there is a subgraph *G* with 2|*U*| + *K* vertices to cover the shortest path distance of |*U*| vertex pairs (*xu,zu*), *u* ∈ *U*.

The proof of this claim is as follows. Assume the set-cover problem is satisfiable, let *C*1*,*···*Ck*(*k* ≤ *K*) be the *k* candidate sets which covers the ground set, i.e., ∪*Ci* = U. Let *YC* include all the vertices in *Y* corresponding to *C*1*,*··· *,Ck*. It is easy to observe the induced subgraph of *G*[*X* ∪ *YC* ∪ *Z*] can recover the distances of all |*U*| pairs (*xu,zu*), *u* ∈ *U*. Note that their distances in the original graph *G* and the induced subgraph *G*[*X* ∪*YC* ∪ *Z*] are all equal to 2.

From the other direction, let *Gs* be the subgraph with 2|*U*| + *K* vertices which recovers the distances of these |*U*| pairs. Since the vertices in the pairs have to be included in the subgraph (otherwise, the distance can not be explicitly recovered), the additional *K* vertices can only come from the vertex set *Y* (there are 2|*U*| in the vertex pairs from *X* and *Z*). Note that the distance of (*xu,zu*) in the original graph is 2 and to recover that, a vertex *yC* in *Y* has to appear in the subgraph so that both (*xu,yC*) and (*yC,zu*) are in the subgraph (and in the original graph). This indicates the corresponding candidate set *C* covers element *u*. Since there are at most *K* vertices in *Y* , there are at most *K* candidates needed to cover all

the ground set *U*. ✷

Based on similar reduction, we can also prove that finding the minimal distance-preserving subgraph in terms the number of the edges is also an NP-hard problem. Due to simplicity, we will not further explore this alternative in the paper.

## Algorithm for Hub-Network Discovery

In the subsection, we will discuss an efficient approach for discovering the distance-preserving subgraph and the hub-network. To simplify our discussion, we focus on extracting the hub-network, though the approach is directly applicable to any collection of vertex pairs (and thus the general distance-preserving subgraph). Recall that in the hub-network discovery problem, given a set *H* of hubs and a collection *D* of hub-pairs whose distance is no more than *k* (for *k*-degree shortest path search), then the goal is to recover the distance for the pairs in *D* using a minimal (distancepreserving) subgraph.

To tackle the hub-network (and the distance-preserving subgraph) efficiently, we make the following simple observation. For any vertex pairs (*x,y*) in *D*, if there is another hub *z*, such that *d*(*x,y*) = *d*(*x,z*) + *d*(*z,y*), then we refer to the vertex pair (*x,y*) as a *composite pair*; otherwise, it is a *basic pair*, i.e., any shortest path connecting *x* and *y* does not contain a hub in *H*. Let *Db* ⊆ *D* be the set of basic pairs. Given this, it is easy to see that *if a subgraph can recover all the vertex pairs in Db, then it is a distance-preserving subgraph of D (and thus the hub-network)*. This indicates that we only need to focus on the basic pairs (*Db*) as the distances of composite pairs can be directly recovered using the paths between basic pairs.

Considering this, at the high level, the algorithm of the hubnetwork discovery performs a BFS-type traversal from each hub *h* and it accomplishes the two tasks: 1) during the BFS, all basic pairs including *h*, i.e., (*h,v*), *v* ∈ *H*, should be recognized and collected; and 2) once a basic pair (*h,v*) is identified, the algorithm will select a “good” shortest path which consists of the minimal number of “new” vertices (not included in the hub-network yet). In other words, as we traverse the graph from each hub, we gradually augment the hub-network with new vertices to recover the distance (shortest path) of the newly found basic pairs.

**Recognizing basic pairs:** To quickly determine whether the (*h,v*) is a basic pair during the BFS process starting from hub *h*, we utilize the following observation: *Let vertex y lie on a shortest path from hub h to non-hub vertex v with distance d*(*h,v*) − 1 *(i.e., y is one hop closer than v with respect to h). If there is a hub h*′ *appearing in a shortest path from h to y (h*′ *and y may not be distinct), h*′ *definitely lies on a shortest path from h to v and* (*h,v*) *is a composite pair (not basic pair).* Based on this observation, we simply maintain a binary flag *b*(*v*) to denote whether there is another hub appearing in a shortest path between *h* and *v*. Specifically, its update rule is as follows: *b*(*v*) = 0 (not basic pair) if *v* itself is a hub or *b*(*y*) = 0 (*y* is *v*’s parent in the BFS, i.e., *d*(*h,y*) = *d*(*h,v*) − 1 and *d*(*y,v*) = 1). Thus, during the BFS traversal, when we visit vertex *v*, if its flag *b*(*v*) = 1 (true) meaning there is no other hubs lying on the shortest path between *h* and *v* and we are able to recognize it is a basic pair.

**Selecting a “good” shortest path between basic pairs:** To select a good shortest path between basic pairs *h* and *v*, a basic measurement is the number of “new vertices” that need to be added to the hub-network. As a greedy criterion, the fewer that need to be added, the better is the path. To compute this, for any shortest path from starting point *h* to *v*, a score *f* records the maximal number of vertices which are already in the hub-network. This measure can be easily maintained incrementally. Simply speaking, its update rule is as follows: *f*(*v*) = *maxf*(*u*) + 1 if *v* itself is in the hubnetwork or *f*(*v*) = *maxf*(*u*), where *u* is *v*’s parent in the BFS (a shortest path from *h* to *v* go through *u* and *u* directly links to *v*). Also vertex *v* records *u* which has the maximal *f* for tracking such a shortest path (with maximal number of vertices in the hubnetwork). Finally, we note that only for vertices *v* with *b*(*v*) = 1, i.e., when the shortest path between *h* and *v* does not go through any other hub, does a score *f* need to be maintained. Otherwise, *v* and its descendents cannot produce any basic pairs.

**Overall Algorithm:** The outline of this BFS-based procedure for discovering the hub-network is described in Algorithm 1. Here *H⋆* is the set recording the vertices in the hub-network. Initially, *H⋆* = *H* and then new vertices will be added during the processing. Note that in the queue for BFS traversal (Line 3), we always visit those vertices with *b*(*u*) = 0, i.e., they and any of their descendents (in the BFS traversal) will not form a basic pair, and thus the score *f* does not need to be maintained for them. Once a hub is visited and it initially has *b*(*u*) = 1, then (*h,u*) is a basic pair (Line 5); we will extract the shortest path which has the maximal number of vertices in the hub-network and add the new vertices to *H⋆* (Line 6). Now, since the descendent of this hub (in the BFS traversal) will not form a basic pair, we simply change its flag to false, i.e., *b*(*u*) = 0 (Line **Algorithm 1** BFSExtraction(*G* = (*V,E*), *h*, *H*, *H⋆*)

|  |  |
| --- | --- |
| 1: Initialize *b*(*u*) ← 1; *f*(*u*) ← 0 for each vertex *u*;  2: *level*(*h*) ← 0; *Q* ← {*h*} {queue for BFS};  3: **while** *Q* 6= ∅ {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 level*k* for *k*-degree shortest 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** | |

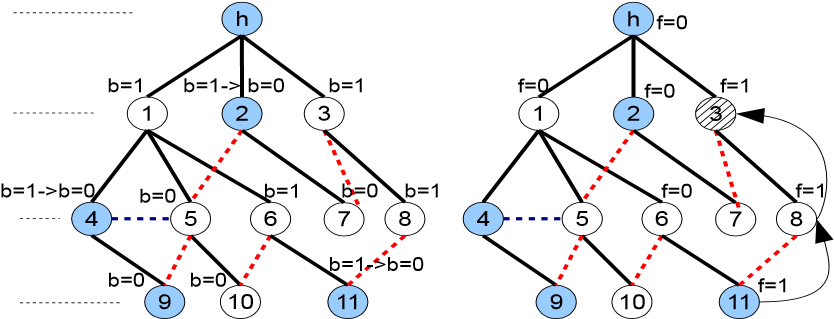


Figure 2: Incremental Maintenance of flag *b* and score *f*

7). Also, since we are only interested in the shortest path within *k*-hop, we will not expand any vertex with distance to *h* to be *k*

(Lines 9−11). Before we expand the neighbors of *u*, we also need to update its *f* score based on whether *u* itself is in the hub-network (Line 12 − 14).

The complete expansion of a vertex *u* is from Line 15 to 28. We will visit each of its neighbors *v*. If *v* has not been visited, we will add it to the queue for future visiting (Line 16 − 18). Then we perform the incremental update of flag *b*(*v*) and score *f*(*v*). Flag *b*(*v*) will be turned off if *b*(*u*) = 0 (Line 20 − 22) and if *f*(*u*) is larger than *f*(*v*), i.e., the shortest path from *h* to *u* has the largest number of vertices so far in the hub-network. Vertex *v* will record *u* as the parent (for shortest path tracking) and *f*(*v*) is updated (Line 24 − 26). This procedure will be invoked for each hub in *H*.

EXAMPLE 4.2. *Figure 2 illustrates the flag b and score f in the BFS process. Here the vertices h,* 2*,* 4*,* 9*, and* 11 *are hubs. In Figure 2 (a),* (*h,*2)*,* (*h,*4)*, and* (*h,*11) *are basic pairs; the flag b changes from b* = 1 *originally to b* = 0 *(Lines* 5*-*7*). After the flag b of* 2*,*4*, and* 11 *changes to false (b* = 0)*, all their descendents in the BFS traversal become false. For instance, the flag b of vertex* 5 *is false as it is also considered hub* 2*’s descendent. In Figure 2(b), the shaded vertex* 3 *indicates it is already included in the hub-network (*3 ∈ *H⋆). Therefore, vertex* 11 *points to vertex* 8 *(parent(11)=8 and parent(8)=3) as its f score is higher than the that of vertex* 6*.*

THEOREM 2. *If we invoke Algorithm 1 for each h* ∈ *H, then the induced subgraph G*[*H⋆*] *is a hub-network of H with respect to the k-degree shortest path.*

**Proof Sketch:** The correctness of the algorithm can be derived from the following two observations: 1) for any basic pair (*h,u*) with distance no more than *k*, there is at least one shortest path in *G*[*H⋆*] as the algorithm explicitly extracts a shortest path and adds all its vertices to *H⋆*; 2) for any composite pair (*h,h*′) with distance no more than *k*, then it can always be represented as a sequence of basic pairs, which has at least one shortest path in *G*[*H⋆*]. Thus, for any hub pair (*h,h*′) with distance no more than *k*, their distance (at least one shortest path) is preserved in the induced subgraph *G*[*H⋆*]. ✷

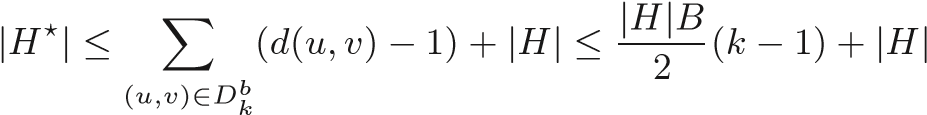
The computational complexity for hub-network discovery as described in Algorithm 1 is basically equivalent to that of a simple BFS procedure. The overall procedure takes *O*(P*h*∈*H*(|*Nk*(*h*)|+|*Ek*(*h*)|)) time, where *H* is the hub set, and *Nk*(*h*) and *Ek*(*h*) are the number of vertices and edges, respectively, in *u*’s k-degree neighborhood. We also note that this algorithm works correctly for both undirected and directed graphs. Interestingly, we note the following property of applying Algorithm 1 for an undirected graph.

LEMMA 1. *Let* (*u,v*) *be a basic hub pair in an undirected graph. Consider Algorithm 1 performs BFS from u first and it discovers the shortest path SP*(*u,v*)*. When it performs BFS from v and discovers the symmetric basic pair* (*v,u*)*, the algorithm will not add any additional new vertices.*

**Proof Sketch:** The score *f* guarantees *f*(*v*) = |*SP*(*v,u*)|=|*SP*(*u,v*)| and thus a shortest path as “good” as *SP*(*u,v*) will be extracted which does not need to add any new vertices to *H⋆*. ✷

This observation leads to the simple bound constraint of the hubnetwork (the final size of *H⋆*) and the result of Algorithm 1 will match such a bound.

LEMMA 2. *Let Dkb* ⊆ *Dk* ⊆ *H* × *H be the set of all unique basic hub pairs whose distance is no more than k, then,*

*,*

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

**Proof Sketch:** The term P(*u,v*)∈*Dkb*(*d*(*u,v*) − 1) corresponds to the definition that any basic pair needs to recover only one shortest path; this also corresponds to the worst case scenario in Algorithm 1, where for any basic pair, all non-hub vertices along a new shortest path need to be added to *H⋆*. Note that for undirected graph *Dkb* treats basic pairs (*u,v*) and (*v,u*) as a single one. This directly leads to the term |*H*|*B/*2(*k*−1), which contemplates the maximal distance between any basic hub pair is *k* and only one shortest path needs to be recovered for symmetric basic pairs (*u,v*) and (*v,u*). Algorithm 1 also holds that (Lemma 1). Note that the result holds for directed graph as well where *B* is the total degree of both incoming and outgoing edges. ✷

# HUB-NETWORK BASED SEARCH

In this section, we describe the hub-network based bidirectional BFS. The main challenge here is given a hub-network, how we can leverage it to maximally reduce the expansion of hubs and still guarantee to discover the correct *k*-degree shortest path? Recall that a key reason for introducing the hub-network is to use it to constraint the expansion of hubs. Thus, a basic search principle

is that *any hub will only visit its neighbors in the hub-network*. But what about any non-hub vertices *v* in the hub-network, such as *v* ∈ *H⋆* \ *H*? Should they be expanded only within the hubnetwork or should they be treated as the remaining vertices outside the hub-network? Furthermore, in traditional bidirectional BFS, when two searches (forward and backward) meet for the first time, the shortest path is discovered. Unfortunately, this does not necessarily hold if the hub is not fully expanded and thus the question becomes: what should be the correct stop condition? The stop condition is crucial as it determines the search space and the correctness of discovering the exact shortest path.

In the following, we first describe the hub-network based bidirectional BFS algorithm (Subsection 5.1) and then we prove its correctness and discuss its search cost (Subsection 5.2).

## HN-BBFS Algorithm

The Hub-Network based Bidirectional BFS (HN-BBFS) algorithm consists of a two-step process: 1) (**Meeting step**) A bidirectional search will traverse both hub-network and remaining graphs until the forward and backward searches meet at the first common vertex; 2) (**Verification step**) Next, the searches continues in the remaining graphs (not hub-network) to verify whether the path discovered in the first step is shortest. If not, this step will discover an alternative shortest path.

**Expansion Rule:** In the Meeting step, the forward (backward) BFS follows the following rules to expand vertex *v* in *G*: 1) if a vertex is a hub, then it only expands its neighbors in the hub-network; 2) if a vertex is a regular vertex (not in the hub-network), then it expands all its neighbors; 3) for a vertex is a non-hub vertex but in the hubnetwork, *H⋆*\*H*, if the BFS traversal first reaches it through a hub, then it only expands its neighbors in the hub-network; otherwise, it is considered a regular vertex (no shortest path from start (end) vertex to it going through a hub). In the Verification step, both forward and backward BFS traversals will continue but they will not need to expand any hub, and any regular vertex and non-hub vertices in the hub-network will expand all their neighbors in the entire network.

**Stop Condition:** The stop condition for the forward (backward) BFS in the Verification step is as follows. Let *dist* be the shortest path distance discovered so far; let *dhs* (*dht* ) be the distance between *s* (*h*) to its closest hub *h*; let *levelf* (*levelb*) be the current level being traversed by forward (backward) BFS. Then, the forward (backward) BFS will stop when the following condition is met:

*dist* ≥ *levelf* + *dhs* + 1 (*dist* ≥ *levelb* + *dht* + 1) (1)

**Overall Algorithm:** Hub-Network based Bidirectional BFS (HNBBFS) is sketched in Algorithm 2. Note that *BackwardSearch* is essentially the same as *ForwardSearch* and is omitted for simplicity. Initially, *dist* is set to be *k*+1 for *k*-degree shortest path search (indicating no path within *k*-hops) and the *met* condition is false (Line 2).

The first step (Meeting Step) is carried out by the first while loop (Lines 3 − 6), where a forward search and a backward search are employed in an alternating manner. In *ForwardSearch* (and *BackwardSearch*), a vertex in the corresponding queue *Qf* (*Qb*) is expanded. The expansion rule as described earlier is used in Line 15. Basically, if a vertex is a hub or is in the hub-network, *H⋆* \*H*, but the BFS traversal first reaches it through a hub (there is a shortest path from *s* to *u* via a hub), it is considered “in-hub-network”. Otherwise, it is “out-hub-network”. For an in-hub-network vertex, BFS only expands its neighbors in the hub-network. Note that recognizing these “in-hub-network” vertices is straightforward and can be **Algorithm 2** HN-BBFS(*G*, *G*[*H⋆*], *s*, *t*)

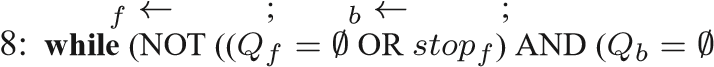


1: *Qf* ← {*s*}; *Qb* ← {*t*}; {Queues for forward and backward search} 2: *dist* ← *k* +1; *met* ← *false*;

3: **while** (*Qf* 6= ∅ AND *Qb* 6= ∅) AND NOT *met* AND *d*(*s,Qf.top*)+ *d*(*Qb.top,t*) *< dist* **do**

4: ForwardSearch(*Qf,false*); {not Verification Step} 5: BackwardSearch(*Qb,false*);

6: **end while**

7: *stop false stop false*

OR *stopb*))) **do**

9: NOT *stopf*: ForwardSearch(*Qf,true*); {true: Verification Step}

10: NOT *stopb*: BackwardSearch(*Qb,true*)

11: **end while**

12: **return** *dist* and shortest path;

**Procedure** ForwardSearch(*Qf* ,*V erification*)

13: *u* ← *Qf.pop*() {if *V erification* is true, only out-hub-network vertices will be visited}

14: *u* is set to be visited by forward search;

15: **for all** *v* ← *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* ← *true* |
| 21: | **end if** |
| 22: | **end if** |
| 23: | **end if** |
| 24: | **if** *v* is not visited AND NOT (*V erification* and *v* ∈ *H*) **then** |
| 25: | *Qf.push back*(*v*); |
| 26: | **end if** |
| 27: | **if** *V erification* AND *dist* ≥ *d*(*s,v*)+ *dht* +1 **then** |
| 28: | *stopf* ← *true*; |
| 29: | **end if** |

30: **end for**



incrementally computed (similar to using the flag *b* in Algorithm 1). Once a forward (backward) search visits a vertex already visited by the backward (forward) search, a candidate shortest path is discovered and *met* is set to be true. Note that when *V erification* is false (at the first step), every vertex (both hubs and non-hubs) will be visited and expanded.

Once *met* turns true, the second step (Verification Step) is carried out by the second while loop (Lines 8−11). Before the forward stop condition is met (*stopf* is false), the ForwardSearch will continue. However, only out-hub-network vertices will be visited and expanded (Line 13 and Lines 24−26). Also, during the expansion process, the candidate shortest path can be updated (Lines 17−19). Finally, when the stop condition is met (Line 26: *d*(*s,v*) is the current BFS level being expanded, thus *levelf*), *stopf* will become true and no forward search will not performed (Line 9). Note that *dhs* (*dht* ) can be easily computed during the BFS traversal: the first time a hub is visited, its distance to *s* is recorded as *dhs*.

## Correctness and Search Cost

We now discuss the correctness of HN-BBFS (Algorithm 2) and then its search cost (especially in terms of the new Stop condition, Formula 1). To prove the correctness of HN-BBFS, we will make the following important observations:

LEMMA 3. *For any hub h* ∈ *H, during the first step (Meeting Step), the distance d*(*s,h*) *computed using the forward BFS search, i.e., the number of traversal levels to reach h, is the exact shortest path distance between s and h. The same holds for d*(*h,t*) *for the backward BFS traversal.*

**Proof Sketch:** If *s* is a hub, then based on the hub-network definition, this clearly holds. If *s* is not a hub, then one of the following two cases must hold: 1) All shortest paths between (*s,h*) do not contain a hub except *h*, so the forward BFS finds the shortest path distance *d*(*s,h*) by traversing only non-hub vertices in the original graph; 2) There is a shortest path between (*s,h*) containing another hub, so there is always *h*′, such that (*s,h*′) does not contain any hubs and (*h*′*,h*) can be discovered in the hub-network. ✷

Lemma 3 demonstrates the power of the hub-network and shows that HN-BBFS can correctly calculate the shortest path (distance) between query vertices to hubs (and between hubs). However, despite this, the candidate shortest path being discovered at the first meeting vertex may not be the exact one. The following lemma categorizes the exact shortest paths if they are shorter than the candidate shortest path discovered in the first step (Meeting Step).

LEMMA 4. *Assuming u is the meeting vertex where forward and backward search first meet (Lines* 22 − 26 *in Algorithm 2), the candidate shortest path is denoted as SP*(*s,u,t*) *and the distance dist is d*(*s,u*) + *d*(*u,t*)*. If there is a shorter path, then it must contain a hub h, such that the exact shortest path can be represented as two segments SP*(*s,h*) *and SP*(*h,t*)*. Moreover, either 1) SP*(*s,h*) *contains no hub other than h with distances d*(*s,h*) ≥ *d*(*s,u*) *and d*(*h,t*) *< d*(*u,t*)*, or 2) SP*(*h,t*) *contains no hub other than h with distances d*(*s,h*) *< d*(*u,t*) *and d*(*h,t*) ≥ *d*(*u,t*)*.*

**Proof Sketch:** We prove this by way to contradiction. It the lemma does not hold, then the following two types of paths cannot be shorter than the discovered candidate shortest path: 1) there is no hub in the exact shortest path *SP*(*s,t*), and 2) there are two hubs *hs* and *ht*, such that the shortest path has three segments: *SP*(*s,hs*), *SP*(*hs,ht*) and *SP*(*ht,t*) where *d*(*s,hs*) *< d*(*s,u*) and *d*(*ht,t*) *< d*(*u,t*). For the first case, the bidirectional BFS should be able to find such a path (if they are shorter than the candidate *SP*(*s,u,t*)) earlier as it only involves visiting non-hub vertices in the graph. For the second case, based on Lemma 3, Algorithm 2 computes the exact *d*(*s,hs*) and *d*(*ht,t*) before the two BFS met at *u* and the hub-network encodes the correct distance between *d*(*hs,ht*). Thus, if *d*(*s,hs*) + *d*(*hs,ht*) + *d*(*ht,t*) *< d*(*s,u*)+*d*(*u,t*), this shortest path should be discovered (met at an in-hub-network vertex) during the first step (Meeting Step). Since both cases are impossible, the lemma holds. ✷

THEOREM 3. *The Hub-Network based Bidirectional BFS approach (HN-BBFS, Algorithm 2) guarantees the discovery of the exact k-degree shortest path.*

**Proof Sketch:** Basically, we need show that when the stop condition is met, no shorter alternative paths exists. By Lemma 4, if a shortest path exists that is better than the candidate shortest path *SP*(*s,u,t*)), it must follow one of two simple formats. These formats suggest we only need to extend out-hub-network vertices until they meet a hub already visited from the other direction (*d*(*s,hs*) *< d*(*s,u*) or *d*(*ht,t*) *< d*(*u,t*)). If such a path can be found, it must be shorter than the already discovered distance *dist*, i.e., *dist > levelf* + 1 + *dht* (the best case situation is when the shortest path extends from the current step by one step to a hub closest to the query vertices). Clearly, if this does not hold, any shortest path in this format will not be smaller than *dist*. ✷

In classical Bidirectional search, once both directions meet at a common vertex, the search can be stopped and the exact shortest path is discovered. However, in HN-BBFS, in order to reduce the expansion of hubs, some additional traversal (Verification Step) has to be taken. Clearly, if we need to walk *k/*2 additional steps, then the benefit of HN-BBFS can be greatly compromised.

So, what is the average number of steps HN-BBFS needs to take for a typical (random) query in the Verification step? The number is close to *zero* or at most one. To illustrate, first consider the distance between two vertex pairs to be 6 (since most distances are less than that in social networks [39]), and assume *s* and *t* are not hubs (because there are few hubs) but each of them has a direct hubneighbor *dhs* = 1 (*dht* = 1). Note that both directions typically traverse at most three steps, i.e., *levelf* = *levelb* = 3. Thus, at most one extra step needs to be taken in this case to make the stop condition true: *dist*−*levelf* −*dht* −1 ≥ 0, where *levelf* = 4. Similarly, let us consider the distance to be 4 and assume each direction has taken 2 steps in the Meeting Step. In this case, there is no need to take an additional step (assuming *s* and *t* are not hubs), and we can immediately recognize that the candidate shortest path is indeed the exact one. Finally, we note that when *dist*−*levelf* −*dht* −1 = 1, i.e., the last step of BFS for Verification, there is no need to expand all the neighbors of a given vertex. Only its immediate hubneighbors need to be expanded and checked (Lemma 4 and Theorem 3). To facilitate this, the neighbors of regular vertices can be reorganized so that the hub-neighbors and non-hub-neighbors are separately recorded.

# HUB2-LABELING FOR SHORTEST PATH COMPUTATION

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.

## Hub2-Labeling Framework

Hub2-Labeling replaces the Hub-Network with a *Hub*2 distance matrix and *Hub Labeling*.

**Hub**2**:** The distance matrix between hub pairs (referred to as Hub2) is precomputed and stored in main memory. Indeed, only the distances of pairs with distance no more than *k* need to be computed for *k*-degree shortest path. As we discussed before, nowadays a desktop computer with moderate memory size can easily hold such a matrix for 10*K* (or more) of hubs.

**Hub Labeling:** In order to effectively utilize the distance matrix, each vertex *v* in the graph also records a small portion of hubs, referred to as the *core-hubs*, along with the distances. Basically, those core-hubs along with the distance matrix can help quickly estimate the upper-bound of distance between the query vertex pairs and can be used for bounding the search step of bidirectional BFS. Now, we formally define the *core-hubs*.

DEFINITION 3. **(Core-Hubs)** *Given graph G* = (*V,E*) *and a collection H of hubs, for each vertex v, we say vertex h* ∈ *H is a core-hub for v if there is no other hub h*′ ∈ *H such that d*(*v,h*) = *d*(*v,h*′) + *d*(*h*′*,h*)*. Formally, L*(*v*) = {*h* ∈ H : ∄*h*′ ∈ H*,d*(*v,h*) = *d*(*v,h*′) + *d*(*h*′*,h*)}*.*

Simply speaking, if no other vertex *h*′ appears in any shortest path between *v* and *h*, *h* is *v*’s core-hub. Note that a pair (*v,h*), where *v* ∈ *L*(*v*), is similar to a basic pair in the hub-network (Subsection 4.2). The original basic pair definition only refers to hub pairs, but here it is being extended to vertex pairs with one hub and one non-hub vertex.

EXAMPLE 6.1. *Figure 1(c) illustrate the core-hubs (along with the distance) for each non-hub vertices in the original graph (Figure 1(a)). Here the hubs are* 4*,* 6*,* 8*,* 12*,* 17*,* 18*, and* 19*. For instance, Vertex* 1 *only needs to record core-hubs* 4*,* 6*,* 12 *and* 19*, and it can reach hubs* 8 *and* 17 *through them in some shortest path.*

Using the core-hubs *L* and distance-matrix Hub2 , we can approximate the distance and the shortest path for vertex pair (*s,t*) in the following fashion:

*dH*(*s,t*) = *minx*∈*L*(*s*)∧*y*∈*L*(*t*){*d*(*s,x*) + *d*(*x,y*) + *d*(*y,t*)} (2)

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

The construction of the distance matrix Hub2 and the labeling of core-hubs are also rather straightforward. The BFS procedure in Algorithm 1 can be easily adopted: 1) each BFS performs *k* steps and thus the distance matrix can be directly constructed; 2) when a vertex *v* has flag *b* = 1 (basic pair) from BFS traversal of *h*, we simply append *h* to *L*(*v*). Thus, the total computational complexity of the pre-computation is *O*(P*h*∈*H*(*Nk*(*h*)+*Ek*(*h*))) time, where *H* is the hub set and *Nk*(*h*) and *Ek*(*h*) are the number of vertices and edges, respectively, in *u*’s k-degree neighborhood. We note that for directed graphs, we will compute both *Lin*(*v*) and *Lout*(*v*), one for incoming core-hubs (*h,v*) and the other for outgoing core-hubs (*v,h*). To construct such labels, we need perform both forward and backward BFS from each hub.

The overall memory cost of Hub2-Labeling is the sum of the cost of the distance matrix (Hub2) together with the core-hub labeling for each vertex (*L*(*v*)): P*v*∈*V O*(|*L*(*v*)|) + *O*(|*H*|2). This turns out to be rather affordable. In the experimental study, we found that for most of the real social networks, the core-hubs of each vertex *v* is only a small portion of the total hubs (in most case, less than or close to 2%). Thus, the Hub2-Labeling can easily handle graphs with more than 10*K* hubs. Furthermore, since the second term (the size of the distance matrix) is stable, as the number of vertices increases in the original graph, the first term will scale linearly with respect to |*V* |.

## Hub2-Labeling Query Processing

To compute the *k*-degree shortest path between vertex pair (*s,t*), the online query process in Hub2-Labeling consists of two steps: **Step 1 (Distance Estimation):** Using the distance matrix Hub2 and core-hubs labeling *L*(*s*) and *L*(*t*), the distance *dH*(*s,t*) is estimated (Formula 2).

**Step 2 (Hub-Pruning Bidirectional BFS (HP-BBFS)):** A bidirectional BFS from *s* and *t* is performed and the search step is constrained by the minimum between *k* (for *k*-degree shortest path) and *dH*(*s,t*). In particular, none of the hubs need to be expanded during the bidirectional search. Mathematically, the Hub-Pruning Bidirectional BFS is equivalent to performing a typical Bidirectional BFS on the non-hub induced subgraph, *G*[*V* \ *H*] of *G*.

THEOREM 4. *The two-step Hub*2*-Labeling query process can correctly compute the k-degree shortest path in graph G.*

**Proof Sketch:** We observe that any vertex pair with distance no more than *k* can be categorized as: 1) vertex pairs having at least one shortest path passing through at least one hub in *H*; and 2) vertex pairs whose shortest paths never pass through any hub.

For any vertex pair (*s,t*) with distance no greater than *k* (*d*(*s,t*) ≤

*k*), if there exists one hub *x*′ ∈ H satisfying *d*(*s,t*) = *d*(*s,x*′) + *d*(*x*′*,t*), then, we can always find *x* ∈ *LH*(*s*) and *y* ∈ *LH*(*t*) such that *d*(*s,t*) = *d*(*s,x*) + *d*(*x,y*) + *d*(*y,t*). In other words, Step 1 (distance estimation), which uses the distance-matrix Hub2 and core-hub labeling, can handle this category. Also, the Step 2 will help confirm the shortest path belongs to this category (cannot find a shorter one).

If an approximate shortest path computed in Step 1 is not an exact one, then the shortest path does not involve any hub. Thus Step 2 can guarantee to extract an exact shortest path using the bidirectional search in the non-hub induced subgraph *G*[*V* \*H*]. ✷

The time complexity of online query processing of a pair *s* and *t* can be written as *O*(|*L*(*s*)||*L*(*t*)| + *Nk/*2(*s*|*G*[*V* \*H*])+ *Ek/*2(*s*|*G*[*V* \ *H*])+*Nk/*′ 2(*t*|*G*[*V* \*H*])+ *Ek/*′ 2(*t*|*G*[*V* \*H*])), where |*L*(*s*)||*L*(*t*)| is the distance estimation cost and the remaining terms are the cost of bidirectional search. *Nk/*2 (*Nk/*′ 2) and *Ek/*2 (*Ek/*′ 2) are the number of vertices and edges in the *k/*2-neighborhood (reversed neighborhood which follows the incoming edges) of the non-hub induced subgraph *G*[*V* \ *H*]. Since the hubs are excluded, the cost of hubpruning bidirectional BFS is significantly smaller than that on the original graph.

However, if the number of core-labels is large, then the distance estimation can be expensive (a pairwise join on *L*(*s*) and *L*(*t*) is performed). To address this issue, the core-hubs in *L*(*u*) can be organized in a level-wise fashion, each level corresponding to their distance to *u*, such as *L*1(*u*)*,L*2(*u*)*,*···*Lk*(*u*). Using such a level-wise organization, we can perform a much more efficient distance estimation: the pairwise joins first performed between *L*1(*s*) and *L*1(*t*); then on (*L*1(*s*)*,L*2(*t*)), (*L*2(*s*)*,L*1(*t*)), (*L*2(*s*)*,L*2(*t*)), etc. Given this, let us denote *d* to be the shortest path length obtained by pairwise join so far. Assuming we are currently working on (*Lp*(*s*)*,Lq*(*t*)), if *d < p* + *q*, then we terminate the pairwise join immediately. This is because it is impossible for (*Lp*′(*s*)*,Lq*′(*t*)) to produce better results since *p*′+*q*′ ≥ *p*+*q > d*. This early termination strategy based on the level-wise organization can help us effectively prune unnecessary pairwise join operations and improve the query efficiency.

# EXPERIMENTAL EVALUATION

In this section, we empirically evaluate the performance of our algorithm on a range of large real social networks. In particular, we will compare the Hub-Network approach (denoted as **HN**) and Hub2-Labeling approach (denoted as **HL**) with the following methods: 1) basic breadth-first search (denoted as **BFS**); 2) bidirectional breadth-first search (denoted as **BiBFS**); 3) the Sketch algorithm [9] (denoted as **S***⋆*), the state-of-the-art approximate distance estimation algorithm; 4) the TreeSketch method [16] (denoted as **TS***⋆*), which utilizes a tree to improve the approximation accuracy of Sketch based shortest path computation. Here the symbol *⋆* also indicates it is an approximation method.

In addition, we have also tested the two latest exact shortest path distance methods, including tree decomposition based shortest path computation [40] and the highway-centric labeling approach [19] based on authors’ provided implementation. However, neither of them can work on the graphs used in this study. This is as expected as their indexing cost is very high (tree decomposition or set-cover approach) and they are mainly focusing on very sparse graphs.

We also tested RigelPath, another recent approach on approximate shortest path discovery in social networks [42]. However, its query performance is slower than that of Sketch (also confirmed in their own study [42]). Furthermore, its current implementation only focuses on undirected graphs, wheres most of the real benchmarking networks are directed. Thus, we do not report RigelPath’s experimental results here.

We implemented our algorithms in C++ and the Standard Template Library (STL).The implementation of sketch-based approaches (including **S**∗ and **TS**∗) is kindly provided by authors [16] (also implemented in C++). All experiments were run on a Linux server with 2.48GHz AMD Opteron processors and 32GB RAM.

In experiments, we are interested in two important measures: query time and preprocessing cost, which consists of precomputation time and indexing size. To measure the query time, we randomly generate 10*,*000 vertex pairs and obtain the average running time for each query. For the index size, since all Sketch indices are stored in RDF format, their indexing sizes are measured in terms of the corresponding RDF file size. If the preprocessing cannot be finished in 48 hours,we will stop it and record “-” in the table of results. Furthermore, we note that all Sketch-based benchmarks can only approximate shortest paths, where approximation accuracy is influenced by an iterative sampling procedure. A parameter *r* is specified to determine the number of sampling iterations, which leads to 2*r* log|*V* | sketches for each vertex. To make a fair comparison with exact query schemes, we set *r* = 2 as suggested in [16] which can produce sketches with good approximation accuracy and efficient query processing. Also, in this study, we focus on comparing their query time again the new approaches despite they are only able to provide approximate solution whereas our approaches can provide the exact solution.

The benchmarking datasets are listed in Table 1. Most of them are gathered from online social networks, with the number of vertices ranging from several tens of thousands to more than 10 million. Others also exhibit certain properties commonly observed in social networks, such as small diameter and relatively high average vertex degree. All datasets are downloadable from Stanford Large Network Dataset Collection [[1]](#footnote-1), Max Planck Institute’s Online Social Network Research Center [[2]](#footnote-2), and Social Computing Data Reposi-

tory at Arizona State University [[3]](#footnote-3).

In Table 1, we present important characteristics of all real datasets,



where *δ* is average vertex degree (i.e., 2|*E*|*/*|*V* |) and *d*0*.*9 is 90percentile effective diameter [24]. Finally, in the experimental study, we focus on the 6-degree shortest path queries (*k* = 6) as they are the most commonly used and also the most challenging one.

## Experimental Results

In the following, we report effectiveness and efficiency of the shortest path computation algorithms from different perspectives: **Query Results on Random Queries** In this experiment, we randomly generate 10*,*000 vertex pairs with various distances and execute all algorithms on these queries to study their performance. Here, we select 10*,*000 vertices with highest vertex degree as hubs. Table 3 presents the average query time for 10*,*000 queries on all the methods and Table 4 highlights the average query time for those vertex pairs whose distance is no less than 4 (longer path) as these are the more challenging ones (the longer the path, the likely more hubs will be expanded). Note that for BFS and two sketch methods Sketch(S∗) and TreeSketch(TS∗), we use the *millisecond* (10−3) as the unit, as they typically have much longer query time, and for BiBFS and our new approaches, Hub-Network (HN) and Hub2Labeling (HL) approaches, we use the *microsecond* (10−6) as the unit, as they are much faster. Their corresponding average search space per query is reported in Table 5, where column “HP-BBFS” records the average number of vertices visited by HP-BBFS (HubPruning Bidirectional BFS) in Hub2-Labeling (HL) and column “Join” records the average times of pairwise join on the core-hubs labeling *L*(*s*) and *L*(*t*) in HL. We make the following observations on the query time and average search space:

1) The Hub2-Labeling (HL) is clearly the winner among all algorithms, which is on average more than 2000 times faster than BFS. In most of the social networks, like As-skitter and WikiTalk, the average query time of Hub2-Labeling (HL) is only tens of microseconds (10−8 second), and except for one (Orkut), all of tham are less than 1ms. Overall, Hub2-Labeling (HL) is on average 23 times faster than BiBFS. Specifically, we observe that compared to BiBFS, the Hub-Pruning Bidirectional Search (HP-BBFS) of achieves significant improvement in terms of search space, which is around 800 times smaller than BiBFS (Table 5). 2) The Hub-Network (HN) is on average about 2 times faster than BiBFS (with no additional storage cost but reorganizes the network structure). It is about two orders of magnitude faster than BFS but is about 10 times slower than the Hub2-Labeling approach. 3) Sketch (S*⋆*) is on average about 10 times faster than BFS but it fails to run on a few datasets. The TreeSketch (TS*⋆*) is on average 70 times slower than Sketch. Both Hub-Network and HubLabeling approaches are are on average more than two orders of magnitude faster than Sketch, the fastest approximation method. 4) For long distance queries *d*(*u,v*) ≥ 4 the exact shortest path approaches require longer query time (Table 4). However, the increase for the Hub-Network (HN) and Hub2-Labeling (HL) are smaller than BFS and BiBFS. Also, it is interesting to observe the approximate shortest path approaches do not show performance decrease though both of them are still very slow.

**Preprocessing Cost:** Table 6 shows preprocessing cost of the Sketchbased approach along with HL, consisting of indexing size and precomputation time. The first column S*⋆* records the index size (MB) for the Sketch method. The second column HL*total* records total index size of Hub2-Labeling (HL), which is the sum of core-hubs



labeling cost and distance matrix size. Column |*L*(*v*)| record the average number of core-hubs stored by each vertex. Remarkably, the core-hub labeling scheme in Hub2-Labeling (HL) is very effective, as there is a very small portion of core-hubs recorded by each vertex. In most of the network, the average number of corehubs per vertex is no more than 2% of the total hubs. In particular, for network WikiTalk, only 2*.*5 core-hubs are stored in each vertex on average, which potentially leads to efficient query answering. However, for LiveJournal, the Hub2-Labeling is too expensive to be materialized in the main memory. In terms of precomputation time, Hub2-Labeling can be constructed faster than Sketch on 7 out of 10 networks. The construction time of HubNetwork (HN) is average more than three times faster than the Hub2-Labeling (HL), and it does not need any additional memory cost.

**Impacts of Hub Number:** In this experiment, we study the effect of different number of hubs on query performance. Here, we vary the hub-set size from 5*,*000 to 15*,*000 and conduct the experiment on 10*,*000 randomly generated queries with various distances. Table 2 shows the average query time of Hub-Network (HN) and Hub2-Labeling approaches using different number of hubs. In most of these networks, the best query performance is achieved when the number of hubs lies between 10*K* and 15*K*. Though a large number of hubs may potentially help reduce the search space of the bidirectional search in Hub2-Labeling (HL), it may also increase the size of core-hubs associated with each vertex. We observe that the query performance obtained by using 10*K* hub is comparable to the best one). Note that here due to space limitation, we do not report the detailed precomputation cost in terms of construction time and index size (for Hub2-Labeling). Overall, as the number of hub increases, most large networks, show an increasing trend regarding the average index size. Interestingly, when hub-set size increases, significant reduction of average index size is observed on WikiTalk. This is in part explained by its very small diameter. In terms of the precomputation time, as more hubs are chosen, the computational

cost of Hub-Network and Hub2-Labeling becomes larger, because more BFS needs to performed. Indeed, the precomputation time increases almost linearly with respect to the hub-set size.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | |  |  |  |  |  | | --- | --- | --- | --- | --- | | Dataset | |*V* | | |*E*| | *δ* | *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 | | |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 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 1: Network Statistics 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 | | |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | 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 3: Average Query Time on Random Query 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 | | |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | Dataset | Indexing Cost | | | Preproc.Time(min) | | | | S∗(MB) | HL*all*(MB) | |*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 5: Average Search Space on Random Query 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 |

**Hub-Network Statistics:** Finally, we report the basic statistics of the discovered distance preserving Hub-Network. Specifically, we are introduced in two following two questions: 1) given a set of hubs, how large the hub-network will be? What is the size of |*H⋆*|? 2) what are the degree difference between the hubs in the original network and in the Hub-Network? Do we observe a significant degree decreasing? To answer these two questions, in Table 7, we report |*H⋆*| (the number of total vertices in the hub-network), *d*1(*H*)

the average degree of hubs in the original graph, and *d*2(*H*), the average degree of hubs in the extracted hub-network, with respect to 5*K*, 8*K*, 10*K* and 15*K* hubs. We observe for most graphs, the size of |*H⋆*| is a few times larger than the hub number; however, for Orkut, LiveJournal, and Twitter, the hub network becomes quite large at 10*K* and 15*K* hubs. Also, in general, the degree of hubs in the hub-network has been lowered and on several graphs, the average degree is reduced smaller than 1/3 of the original average degree. We also observe that the ability of lowering degree is correlated with the search performance: the better the hub degree is lowered, the better query performance improvement we can get from the Hub-Network based bidirectional BFS.

# CONCLUSION

In this paper, we introduce a set of novel techniques centered on hubs for *k*-degree shortest path computation in large social networks. The Hub-Network and Hub2-Labeling algorithms can help significantly reduce the search space. The extensive experimental study demonstrates that these approaches can handle very large networks with millions of vertices, and its query processing is much faster than online searching algorithms and Sketch-based approaches, the state-of-the-art shortest path approximation algorithms. To the best of our knowledge, this is the first practical study on computing exact shortest paths on large social networks. In the future, we will study how to parallelize the index construction and query answering process. We also plan to investigate how to compute *k*-degree shortest path on dynamic networks.

# REFERENCES

1. I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck. A hub-based labeling algorithm for shortest paths in road networks. In *Proceedings of the 10th international conference on Experimental algorithms*, 2011.
2. I. Abraham, A. Fiat, A. V. Goldberg, and R. F. Werneck. Highway dimension, shortest paths, and provably efficient algorithms. In *SODA ’10*, 2010.
3. H. Bast, S. Funke, P. Sanders, and D. Schultes. Fast Routing in Road Networks 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 for dijkstra’s algorithm. *J. Exp. Algorithmics*, 15, March 2010.
5. James Cheng, Yiping Ke, Shumo Chu, and Carter Cheng. Efficient processing of distance queries in large graphs: a vertex cover approach. In *SIGMOD 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, 2012.
7. Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. Reachability and 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. *Phys. Rev. Lett.*, 90, Feb 2003.
9. A. Das Sarma, S. Gollapudi, and R. Najork, M.and Panigrahy. A sketch-based distance oracle for web-scale graphs. In *WSDM ’10*, 2010.
10. D.Djokovic. Distance-preserving subgraphs of hypercubes. *Journal of Combinatorial 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. 2009.
12. E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische 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.

1. R. Geisberger, P. Sanders, D. Schultes, and D. Delling. Contraction hierarchies: faster and simpler hierarchical routing in road networks. In *Proceedings of the 7th international conference on Experimental algorithms*, 2008.
2. A. V. Goldberg and C. Harrelson. Computing the shortest path: A search meets graph theory. In *SODA ’05*, 2005.
3. Andrey Gubichev, Srikanta Bedathur, Stephan Seufert, and Gerhard Weikum. Fast and accurate estimation of shortest paths in large graphs. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, CIKM ’10, pages 499–508, 2010.
4. R. J. Gutman. Reach-based routing: A new approach to shortest path algorithms optimized for road networks. In *ALENEX/ANALC*, pages 100–111, 2004.
5. R. Jin, Y. Xiang, N. Ruan, and D. Fuhry. 3-hop: a high-compression indexing scheme for reachability query. In *SIGMOD’09*, 2009.
6. Ruoming Jin, Ning Ruan, Yang Xiang, and Victor E. Lee. A highway-centric labeling approach for answering distance queries on large sparse graphs. In *SIGMOD Conference*, pages 445–456, 2012.
7. N. Jing, Y. Huang, and E. A. Rundensteiner. Hierarchical encoded path views for path query processing: An optimal model and its performance evaluation. *TKDE*, 10(3):409–432, 1998.
8. S. Jung and S. Pramanik. An efficient path computation model for hierarchically structured topographical road maps. *TKDE*, 14(5):1029–1046, 2002.
9. J. Kleinberg, A. Slivkins, and T. Wexler. Triangulation and embedding using small sets of beacons. In *FOCS ’04*, 2004.
10. H. Kriegel, P. Kro¨ger, M. Renz, and T. Schmidt. Hierarchical graph embedding for efficient query processing in very large traffic networks. In *SSDBM ’08*, 2008.
11. 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.
12. Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *CoRR*, abs/0810.1355, 2008.
13. C. Castillo M. Potamias, F. Bonchi and A. Gionis. Fast shortest path distance estimation in large networks. In *CIKM ’09*, 2009.
14. Alan Mislove, Hema Swetha Koppula, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee. Growth of the flickr social network. In *Proceedings of the 1st ACM SIGCOMM Workshop on Social Networks (WOSN’08)*, August 2008.
15. T. S. Eugene Ng and H. Zhang. Predicting internet network distance with coordinates-based approaches. In *INFOCOM*, 2001.
16. Miao Qiao, Hong Cheng, Lijun Chang, and Jeffrey Xu Yu. Approximate shortest distance computing: A query-dependent local landmark scheme. In *ICDE*, 2012.
17. H. Samet, J. Sankaranarayanan, and H. Alborzi. Scalable network distance browsing in spatial databases. In *SIGMOD’08*, 2008.
18. P. Sanders and D. Schultes. Highway hierarchies hasten exact shortest path queries. In *17th Eur. Symp. Algorithms (ESA)*, 2005.
19. J. Sankaranarayanan, H. Samet, and H. Alborzi. Path oracles for spatial networks. *PVLDB*, 2, August 2009.
20. R. Schenkel, A. Theobald, and G. Weikum. HOPI: An efficient connection index for complex XML document collections. In *EDBT*, 2004.
21. S. Shekhar, A. Fetterer, and B. Goyal. Materialization trade-offs in hierarchical shortest path algorithms. In *SSD ’97*, 1997.
22. Y. Tao, C. Sheng, and J. Pei. On *k*-skip shortest paths. In *SIGMOD’11*, 2011.
23. M. Thorup and U. Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, 2005.
24. [http://entitycube.research.microsoft.com.](http://entitycube.research.microsoft.com/)
25. [http://www.6-degreeonline.com.](http://www.6-degreeonline.com/)

[39]

[http://www.sysomos.com/insidetwitter/sixdegrees/.](http://www.sysomos.com/insidetwitter/sixdegrees/)

1. Fang Wei. Tedi: efficient shortest path query answering on graphs. In *SIGMOD Conference*, pages 99–110, 2010.
2. Xiaohan Zhao, Alessandra Sala, Christo Wilson, Haitao Zheng, and Ben Y. Zhao. Orion: Shortest path estimation for large social graphs. In *Proceedings of the 3rd Workshop on Online Social Networks (WOSN 2010)*, 2010.
3. Xiaohan Zhao, Alessandra Sala, Haitao Zheng, and Ben Y. Zhao. Efficient shortest paths on massive social graphs. In *Proceedings of 7th International Conference on Collaborative Computing:*

*Networking, Applications and Worksharing (CollaborateCom)*, 2011.

1. http://snap.stanford.edu/data/index.html [↑](#footnote-ref-1)
2. http://socialnetworks.mpi-sws.org/ [↑](#footnote-ref-2)
3. http://socialcomputing.asu.edu/datasets/ [↑](#footnote-ref-3)