通过剪枝地标标记实现大型网络上的快速精确最短路径距离查询

Takuya Akiba 东京大学 日本东京 113-0033
t.akiba@is.s.u-tokyo.ac.jp

岩田洋一
东京大学 日本东京 113-0033
y.iwata@is.s.u-tokyo.ac.jp

吉田雄一
国立信息学研究所，Preferred Infrastructure, Inc.东京，101-8430，日本
yyoshida@nii.ac.jp

摘要

我们为大规模网络上的最短路径距离查询提出了一种新的精确方法。我们的方法通过对每个顶点进行广度优先搜索来预先计算顶点的距离标签。乍看之下，这种方法似乎太明显、太低效，但这里引入的关键要素是在广度优先搜索过程中进行*剪枝*。虽然我们仍能根据标签回答出任意一对顶点的正确距离，但却出人意料地减少了搜索空间和标签的大小。此外，我们还展示了利用位操作可以同时执行 32 或 64 次广度优先搜索。我们通过实验证明，在各种大规模真实世界网络中，这两种技术的结合既高效又稳健。特别是，我们的方法可以处理具有数亿条边的社交网络和网络图，这比以前的精确方法的极限大两个数量级，而且查询时间与以前的方法相当。

类别和主题描述符

E.1 [**数据**]：数据结构-图*和网络*

一般条款

算法、实验、性能

关键词

图、最短路径、查询处理

1. 引言

*距离查询*询问的是图中两个顶点之间的距离。毫无疑问，回答距离查询是图上最基本的操作之一，而且应用广泛。例如，在社交网络中，两个用户之间的距离被认为是亲密程度的标志，并被用于社交敏感搜索，帮助用户找到更多相关的用户或内容 [40, 42]，或帮助用户找到更多相关的用户或内容 [41, 43]。

SIGMOD'13 2013 年 6 月 22-27 日，美国纽约。Copyright 2013 ACM 978-1-4503-2037-5/13/06 ...$15.00.

分析有影响力的人物和社区 [19, 6]。在网络图上，网页之间的距离是相关性的指标之一，在上下文感知搜索中用于给与当前访问网页更相关的网页更高的排名 [39, 29]。距离查询的其他应用还包括链接数据中的 *top-k* 关键字查询 [16,37]、发现代谢网络中化合物之间的最佳路径 [31, 32]，以及计算机网络中的资源管理 [28, 7]。

当然，我们可以使用广度优先搜索（BFS）或 Dijkstra 算法计算每个查询的距离。但是，对于大型图而言，这些算法需要花费一秒以上的时间，速度太慢，无法用作这些应用的构建模块。特别是社会敏感搜索或情境感知搜索等应用，由于涉及用户之间的实时交互，因此应该具有较低的延迟，而这些应用需要大量顶点对之间的距离来为每个搜索查询排序。因此，对距离查询的响应速度应更快，例如在微秒级。

另一种极端的方法是事先计算所有顶点对之间的距离，并将其存储在索引中。虽然我们可以立即回答距离查询，但这种方法也是不可接受的，因为预处理时间和索引大小都是二次方，大得不切实际。由于海量图数据的出现，在这两种极端方法之间设计出更适中、更实用的方法已经引起了数据库界的强烈兴趣 [12,29,41,38,4,30,17]。

一般来说，现实世界的网络有两大类图：一类是道路网络，另一类是复杂网络，如社交网络、网络图、生物网络和计算机网络。对于道路网络，由于其结构更容易掌握和利用，研究已经非常成功。现在，对于美国的整个公路网来说，公路网的距离查询可以在不到一微秒的时间内完成[1]。

相比之下，在复杂网络上回答距离查询仍然是一个极具挑战性的问题。由于道路网络的结构完全不同，因此用于道路网络的方法在这些网络上表现不佳。针对这些网络已经提出了几种方法，但它们都存在可扩展性差的缺点。它们至少需要几千秒或几万秒的时间来索引具有数百万条边的网络 [41, 4, 2, 17]。

为了处理更大的复杂网络，除了这些精确方法外，我们还研究了近似方法。也就是说，我们不必总是回答正确的距离。这些方法的成功之处在于可扩展性更强，而且计算量非常小。

随机查询的平均相对误差。然而，其中一些方法回答查询需要几毫秒 [15, 38, 30]，比其他方法慢了三个数量级。还有一些方法只需几毫秒就能回答查询[29, 40]，但据报道，这些方法对接近的顶点对的精度并不高[30, 4]。这一缺点可能对社会敏感搜索或上下文感知搜索等应用至关重要，因为在这些应用中，距离查询被用来区分相近的项目。

1.1 我们的贡献

为了解决这些问题，我们在本文中提出了一种在复杂网络中回答距离查询的新方法。本文提出的方法是一种精确方法。也就是说，它总是能准确回答距离查询。与以前的精确方法相比，它具有更好的可扩展性，可以处理数亿条边的图。不过，查询时间非常短，约为 10 微秒。虽然我们的方法可以处理有向图和/或加权图，但在下文中，为了简化说明，我们假设为无向、无加权图。

我们的方法基于*距离标签*或*距离感知 2 跳覆盖*的概念。2 跳覆盖的概念如下。我们为每个顶点 u 挑选一个候选顶点集 $C(u)$，这样每对顶点（u, v）之间的最短路径上至少有一个顶点 w∈ $C(u) \cap C(v)$。对于每个顶点 u 和顶点 $w \in C(u)$，我们预先计算它们之间的距离 $d_G(u, w)$。我们说集合 $L(u) = \{(w, d_G(u, w))\}_{w \in C(u)}$ 是 u 的*标签*。使用标签，很明显两个顶点 u 和 v 之间的距离 $_{dG}(u, v)$ 可以计算为 $min\{\delta+\delta' \mid (w, \delta) \in L(u), (w, \delta') \in L(v)\}$。标签系列 $\{L(u)\}$ 称为 *2 跳覆盖*。距离标注也常用于以前的精确方法 [13,12,2,17]，但我们提出了一种全新的、不同的标注计算方法，称为*剪枝地标标注*。

我们的方法思路简单而激进：从每个顶点开始，我们进行广度优先搜索，并将距离信息添加到访问顶点的标签中。当然，如果我们天真地实现这个想法，我们需要 $O(nm)$ 的预处理时间和 $O(n^2)$ 的空间来存储索引，这是不可接受的。这里，n 是顶点的数量，m 是边的数量。我们使这种方法实用化的关键想法是在广度优先搜索过程中进行*剪枝*。假设 S 是一个顶点集合，并假设我们已经有了一些标签，可以回答两个顶点之间的最短路径是否经过 S 中的一个顶点。假设我们从 v 开始进行 BFS，并访问 u。如果有一个顶点 w∈ S，使得 $d_G(v, u) = d_G(v, w) + d_G(w, u)$，那么我们就剪枝 u。也就是说，我们不遍历任何来自 u。正如我们在第 4.3 节中所证明的，在对 v 进行剪枝 BFS 之后，如果两个顶点之间的最短路径经过 Su{v} 中的一个顶点，那么标签就能回答这两个顶点之间的距离。

有趣的是，我们的方法结合了之前三种不同成功方法的优点：基于地标的近似方法 [29, 38, 30]、基于树分解的精确方法 [41,4] 和基于标记的精确方法 [13, 12, 2]。基于地标的近似方法通过利用复杂网络中存在的高顶点 [29]来达到可再标记的精度。这一事实也是我们的剪枝功能强大的主要原因：通过进行

表 1：以往方法和拟议方法在精确距离查询方面的实验结果摘要。

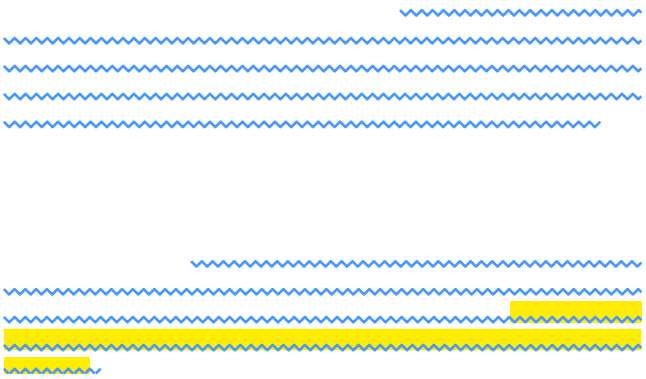
方法	网络	$\|V\|$	$\|E\|$	索引g	查询
泰迪	计算机	22 K	46 K	17 s	4.2 微秒
[41]	社会	0.6 M	0.6 M	2,226 s	55.0 微秒
HCL	社会	7.1 K	0.1 M	1,003 s	28.2 微秒
[17]	引用	0.7 M	0.3 M	253,104 s	0.2 微秒
TD	社会	0.3 M	0.4 M	9 s	0.5 微秒
[4]	社会	2.4 M	4.7 M	2,473 s	0.8 微秒
HHL	计算机	0.2 M	1.2 M	7,399 s	3.1 微秒
[2]	社会	0.3 M	1.9 M	19,488 s	6.9 微秒
	网络	0.3 M	1.5 M	**4 s**	0.5 微秒
PLL	社会	2.4 M	4.7 M	**61 s**	0.6 微秒
(本作品)	社会	1.1 M	**114 M**	15,164 s	15.6 微秒
	网络	7.4 M	**194 M**	6,068 s	4.1 微秒

首先从这些中心顶点开始进行广度优先搜索，之后我们就可以大幅削减广度优先搜索。基于树分解的方法通过分解低树宽的树状边缘来利用网络的核心-边缘结构 [10, 27]。虽然我们的方法没有明确使用树分解，但我们证明了我们的方法可以有效地处理小树宽的图。这一处理过程表明，我们的方法也利用了核心-边缘结构。与其他基于标签的方法一样，我们的索引数据结构简单，由于内存访问的局部性，查询处理非常迅速。

尽管这种剪枝地标标记方案本身已经很强大，但我们提出了另一种具有不同优势的标记方案，并将它们结合起来，进一步提高了性能。我们利用寄存器字中 b 位数通常为 32 或 64 的特性，证明广度优先搜索的标注方法可以通过比特并行方式实现，而且我们可以同时对这 b 位进行比特操作。利用这种技术，我们可以在 $O(m)$ 时间内同时对 $b + 1$ 个顶点执行 BFS。一开始，这种比特并行标注（无剪枝）比剪枝地标标注效果更好，因为剪枝不会经常发生。请注意，我们说的不是线程级并行，我们的比特并行实际上是以 $b + 1$ 的系数降低了计算复杂度。除了这两种标记方案，我们还可以使用线程级并行。

正如我们在实验结果中证实的那样，在精确距离查询方面，我们的方法优于其他最先进的方法。特别是，它的扩展能力明显优于以前的方法。索引数百万条边的网络只需几十秒。这个索引时间比以前的方法快了两个数量级，以前的方法至少需要数千秒，甚至一天以上。此外，我们的方法还成功处理了数亿条边的网络，这也比之前用于精确方法实验的网络大两个数量级。对于相同规模的网络，我们的查询时间也优于之前的方法，而且我们证实，查询时间不会随着网络规模的增大而迅速增加。我们还证实，我们方法的索引大小与其他方法相当。

在表 1 中，我们总结了我们的实验结果和这些论文中之前介绍的精确方法的实验结果。我们列出了现实世界中最大的两个复杂的



每篇论文中的网络。在实验中，我们进一步将我们的方法与分层中心标记法[2]和基于树分解的方法[4]进行了比较。

在第 2 节中，我们将介绍有关精确和近似距离查询的相关工作。在第 3 节中，我们给出了本文中使用的定义和概念。第 4 节专门介绍我们的第一个方案--剪枝地标标注。我们将在第 5 节解释我们的第二个方案--位并行标注。在第 6 节中，我们提到了通过稍微修改我们的方法可以处理的距离查询变体。我们将在第 7 节展示实验结果，并在第 8 节做出总结。

2. 相关工作

2.1 精确方法

对于社交网络和网络图等复杂网络的精确距离查询，最近提出了几种方法。

这些方法中的大部分都可以认为是基于 2 跳覆盖的思想[13]。高效寻找小型 2 跳覆盖是一个具有挑战性的长期问题[13, 2]。最新的方法之一是*分层中心标记法* [2]、该方法基于一种道路网络方法 [1]。另一种与 2 跳覆盖相关的最新方法是*以高速公路为中心的*覆盖法 [17]。在这种方法中，我们首先计算生成树 T，并将其作为 "高速公路"。也就是说，在计算两个顶点 u 和 v 之间的距离 $_{dG}(u, v)$ 时，我们输出 $_{dG}(u, w1) + dT (w1, w2) + _{dG}(w2, v)$ 的最小值，其中 w_1 和 w_2 分别是 u 和 v 的标签中的顶点，d_T (-, -) 是生成树 T 上的距离度量。

据报道，基于*树分解*的方法也很有效 [41,4]。图 G 的树分解是一棵树 T，其中每个顶点都与 G 中的一个顶点集合相关联，称为*一个包* [35]。此外，包含 G 中一个顶点的袋集构成 T 中的一个连通部分。它启发式地计算树分解，并存储每个包的最短距离矩阵。根据这些信息计算距离并不难。最大包的大小越小，这种方法的效率就越高。由于网络的核心-边缘结构[10, 27]，这些网络可以分解成一个大包和许多小包，最大包的大小虽然不小，但也适中。

2.2 近似方法

为了获得比这些精确方法更高的可扩展性，人们还研究了近似方法，这种方法并不总能回答正确的问题。

主要的方法是*基于地标的方法* [36, 40]。这些方法的基本思想是选择一个顶点子集 L 作为地标，并预先计算每个地标 $l \in L$ 与所有顶点 $u \in V$ 之间的距离 $d_G (l, u)$。当查询两个顶点 u 和 v 之间的距离时，我们会回答地标 $l \in L$ 上的最小 $d_G (u, l) + d_G (l, v)$ 作为估计值。一般来说，每次查询的精度取决于实际最短路径是否经过地标附近。因此，通过选择中心顶点作为地标，估算的精度要比随机选择地标高得多 [29, 11]。然而，对于距离较近的线对，精度仍然比平均值差很多，因为它们之间的最短路径长度较小，不太可能经过地标附近[4]。

为了进一步提高精确度，我们采用了以下几种技术

表 2：常用符号。

符号	说明
$G = (V, E)$	A 图
n	图 G 中的顶点数
m	图 G 中的边数
$N_G(v)$	图 G 中顶点 v 的邻居
$d_G(u, v)$	图 G 中顶点 u 与 v 之间的距离
$P_G(u, v)$	最短路径上所有顶点的集合 图 G 中顶点 u 和 v 之间的

[15,38,30]。它们通常存储根植于地标的最短路径树，而不是仅仅存储与地标的距离。为了回答查询，它们从最短路径树中提取路径作为候选最短路径，并通过寻找循环或捷径来改进路径。虽然它们大大提高了准确性，但查询时间却慢了三个数量级。

3. 预览

3.1 注释

表 2 列出了本文中经常使用的符号。本文主要关注以图建模的网络。让 $G = (V, E)$ 是一个具有顶点集 V 和边集 E 的图。当图的上下文很清楚时，我们用符号 n 和 m 分别表示顶点数 $|V|$ 和边数 $|E|$。我们还用 $V(G)$ 表示 G 的顶点集，用 $E(G)$ 表示 G 的边集。我们用 $N_G(v)$ 表示顶点 $v \in V$ 的邻居。也就是说，$N_G(v) = \{u \in V \mid (u, v) \in E\}$。

让 $d_G(u, v)$ 表示顶点 u、v 之间的距离。如果 u 和 v 在 G 中断开，我们定义 $d_G(u, v) = \infty$。图中的距离是一种度量，因此它满足三角形不等式。也就是说，对于任意三个顶点 s、t 和 v、

$$d_G(s, t) \le d_G(s, v) + d_G(v, t)、 \quad (1)$$

$$d_G(s, t) \ge |d_G(s, v) - d_G(v, t)| . \quad (2)$$

我们定义 $P_G(s, t) \subseteq V$ 为顶点 s 和 t 之间最短路径上所有顶点的集合、

$$P_G(s, t) = \{v \in V \mid d_G(s, v) + d_G(v, t) = d_G(s, t)\} .$$

3.2 问题定义

本文研究以下问题：给定一个图 G，构建一个索引来高效回答距离查询，即询问任意一对顶点之间的距离。为便于阐述，我们主要考虑未有向图、无权重图。不过，我们的算法可以很容易地扩展到有向图和/或加权图，我们将在第 6 节讨论这一扩展。此外，我们的方法不仅能计算距离，还能计算最短路径。第 6 节也将讨论这一扩展。

3.3 标签和 2-Hop 封面

2 跳覆盖法[13, 12, 2]的一般框架如下，有时也称为*标签法*。我们的方法也遵循这一框架。

对于每个顶点 v，我们都会预先计算一个*标签*，*标记*为 $L(v)$，它是一组*标签*对（u, δ_{uv}），其中 u 是一个顶点，$\delta uv = d_G(u, v)$。我们有时称标签集 $\{L(v)\}_{v \in V}$ 为*索引*。要回答顶点 s 和

t 时，我们计算并回答如下定义的查询（s，t，L）、

查询（s，t，L）=

$$\min\{\delta_{vs} + \delta_{vt} \mid (v, \delta_{vs}) \in L(s), (v, \delta_{vt}) \in L(t)\}.$$

如果 $L(s)$和 $L(t)$不共享，我们定义 $\text{QUERY}(s, t, L) = \infty$。

任何顶点。对于任何一对顶点 s 和 t，如果 $\text{QUERY}(s, t, L) = d_G(s,t)$，我们就称 L 为 G 的（*距离感知*）2 *跳覆盖*。

中的顶点排序。然后，我们可以计算
查询（s, t, L）使用合并-连接-查询，耗时 $O(|L(s)| + |L(t)|)$
类似的算法。

4. 算法描述

4.1 天马行空的地标标记

我们从以下简单的方法开始。首先，我们从每个顶点开始进行 BFS，并存储所有线对之间的距离。虽然这种方法过于明显，效率也不高，但为了说明下一种方法，我们还是要解释一下细节。

设 $V = \{v1, v2, ., vn\}$。我们从空索引 L_0 开始，其中 $L_0(u) = \emptyset$ 适用于每个 $u \in V$。假设我们按照 $v1, v2, \ldots, vn$ 的顺序从顶点导出 BFS。，vn *的顺序导出 BFS*。之后从顶点 v_k 开始的*第* k 个 BFS，我们将从 v_k 到所到达顶点的标签的距离相加，即 $L_k(u) = L_{k-1}(u) \cup \{(v_k, d_G(v_k, u))\}$ for each $u \in V$ with $d_G(v_k, u) /= \infty$。我们不改变未到达顶点的标签，也就是说，对于 $d_G(v_k, u) = \infty$ 的每个 $u \in V$，$L_k(u) = L_{k-1}(u)$。

L_n 是最终索引。显然，对于任何一对顶点 s 和 t，$\text{QUERY}(s, t, L_n) = d_G(s, t)$，因此，$L_n$ 是精确距离查询的直接 2 跳覆盖。这是因为，如果 s 和 t 是可达的，那么 $(s, 0) \in L_n(s)$ 和 $(s, d_G(s, t)) \in L_n(t)$ 举例来说。

这种方法可视为第 2.2 节中提到的基于地标的近似方法的变种。标准的基于地标的方法可以看作是一种预先计算 L_l 而不是 L_n 的方法，通过 $\text{QUERY}(s, t, L_l)$ 来估计 s 和 t 之间的距离，其中 l n 是表示地标的数量的参数。

4.2 经修剪的地标标签

然后，我们将*剪枝*方法引入幼稚方法。与上述方法类似点我们从为 $v1$、$v2$、.，vn。我们从一个空索引 L，并从 L 创建索引 L''_k L'_{k-1} 使用

算法 1 从 $v_k \in V$ *中*剪枝 BFS，创建索引 L'_k。

1：**程序** $\text{PRUNEDBFS}(G, v_k, L'_{k-1})$
2: $Q \to$ 只有一个元素 v 的队列。
3: $P[v_k] \to 0$，对于所有 $v \in V(G) \setminus \{v_k\}$，$P[v] \to \infty$。
4: $L'_k[v] \to L'_{k-1}[v]$ 适用于所有 $v \in V(G)$。

5: **while** Q is not empty **do**
6: 从 Q 中 Dequeue u。 瓶颈是 7 号线
7: **if** $\text{QUERY}(v_k, u, L'_{k-1}) \le P[u]$ **then**

8: **继续**
9: $L'_k[u] \to L'_{k-1}[u] \cup \{(v_k, P[v_k])\}$

10: **对于所有** $w \in NG(v)$ s.t. $P[w] = \infty$ **do**
11: $P[w] \to P[u] + 1$。 第10行应该是NG(u)
12: 将 w 查询到 Q 13
: **返回** L'_k

算法 2 通过剪枝 BFS 计算 2 跳覆盖索引。1: 预处理（G）**过程**
2: $L'_0[v] \to \emptyset$ for all $v \in V$
(G).3: **for** k = 1, 2, ., n **do**
4: $L'_k \to \text{PRUNEDBFS}(G, v_k, L'_{k-1})$

5: **返回** L'_n

从 ver- tex v_k 中剪枝的*第* k 个 BFS 所获得的信息。

访问顶点 6 时，由于 $\text{QUERY}(2, 6, L') = {}_{dG}(2, 1) + {}_{dG(}1, 6)$ $= 3 = {}_{dG}(2，6)$，因此我们剪去顶点 6，不从它开始遍历边。我们还删除了顶点 1 和 12。随着所执行 BFS 的数量增加，我们可以确认搜索空间越来越小（图 1c 、1d 和 1e）。

4.3 正确性证明

下面，我们将证明这种方法能计算出正确的 2 跳覆盖索引，即对于任意一对顶点 s 和 t ，$\text{QUERY}(s, t, L'_{k-1})$ ，我们对 BFS 进行如下剪裁。假设我们有一个索引 L'_{k-1}，我们从 v_k 创建一个新的索引 L_k。假设我们正在访问一个顶点 u，其索引为距离 δ。如果 $\text{QUERY}(v_k, u, L'_{k-1}) \le \delta$，那么我们剪去 u，即也就是说，我们不把 (v_k, δ) 加入 $L'_k(u)$ （即 $L'_k(u) = L'_{k-1}(u)$）并且我们不会从顶点 u 开始遍历任何一条边。否则我们设置 $L'_k(u) = L'_{k-1}(u) \cup \{(v_k, \delta)\}$ 并遍历所有边从顶点 u 开始。与前一种方法一样，我们还设置 $L'_k(u) = L'_{k-1}(u)$，适用于所有不在 V'_k 中的顶点 $u \in V$。

在第 k 个经过剪枝的 BFS 中被访问。该算法执行经过剪枝的 BFS 的算法描述为算法 1，整个预处理算法描述为算法 2。

图 1 显示了剪枝 BFS 的示例。从顶点 1 出发的第一个剪枝 BFS 访问了所有顶点（图 1a）。在从顶点 2 开始的下一次剪枝 BFS（图 1b）中，当我们

$L') = d_G(s, t)$。

定理 4.1. *对于任意* $0 \le k \le n$ *和任意一对顶点* s *和* t ，$\text{QUERY}(s, t, L'_k) = \text{QUERY}(s, t, L_k)$.

证明由于 $L'_0 = L_0$ ，所以 $k = 0$ 时定理成立。现在我们假设 $0, 1, ., k - 1$ 时成立，并证明对 k 也成立。

这就是 {v_k, P[u], u,parent}

假设 s, t 是一对顶点。我们假设这对顶点在 G 中是可到达的，否则答案∞ 显然可以得到。让 j 是最小的数，使得 $(v_j, \delta_{vjs}) \in L_k(s)$ ，$(v, \delta_{jvjt}) \in L_k(t)$ 且 $\delta_{vjs} + \delta_{vjt} = \text{QUERY}(s, t, L_k)$。我们证明 (v_j, δ_{vjs}) 和 (v_j, δ_{vjt}) 也包含在 $L'_k(s)$ 和 $L'_k(t)$ 中。这立即导致到 $\text{QUERY}(s, t, L'_k) = \text{QUERY}(s, t, L_k)$。由于对称性在 s 和 t 之间，我们证明 $(v_j, \delta_{vjs}) \in L_k(s)$。

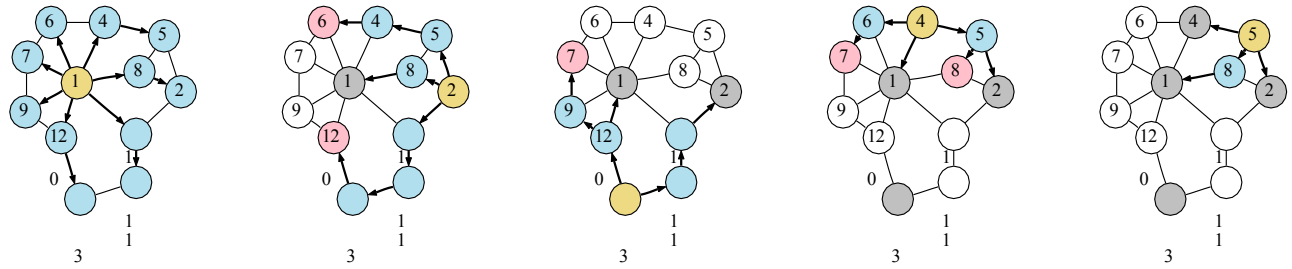
首先，对于任意 $i < j$，我们通过矛盾证明 $v_i \notin P_G(v_j, s)$。如果我们假设 $v_i \in P_G(v_j, s)$ ，根据不等式 1

$$\begin{aligned}\text{QUERY}(s, t, L_k) &= d_G(s, v_j) + d_G(v_j, t) \\ &= d_G(s, v_i) + d_G(v_i, v_j) + d_G(v_j, t) \\ &\ge d_G(s, v_i) + d_G(v_i, t)。\end{aligned}$$

由于 $(v_i, d_G(s, v_i)) \in L_k(s)$ 和 $(v_i, d_G(t, v_i)) \in L_k(t)$，因此

因此，对于任何 $i < j$，$v_i \notin P_G(v_j, s)$ 都成立。

现在我们证明 $(v_j, d_G(v_j, s)) \in L'_k(s)$。实际上，我们证明了一个更一般的事实：对于所有 $u \in P_G(v_j, s)$ ，$(v_j, d_G(v_j, u)) \in L'_k(u)$。注意 $s \in P_G(v_j, s)$。假设我们



(a) 从 ver- tex 1 开始的第一次 BFS。我们访问了所有顶点。

(b) 从顶点 2 开始的第二次 BFS。我们没有为五个顶点添加标签。

(c) 从顶点 3 开始的第三次 BFS。我们只看到了顶点的下半部分。

(d) 从顶点 4 开始的第四次 BFS。这次我们只访问了高半部分。

(e) 第 5 版的第五次 BFS。搜索空间更小了。

图 1：经过剪枝的 BFS 示例。黄色顶点表示根，蓝色顶点表示我们访问过并标注的顶点，红色顶点表示我们访问过但已剪枝的顶点，灰色顶点表示已用作根的顶点。

正在对 v_j 进行*第* j *次*剪枝 BFS，以创建 L_j。让 u∈ P_G (v_j, s)。因为对于任意 $i < j$，$P_G(v_j, u) \subseteq P_G(v_j, s)$ 和 $v_i \notin P_G(v_j, s)$，所以对于任意 $i < j$，我们有 $v_i \notin P_G(v_j, u)$。

$i < j$。因此，QUERY（v_j，u，$_{j-1}$）$> d_G(v_j, u)$ 成立 L'。

因此，我们会访问所有顶点 $u \in P_G(v_j, s)$，无需剪枝、由此可知，$(v_j, d_G(v_j, u)) \in L'_j(u) \subseteq L'_k(u)$。□

作为推论，通过将定理实例化为 $k = n$，证明我们的方法是一种精确的差分查询方法。

推论 4.1.*对于任何一对顶点 s 和 t*、

$$\text{QUERY}(s, t, L'_n) = d_G(s, t).$$

4.4 顶点排序策略

他们使用 "度 "来排列顶点

4.4.1 *动机*

在上述算法描述中，我们按照 $_{v1}$、$_{v2}$、. . .. 的顺序对顶点进行了剪枝 BFS。，$_{vn}$ *的顺序进行剪枝 BFS*。我们可以自由选择顺序，而且正如我们将在第 7.3.4 节中看到的实验结果所示，顺序对该方法的性能至关重要。

要决定顶点的顺序，我们应该首先选择*中心*顶点，因为许多最短路径都会经过这些顶点。由于我们希望尽可能地删减后面的 BFS，因此我们希望前面的 BFS 能*覆盖*更多的顶点对。也就是说，较早的标签应该为尽可能多的顶点对提供正确的距离，因此较早的顶点应该是那些有许多最短路径经过的顶点。

这个问题与 [29] 中讨论的为基于地标的近似方法选择好地标的问题非常相似。在该问题中，我们也希望选择接近度我们对顶点进行排序，从最接近中心度最高的顶点开始。由于计算所有顶点的精确接近中心度需要花费 $O(nm)$ 的时间，这太过苛刻。

在大规模网络中，我们将接近度近似为

通过随机抽取少量顶点来确定中心性并计算这些顶点到所有顶点的距离。

4.5 高效实施

4.5.1 *预处理（算法 1）* 和

索引：首先，在上述描述中，我们将 L' $_{k-1}$ 好的地标，以便有许多最短路径经过这些顶点或附近的顶点。

4.4.2 *战略*

根据基于地标方法的研究结果 [29]，我们提出并研究了以下三种策略。在实验中，我们基本上使用度策略，并在第 7.3.4 节中进行了经验比较。

随机：我们随机排列顶点。我们使用这种方法

′ 为便于解释，我们将 L' 分别复制到 L'。不过，只需保留一个索引，并在每次剪枝 BFS 后添加标签，就能轻松避免这种复制。

初始化：另一个重要注意事项是避免为每个剪枝 BFS 进行 $O(n)$ 时间的初始化。这种方法之所以高效，是因为剪枝 BFS 的搜索空间比整个图要小得多。然而，如果我们花费 $O(n)$ 时间进行初始化，这将成为瓶颈。在初始化过程中，我们要做的是将存储暂定距离的数组中的所有值都设置为∞ （第 3 行）。我们可以通过以下方法避免 $O(n)$ 时间的初始化。在进行第一次剪枝 BFS 之前，我们将数组 P 中的所有值都设为 ∞。(这需要 $O(n)$ 时间，但我们只做一次）。然后，在每次剪枝 BFS 的过程中，我们存储我们访问过的所有顶点，并在每次剪枝 BFS 之后，将我们访问过的所有顶点 v 的 $P[v]$ 设为 ∞。

作为显示其他战略重要性的基线。

度数：我们从程度较高的顶点开始排序。这种策略的理念是，顶点越高，则

度与许多其他顶点有更强的联系，并且因此，会有许多最短路径经过它们。

数组：对于存储暂定距离的数组，最好使用 8 位整数。因为我们感兴趣的网络是小世界网络，8 位整数足以表示距离。使用 8 位整数，该数组可以放入最新计算机的低级缓存存储器中，从而通过减少缓存丢失来提高速度。

查询：要评估用于剪枝的查询（第 7 行），使用不同于普通算法的算法会更快，因为我们可以利用这样一个事实，即我们发出的许多查询的一个端点总是 v_k。在从 v_k 开始第 k 个剪枝 BFS 之前，我们准备一个长度为 n 的数组 T，用 ∞ 进行初始化，并为所有 $(w, \delta_{wvk}) \in L'_{k-1}(v_k)$ 设置 $T[w] = \delta_{wvk}$。要评估 $\text{QUERY}(v_k, u, L'_{k-1})$，对于所有 $(w, \delta_{wu}) \in L'_{k-1}(u)$ 时，我们计算 $\delta_{wu} + T[w]$，并返回最小值。虽然普通查询算法需要 $O(|L'_{k-1}(v_k)| + |L'_{k-1}(u)|)$ 时间，该算法的运行时间为 $O(|L'_{k-1}(u)|)$。由于第 7 行是算法的瓶颈，因此这种技术可将预处理速度提高约两倍。请注意，T 应

与上述原因相同，数组 T 应使用 8 位整数表示，数组 P 也应避免使用 $O(n)$ 时初始化。

预取：遗憾的是，我们无法将索引和邻接表放入大规模网络的缓存内存中。不过，我们可以手动预取它们，以减少缓存丢失，因为我们即将访问的顶点都在队列中。手动预取可将预处理速度提高约 20%。

线程级并行：与并行 BFS 算法 [3] 一样，剪枝 BFS 算法也可以并行化。不过，为了进行简单的实验分析并与之前的方法进行公平比较，我们在实验中没有对我们的实现进行并行化。

标签排序：在使用类似合并连接的算法回答查询时，需要按照顶点对标签中的数据对进行排序。但实际上，我们并不需要明确地排序，只需存储顶点的等级而不是顶点。也就是说，在第 i 次剪枝的 BFS 中，当从顶点 u 开始添加一对 (u, δ) 时，我们会添加一对 (i, δ)。然后，由于数据对是从等级较低的顶点添加到等级较高的顶点，所有标签都会自动排序。

4.5.2 查询

哨兵：我们为每个 $v \in V$ 的标签 $L(v)$ 添加一个虚拟条目 (n, ∞)。这个虚拟条目确保我们在扫描两个标签时，最终能找到相同的顶点 n。因此，我们可以避免分别测试是否扫描到最后。

数组：对于每个标签 $L(v)$，分别存储顶点数组和距离数组会更快，因为只有当顶点匹配时才会用到距离[1]。我们还将数组与缓存行对齐。

4.6 理论特性

4.6.1 最小化

定理 4.2 *设 L' 是第 4.2 节中定义的索引。L' 是最小的，因为对于任何顶点 v 和任何顶点对 $(u, \delta_{uv}) \in L'(v)$，都有一对顶点 (s, t)，使得如果我们将 (u, δ_{uv}) 从 $L_n(v)$ 中删除，我们就无法回答 s 和 t 之间的正确距离。*

证明。设 $v_i \in V$ 且 $(v_j, \delta_{v_i v_j}) \in L'(v_i)$。这意味着 $j < i$。我们可以证明，如果从 $L'(v_i)$ 中删除 $(v_j, \delta_{v_i v_j})$，那么我们就无法回答 v_i 和 v_j。我们认为，对于任意 $k \neq j$，要么 (i) $(v_i, \delta_{v_i v_k}) \notin L'(v_i)$ 或 $(v_k, \delta_{v_k v_j}) \notin L'(v_j)$ 成立，或 (ii) $d_G(v_i, v_k) = d_G(v_i, v_j) + d_G(v_j, v_k)$ 成立。假设 $k < j$，并假设 (ii) 不成立。那么，(i) 必须成立，否则第 j 个 BFS 应该剪去顶点 v_i 和 $(v_j, \delta_{v_j v_i}) \notin L'(v_i)$。假设 $k > j$，并假设 (ii) 不成立。那么，$v_k \in P_G(v_i, v_j)$，因此 $(v_j, \delta_{v_k v_j}) \in L'(v_i)$ 和 $(v_k, \delta_{v_i v_k}) \in L'(v_k)$。因此，$d_G(v_i, v_k) = d_G(v_i, v_j) - d_G(v_k, v_j)$，这与 (ii) 不成立的假设相矛盾。□

从下面的定理可以看出，我们的方法对于基于地标的方法能够以如此高的预精度回答距离问题的网络是有效的。从下面的定理中我们可以看出，我们的方法对于基于地标的方法能以如此高的预精度回答其距离的网络是有效的，而且我们的方法还能利用这些中心顶点的存在。

定理 4.3. *如果我们假设基于地标的标准近似方法可以用 k 个地标回答 $(1 - G)n^2$ 对（共 n^2 对）的正确距离，那么剪枝地标标注方法给出的索引平均标签大小为 $O(k + Gn)$。*

证明草图。首先对 k 个地标顶点进行剪枝 BFS 后，索引中最多添加 Gn^2 对，因为我们从不添加根据当前标签可以知道距离的对。 □

4.6.3 利用条纹的小树宽

最后，我们从理论上证明，我们的方法也能有效利用树状边缘。正如我们在第 2.1 节中提到的，有人提出了基于树分解的距离查询方法 [41, 4]。这两种方法都扩展了对小树宽图有效的方法，并通过树分解利用了现实世界网络中的低树宽边缘。有趣的是，虽然我们没有明确使用树分解，但我们可以证明我们的方法可以高效处理小树宽的图。因此，我们的方法也隐含地利用了现实世界网络的这一特性。关于树宽和树分解的定义，请参见 [35]。

定理 4.4 *设 w 是 G 的树宽。有一种顶点顺序，剪枝地标标注方法预处理需要 $O(wm \log n + w^2 n \log^2 n)$ 时间，存储索引需要 $O(wn \log n)$ 空间，回答每个查询只需 $O(w \log n)$ 时间。*

证明草图关键要素是树分解的中心点去组成 [18]。首先，我们从中心点袋中的所有顶点剪枝 BFS。然后，剪枝后的 BFS 永远不会超出该袋。因此，我们可以在划分树分解时考虑分成不相连的部分，每个部分最多有一半的袋。我们递归地重复这一过程。递归次数最多为 $O(\log n)$。由于每个递归深度最多给每个顶点添加 w 对，因此递归次数为每个标签中的线对数量为 $O(w \log n)$。在每个深度再的剪枝 BFS 所消耗的总时间。当前分量为 $O(wm + w^2 n \log n)$，其中 $O(wm)$ 为是遍历边的时间，$O(w n \log n)$是

$\in L'(v_k)$，因此*第* k 个 BFS 删除顶点 v_j，导致 $(v_k, \delta_{v_{kvj}}) /\in L'(_{vj})$。

4.6.2 利用高中心顶的存在 然后，我们将我们的方法与基于地标的方法进行比较，以说明我们的方法也能利用*高中心顶*的存在。高度中心顶点。我们考虑了标准的地标的方法 [29,40]，这种方法不使用任何路径启发式。正如我们在第 2.2 节所述，通过选择中心顶点作为进行剪枝测试。

5. 位并行标签

为了进一步加快预处理和查询的速度，我们提出了一种利用位级并行的优化方法。位并行方法是指在同一字中的不同位上执行不同计算的方法，以利用计算机可以同时对一个字执行位操作这一事实。在当时的计算机中，字长通常为 32 或 64。

在下文中，我们将计算机字的位数记为 b，并假设对长度为 b 的位向量进行比特运算可以在 $O(1)$ 时间内完成。我们提出了一种进行 BFS 并从 $b + 1$ 根计算标签的算法

位并行不适合加权图，参见第 6 节

算法 3 来自 $r \in V$ 和 $S_r \subseteq N_G(r)$ 的比特并行 BFS。

```
1: 程序 BP-BFS (G, r, S)_r
2:   (P[v], S_r^{-1}[v], S_r^0[v]) → (∞, ∅, ∅) for all v ∈ V
3:   P[r], S_r^{-1}[r], S^0[r]) → (0, ∅, ∅)
4:   (P[v], S_r^{-1}[v], S_r^0[v]) → (1, {v}, ∅) for all v∈ S r_r
5:   Q_0, Q_1 → 空队列
6:   将 r 排序到 Q_0
7:   对于所有 v∈ S，将 v Enqueue
到 Q_1r
8:   while Q_0 is not empty do
9:     E_0 → ∅ 和 E_1 → ∅
10:    while Q_0 is not empty do
11:      从 Q_0 中 Dequeue v。
12:      对所有 u∈N_G(v) 做
13:        如果 P[u] = ∞ ∨ P[u] = P[v] + 1
，那么
14:          E_1 → E_1 ∪ {(v, u)}
15:          如果 P[u] = ∞，那么
16:            P[u] → P[v] + 1
17:            Enqueue u onto Q_1。
18:        else if P[u] = P[v] + 1 then
19:          E_0 → E_0 ∪ {(v, u)}
20:    对于所有 (v, u) ∈ E_0
       做
21:      S_r^0[u] → S_r^0[u] ∪ S_r^{-1}[v]
22:    对于所有 (v, u) ∈ E_1
do
23:      S_r^{-1}[u] → S_r^{-1}[u] ∪ S_r^{-1}[v]
24:      S_r^0[u] → S_r^0[u] ∪ S_r^0[v]
25:    Q_0 → Q_1 和 Q_1 → ∅
26:  return (P, S_r^{-1}, S_r^0)
```

同时，只需 $O(m)$ 时间。此外，我们还提出了一种方法，通过其中一个 $b+1$ 个顶点，在 $O(1)$ 时间内回答任意一对顶点的距离查询。

5.1 位并行标签

为了描述预处理算法和查询算法，我们首先要定义索引中存储的内容。

正如我们在下一小节中解释的那样，我们从顶点 r 及其大小最多为 b 的邻居子集 $S \subseteq N(r)$ 开始进行比特并行 BFS。

$$S_r^i(v) = \{u \in S \mid d_G(u, v) - d_G(r, v) = i\}$$

由于 S_r 中的顶点是 r 的邻居，因此对于任意顶点 $u \in S_r$ 和任意顶点 $v \in V$，$|d_G(u, v) - d_G(r, v)| \le 1$。因此，对于每个 $v \in V$，S_r 可以划分为 $S_r^{-1}(v)$、$S_r^0(v)$ 和 $S_r^{+1}(v)$。也就是说，$S_r^{-1}(v) \cup S_r^0(v) \cup S_r^{+1}(v) = S$。

我们计算*位并行标签*，并将其存储在数据库中。对于每个顶点 $v \in V$，我们会预先计算一个位并行标签。标记为 $L_r(v)$。$L_r(v)$ 是一组四元组

$$(r, \delta_{rs}, S_r^{-1}(s), S_r^0(s)) \in L_{BP}(s)$$

和

$$(r, \delta_{rt}, S_r^{-1}(t), S_r^0(t)) \in L_{BP}(t)$$

，我们从这些四元组中通过 $\{r\} \cup$ 中的一个顶点计算 s 和 t 之间的距离。也就是说，我们计算 $\delta = \min_{u \in \{r\} \cup S_r} \{d_G(s, u) + d_G(u, t)\}$。

在第 4.1 节中进行了讨论。我们将在第 5.4 节介绍剪枝。

让 $r \in V$ 是一个顶点，$S_r \subseteq N_G(r)$ 是

我们将解释一种算法。

计算所有 $v \in V$ 的 $d_G(r, v)$、$S_r^{-1}(v)$ 和 $S_r^0(v)$，即可从 $\{r\} \cup S_r$ 到达。算法描述如下

算法 3.基本上，我们从 r 计算集合 S_r^{-1} 和 S_r^0，进行 BFS

。

假设 v 是一个顶点。假设我们已经计算出 $S_r^{-1}(w)$ for all w such that $d_G(r, w) < d_G(r, v)$. 我们可以计算 $S_r^{-1}(v)$ 的方法如下、

$$\{u \in S_r \mid u \in S_r^{-1}(w),\ w \in N_G(v),\ d_G(r, w) = d_G(r, v) - 1\},$$

因为如果 u 位于 $S_r^{-1}(v)$，则 $d_G(u, v) = d_G(r, v) - 1$，因此 u 位于从 r 到 v 的最短路径之一上。同样，假设我们已经计算了所有 w 的 $S_r^{-1}(w)$，使得 $d_G(r, w) \le d_G(r, v)$，以及所有 w 的 $S_r^0(w)$，使得 $d_G(r, w) < d_G(r, v)$，我们可以计算 $S_r^0(v)$ 如下、

$$\{u \in S_r \mid u \in S_r^0(w),\ w \in N_G(v),\ d_G(r, w) = d_G(r, v) - 1\}$$
$$\cup \{u \in S_r \mid u \in S_r^{-1}(w),\ w \in N_G(v),\ d_G(r, w) = d_G(r, v)\}.$$

因此，在进行广度优先搜索的同时，我们可以通过动态编程交替计算 S_r^{-1} 和 S_r^0 的距离依次增大。

针对所有 $v \in V$ 计算 $S_r^{-1}(v)$，使得 $d_G(r, v) = 1$，接下来对于所有 $v \in V$ 且 $d_G(r, v) = 1$ 的情况，我们计算 $S_r^0(v)$，然后对于所有 $v \in V$ 且 $d_G(r, v) = 2$ 的情况，我们计算 $S_r^{-1}(v)$、

接下来，我们针对所有 $v \in V$ 计算 $S_r^0(v)$，使得 $d_G(r, v) = 2$，以此类推。请注意，对集合的运算可以在 $O(1)$ 用比特矢量表示集合，并使用比特

业务。

5.3 位并行距离查询

处理一对顶点 s 和 t 之间的距离查询时，与处理普通标签一样，我们扫描位并行标签 $L_{BP}(s)$ 和 $L_{BP}(t)$。对于共享相同根顶点的每对四元组，

$$(r, \delta_{rs}, S_r^{-1}(s), S_r^0(s)) \in L_{BP}(s)$$

和

$$(r, \delta_{rt}, S_r^{-1}(t), S_r^0(t)) \in L(t)$$

，我们从这些四元组中通过 $\{r\} \cup$ 中的一个顶点计算 s 和 t 之间的距离。

一种简单的方法是计算所有 u 的 $d_G(s, u)$ 和 $d_G(u, t)$ 并取最小值，这需要 $O(|S_r|)$ 时间。然而，我们提出了一种利用位操作在 $O(1)$ 时间内计算 δ 的算法。

具体方法如下。如果 $S_r^{-1}(s) \cap S_r^{-1}(t) \neq \emptyset$，那么 $\delta = \tilde{\delta} - 2$。否则，如果 $S_r^0(s) \cap S_r^{-1}(t) \neq \emptyset$ 或 $S_r^{-1}(s) \cap S_r^0(t) \neq \emptyset$，则 $\delta = \tilde{\delta} - 1$，否则 $\delta = \tilde{\delta}$。注意计算集合的交集可以用位运算 AND 来完成操作。因此，所有这些操作都可以在 $O(1)$ 时间。因此，计算距离 δ 的时间为 $O(1)$，我们回答每个查询的总时间为 $O(|L_{\mathrm{BP}}(s)| + |L_{\mathrm{BP}}(t)|)$ 时间。

设 $\tilde{\delta} = d_G(s, r) + d_G(r, t)$。由于 $\tilde{\delta}$ 是且 $d_G(s, u) \geq d_G(s, r) - 1$, $d_G(u, t) \geq d_G(r, t) - 1$ for all $u \in S_r$, $\tilde{\delta} - 2 \leq \delta \leq \tilde{\delta}$. 因此，我们要做的是判断距离 δ 是 $\tilde{\delta}-2$，$\tilde{\delta}-1$，还是 $\tilde{\delta}$。

$(u, \delta_{uv}, S_u^{-1}(v), S_u^0(v))_{u \in V}^{\mathrm{BP}}$，其中 $u \in V$ 是一个顶点，$\delta_{uv} = d_G(u, v)$ 和 $S_u(v) \in S_r$ 的定义如上。我们存储 $S_u(v)$ 和 $S_u^0(v)$ 的 b 比特向量。请注意，$S_u^{+1}(v)$ 可以是得到的 $S_u(S_u^{-1}(v) \cup S_u^0(v))_{u \in V}^{\mathrm{BP}}$，但实际上我们并没有在查询算法中使用 $S_u^{+1}(v)$。

为了用 b 个比特的比特向量描述 S_r 的子集，我们为 S_r 中的每个顶点分配一个介于 1 和 $|S_r|$ 之间的唯一编号，并通过设置第 i 个比特来表示第 i 个顶点的存在。

5.2 位并行 BFS

我们撇开第 4.2 节中讨论的剪枝处理不谈，我们制作了一个比特并行版的天真标注方法

5.4 修剪标签介绍

现在我们讨论如何将这种比特并行标注方法与第 4.2 节中讨论的剪枝标注方法结合起来。我们提出一种简单高效的方法如下。

首先，我们进行 t 次不剪枝的位并行 BFS，其中 t 是一个参数。然后，我们使用位并行标签和普通标签进行剪枝 BFS。

这种方法利用了剪枝法和比特并行标注法的不同优势。一开始，剪枝的作用不大，剪枝后的 BFS 会访问大部分顶点。因此，我们不使用剪枝标注法，而是使用位并行标注法来有效地覆盖更多的顶点对。跳过徒劳的剪枝测试也有助于提高速度。

作为比特并行 BFS 的根和邻居集，我们倾向于贪婪地使用优先级最高的顶点：我们选择一个优先级最高的顶点作为剩余顶点中的根 r，并选择最多 b 个优先级最高的顶点作为剩余邻居集 S_r。

正如我们在第 7 节的实验结果中所看到的，这种方法改善了预处理时间、索引大小和查询时间。此外，正如我们在实验中证实的那样，如果我们不设置过大的 t 值，至少不会影响性能。因此，我们不必太认真地寻找一个合适的 t 值，而且我们的方法仍然很容易使用。

6. 变体和扩展

最短路径查询： 不仅要回答距离问题，还要回答最短路径问题，我们存储的是元组集，而不是成对的数据集作为标签。标签 $L(v)$ 是一组三元组 (u, δ_{uv}, p_{uv})，其中 $p_{uv} \in V$ 是经过剪枝的广度优先搜索中 u 的父节点我们可以通过从 v 到父代的上升树来恢复 v 和 u 之间的最短路径。

加权图： 要处理加权图，唯一必要的改变是执行剪枝 Dijkstra 算法，而不是剪枝 BFS。位并行标记不能用于加权图。

有向图： 在处理有向图时，我们首先将 $d_G(u, v)$ 重定义为 u 到 v 的距离。然后，我们为每个顶点存储两个标签 $L_{\text{OUT}}(v)$ 和 $L_{\text{IN}}(v)$。标签 $L_{\text{OUT}}(v)$ 是一对 (u, δ_{vu}) 的集合，其中 $u \in V$ 和 $\delta_{vu} = d_G(v, u)$；标签 $L_{\text{IN}}(v)$ 是一对 (u, δ_{uv}) 的集合，其中 $u \in V$ 和 $\delta_{uv} = d_G(u, v)$。我们可以用 $L_{\text{OUT}}(s)$ 和 $L_{\text{IN}}(t)$ 来回答顶点 s 到顶点 t 的距离。为了计算这些标签，我们从每个顶点出发，进行两次剪枝 BFS：一次在正向，一次在反向。

基于磁盘的查询回答： 要回答距离查询，我们的查询算法只需参考两个连续区域。因此，如果索引驻留在磁盘上，我们只需进行两次磁盘寻道操作就能回答查询，这仍然比内存中的 BFS 快得多。

7. 实验

我们在一台配备英特尔至强 X5670（2.93 GHz）和 48GB 主内存的 Linux 服务器上进行了实验。提出的方法是用 C++ 实现的。我们使用 8 位整数表示距离，32 位整数表示顶点，64 位整数进行位并行 BFS。对于顶点排序，我们主要使用 DEGREE 策略，除非使用其他策略，否则不指定顶点排序策略。对于查询时间，我们一般报告 1,000,000 次随机查询的平均时间。

表 4：数据集

数据集	网络	$\|V\|$	$\|E\|$
Gnutella	计算机	63 K	148 K
Epinions	社会	76 K	509 K
Slashdot	社会	82 K	948 K
Notredame	网络	326 K	1.5 M
WikiTalk	社会	2.4 M	4.7 M
Skitter	计算机	1.7 M	11 M
印 度	网络	1.4 M	17 M
MetroSec	计算机	2.3 M	22 M
图片库	社会	1.8 M	23 M
好莱坞	社会	1.1 M	114 M
印度支那	网络	7.4 M	194 M

7.1 数据集

为了证明我们的方法的效率和鲁棒性，我们在各种真实世界的网络上进行了实验：五个社交网络、三个网络图和三个计算机网络。我们将所有图都视为无向、无权重图。基本上，我们使用了五个较小的数据集来比较我们提出的方法和以往方法的性能，并分析这些方法的行为，同时使用了六个较大的数据集来展示我们提出的方法的可扩展性。表 4 列出了网络类型、顶点和边的数量。

7.1.1 详细说明

Gnutella这是根据 2002 年 8 月 Gnutella P2P 网络的快照绘制的图 [34]。

Epinions： 该图是 Epin-ions (www.epinions.com) 中的在线社交网络，每个顶点代表一个用户，每条边代表一种信任关系[33]。

Slashdot： 这是 2009 年 2 月获得的 Slashdot （slashdot.org）在线社交网络。顶点对应用户，边对应用户之间的朋友/朋友链接[23]。

NotreDame： 这是 1999 年收集的圣母大学（域名 nd.edu）网页之间的网络图 [5]。

WikiTalk： 这是维基百科编辑者之间的在线社交网络 (www.wikipedia.org)，截至 2008 年 1 月，该网络是通过讨论页上的编辑交流创建的[21, 20]。

Skitter： 这是一张互联网拓扑图，由 Skitter [22] 在 2005 年运行的跟踪路由创建而成。

印度： 这是 2004 年抓取的.in 域名页面之间的网络图 [9, 8]。

MetroSec： 这是一个由 MetroSec 捕获的互联网流量构建的图。每个顶点代表一台计算机，如果两个顶点分别作为发送方和目的地出现在数据包中，则这两个顶点会被链接起来 [24]。

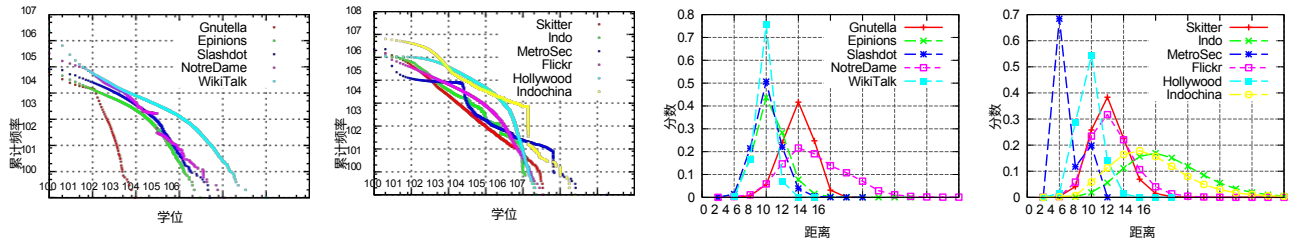
Flickr： 这是一个照片共享网站 Flickr（www.flickr.com ）中的在线社交网络[26]。

印度支那： 这是一张印度支那国家域网页的网络图，于 2004 年抓取[9, 8]。

好莱坞： 这是一个电影演员社交网络。如果两个演员在 2009 年之前共同出演过一部电影，他们就会被链接起来[9, 8]。

7.1.2 统计资料

首先，我们研究了网络的度分布，因为当我们使用度策略进行顶点排序时，顶点的度在我们的方法中起着重要作用。



(a) 较小的五个数据集的度数分布。 (b) 较大的六个数据集的度数分布。 (c) 较小的五个数据集的距离分布。 (d) 较大的六个数据集的距离分布。

图 2：数据集的属性。

表 3：在真实数据集上，建议方法与之前方法的性能比较。IT 表示索引时间，IS 表示索引大小，QT 表示查询时间，LN 表示每个顶点的平均标签大小。DNF 表示一天内未完成或内存耗尽。

数据集	经修剪的地标标签				分层枢纽标签法 [2]				树形分解 [4]			
	信息技术	IS	QT	LN	信息技术	IS	QT	LN	信息技术	IS	QT	外勤部
Gnutella	54 s	209 MB	5.2 微秒	644+16	245 s	380 MB	11 微秒	1,275	209 s	68 MB	19 微秒	3.2 毫秒
Epinions	1.7 s	32 MB	0.5 微秒	33+16	495 s	93 MB	2.2 微秒	256	128 s	42 MB	11 微秒	7.4 毫秒
Slashdot	6.0 s	48 MB	0.8 微秒	68+16	670 s	182 MB	3.9 微秒	464	343 s	83 MB	12 微秒	12 毫秒
圣母院	4.5 s	138 MB	0.5 微秒	34+16	10,256 s	64 MB	0.4 微秒	41	243 s	120 MB	39 微秒	17毫秒
WikiTalk	61 s	1.0 GB	0.6 微秒	34+16	DNF	-	-	-	2,459 s	416 MB	1.8 微秒	197 毫秒
Skitter	359 s	2.7 GB	2.3 微秒	123+64	DNF	-	-	-	DNF	-	-	190 毫秒
印度	173 s	2.3 GB	1.6 微秒	133+64	DNF	-	-	-	DNF	-	-	150 毫秒
MetroSec	108 s	2.5 GB	0.7 微秒	19+64	DNF	-	-	-	DNF	-	-	150 毫秒
图片库	866 s	4.0 GB	2.6 微秒	247+64	DNF	-	-	-	DNF	-	-	361 毫秒
好莱坞	15,164 s	12 GB	15.6 微秒	2,098+64	DNF	-	-	-	DNF	-	-	1.2 s
印度支那	6,068 s	22 GB	4.1 微秒	415+64	DNF	-	-	-	DNF	-	-	1.5 s

图 2a 和图 2b 是度数累积分布的对数图。不出所料，我们可以确认所有这些网络一般都呈现出幂律程度分布。

然后，我们还研究了距离的分布。图 2c 和 2d 显示了 1,000,000 随机顶点对的距离分布。从图中可以看出，这些网络也是小世界网络，即平均距离非常小。

7.2 性能

首先，我们介绍了我们的方法在真实世界数据集上的 性能，以显示我们的方法的效率和鲁棒性。表 3 显示了我们的方法在数据集上的性能。IT 表示预处理时间，IS 表示索引大小，QT 表示 1,000,000 次随机查询的平均查询时间，LN 表示每个顶点的平均标签大小，格式为正常标签大小（左）加上位并行标签大小（右）。我们将前五个数据集进行比特并行 BFS 的次数设为 16 次，其余数据集为 64 次。

在表 3 中，我们还列出了两种最先进的现有方法的性能。一种是*分层中心标注法*[2]，也是基于距离标注法。另一种是基于树分解的方法 [4]，它是 TEDI [41] 的改进版。对于以前的这些方法，我们使用了这些方法作者的实现，都是用 C++ 编写的。分层集线器标注的实验是在一台 Windows 服务器上进行的，该服务器配有两个英特尔至强 X5680（3.33GHz）和 96GB 主内存。基于树分解方法的实验在上述环境中进行。我们还描述了通过广度优先搜索计算距离所需的平均时间。

1,000 对随机顶点。在这四种方法（包括本文提出的方法）中，只有分层集线器标注[2]的预处理是并行的，使用了全部 12 个内核。其他计时结果均为顺序结果。

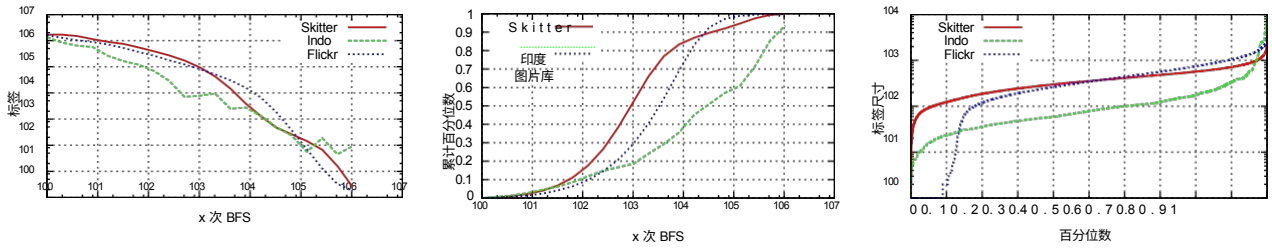
7.2.1 预处理时间和可扩展性

我们特别强调了预处理时间的大幅改进，从而大大提高了可扩展性。首先，我们成功地预处理了最大的两个数据集好莱坞和印度支那，这两个数据集拥有数百万个顶点和数亿条边，预处理时间适中。这比我们能处理的图大小提高了两个数量级，因为正如我们在表 1 中列出的，其他现有的精确距离查询方法需要数千或数万秒来预处理具有数百万条边的图。

对于接下来的四个有数千万条边的数据集，用时不到一千秒，而之前的方法在一天后仍未完成或内 存耗尽。对于较小的 6 个数据集，他们用时最多为 1 分钟，其中大部分数据集比以前的方法至少快 50 倍。

7.2.2 查询时间

平均查询时间一般为微秒，最多为 16 微秒。对于几乎所有较小的 5 个数据集，建议方法的查询时间都快于之前方法的查询时间。事实上，从表 1 中我们也可以看出，对于这些大小的图，我们方法的查询时间与所有现有方法相当。此外，我们还可以确认，对于较大的网络，查询时间不会增加太多。

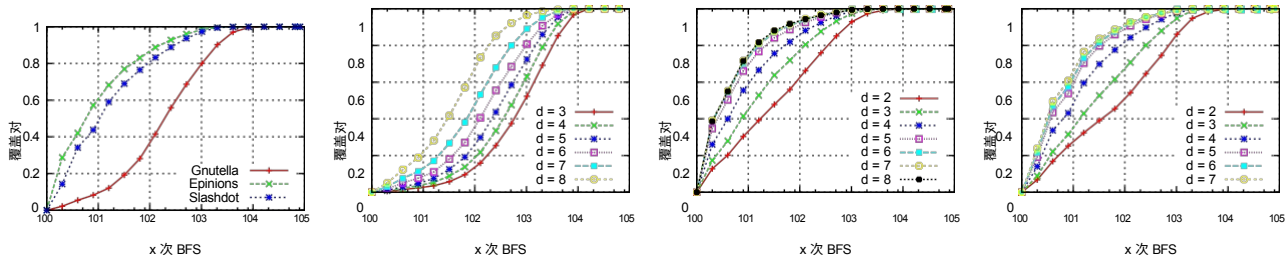


(a) 每个剪枝 BFS 中标注的顶点数量。

(b) 每个剪枝 BFS 中标注顶点数量的累积分布。

(c) 标签尺寸的分布。

图 3：剪枝和标签大小的影响。



(a) 平均

(b) 距离方面，Gnutella

(c) 距离方面，Epinions

(d) 距离方面，Slashdot

图 4：可通过索引回答其距离的顶点对的比例与已执行剪枝 BFS 的数量对比。

7.2.3 索引大小

至于较小的五个网络，结果表明我们的方法在索引大小方面与之前的方法相当。不过，尽管如今拥有数十 GB 内存的计算机既不罕见也不昂贵，但缩小索引大小仍是下一步研究的重要课题。

7.3 分析

接下来，我们将分析我们的方法的行为，以研究我们的方法为何高效。

7.3.1 经修剪的 BFS

首先，我们研究标签是如何计算和存储的。图 3a 显示了每个经过剪枝的 BFS 中添加到标签的距离数，图 3b 显示了其累积分布，即每一步之前存储的距离与最后存储的所有距离之比。在这些实验中，我们没有使用比特并行 BFS。

从这些图中，我们可以确认剪枝的 巨大影响。图 3a 显示，每次 BFS 中添加到标签上的距离数量都在迅速减少。例如，在进行了 1,000 次 BFS 后，所有三个数据集只有不到 10% 的顶点的 标签被添加了距离，而在进行了 10,000 次 BFS 后，所有三个数据集只有不到 1% 的顶点的标签被添加了距离。图 3b 还显示，大部分标签都是在开始时计算的。

7.3.2 标签尺寸

图 3c 显示了整个预处理后标签大小的分布情况，按大小从大到小排序。我们可以观察到，不同顶点的标签大小差别不大，很少有顶点的标签比平均值大得多。这说明我们的方法的查询时间相当稳定。

如果您担心有异常大的顶点，可以预先计算这些顶点与所有顶点之间的距离，然后直接回答，因为这类顶点的数量很少，如图 3c 所示。

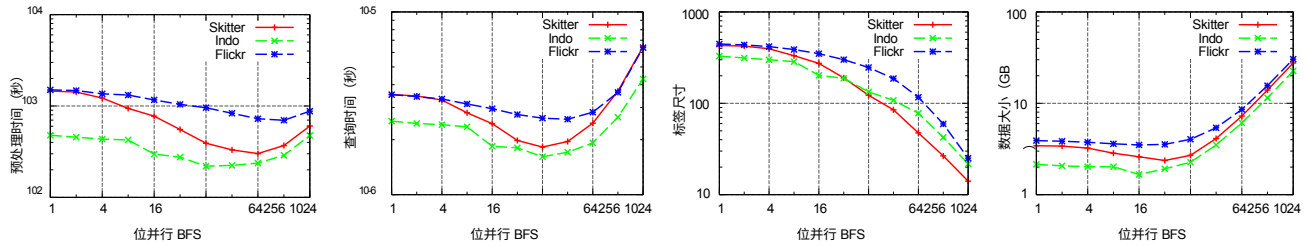
7.3.3 配对覆盖范围

图 4a 显示了每一步中被*覆盖的*顶点对（即当前标签能正确回答其距离的顶点对）的比率。我们使用了 1,000,000 个随机配对来估算这些比率。我们可以发现，大多数线对在一开始就被覆盖了。这表明，有很大一部分线对的最短路径只经过一小部分中心顶点，而这些中心顶点是由度策略选择的。这就是基于地标的近似方法具有良好精度的原因，也是我们的剪枝工作如此有效的原因。

图 4b、4c 和 4d 展示了每一步中按距离分类的顶点对的覆盖比例。从图中可以看出，远处的顶点对通常比近处的顶点对更早被覆盖。这就是为什么基于地标的近似方法对近距离顶点对的精确度远远低于对远距离顶点对的精确度。另一方面，我们的方法积极利用了这一特性：由于远距离的线对在一开始就被覆盖，我们可以在处理其他线对时剪除远距离的顶点，从而实现快速预处理。

7.3.4 顶点排序策略

接下来我们看看顶点排序策略的效果。第 5 章描述了使用第 4.4 节中描述的不同顶点排序策略时每个顶点标签的平均大小。在这些实验中，我们没有使用比特并行 BFS。正如我们所看到的，"度"（Degree）策略和 "接近度"（Closeness）策略的结果差别不大。Degree 策略可能略胜一筹。另一方面



(a) 预处理时间　(b) 查询时间　(c) 正常标签的平均尺寸　(d) 索引大小

图 5：性能与比特并行 BFS 数量的关系。

表 5：不同顶点排序策略下每个顶点标签的平均大小。

数据集	随机	学位	亲近
Gnutella	6,171	781	865
Epinions	7,038	124	132
Slashdot	8,665	216	234
圣母院	DNF	60	82
WikiTalk	DNF	118	158

随机策略的结果远不如其他两种策略。这说明我们可以通过度策略和接近度策略成功捕捉到中心顶点。

7.3.5　位并行 BFS

最后，我们来看看第 5 节中讨论的位并行 BFS 的效果。图 5 显示了我们的方法在进行不同次数的比特并行 BFS 时的性能。

图 5a 展示了预处理时间。图中显示，如果使用适当数量的比特并行 BFS，预处理时间会缩短 2 到 10 倍，从而进一步提高我们方法的可扩展性。图 5b 展示了查询时间。我们可以确认查询时间也变快了。图 5c 显示了每个顶点的法线平均大小。随着比特并行 BFS 数量的增加，比特并行 BFS 计算出的特殊标签覆盖了许多顶点对，正常标签的大小也随之减小。图 5d 显示了索引的大小。随着位并行 BFS 数量的增加，索引大小也会减小。

这些数据的另一个重要发现是，我们方法的性能对比特并行 BFS 的数量并不太敏感。如图所示，除非我们选择过大的数量，否则我们方法的性能不会变差太多。在不同的网络中，适当的参数似乎是通用的。因此，我们的方法仍然易于使用这种比特并行技术。

8.　结 论

在本文中，我们提出了一种新颖高效的方法，用于大型图上的精确最短路径距离查询。我们的方法基于对顶点进行距离标注，这在现有的精确距离查询方法中很常见，但我们的标注算法基于全新的理念。我们的算法通过剪枝从所有顶点进行广度优先搜索（BFS）。虽然算法简单，但我们的剪枝却出人意料地减少了搜索空间和标签，从而实现了快速的预处理时间、较小的索引规模和较快的查询时间。此外，我们还提出了另一种利用比特级并行性的标注方案，它可以很容易地与剪枝标注方法结合起来，以进一步

提高性能。在各种类型的大规模真实世界网络上的大量实验结果证明了我们方法的高效性和鲁棒性。特别是，我们的方法可以处理数亿个顶点的网络，这比以前方法的极限大两个数量级，而且索引大小和查询时间都相当可观。

我们计划研究处理更大图形的方法，因为主内存可能无法容纳索引和/或 图形。第一种方法是利用明显的部分和对称性缩小图，从而减小索引大小[30, 14]，并通过为最短路径树制作共同子树字典来压缩标签[1]。另一种方法是基于磁盘或分布式实现。正如我们在第 6 节中所说，基于磁盘的查询回答是显而易见的，也是现成的，其挑战尤其在于预处理。不过，由于我们的预处理算法是基于 BFS 的简单算法，因此我们可以利用现有的大量 BFS 研究成果。特别是，由于剪枝可以在本地完成，预处理算法将在基于 BSP 模型的分布式图处理平台上运行良好[25]。

9. 致谢

我们要感谢今井浩（Hiroshi Imai）对稿件的批判性阅读，感谢丹尼尔-戴林（Daniel Delling）为我们提供分层集线器标记的实验结果[2]。我们还要感谢匿名审稿人对论文提出的建设性改进意见。吉田雄一的研究得到了 JSPS 研究活动启动补助金（24800082）、文部科学省创新领域科学研究补助金（24106001）以及 JST、ERATO、Kawarabayashi Large Graph Project 的支持。

10. 参考文献

[1] I.Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck.道路网络中最短路径的基于集线器的标注算法。In *SEA*, pages 230-241, 2011.

[2] I.Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck.最短路径的分层集线器标签。In *ESA*, pages 24-35.2012.

[3] V.Agarwal、F. Petrini、D. Pasetto 和 D. A. Bader。多核处理器上的可扩展图探索。在 *SC 中*，第 1-11 页，2010 年。

[4] T.秋叶、C. 萨默和 K. 川林。复杂网络的最短路径查询：利用核心外的低树宽。在 *EDBT* 中，第 144-155, 2012.

[5] R.Albert, H. Jeong, and A. L. Barabasi.万维网的直径。*Nature*, 401:130-131, 1999.

[6] L.Backstrom、D. Huttenlocher、J. Kleinberg 和 X.Lan.大型社交网络中的群体形成：成员、成长与演化。In *KDD*, pages 44-54, 2006.

[7] S.Boccaletti, V. Latora, Y. Moreno, M. Chavez, and D.Hwang.复杂网络：结构与动力学。*物理报告*，424（4-5）：175-308，2006.

[8] P.Boldi, M. Rosa, M. Santini, and S. Vigna.分层标签传播：用于压缩社交网络的多分辨率无坐标排序。在 *WWW* 中，第 587-596 页，2011 年。

[9] P.Boldi 和 S. Vigna.网络图框架 I：压缩技术。在 *WWW* 中，第 595-602 页，2004 年。

[10] D.S. 卡拉韦、M. E. J. 纽曼、S. H. 斯特罗加茨和 D.J. Watts.网络鲁棒性与脆弱性：随机图上的周而复始。*Physical Review Letters*, 85:5468-5471, 2000.

[11] W.Chen, C. Sommer, S.-H. Teng, and Y. Wang.Teng, and Y. Wang.一种紧凑的路由方案和近似距离
幂律图的甲骨文。*Talg*, 9(1):4:1-26, 2012.

[12] J.Cheng 和 J. X. Yu.在线精确最短距离查询处理。在 *EDBT* 中，第 481-492 页，2009 年。

[13] E.科恩、E. 哈尔佩林、H. 卡普兰和 U. 兹威克。通过 2 跳标签进行可达性和距离查询在 *SODA*, 第 937-946 页，2002 年。

[14] W.W. Fan, J. Li, X. Wang, and Y. Wu.查询保留图压缩。*SIGMOD*》，第 157-168 页，2012 年。

[15] A.Gubichev, S. Bedathur, S. Seufert, and G. Weikum. 快速准确地估计大型图中的最短路径。In *CIKM*, pages 499-508, 2010.

[16] H.He, H. Wang, J. Yang, and P. S. Yu.眨眼：图上的有序关键词搜索。2007 年，*SIGMOD*，第 305-316 页。

[17] R.Jin, N. Ruan, Y. Xiang, and V. Lee.一种以高速公路为中心的标注方法，用于回答大型稀疏图上的距离查询。*SIGMOD*》，第 445-456 页，2012 年。

[18] C.Jordan.Sur les assemblages de lignes.*J. Reine Angew Math*, 70:185-190, 1869.

[19] D.Kempe, J. Kleinberg, and E. Tardos.通过社交网络最大化影响力传播。在 *KDD* 中，第 137-146 页，2003 年。

[20] J.Leskovec, D. Huttenlocher, and J. Kleinberg. 预测在线社交网络中的正面和负面链接。*WWW*, 第 641-650 页，2010 年。

[21] J.Leskovec, D. Huttenlocher, and J. Kleinberg.社交媒体中的签名网络。In *CHI*, pages 1361-1370、2010.

[22] J.Leskovec, J. Kleinberg, and C. Faloutsos.图形随时间变化：致密化规律、直径缩小与可能的解释。In *KDD*, pages 177-187, 2005.

[23] J.Leskovec, K. Lang, A. Dasgupta, and M. Mahoney. 大型网络中的群落结构：自然聚类大小和不存在定义明确的大型聚类。*互联网数学*》，6（1）：29-123，2009 年。

[24] C.Magnien、M. Latapy 和 M. Habib。快计算大规模图直径的经验紧界。*J. Exp.算法*，13:10:1.10-10:1.9，2009 年 2 月。

[25] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert、

I.Horn, N. Leiser, and G. Czajkowski.Pregel: a 大规模图形处理系统。2010年，*SIGMOD*，第135-146页。

[26] A.Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee.在线社交网络的测量与分析。In *IMC*, pages 29-42, 2007.

[27] M.E. J. Newman, S. H. Strogatz, and D. J. Watts.任意度分布的随机图及其应用。*物理评论 E*、64(2):026118 1-17, 2001.

[28] R.Pastor-Satorras and A. Vespignani.*互联网的演变与结构：统计物理学方法*。剑桥大学出版社，2004 年。

[29] M.Potamias, F. Bonchi, C. Castillo, and A. Gionis. 大型网络中的快速最短路径距离估计。2009年，*CIKM*，第867-876页。

[30] M.Qiao, H. Cheng, L. Chang, and J. X. Yu. 近似最短距离计算A 依赖查询的局部地标方案。In *ICDE*, pages 462-473, 2012.

[31] S.A. Rahman、P. Advani、R. Schunk、R. Schrader 和 D. Schomburg。代谢途径分析网络服务（立方体中的途径猎人工具）。*生物信息学*》，21（7）：1189-1193，2005 年。

[32] S.S. A. Rahman 和 D. Schomburg.观察代谢途径的局部和全局特性：代谢网络中的 "负荷点 "和 "堵塞点"。*生物信息学*》，22（14）：1767-1774，2006 年。

[33] M.Richardson, R. Agrawal, and P. Domingos.语义网络的信任管理。In *ISWC*, volume 2870, pages 351-368.2003.

[34] M.Ripeanu, A. Iamnitchi, and I. Foster.Gnutella 网络映射。*IEEE Internet Computing*、6(1):50-57, Jan. 2002.

[35] N.Robertson and P. D. Seymour.图形未成年人。III.平面树宽。*J. Comb.Theory, Ser*、36(1):49-64, 1984.

[36] L.Tang and M. Crovella.互联网虚拟地标。*SIGCOMM*, pages 143-152, 2003.

[37] T.Tran、H. Wang、S. Rudolph 和 P. Cimiano。图形 (RDF) 数据上高效关键词搜索的候选查询 Top-k 探索。In *ICDE*, pages 405-416, 2009.

[38] K.Tretyakov, A. Armas-Cervantes、L.Garc´ıa-Ban˜uelos, J. Vilo, and M. Dumas.超大图中基于全动态地标的最短路径距离快速估算。In *CIKM*, pages 1785-1794, 2011.

[39] A.Ukkonen, C. Castillo, D. Donato, and A. Gionis.用上下文信息搜索维基百科。In *CIKM*, pages 1351-1352, 2008.

[40] M.V. Vieira、B. M. Fonseca、R. Damazio、P. B. Golgher、D. d. C. Reis 和 B. Ribeiro-Neto。社交网络中的高效搜索排名。In *CIKM*, pages 563-572, 2007.

[41] F.Wei.Tedi：高效的图上最短路径查询应答。*SIGMOD*，第 99-110 页，2010 年。

[42] S.A. Yahia、M. Benedikt、L. V. S. Lakshmanan 和 J.Stoyanovich.协作标签网站中的高效网络感知搜索。*PVLDB*，1（1）：710-721，2008.