華中科技大學

本科生毕业设计（论文）参考文献译文本

译文出处：

Takuya Akiba, Yoichi Iwata, Yuichi Yoshida. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. Proceedings of SIGMOD 2013.

院　　系 计算机科学与技术学院

专业班级 信息安全 1301 班

姓　　名 王梦鸽

学　　号 U201315120

指导教师 陈汉华

2017 年 3 月

译文要求

一、 译文内容须与课题（或专业内容）联系，并需在封面注明详细出处。

二、 出处格式为

图书：作者.书名.版本（第×版）.译者.出版地：出版者，出版年.起页～止页

期刊：作者.文章名称.期刊名称，年号，卷号（期号）：起页～止页

三、 译文不少于5000汉字（或2万印刷符）。

四、 翻译内容用五号宋体字编辑，采用A4号纸双面打印，封面与封底采用浅蓝色封面纸（卡纸）打印。要求内容明确，语句通顺。

五、 译文及其相应参考文献一起装订，顺序依次为封面、译文、文献。

六、 翻译应在第七学期完成。

译文评阅

导师评语

应根据学校“译文要求”，对学生译文翻译的准确性、翻译数量以及译文的文字表述情况等做具体的评价后，再评分。

评分：_____（百分制）　　指导教师(签名)：_____

年　　月　　日

大规模网络中基于剪枝标记的快速最短路径距离查询

摘要

我们提出了一个新的精确方法在大规模网络的最短路径距离查询。我们的方法预计算顶点的距离标签从每个顶点进行广度优先搜索。乍一看过于明显和效率太低，这里介绍的关键因素是修剪广度优先搜索。虽然我们仍然可以回答从标签的任何顶点对正确的距离，它惊人地减少了搜索空间和大小的标签。此外，我们表明，我们可以执行 32 或 64 的宽度优先搜索的同时利用位操作。我们的实验表明，COM 这两种技术的组合是各种大规模真实网络的有效性和鲁棒性。在特别大，我们的方法可以处理社会网络和数以百万计的边缘网络图，这是两个数量大于以往的精确方法的限制级，具有可比性的查询时间以前的方法。

类别和主题描述符

译文：[数据]数据结构图和网络的一般条款

算法，实验，性能

关键词

图，最短路径，查询处理

1 简介

距离查询要求图中两个顶点之间的距离。毫无疑问，应答距离查询是图上最基本的运算之一，具有广泛的应用范围。例如，在社会网络工程，两个用户之间的距离被认为是表示亲近，并用于社会敏感的搜索，以帮助用户找到更多的相关用户或内容[40,42]，或分析影响人们和社区[19,6]。在网络图，网页之间的距离是一个指标的相关性，并使用上下文感知搜索得到更高的排名的网页更关系到目前访问网页[39,29]。远程查询-其他应用包括 top-k 关键字查询关联数据[16,37]，在代谢网络之间的化合物的最佳途径的作品[31,32]发现，在计算机网络[28,7] 资源管理。

当然，我们可以使用广度优先搜索（BFS）或 Dijkstra 算法来计算每个查询的距离。 然而，对于大图形来说，它们需要超过一秒钟，这对于用作这些应用程序的构建块太慢了。 特别地，诸如社交敏感搜索或上下文感知搜索之类的应用应该具有低延迟，因为它们涉及用户之间的实时交互，而它们需要在多对顶点之间的距离以对于每个搜索查询对项目进行排序。因此，应该更快地回答距离查询，例如微秒。

另一种极端方法是预先计算所有顶点对之间的距离，并将它们存储在索引中。 虽然我们可以即时回答距离查询，但这种方法也是不可接受的，因为预处理时间和索引大小是二次

的且不切实际的大。由于巨大的图形数据的出现，在这两种极端方法之间设计更为中性和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个主要的图类：一个是道路网络，另一个是复杂的网络，如社会网络，网络图，生物网络工程和计算机网络。道路网络，因为它更容易掌握和利用它们的结构，研究已经非常成功。现在，道路网络的距离查询可以在不到一微秒的时间内处理完整的美国公路网[1]。

相比之下，在复杂网络上回答距离查询仍然是一个非常具有挑战性的问题。 道路网络的方法在这些网络上表现不佳，因为它们的结构完全不同。对于这些网络已经提出了几种方法，但是它们具有可扩展性差的缺点。他们花费至少几千秒或几万秒来索引具有数百万边缘的网络[41,4,2,17]。

为了处理更大的复杂网络，除了这些确切的方法，近似方法也被研究。也就是说，我们不总是必须回答正确的距离。它们在随机查询的更好的可扩展性和非常小的平均相对误差方面是成功的。然而，这些方法中的一些需要毫秒来回答查询[15,38,30]，这比其他方法慢约三个数量级。一些其他方法在微秒内回答查询[29,40]，但据报道这些方法对于接近的顶点对的精度不高[30,4]。 这个缺点可能对于诸如社交敏感的搜索或上下文感知搜索的应用是至关重要的，因为在这些应用中，距离查询被用于区分关闭项目。

1.1 我们的贡献

为了解决这些问题，在本文中，我们提出了一种新的方法来回答复杂网络中的距离查询。该方法是一种精确的方法。也就是说，它总是回答准确的查询距离。它比以前的精确方法具有更高的可扩展性，可以处理数以百万计的边的图形。然而，查询时间非常短，大约十微秒。虽然我们的方法可以处理指示和/或加权图作为我们提到后，在下面的内容中，我们假设无向加权图，简单的论述。

我们的方法是基于距离标记或距离感知的 2 跳盖的概念。 2 跳盖的想法如下。 对于每个顶点 u，我们拾取候选顶点集合 C（u），使得每对顶点（u，v）在最短路径上具有至少一个顶点 w∈C（u）∩C 对。 对于每个顶点 u 和顶点 w∈C（u），我们预先计算它们之间的距离 dG（u，w）。 我们说集合 L（u）= {（w，dG（u，w））}w∈C（u）是 u 的标签。 使用标签，清楚的是，两个顶点 u 和 v 之间的距离 dG（u，v）可以计算为 min { δ + δ '| （w，δ）∈L（u），（w， δ'）∈L（v）}。 标签族{L（u）}被称为 2 跳盖。 距离标记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和不同的方法来计算标记，称为修剪

的地标标记。

我们的方法的想法是简单而相当激进：从每个顶点，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。当然，如果我们天真地实现这个想法，我们需要 O（nm）预处理时间和 O（n2）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。我们使这个方法实际的关键思想是在广度优先搜索期间修剪。设 S 是一组顶点，假设我们已经有标签，它们可以回答两个顶点之间的正确距离，如果它们之间的最短路径通过 S 中的顶点。假设我们从 v 进行 BFS 并访问 u。如果有一个顶点 w∈S 使得 dG（v，u）= dG（v，w）+ dG（w，u），那么我们修剪 u。也就是说，我们不会从 u 中遍历任何边。正如我们在 4.3 节中证明的，在从 v 中删除 BFS 之后，标签可以回答两个顶点之间的距离，如果它们之间的最短路径通过 S∪{v}中的顶点。

有趣的是，我们的方法结合了三种不同的以前的成功方法的优点：基于地标的近似方法[29,38,30]，基于树分解的精确方法[41,41]和基于标签的精确方法[13，12 ，2]。基于地标的近似方法通过利用复杂网络中的高度中心顶点的存在来实现可重复的精度[29]。这个事实也是我们修剪的力量的主要原因：通过首先从这些中心顶点进行广度优先搜索，之后我们可以彻底修剪广度优先搜索。基于树分解的方法通过分解低树宽的树状条纹来利用网络的核心结构[10,27]。虽然我们的方法没有明确使用树分解，我们证明我们的方法可以有效地处理小树宽的图。这个过程表明我们的方法也利用核心 - 边缘结构。与其他基于标签的方法一样，我们的索引的数据结构很简单，并且由于存储器访问的局部性，查询处理非常快。

虽然这个修剪的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强度的标记方案，并将它们组合起来以进一步提高性能。也就是说，我们表明，通过宽度优先搜索的标记可以以位并行方式实现，其利用寄存器字中的位 b 的数量通常为 32 或 64 的属性，并且我们可以执行位操作这些 b 位同时。通过这种技术，我们可以在 O（m）时间同时从 b + 1 个顶点执行 BFS。在开始时，这种位并行标记（不修剪）比修剪的地标标记更好，因为修剪不会发生很多。注意，我们不是在讨论线程级并行性，我们的位并行化实际上减少了计算复杂度的因子 b + 1。除了这两种标记方案之外，我们还可以使用线程级并行性。

正如我们在实验结果中所证实的，我们的方法优于其他用于精确距离查询的最先进的方法。特别地，它具有比以前的方法明显更好的可扩展性。对于具有数百万个边缘的索引网络，只花费了几十秒钟。这种分度时间比以前的方法快两个数量级，花费至少几千秒甚至一天以上。此外，我们的方法成功地处理具有数亿个边缘的网络，其比之前在精确方法的实验中使

用的网络大两个数量级。对于大小相同的网络，查询时间也优于先前的方法，并且我们确认查询时间对于网络的大小不会快速增加。我们还确定了我们的方法的索引的大小与其他方法相当。

表 1 以前的方法和精确的距离查询的方法的实验结果总结

Method	Network	$\|V\|$	$\|E\|$	Indexing	Query
TEDI [41]	Computer	22 K	46 K	17 s	4.2 μs
	Social	0.6 M	0.6 M	2,226 s	55.0 μs
HCL [17]	Social	7.1 K	0.1 M	1,003 s	28.2 μs
	Citation	0.7 M	0.3 M	253,104 s	0.2 μs
TD [4]	Social	0.3 M	0.4 M	9 s	0.5 μs
	Social	2.4 M	4.7 M	2,473 s	0.8 μs
HHL [2]	Computer	0.2 M	1.2 M	7,399 s	3.1 μs
	Social	0.3 M	1.9 M	19,488 s	6.9 μs
PLL (this work)	Web	0.3 M	1.5 M	**4 s**	0.5 μs
	Social	2.4 M	4.7 M	**61 s**	0.6 μs
	Social	1.1 M	**114 M**	15,164 s	15.6 μs
	Web	7.4 M	**194 M**	6,068 s	4.1 μs

在表 1 中，我们总结了我们的实验结果和这些论文中提出的以前的确切方法的结果。我们列出了每篇论文中最大的两个现实世界复杂网络的结果。 在我们的实验中，我们进一步比较我们的方法与分层集线器标签[2]和基于树分解的方法[4]。

在第 2 节中，我们描述了有关的确切和近似距离查询的成果。在第 3 节中，我们给出了本文使用的定义和概念。第 4 节致力于解释我们的第一个计划，修剪地标标记。我们解释我们的第二个方案，位并行标记，在第 5 节。在第 6 节中，我们提到的变体的距离查询，我们可以通过稍微修改我们的方法处理。我们在第 7 节解释我们的实验结果，并在第 8 节结束。

2 相关的工作

2.1 精确方法

对于复杂的网络，如社会网络和网络图的精确距离查询，最近提出了几种方法。

这些方法的大部分可以被认为是基于 2 跳盖的想法[13]。 寻找小的 2 跳覆盖有效是一个具有挑战性和长期存在的问题[13,12，2]。 最新的方法之一是分层集线器标签[2]，其基于道路网络的方法[1]。 另一种与 2 跳覆盖有关的最新方法是公路中心平衡[17]。 在这种方法中，我们首先计算生成树 T 并将其用作“高速公路”。 也就是说，当计算两个顶点 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 w1 和 w2 是顶点 分别在 u 和 v 的标签中，dT（ •，•）是生成树 T 上的距离度量。

基于树分解的方法也被报道是有效的[41,42]。 图 G 的树分解是树 T，每个顶点与 G 中

的一组顶点相关联，称为 bag [35]。 此外，包含 G 中的顶点的袋子集合在 T 中形成在 T 中的连通分量。其启发式地计算树分解并存储每个袋子的最短距离矩阵。 计算与此信息的距离并不难。 最大袋的尺寸越小，该方法效率越高。 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袋子和许多小袋子，最大的袋子的大小虽然不小但是适中的。

2.2 近似方法

为了获得比这些确切方法更大的可扩展性，还研究了不总是回答正确距离的近似方法。

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顶点的子集 L 作为界标，并预先计算每个界标 l∈L 和所有顶点 u∈V 之间的距离 dG（l，u）。 当查询两个顶点 u 和 v 之间的距离时，我们回答在地标 l∈L 上的最小 dG（u，l）+ dG（l，v）作为估计。通常，每个查询的精度取决于实际最短路径是否在地标附近通过。 因此，通过选择中心顶点作为地标，估计的准确性比选择地标的随机性更好[29,11]。 然而，对于接近的对，精度仍然比平均值差很多，因为它们之间的最短路径的长度很小，并且它们不太可能通过附近的地标[4]。

为了进一步提高测量精度，几项技术被提出[15,38,30]。它们通常存储以地标为根的最短路径树，而不是仅存储距地标的距离。 为了回答查询，它们从最短路径树中提取路径作为最短路径的候选，并通过找到循环或快捷方式来改进它们。 虽然它们显着提高了精度，但查询时间变慢了三个数量级。

3.前言

3.1 符号

表 2 列出了本文中经常使用的符号。在本文中，我们主要关注被建模为图形的网络。令 G =（V，E）是具有顶点集合 V 和边集合 E 的图形。我们使用符号 n 和 m 来表示顶点数目| V | 和边| E |的数量，当图从上下文中清楚时。 我们还将 G 的顶点集合表示为 V（G），将 G 的边缘集合表示为 E（G）。 我们用 NG（v）表示顶点 v∈V 的邻居。 也就是说，NG（v）= {u∈V| （u，v）∈E}。

让 DG（u，v）表示顶点之间的距离 u，v 如果 u 和 V 断开 G，我们定义 DG（u，v）= H。图的距离是一个度量，满足三角不等式。也就是说，对于任意三个顶点，T 和 V，

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

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

我们定义 PG（s，t）⊆V 作为顶点 s 和 t 之间的最短路径上所有顶点的集合。 换一种说法，

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

3.2 问题的定义

本文研究了以下问题：给定一个图 G，构造一个索引，有效地回答距离查询，它要求能回答任意一对顶点之间的距离。

简单的论述，我们主要考虑无向无权图。然而，我们的算法可以轻松地扩展到有向和/或加权图，我们在第 6 节讨论这个扩展。此外，我们的方法不仅可以回答距离，也可以回答最短路径。这个扩展也在第 6 节中讨论。

3.3 标签和 2 跳盖

2 跳盖的一般框架[13,12,2]，或有时称为标记方法如下。 我们的方法也遵循这个框架。

对于每个顶点 v，我们预先计算标记为 L（v）的标签，其是一对对（u， δ uv），其中 u 是顶点， δ uv= dG（u，v）。 我们有时将这组标签{L（v）}v∈V 作为索引。 为了回答顶点 s 和 t 之间的距离查询，我们计算并回答如下定义的 Query（s，t，L）

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

如果 L（s）和 L（t）不共享任何顶点，我们定义 Query（s，t，L）=∞。 如果 Query（s，t，L）= dG（s，t）对于任何顶点对 s 和 t，我们称为 L a（距离感知）2 跳盖。

对于每个顶点 v，我们存储标签 L（v），使得其中的对按其顶点排序。 然后，我们可以使用 merge-join-like 算法在 O（| L（s）| + | L（t）|）时间计算 Query（s，t，L）。

4 算法描述

4.1 朴素剪枝标记法

我们从以下 naive 方法开始。作为指数，我们从每个顶点进行 BFS 并存储所有对之间的距离。虽然这种方法太明显和低效，对于下一个方法的阐述，我们解释细节。

令 V = {v1，v2，...，vn}。我们从空索引 L0 开始，其中对于每个 u∈V，L0（u）=∅。假设我们按照 v1，v2，...的顺序从顶点开始 BFS。 。 。 ，vn。在从顶点 vk 开始的第 k 个 BFS 之后，我们增加从 vk 到到达顶点的标签的距离，即 Lk（u）= Lk-1（u）∪{（vk，dG（vk，u））} for 每个 u∈V 与 dG（vk，u）≠∞。我们不改变未知顶点的标签，即对于每个具有 dG（vk，u）=∞的 u∈V，Lk（u）= Lk-1（u）。

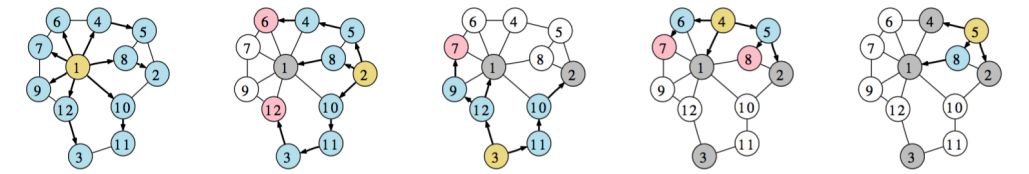
Ln 是最终索引。显然，对于任何顶点对 s 和 t 的查询（s，t，Ln）= dG（s，t），因此 Ln 是用于精确距离查询的正确的 2 跳覆盖。这是因为，如果 s 和 t 可达，则例如（s，0）∈Ln（s）和（s，dG（s，t））∈Ln（t）。

这种方法可以被认为是基于地标的近似方法的变体，我们在 2.2 节中提到。基于标准地标的方法可以被视为预先计算 L1 而不是 Ln 并且通过 Query（s，t，L1）估计 s 和 t 之间的距离的方法，其中 ln 是表示地标数量的参数。

4.2 修剪地标标记

然后，我们引入修剪到天真的方法。与上述方法类似，我们按照 v1，v2，... vn 的顺序从顶点开始修剪 BFS。我们从空索引 L'0 开始，并使用从第 t 个修剪的 BFS 获得的信息从 L'k-1 创建索引 L'k。

我们修剪 BFS 如下。假设我们有一个索引 Lk-1，我们从 vk 进行一个 BFS 以创建一个新的索引 L'k。假设我们访问一个距离为 δ 的顶点 u。如果 Query（vk，u，L'k-1）≤ δ，则我们修剪 u，也就是说，我们不对 L'k（u） 'k-1（u)），我们不从顶点 u 遍历任何边。否则，我们设置 L'k（u）= L'k-1（u）∪{（vk， δ）}并像往常一样从顶点 u 遍历所有边。与前面的方法一样，我们还针对在第 k 个剪枝 BFS 中没有访问的所有顶点 u∈V 设置 L'k(u)= L'k-1（u)。执行修剪的 BFS 的该算法被描述为算法 1，并且整个预处理算法被描述为算法 2。



(a) First BFS from vertex 1. We visited all the vertices.

(b) Second BFS from vertex 2. We did not add labels to five vertices.

(c) Third BFS from vertex 3. We only visited the lower half of the vertices.

(d) Fourth BFS from vertex 4. This time we only visited the higher half.

(e) Fifth BFS from vertex 5. The search space was even smaller.

图 1 修剪分别为例。黄色的顶点表示根，蓝色的顶点表示那些我们访问和标记，红色的顶点表示那些我们访问，但修剪，灰色顶点表示那些已被用作根。

图 1 显示了修剪的 BFS 的示例。来自顶点 1 的第一个剪枝 BFS 访问所有顶点（图 1a)。在从顶点 2（图 1b）的下一个剪枝 BFS 期间，当我们访问顶点 6 时，由于 Query（2,6，L'1）= dG（2,1）+ dG（1,6）= 3 = dG ，6)，我们修剪顶点 6，我们不从它的边缘。我们还修剪顶点 1 和 12.随着执行的 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)。

Algorithm 1 Pruned BFS from $v_k \in V$ to create index L'_k.

```
1: procedure PRUNEDBFS(G, v_k, L'_{k-1})
2:     Q ← a queue with only one element v_k.
3:     P[v_k] ← 0 and P[v] ← ∞ for all v ∈ V(G) \ {v_k}.
4:     L'_k[v] ← L'_{k-1}[v] for all v ∈ V(G).
5:     while Q is not empty do
6:         Dequeue u from Q.
7:         if QUERY(v_k, u, L'_{k-1}) ≤ P[u] then
8:             continue
9:         L'_k[u] ← L'_{k-1}[u] ∪ {(v_k, P[v_k])}
10:        for all w ∈ N_G(v) s.t. P[w] = ∞ do
11:            P[w] ← P[u] + 1.
12:            Enqueue w onto Q.
13:    return L'_k
```

Algorithm 2 Compute a 2-hop cover index by pruned BFS.

```
1: procedure PREPROCESS(G)
2:     L'_0[v] ← ∅ for all v ∈ V(G).
3:     for k = 1, 2, ..., n do
4:         L'_k ← PRUNEDBFS(G, v_k, L'_{k-1})
5:     return L'_n
```

4.3 正确性证明

在下面，我们证明这个方法计算任何一对顶点 s 和 t 的正确的 2 跳覆盖索引，即 Query（s，t，Ln）= dG（s，t）。

定理 4.1。对于任何 0≤k≤n 并且对于任何顶点对 s 和 t，Query（s，t，L'k）= Query（s，t，Lk）。

证明。我们通过数学 inductonk 证明了定理。由于 L'0 = L0，itistruefork = 0。 Nowwe 假设它保持 0,1，...，k-1 并且证明它也适用于 k。

令 s，t 是一对顶点。我们假设这些顶点在 G 中是可达的，因为否则可以明显地获得答案∞。令 j 是使得（vj， δ vjs）∈Lk（s），（vj， δ vjt）∈Lk（t）和 δ vjs+ δ vjt= Query（s，t，Lk）的最小数。我们证明（vj， δ vjs）和（vj， δ vjt）也包括在 L'k（s）和 L'k（t）中。这立即导致 Query（s，t，L'k）= Query（s，t，Lk）。由于 s 和 t 之间的对称性，我们证明（vj， δ vjs）∈L'k（s）。

首先，对于任何 i <j，我们通过矛盾证明 vi∉PG（vj，s）。如果我们假设 vi∈PG（vj，s），从不等式 1

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

由于（vi，dG（s，vi））∈Lk（s）和（vi，dG（t，vi））∈Lk（t），这与 j 的最小值的

假设相矛盾。 因此，对于任何 i <j，vi∈PG（vj，s）成立。

现在我们证明（vj，dG（vj，s））∈Lk（s）。 实际上，我们证明一个更一般的事实：（vj，dG（vj，u））∈L'k（u）对于所有 u∈PG（vj，s）。 注意 s∈PG（vj，s）。 假设我们从 vj 进行第 j 个剪枝的 BFS 以创建 Lj。 令 u∈PG（vj，s）。 由于对于任何 i <j，PG（vj，u）∈PG（vj，s）和 vi∈∈PG（vj，s），所以对于任何 i <j，我们有 PG（vj，u）。 因此，Query（vj，u，Lj-1）> dG（vj，u）成立。 因此，我们访问所有顶点 u∈PG（vj，s）而不修剪，并且随后是（vj，dG（vj，u））∈L'j（u）⊆L'k（u）。

作为推论，通过实例化定理 k = n，我们的方法被证明是一个精确的距离查询方法。

推论 4.1 对于任何一对顶点 s 和 t，

$$\textsc{Query}(s,t,L'_n) = d_G(s,t)$$

4.4 顶点排序策略

4.4.1 动机

在上面的算法描述，我们从顶点分别为 V1，V2，。..，VN 进行了修剪。我们可以自由选择，而且 BFS 的顺序是该方法的性能至关重要，正如我们将看到在第 7.3.4 实验结果。

为了决定顶点的顺序，我们应该在许多最短路径通过这些顶点的意义上首先选择中心顶点。 由于我们希望尽可能多地修剪后面的 BFS，因此我们希望通过更早的 BFS 覆盖更大部分的顶点对。 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许多最短路径通过的顶点。

这个问题与为基于地标的近似方法选择好的地标的问题非常相似，这在[29]中有很好的讨论。 在这个问题中，我们还要选择好的地标，以便许多最短路径通过这些顶点或附近的顶点。

4.4.2 策略

基于地标方法的结果[29]，我们提出并研究以下三个策略。 在实验中，我们基本上使用 Degree 策略，并在第 7.3.4 节中根据经验进行比较。

随机：我们随机排序顶点。 我们使用这种方法作为基线显示其他策略的意义。

度：我们从具有较高程度的那些订单顶点。 这个策略背后的想法是，具有更高程度的顶点与许多其他顶点具有更强的连接，因此许多最短路径将通过它们。

4.5 执行效率

4.5.1 预处理（算法 1）

索引：首先，在上面的描述中，我们分别处理 L'k-1 和 L'k，并解释为好像为了简化解释将 L'k-1 复制到 L'k。但是，通过只保留一个索引并在每个剪枝的 BFS 之后向其添加标签，可以容易地避免此副本。

初始化：另一个重要的注意事项是避免每次修剪的 BFS 的 O（n）时间初始化。这种方法的高效的原因是修剪的 BFS 的搜索空间比整个图小得多。然而，如果我们花 O（n）时间进行初始化，它将是瓶颈。我们在初始化中要做的是将存储临时距离的数组中的所有值设置为∞（第 3 行）。我们可以避免 O（n）时间初始化如下。在我们进行第一次修剪的 BFS 之前，我们将数组 P 中的所有值设置为∞。 （这需要 O（n）时间，但是我们只做一次。）然后，在每个修剪的 BFS 期间，我们存储我们访问的所有顶点，并且在每个修剪的 BFS 之后，对于所有每个顶点 v 设置 P [v]已访问。

数组：对于存储临时距离的数组，最好使用 8 位整数。由于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，该阵列适合于最近计算机的低级高速缓存存储器，从而通过减少高速缓存缺失而加速。

查询：为了评估修剪的查询（第 7 行），使用与正常算法不同的算法更快，因为我们可以利用这里的事实，我们发出其一个端点总是 vk 的许多查询。在从 vk 开始第 k 个删减 BFS 之前，我们准备长度为 n 的数组 T，用∞初始化，并对所有（w， δ wvk）∈L'k-1（vk）设置 T [w] = δ wvk。 ToevaluateQuery（vk，u，L'k-1），forall（w， δ wu）∈L'k-1（u），我们计算 δ wu+ T [w]并返回最小值。虽然正常查询算法需要 O（| Lk-1（vk）| + | L'k-1（u）|）时间，该算法运行在 O 由于第 7 行是算法的瓶颈，这种技术加速预处理大约两次。注意 T 应该

由于与上述相同的原因由 8 位整数表示，并且对于阵列 P 应当以相同的方式避免阵列 T 的 O（n）时间初始化。

预取：不幸的是，我们不能将索引和邻接列表放入大型网络的缓存内存中。然而，我们可以手动预取它们以减少高速缓存未命中，因为我们很快将访问的顶点在队列中。手动预取可将预处理速度提高约 20％。

线程级并行性：与并行 BFS 算法[3]一样，修剪的 BFS 算法也可以并行化。然而，对于简单的实验分析和与以前的方法公平的比较，我们没有并行实验在实验中。

排序标签：当应用合并连接类算法来回答查询时，标签中的对需要按照语法进行排序。然而，实际上我们不需要通过存储顶点而不是顶点的排序显式排序。也就是说，当从顶点 u

在第 i 个剪枝 BFS 中添加一对（u，δ）时，我们添加一对（i，δ）。然后，由于从低等级的顶点到高等级的顶点添加对，所以所有标签被自动地排序。

4.5.2 查询

Sentinel：我们为每个 v∈V 的标签 L（v）添加一个虚项（n，∞）。这个虚项保证了当扫描两个标签时，我们在结束时找到相同的顶点 n。因此，我们可以避免单独测试我们是否已经扫描到结束。

数组：对于每个标签 L（v），分别存储顶点数组和距离数组更快，因为只有当顶点匹配[1]时才使用距离。我们还将数组与高速缓存行对齐。

4.6 理论性质

4.6.1 极小

定理 4.2。令 L'n 为第 4.2 节中定义的索引。在任何顶点 v 和任何对（u，δuv）∈L'n（v）的情况下，存在一对顶点（s，t），使得如果我们去除，δuv）从 L'n（v），我们不能回答 s 和 t 之间的正确距离。

证明。令 vi∈V 和（vj，δvjvi）∈L'n（vi）。这意味着 j <i。我们表明，如果我们从 L'n（vi）中删除（vj，δvjvi），那么我们不能回答 vi 和 vj 之间的正确距离。我们声称，对于任何 k≠ j，或者（i）（vk，δvkvi）∉L'n（vi）或（vk，δvkvj）∉L'n vi，vk）+ dG（vk，vj）> dG（vi，vj）。假设 k <j 并且假设（ii）不成立。然后，（i）必须保持，因为否则第 j 个 BFS 应该已经修剪顶点 vi 和（vj，δvjvi）∈∈Ln（vi）。假设 k> j 并且假设（ii）不存在。然后，vk∈PG（vi，vj），因此（vj，δvjvk）∈Lj（vk），因此第 k 个 BFS 修剪顶点 vj，导致（vk，δvkvj）∉L'n

4.6.2 利用高中心顶点的存在

然后，我们比较我们的方法与基于地标的方法，以表明我们的方法也可以利用高度中心顶点的存在。我们考虑基于标准地标的方法[29,40]，它不使用任何路径启发式。正如我们在 2.2 节中所说，通过选择中心顶点

地标，它获得了显着的平均精度的现实世界的网络。从以下定理，我们可以观察到，我们的方法对于其距离可以由具有这样高精度的基于地标的方法来回答的网络是有效的，并且我们的方法也可以利用这些中心顶点的存在。

定理 4.3。如果我们假设标准的基于地标的近似方法可以使用 k 个地标回答到（1-ε）n2 对（n2 个对中）的正确距离，则修剪的地标标记方法给出具有平均标签大小 O（k + εn）。

证明草图。在首先从 k 个地标顶点进行修剪的 BFS 之后，最多将 ε n2 对总共添加到索引，因为我们从不添加其距离可以从当前标签回答的对。

4.6.3 利用小树枝宽度

最后，我们展示了一个理论证据，我们的方法也可以有效地利用树状条纹。正如我们在 2.1 节提到的，基于树分解的方法被提出用于距离查询[41,4]。它们都扩展了对于小树宽的图形有效工作的方法，并且它们通过树分解在现实世界网络中利用低树宽的条纹。有趣的是，尽管我们不明确使用树分解，我们可以证明我们的方法可以有效地处理小树宽的图。因此，我们的方法隐含地利用了现实世界网络的这种特性。关于树宽和树分解的定义，见[35]。

定理 4.4 令 w 是 G 的树宽。有修剪的地标标记方法使用的顶点的顺序，用于预处理的 O（wm log n + w2n log2 n）时间，存储具有 O（wnlogn）空间的索引，并在 O（w log n）时间内回答每个查询。

证明草图。关键成分是树分解的中心解构[18]。首先，我们从一个质心袋中的所有顶点开始修剪 BFS。然后，以后修剪的 BFSs 永远不会超越包。因此，我们可以考虑将树分解成不相交的分量，每个分量至多有一半的行李。我们递归地重复这个过程。复现的数目至多为 O（log n）。由于我们在每个递归深度向每个顶点添加至多 w 对，因此每个标签中的对的数量是 O（wlogn）。在每个深度处，修剪的 BFS 从当前分量消耗的总时间是 O（wm + w2n log n），其中 O（wm）是穿越边缘的时间，O（w2nlogn）是修剪的时间测试。

5.并行标签

为了进一步加快预处理和查询，我们提出了一种利用位级并行的优化方法。位并行方法是对同一字中的不同位执行不同计算的方法，以利用计算机可以一次对一个字执行按位操作。字长一般是 32 或 64 在一天的电脑。

在下文中，我们将计算机字中的位数表示为 b，并且假设可以在 O（1）时间内对长度为 b 的位向量进行按位操作。我们建议在 O（m）时间同时执行 BFS 和计算标签从 b + 1 根的算法。此外，我们还提出了一种通过 O（1）时间中的这些 b + 1 个顶点之一来回答任何一对顶点的距离查询的方法。

Algorithm 3 Bit-parallel BFS from $r \in V$ and $S_r \subseteq N_G(r)$.

```
1: procedure 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_r^0[r]) ← (0, ∅, ∅)
4:     (P[v], S_r^{-1}[v], S_r^0[v]) ← (1, {v}, ∅) for all v ∈ S_r
5:     Q_0, Q_1 ← an empty queue
6:     Enqueue r onto Q_0
7:     Enqueue v onto Q_1 for all v ∈ S_r
8:     while Q_0 is not empty do
9:         E_0 ← ∅ and E_1 ← ∅
10:        while Q_0 is not empty do
11:            Dequeue v from Q_0.
12:            for all u ∈ N_G(v) do
13:                if P[u] = ∞ ∨ P[u] = P[v] + 1 then
14:                    E_1 ← E_1 ∪ {(v, u)}
15:                    if P[u] = ∞ then
16:                        P[u] ← P[v] + 1
17:                        Enqueue u onto Q_1.
18:                else if P[u] = P[v] then
19:                    E_0 ← E_0 ∪ {(v, u)}
20:        for all (v, u) ∈ E_0 do
21:            S_r^0[u] ← S_r^0[u] ∪ S_r^{-1}[v]
22:        for all (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 and Q_1 ← ∅
26:    return (P, S_r^{-1}, S_r^0)
```

5.1 位并行标签

为了描述预处理算法和查询算法，我们首先定义我们存储在索引中的内容。

正如我们在下一个小节中所解释的，我们从顶点 r 和其最大大小为 b 的邻居的子集 $S_r \subseteq N_G(r)$进行位并行 BFS。 我们定义

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

由于对于任何顶点 u∈Sr，Sr 中的顶点是 r 的邻居和任何顶点 v∈V，$|d_G(u,v) - d_G(r,v)| \leq 1$ 因此，对于每个 v∈V，Sr 可以被划分为$S_r^{-1}(v), S_r^0(v)$和$S_r^{+1}(v)$。也就是说$S_r^{-1}(v) \cup S_r^0(v) \cup S_r^{+1}(v) = S_r$

我们计算位并行标签并将其存储在索引中。 对于每个顶点 v∈V，我们预先计算表示为 LBP（v）的位并行标签。 LBP（v）是一组四元组$(u, \delta_{uv}, S_u^{-1}(v), S_u^0(v))$，其中 u∈V 是顶点，δ uv = dG（u，v）以及 Su（v）⊆Su 如上定义。 我们通过 b 位的位向量存储$S_u^{-1}(v)$和$S_u^0(v)$。注意$S_u^{+1}(v)$可以通过$S_u \setminus (S_u^{-1}(v) \cup S_u^0(v))$获得，但实际上我们在查询算法中不使用$S_u^{+1}(v)$。

为了通过 b 比特的比特向量来描述 Sr 的子集，我们在 1 和| Sr |之间分配唯一的数 到 Sr 中的每个顶点，并通过设置第 i 个位来表示第 i 个顶点的存在。

5.2 位并行 BFS

我们暂且放弃第 4.2 节中讨论的修剪，我们使用第 4.1 节中讨论的幼稚标签方法的位并行版本。 我们稍后在 5.4 节中介绍修剪。

令 r∈V 为顶点，Sr⊆NG（r）为 r 大小为 b 的邻居的子集。 我们解释一个算法来计算 dG（r，v），$S_r^{-1}(v), S_r^0(v)$的所有 v∈V，可以从{r}∪Sr。 该算法被描述为算法 3.基本上，我们从 r 个计算集合S^{-1}和S^0进行 BFS。

令 v 是顶点。 假设我们已经针对所有 w 计算了$S_r^{-1}(w)$，使得 dG（r，w）<dG（r，v）。我们可以如下计算$S_r^{-1}(v)$

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

因为如果 u 在 S-1（v）中，则 d（u，v）= d（r，v）-1 并且因此 u 在从 r 到 v 的最短路径之一上。 对于所有 w 使得 dG（r，w）≤dG（r，v）和 Sr0（w）对于所有 w 使得 dG（r，w）<dG 如下计算 $S_r^{-1}(v)$

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

因此，与宽度优先搜索一起，我们可以通过以距离 r 的增加的顺序的动态规划来交替地计算S^{-1}和S^0。 也就是说，首先我们为所有 v∈V 计算S^{-1}（v），使得 dG（r，v）= 1，接下来我们计算所有 v∈V 的S^0（v），使 dG（r，v）= 1 ，那么我们为所有 v∈V 计算 Sr（v），使得 dG（r，v）= 2，接下来我们为所有 v∈V 计算S^0（v），使得 dG（r，v）= 2，等等 。 注意，对集合的操作可以通过用位向量表示集合并使用按位操作在 O（1）时间完成。

5.3 位并行距离查询

为了处理一对顶点 s 和 t 之间的距离查询，与正常标记一样，我们扫描位平行标记 LBP（s）和 LBP（t）。 对于共享相同根顶点的每一对四元组，$(r, \delta_{rs}, S_r^{-1}(s), S_r^0(s))$ ∈LBP（s）和$(r, \delta_{rt}, S_r^{-1}(t), S_r^0(t))$∈LBP（t），从这些四元组中，我们通过{r}∪Sr 中的一个顶点计算 s 和 t 之间的距离。 也就是说，我们计算

$$\delta = \min_{u \in \{r\} \cup S_r} \{d_G(s,u) + d_G(u,t)\}$$

一个朴素的方法是为所有 u 计算 dG（s，u）和 dG（u，t），取最小值，这需要 O（| Sr |）时间。 然而，我们提出了一种算法，通过使用逐位运算计算 O（1）时间的 δ 。

令 δ = dG（s，r）+ dG（r，t）。 由于$\tilde{\delta}$是 δ 的上界，并且对于所有 u∈Sr，dG（s，u）≥dG（s，r）-1，dG（u，t）≥dG（r，t），$\tilde{\delta} - 2 \le \delta \le \tilde{\delta}$。 因此，我们要做的是判断距离

δ 是$\tilde{\delta}-2, \tilde{\delta}-1$ 还是$\tilde{\delta}$ 。

这可以如下进行。 如果 Sr（s）∩Sr（t）≠∅，则 δ =$\tilde{\delta}$ -2。否则，如果 S0（s）∩S-1（t）≠∅或 S ）≠∅，则 δ =$\tilde{\delta}$ -1，而否则 δ =$\tilde{\delta}$ 。 注意集合的计算交集可以通过按位 AND 运算来完成。因此，所有这些操作可以在 O（1）时间内完成。 因此，距离 δ 可以在 O（1）时间中计算，并且总的来说，我们可以回答 O（| LBP（s）| + | LBP（t）|）时间中的每个查询。

5.4 修剪标签简介

现在我们讨论如何组合这种位并行标记方法和第 4.2 节中讨论的修剪标记方法。我们提出一种简单有效的方法如下。

首先，我们进行位并行 BFS，而不修剪 t 次，其中 t 是参数。然后，我们使用位并行标签和正常标签来修剪 BFS。

这种方法利用了修剪标记法和位平行标记法的不同强度。在开始，修剪不工作很多，修剪的 BFS 访问大部分的顶点。因此，不是修剪标记，而是使用位平行标记，而不修剪以有效地覆盖较大部分的顶点对。跳过无用的修剪测试的开销也有助于加速。

作为位并行 BFS 的根和邻居集，我们建议贪婪地使用具有最高优先级的顶点：我们选择具有最高优先级的顶点作为剩余顶点中的根 r，并且选择具有最高优先级的 b 个顶点优先级作为剩余邻居中的集合 Sr。

正如我们在第 7 节的实验结果中看到的，这种方法提高了预处理时间，索引大小和查询时间。此外，如我们在实验中所证实的，如果我们不将太大的值设置为 t，至少它不会破坏性能。因此，我们不必太严肃地找到一个适当的 t 值，我们的方法仍然很容易使用。

6.变量和扩展

最短路径查询：为了不仅回答距离而且回答最短路径，我们将元组而不是对作为标签存储。标签 L（v）是三元组（u， δ uv，puv）的集合，其中 puv∈V 是修剪的宽度优先搜索树中 u 的父节点，其中 u 是由从 u 修剪的 BFS 创建的 u。我们可以通过将树从 v 上升到父节点来恢复 v 和 u 之间的最短路径。

加权图：要处理加权图，唯一必要的更改是执行修剪的 Dijkstra 的算法，而不是修剪的 BFS。位并行标记不能用于加权图。

定向图：为了处理有向图，我们首先将 dG（u，v）修改为从 u 到 v 的距离，然后为每个顶点存储两个标记 LOUT（v）和 LIN（v）。标签 LOUT（v）是一组对（u， δ vu），其中 u∈V 和 δ vu= dG（v，u），Label LIN（v）是一组对（u， δ uv） V 和 δ uv= dG（u，v）。

我们可以通过 LOUT（s）和 LIN（t）来回答顶点 s 到顶点 t 的距离。为了计算这些标签，从每个顶点，我们进行两次修剪的 BFS：一次在前向，一次在相反方向。

基于磁盘的查询回答：为了回答距离查询，我们的查询算法只引用两个连续的区域。因此，如果索引是磁盘驻留的，我们可以使用两个磁盘查找操作来回答查询，这仍然会比内存中的 BFS 快得多。

7.实验

我们在具有 Intel Xeon X5670（2.93 GHz）和 48GB 主内存的 Linux 服务器上进行了实验。所提出的方法在 C ++中实现。我们使用 8 位整数表示距离，32 位整数表示顶点，64 位整数表示位并行 BFS。对于顶点排序，我们主要使用 Degree 策略，并且我们不指定顶点排序策略，除非我们使用其他策略。对于查询时间，我们通常报告 1,000,000 个随机查询的平均时间。

表 4 数据集

Dataset	Network	$\|V\|$	$\|E\|$
Gnutella	Computer	63 K	148 K
Epinions	Social	76 K	509 K
Slashdot	Social	82 K	948 K
Notredame	Web	326 K	1.5 M
WikiTalk	Social	2.4 M	4.7 M
Skitter	Computer	1.7 M	11 M
Indo	Web	1.4 M	17 M
MetroSec	Computer	2.3 M	22 M
Flickr	Social	1.8 M	23 M
Hollywood	Social	1.1 M	114 M
Indochina	Web	7.4 M	194 M

7.1 数据集

为了显示我们的方法的效率和鲁棒性，我们对各种现实世界网络进行了实验：五个社交网络，三个网络图和三个计算机网络。我们将所有图形视为无向，未加权的图形。基本上，我们使用五个较小的数据集来比较所提出的方法和以前的方法之间的性能，并分析这些方法的行为，并使用较大的六个数据集来显示所提出的方法的可扩展性。网络的类型，顶点和边缘的数量在表 4 中给出。

7.1.1 详细说明

Gnutella：这是从 2002 年 8 月 Gnutella P2P 网络的快照创建的图[34]。

Epinions：该图是 Epinions（www.epinions.com）上的在线社交网络，其中每个顶点表示一个用户，每个边表示一个信任关系[33]。

Slashdot：这是在 2009 年 2 月获得的 Slashdot（slashdot.org）中的在线社交网络。顶点

对应于用户，边缘对应于用户之间的朋友/敌人链接[23]。

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

WikiTalk：这是维基百科（www.wikipedia.org）编辑之间的在线社交网络，通过交际页面上的编辑创建，直到 2008 年 1 月[21,20]。

Skitter：这是一个由 Traceroutes 创建的 Internet 拓扑图，由 Skitter 在 2005 年运行[22]。

Indo：这是在 2004 年搜索的.in 域的页面之间的网络图[9,8]。

MetroSec：这是由 MetroSec 捕获的 Internet 流量构建的图。每个顶点表示一个计算机，如果两个顶点出现在包中作为发送者和目的地，则它们被链接[24]。

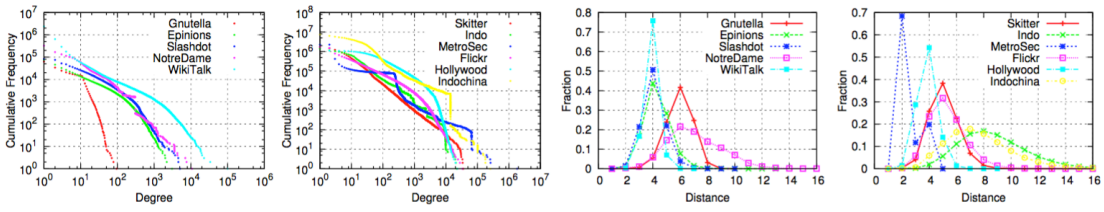
Flickr：这是一个在照片共享网站上的在线社交网络，Flickr（www.flickr.com）[26]。

印度支那：这是一个网页的网页在印度支那国家的国家域，在 2004 年爬行[9,8]。

好莱坞：这是一个电影演员的社交网络。如果两个演员在 2009 年出现在一部电影中[9,8]，则两个演员被联系起来。

7.1.2 统计

首先，我们调查网络的度数分布，因为当我们使用 Degree 策略来顶点排序时，顶点的角度在我们的方法中发挥重要作用。



(a) Degree distribution of smaller five datasets. (b) Degree distribution of larger six datasets. (c) Distance distribution of smaller five datasets. (d) Distance distribution of larger six datasets.

图 2 数据集的属性

表 3 提出的方法和以前的方法之间的实际数据集的性能比较。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 意味着它没有在一天内完成或用完内存。

Dataset	Pruned Landmark Labeling				Hierarchical Hub Labeling [2]				Tree Decomposition [4]			BFS
	IT	IS	QT	LN	IT	IS	QT	LN	IT	IS	QT	
Gnutella	54 s	209 MB	5.2 μs	644+16	245 s	380 MB	11 μs	1,275	209 s	68 MB	19 μs	3.2 ms
Epinions	1.7 s	32 MB	0.5 μs	33+16	495 s	93 MB	2.2 μs	256	128 s	42 MB	11 μs	7.4 ms
Slashdot	6.0 s	48 MB	0.8 μs	68+16	670 s	182 MB	3.9 μs	464	343 s	83 MB	12 μs	12 ms
NotreDame	4.5 s	138 MB	0.5 μs	34+16	10,256 s	64 MB	0.4 μs	41	243 s	120 MB	39 μs	17 ms
WikiTalk	61 s	1.0 GB	0.6 μs	34+16	DNF	-	-	-	2,459 s	416 MB	1.8 μs	197 ms
Skitter	359 s	2.7 GB	2.3 μs	123+64	DNF	-	-	-	DNF	-	-	190 ms
Indo	173 s	2.3 GB	1.6 μs	133+64	DNF	-	-	-	DNF	-	-	150 ms
MetroSec	108 s	2.5 GB	0.7 μs	19+64	DNF	-	-	-	DNF	-	-	150 ms
Flickr	866 s	4.0 GB	2.6 μs	247+64	DNF	-	-	-	DNF	-	-	361 ms
Hollywood	15,164 s	12 GB	15.6 μs	2,098+64	DNF	-	-	-	DNF	-	-	1.2 s
Indochina	6,068 s	22 GB	4.1 μs	415+64	DNF	-	-	-	DNF	-	-	1.5 s

图 2a 和 2b 是度互补累积分布的对数对数图。如预期，我们可以确认所有这些网络一般表现出幂律度分布。

然后，我们还检查了距离的分布。图 2c 和 2d 示出了 1,000,000 个随机对顶点的距离分布。从这些图可以看出，这些网络也是小世界网络，在平均距离非常小的意义上。

7.2 性能

首先，我们介绍我们的方法对真实世界数据集的性能，以显示我们的方法的效率和鲁棒性。表 3 显示了我们的数据集方法的性能。 IT 表示预处理时间，IS 表示索引大小，QT 表示用于 1,000,000 个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，格式为正常标签的大小（左）加上位并行的大小标签（右）。我们设置我们进行位并行 BFSs 的次数为 16 为前五个数据集和 64 其余。

在表 3 中，我们还列出了两种最先进的现有方法的性能。一个是分层集线器标签[2]，其也基于距离标签。另一个是基于树分解[4]，这是 TEDI 的改进版本[41]。对于这些以前的方法，我们使用这些方法的作者的实现，在 C ++中。在具有两个 Intel Xeon X5680（3.33GHz）和 96GB 主内存的 Windows 服务器上进行分层集线器标签实验。在我们的环境中进行基于树分解的方法的实验。我们还描述了通过宽度优先搜索计算距离的平均时间

1,000 个随机对顶点。在这四种方法中，包括提出的方法，只有分层集线器标签的预处理[2]并行使用所有 12 个核心。所有其他定时结果是连续的。

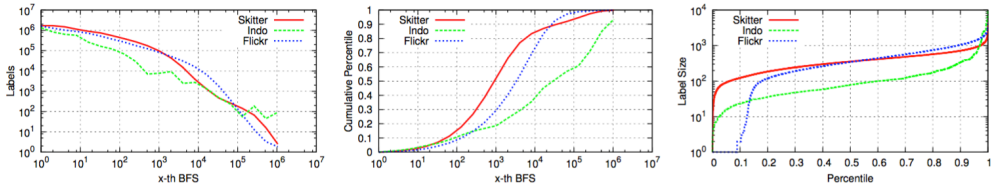
7.2.1 预处理时间和可扩展性

我们的重点特别是在预处理时间的大幅度提高，导致更好的可扩展性。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数百万的顶点和数百万边缘的中等预处理时间的汉堡。这是对我们可以处理的图大小的两个数量级的改进，因为如表 1 中所列，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个边的图。

对于具有数千万条边的接下来的四个数据集，花费少于一千秒，而先前的方法在一天之后没有完成或耗尽存储器。对于较小的六个数据集，他们花了至多一分钟，并且比以前的方法大多至少快 50 倍。

7.2.2 查询时间

平均查询时间通常为微秒，最多为 16 微秒。对于几乎所有较小的五个数据集，所提出的方法的查询时间比先前方法的查询时间快。实际上，从表 1，我们还可以观察到，我们的方法的查询时间与这些尺寸的图的所有现有方法相当。此外，我们可以确认查询时间对于较大的网络不增加太多。

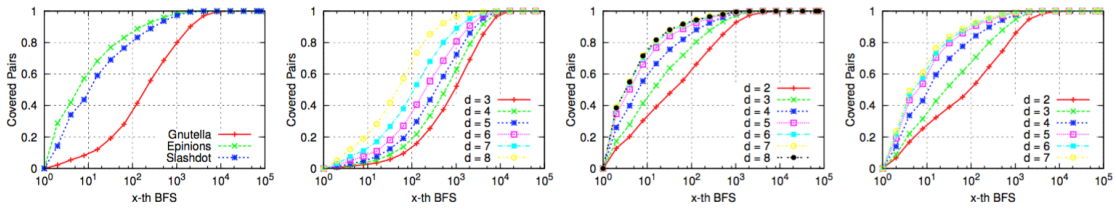


(a) Number of vertices labeled in each pruned BFS.

(b) Cumulative distribution of the number of vertices labeled in each pruned BFS.

(c) Distribution of the sizes of labels.

图 3 剪枝的效果以及标签的大小



(a) Average

(b) Distance-wise, Gnutella

(c) Distance-wise, Epinions

(d) Distance-wise, Slashdot

图 4 其距离可以通过索引来回答的顶点对与执行的删减的 BFS 的数目相对应的比例

7.2.3 索引大小

对于较小的五个网络，结果表明我们的方法与以前的方法相比索引大小。然而，即使现在具有数十吉字节的存储器的计算机既不稀有也不昂贵，减小索引大小可能是下一个重要的研究问题。

7.3 分析

接下来，我们分析我们的方法的行为，以调查我们的方法为什么是高效的。

7.3.1 修剪的 BFS

首先，我们研究如何计算和存储标签。图 3a 显示了添加到每个剪枝 BFS 中的标签的距离的数量，图 3b 显示了它的累积分布，即不晚于每个步骤存储的距离与存储在每个步骤中的所有距离的比率。结束。我们没有使用位并行 BFSs 这些实验。

从这些数字，我们可以确认修剪的巨大影响。图 3a 示出了添加到每个 BFS 中的标签的距离的数量迅速减少。例如，在 1000 次 BFS 之后，对于所有三个数据集，将距离添加到仅小于 10%的顶点的标签，并且在进行 10,000 次 BFS 之后，对于所有三个数据集，距离被添加到标签只有小于 1%的顶点。图 3b 还示出了标签的大部分在开始时被计算。

7.3.2 标签的尺寸

图 3c 示出了在整个预处理之后，以尺寸的升序排序的标签的尺寸的分布。我们可以观察到，每个顶点的标签的大小对于不同的顶点没有太大差异，并且很少的顶点具有比平均值大得多的标签。这说明我们的方法的查询时间相当稳定。

如果您对具有异常大的字母的顶点感到焦虑，则可以预先计算这些顶点和所有顶点之间

的距离，并直接回答它，因为这种顶点的数量很少，如图 3c 所示。

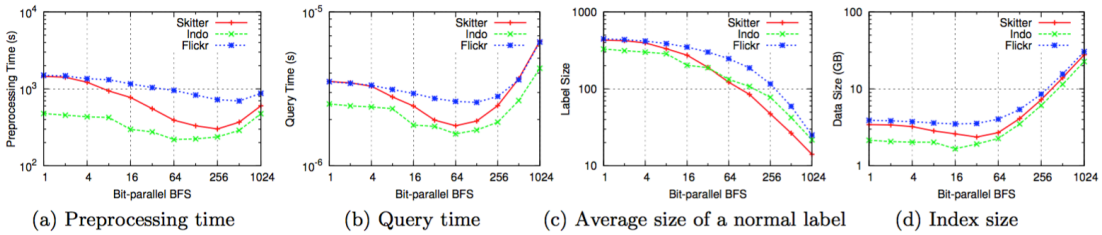
7.3.3 对覆盖

图 4a 示出了在每个步骤处被覆盖的对的对的比率，即，其距离可以被当前标签正确回答的顶点对的比率。我们使用 1,000,000 个随机对来估计这些比率。我们可以看到，大多数对在开始覆盖。这表明，这样的大部分对具有通过中心顶点的这样的小部分的最短路径，其由度策略选择。这就是为什么基于地标的近似方法具有良好的精度，以及为什么我们的修剪工作如此有效的原因。

图 4b，4c 和 4d 示出了每个步骤中被覆盖的顶点对与按距离分类的对的比率。他们表明通常远距离的对被覆盖早于密切的对。这就是为什么用于近对的基于标记的近似方法的精度远远差于远对的精度的原因。另一方面，我们的方法积极利用这个属性：因为远程对在开始覆盖，我们可以修剪远处的顶点，当处理其他顶点，这导致快速预处理。

7.3.4 顶点排序策略

接下来我们看到顶点排序策略的效果。表 5 描述了使用第 4.4 节中描述的不同顶点排序策略的每个顶点的标签的平均大小。我们没有使用位并行 BFSs 这些实验。正如我们所看到的，结果在 Degree 策略和 Closeness 策略之间没有那么不同。Degree 策略可能会稍微好一点。另一方面，随机策略的结果比其他两种策略差得多。这表明通过度和接近策略，我们可以成功捕获中心顶点。



(a) Preprocessing time (b) Query time (c) Average size of a normal label (d) Index size

图 5 针对位并行 BFS 数量的性能

表 5 每个顶点的标签的平均大小与不同的顶点排序策略。

Dataset	RANDOM	DEGREE	CLOSENESS
Gnutella	6,171	781	865
Epinions	7,038	124	132
Slashdot	8,665	216	234
NotreDame	DNF	60	82
WikiTalk	DNF	118	158

7.3.5 位并行 BFS

最后，我们看到第 5 节中讨论的位并行 BFS 的影响。图 5 显示了我们的方法针对不同次数进行位并行 BFS 的性能。

图 5a 示出了预处理时间。它表明，使用适当数量的位并行 BFS，预处理时间获得快两

到十倍，导致进一步增强我们的方法的可扩展性。图 5b 说明了查询时间。我们可以确认查询时间也变得更快。图 5c 示出了每个顶点的正常标记的平均大小。随着我们增加位并行 BFS 的数量，许多对被由位并行 BFS 计算的特殊标签覆盖，并且正常标签的大小减小。图 5d 示出了索引大小。利用适当数量的位并行 BFS，索引大小也减小。

这些图中的另一个重要发现是，我们的方法的性能对于位并行 BFS 的数量的参数不太敏感。如他们所示，我们的方法的性能没有变得更糟，除非我们选择一个太大的数字。正确的参数似乎在不同的网络之间是共同的。因此，我们的方法仍然容易使用这种位并行技术。

8.结论

在本文中，我们提出了一种用于大图上精确最短路径距离查询的新颖有效的方法。我们的方法是基于距离标记到顶点，这是现有的精确距离查询方法的常见的，但我们的标记算法是一个全新的想法。我们的算法通过修剪从所有顶点执行宽度优先搜索（BFS）。虽然算法很简单，我们的修剪令人惊讶地减少了搜索空间和标签，导致快速的预处理时间，小的索引大小和快速的查询时间。此外，我们还提出了另一种利用位级并行性的标记方案，其可以容易地与修剪的标记方法结合以进一步

提高性能。在各种类型的大规模现实世界网络上的广泛的实验结果表明我们的方法的效率和鲁棒性。特别地，我们的方法可以处理具有数亿个顶点的网络，这些顶点比先前方法的限制大两个数量级，具有可比较的索引大小和查询时间。

我们计划研究如何处理更大的图形，其中索引和/或图形可能不适合主存储器。第一种方法是通过减少利用明显部分和对称性的图来减小索引大小[30,14]，并通过使公共子树的字典用于最短路径树来压缩标签[1]。另一种方式是基于磁盘或分布式实现。正如我们在第 6 节中所说，基于磁盘的查询应答是显而易见的，准备就绪，特别是预处理方面的挑战。然而，由于我们的预处理算法是一个基于 BFS 的简单算法，我们可以利用 BFS 上大量的现有工作。特别地，由于修剪可以在本地完成，所以预处理算法将在基于 BSP 模型的分布式图形处理平台上执行良好[25]。

9.致谢

我们要感谢 Hiroshi Imai 对稿件的批判性阅读，Daniel Delling 为我们提供了分层集线器标签的实验结果[2]。我们还要感谢匿名审稿人对改进论文提出的建设性建议。吉田吉一由 JSPS 资助研究活动启动（24800082），MEXT 创新区域科学研究助手（24106001）和 JST，ERATO，川内林大图项目支持。

参考文献

[1] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck. A hub-based labeling algorithm for shortest paths in road networks. In SEA, pages 230–241, 2011.

[2] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck. Hierarchical hub labelings for shortest paths. In ESA, pages 24–35. 2012.

[3] V. Agarwal, F. Petrini, D. Pasetto, and D. A. Bader. Scalable graph exploration on multicore processors. In SC, pages 1–11, 2010.

[4] T. Akiba, C. Sommer, and K. Kawarabayashi. Shortest-path queries for complex networks: exploiting low tree-width outside the core. In EDBT, pages 144–155, 2012.

[5] R. Albert, H. Jeong, and A. L. Barabasi. The diameter of the world wide web. Nature, 401:130–131, 1999.

[6] L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan. Group formation in large social networks: membership, growth, and evolution. In KDD, pages 44–54, 2006.

[7] S. Boccaletti, V. Latora, Y. Moreno, M. Chavez, and D. Hwang. Complex networks: Structure and dynamics. Physics reports, 424(4-5):175–308, 2006.

[8] P. Boldi, M. Rosa, M. Santini, and S. Vigna. Layered label propagation: a multiresolution coordinate-free ordering for compressing social networks. In WWW, pages 587–596, 2011.

[9] P. Boldi and S. Vigna. The webgraph framework I: compression techniques. In WWW, pages 595–602, 2004.

[10] D. S. Callaway, M. E. J. Newman, S. H. Strogatz, and D. J. Watts. Network robustness and fragility: Percolation on random graphs. Physical Review Letters, 85:5468–5471, 2000.

[11] W. Chen, C. Sommer, S.-H. Teng, and Y. Wang. A compact routing scheme and approximate distance oracle for power-law graphs. TALG, 9(1):4:1–26, 2012.

[12] J. Cheng and J. X. Yu. On-line exact shortest distance query processing. In EDBT, pages 481–492, 2009.

[13] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick.

Reachability and distance queries via 2-hop labels. In

SODA, pages 937–946, 2002.

[14] W. Fan, J. Li, X. Wang, and Y. Wu. Query preserving

graph compression. In SIGMOD, pages 157–168, 2012.

[15] A. Gubichev, S. Bedathur, S. Seufert, and G. Weikum. Fast and accurate estimation of shortest paths in large

graphs. In CIKM, pages 499–508, 2010.

[16] H. He, H. Wang, J. Yang, and P. S. Yu. Blinks:

ranked keyword searches on graphs. In SIGMOD,

pages 305–316, 2007.

[17] R. Jin, N. Ruan, Y. Xiang, and V. Lee. A

highway-centric labeling approach for answering distance queries on large sparse graphs. In SIGMOD, pages 445–456, 2012.

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

[19] D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the spread of influence through a social network. In KDD, pages 137–146, 2003.

[20] J. Leskovec, D. Huttenlocher, and J. Kleinberg. Predicting positive and negative links in online social networks. In WWW, pages 641–650, 2010.

[21] J. Leskovec, D. Huttenlocher, and J. Kleinberg. Signed networks in social media. In CHI, pages 1361–1370, 2010.

[22] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: Densification laws, shrinking diameters and possible explanations. In KDD, pages 177–187, 2005.

[23] J. Leskovec, K. Lang, A. Dasgupta, and M. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. Internet Mathematics, 6(1):29–123, 2009.

[24] C. Magnien, M. Latapy, and M. Habib. Fast computation of empirically tight bounds for the diameter of massive graphs. J. Exp. Algorithmics, 13:10:1.10–10:1.9, Feb. 2009.

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

I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In SIGMOD, pages 135–146, 2010.

[26] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In IMC, pages 29–42, 2007.

[27] M. E. J. Newman, S. H. Strogatz, and D. J. Watts. Random graphs with arbitrary degree distributions and their applications. Physical Review E, 64(2):026118 1–17, 2001.

[28] R. Pastor-Satorras and A. Vespignani. Evolution and structure of the Internet: A statistical physics approach. Cambridge University Press, 2004.

[29] M. Potamias, F. Bonchi, C. Castillo, and A. Gionis. Fast shortest path distance estimation in large networks. In CIKM, pages 867–876, 2009.

[30] M. Qiao, H. Cheng, L. Chang, and J. X. Yu. Approximate shortest distance computing: A query-dependent local landmark scheme. In ICDE, pages 462–473, 2012.

[31] S. A. Rahman, P. Advani, R. Schunk, R. Schrader, and D. Schomburg. Metabolic pathway analysis web service (pathway hunter tool at cubic). Bioinformatics, 21(7):1189–1193, 2005.

[32] S. A. Rahman and D. Schomburg. Observing local and global properties of metabolic pathways: 'load points' and 'choke points' in the metabolic networks. Bioinformatics, 22(14):1767–1774, 2006.

[33] M. Richardson, R. Agrawal, and P. Domingos. Trust management for the semantic web. In ISWC, volume 2870, pages 351–368. 2003.

[34] M. Ripeanu, A. Iamnitchi, and I. Foster. Mapping the gnutella network. IEEE Internet Computing, 6(1):50–57, Jan. 2002.

[35] N. Robertson and P. D. Seymour. Graph minors. III. Planar tree-width. J. Comb. Theory, Ser. B, 36(1):49–64, 1984.

[36] L. Tang and M. Crovella. Virtual landmarks for the internet. In SIGCOMM, pages 143–152, 2003.

[37] T. Tran, H. Wang, S. Rudolph, and P. Cimiano. Top-k exploration of query candidates for efficient keyword search on graph-shaped (RDF) data. In ICDE, pages 405–416, 2009.

[38] K. Tretyakov, A. Armas-Cervantes, L.Garc ´ıa-Ban ˜uelos,J.Vilo,andM.Dumas.Fastfully dynamic landmark-based estimation of shortest path distances in very large graphs. In CIKM, pages 1785–1794, 2011.

[39] A. Ukkonen, C. Castillo, D. Donato, and A. Gionis. Searching the wikipedia with contextual information. In CIKM, pages 1351–1352, 2008.

[40] M. V. Vieira, B. M. Fonseca, R. Damazio, P. B. Golgher, D. d. C. Reis, and B. Ribeiro-Neto. Efficient search ranking in social networks. In CIKM, pages 563–572, 2007.

[41] F. Wei. Tedi: efficient shortest path query answering on graphs. In SIGMOD, pages 99–110, 2010.

[42] S. A. Yahia, M. Benedikt, L. V. S. Lakshmanan, and J. Stoyanovich. Efficient network aware search in collaborative tagging sites. PVLDB, 1(1):710–721, 2008.

参考文献原文