X-Hop:传递闭包的多跳数压缩存储和快速可达性查询

舒 虎 崇志宏 倪巍伟 卢 山 徐立臻
(东南大学计算机科学与工程学院 南京 211189)

摘 要 海量图数据上的可达性查询是图数据管理的基本问题。目前解决这个问题的基本方法是对可达关系传递闭包进行压缩存储,再辅以快速查询算法来回答两顶点是否可达。在此基础上,重点研究了稠密图条件下可达传递闭包的高压缩比存储和有效查询算法,提出了多跳(简称为X-Hop)压缩存储方法。通过采用生成树的结构对2-Hop中的中心顶点进行组织,X-Hop存储有效地降低了2-Hop方法中需要记录的索引点数量,从而极大地提高了压缩比。实验证明,X-Hop在索引的规模上要远远小于2-Hop存储,并且在查询效率上也取得优势。

关键词 X-Hop,可达性查询,2-Hop标记,传递闭包压缩
中图法分类号 TP392 **文献标识码** A

X-Hop:Storage of Transitive Closure and Efficient Query Process

SHU Hu CHONG Zhi-hong NI Wei-wei LU Shan XU Li-zhen
(School of Computer Science and Engineering, Southeast University, Nanjing 211189, China)

Abstract Reachability query is one of the fundamental problems of management of massive directed graphs. This paper considered reachability query under the context of dense graphs. We proposed a storage schema, called X-Hop, which compresses the storage of 2-Hop via a multiple-hop schema. By organizing center vertexes in a tree structure, X-Hop storage delivers efficient query process in addition to high compression ratio. Extensive experiments demonstrate the efficiency of our proposal.

Keywords X-Hop, Reachability query, 2-Hop labeling, Transitive closure compression

1 引言

由于其灵活性,图数据模型被广泛应用于开放或复杂环境下的数据建模。例如在Web中常用图来表示网页间的链接关系;在分子生物学中,用图表示蛋白质之间的转换关系;在社交网络中,用图来表示成员间的社会联系。在这些典型的图数据应用中,通常情况下图的规模庞大,其顶点数量一般都在百万级别以上,其建模的复杂联系导致边数急剧增加。因此,稠密海量图数据管理成为目前研究的热点问题,其中可达性查询(图中的顶点 u 能否通过一条路径到达顶点 v)是基本的查询问题。例如,在Web页面的链接关系分析中,常常需要回答一个页面是否被另一个页面直接或间接引用过;在分子生物学中,需要判断一种蛋白质在代谢过程中是否会转化为某种特定的蛋白质;对社交网络中某些规则进行挖掘时,需要检查某对成员之间是否具有某种社会联系。而这些问题都可以抽象为数据图中的可达性问题。

图的深度或广度优先遍历是可达性查询的基本在线算法,其时间开销为 $O(|E|+|V|)$,难以在大规模图上使用。为了提高查询效率,对可达关系的传递闭包(图中所有可达性关系的集合)进行事先存储,将可达性查询转化为对闭包存储的查询。这种方法依靠额外的存储开销来提高查询效率。一个极端方式就是预先计算出图的整个传递闭包并进行存储,用 $O(|V|^2)$ 的存储开销来换取 $O(1)$ 的查询效率。但是,这对于顶点数 n 很大的海量图数据应用是无法实现的。因此,对传递闭包的有效压缩存储成为目前的研究重点,其基本思想是在在线遍历算法和可达性闭包的完全存储这两个极端之间寻求一个平衡,通过适当增加查询开销来明显地减少存储开销。

本文重点研究稠密图的可达传递闭包的高压缩比存储和有效可达性查询算法。一方面,在稠密图中边数量和顶点数量的比值远高于稀疏图,使得目前许多压缩存储的效率明显下降;另外一方面,在网络社区环境、生物信息等领域,联系的复杂性呈现出边数的急剧增加,使得稠密图成为典型的图数据。我们在基本的2-Hop基础上,提出了多跳数的压缩存储方法(简称为X-Hop),通过树结构对2-Hop中的中心顶点进行组织,有效地降低2-Hop方法中需要记录的索引点的数量,从而极大地压缩了可达性传递闭包。其贡献集中表现在以下几个方面:

(1)提出多跳数的压缩存储方法,并通过理论分析和实验验证对多跳数存储的有效性进行研究,发现多跳数压缩存储

到稿日期:2011-08-22 返修日期:2011-12-22 本文受国家自然科学基金(60973023,61003057)资助。
舒 虎(1986-),男,硕士生,主要研究领域为图结构数据管理,E-mail:hushu3488@163.com;崇志宏(1969-),男,副教授,主要研究领域为数据库理论和技术;倪巍伟(1979-),男,副教授,主要研究领域为数据库理论和技术;卢 山(1971-),男,副教授,主要研究领域为网络技术;徐立臻(1963-),男,教授,主要研究领域为数据库理论和技术。

在稠密图上有明显的优势。

(2)设计简单的算法来从多跳数压缩存储上快速地恢复可达性关系,其效率来源于对数量远小于顶点数的中心点进行树结构的组织。这种将存储和查询紧密绑定的方式为用户选择压缩比和查询效率偏好提供了可能的途径。

(3)在多种典型数据集上进行广泛的对比试验,验证 X-Hop 存储方式在存储开销和查询效率方面都有明显的优势,相对于 2-Hop 提高了 10%左右的压缩比,在稠密图情况下查询效率几乎提高一倍。

本文第 2 节简要介绍相关的传递闭包压缩算法;第 3 节介绍 X-Hop 存储结构及其计算;第 4 节介绍 X-Hop 上的查询算法;第 5 节对 X-Hop 的有效性进行实验验证;最后总结全文并对下一步工作进行展望。

2 相关工作

存储传递闭包可高效地回答可达性查询,但其往往需要消耗大量的存储空间。因此大量研究工作集中在了对传递闭包的压缩,同时保证较高的查询效率上。接下来将简要介绍采用该模式解决可达性查询的几种方法。

(1)子结构分解与区间标号　文献[5]提出了链分解的方法来压缩传递闭包,该算法把图分解成两两不相交的链,并对各个顶点按链序号和链内序号进行标号,用以判断两点之间是否可达,链分解主要压缩了同一条链上的传递关系。文献[1]提出了一种计算最优树分解的算法来压缩传递闭包,并证明了采用该算法得到的压缩结构一定优于最优链分解得到的压缩结构。算法计算出原图的一棵生成树,并保存生成树所没有覆盖住的连通关系作为非树边,这样就可以利用生成树与存储的非树边回答可达性查询。针对稀疏图文献[3]提出了改进的树分解算法,它能够在常数时间内回答任意点对间的可达性关系。

结合链分解和树分解,文献[2]提出了 path-tree 的方法来对传递闭包进行压缩,该方法首先对图进行链分解,然后把每条链看成一个节点并按照树结构组织起来,最后对所生成的组合结构进行编码(树编码,链编码)用以快速回答可达性查询,但其压缩率较低。

文献[9]提出了一种随机区间标号的方法来回答大规模图上可达性查询。对于 DAG,该方法对图中的每个节点赋予多组不同的区间标号,以加强对不可达查询的过滤,该方法简单易行,压缩率较高,但不提供查询上界。

(2)中心点路径压缩　Edith Cohen 等最早在文献[4]中提出了 2-Hop 标记(Labelling)方法的思想,即通过少量的中心顶点来压缩任意两顶点之间的连通性,从而达到压缩存储连通传递闭包。但是,在稠密图情况下,其压缩效率明显下降。针对大规模图上的 2-Hop 标记计算难的问题,文献[13]提出了图分割技术来降低计算代价和内存消耗,使其实际计算变得可行。

(3)混合模式　Ruoming Jin 等在文献[8]中提出了 3-Hop 标记方法来提高压缩效率,该方法结合了子结构分解和中心点路径压缩。算法首先对图进行链分解,把每条链作为一条"高速公路",从源顶点到目标顶点的可达关系可以被表达为,源顶点能够进入一条链,通过该链出去而到达目标顶点。该方法提供较高的压缩率,但需要较多的查询开销:在执行性查询时,需要在线计算从源顶点进入哪条"高速公路"(链)以及从链上的哪个顶点出去而到达目标顶点,这就导致查询效率下降。

受到 3-Hop 标记的启发,本文提出了 X-Hop 标记。X-Hop 区别于 3-Hop 的主要点在于通过树结构来组织中心顶点,代替 3-Hop 中的链结构,同时提供精巧的查询算法,在提高压缩比的同时保证了较高的查询效率。下面将详细介绍 X-Hop 方法。

3 X-Hop 存储结构

有向图可以表示为两元组 $G(V,E)$,其中 V 和 E 分别是顶点和有向边的集合。图 G 中两个顶点 u,v 的可达性查询定义为:顶点 u 是否可以到达顶点 v。如果它们之间可达,就简记为 $u \rightarrow v$。

3.1 X-Hop 的基本思想

X-Hop 希望采用多跳可达的方法来提高对图传递闭包的压缩率,同时采用标号的方法保证较高的查询效率,而多跳可达是从基本的两跳可达进行扩展而来,因此首先回顾一下基本的 2-Hop 方法[4]。

定义 1(2-Hop 覆盖)　图 $G(V,E)$ 的 2-Hop 覆盖是一个集合:$S=\{(In(w),w,Out(w))\}$,其中 $w \in V$,$In(w) \subseteq V$,$Out(w) \subseteq V$,使得 G 的传递闭包 $TC \subseteq \bigcup In(w) \odot Out(w)$,其中 $\odot$ 表示集合的笛卡尔积。

最优 2-Hop 覆盖就是使得集合 S 最小的一个 2-Hop 覆盖。由于一个基本的目标就是减少存储开销,因此在没有特别说明时本文所指的都是最优 2-Hop。

对于可达性查询,2-Hop 的基本思想是通过选择部分顶点作为中心点来覆盖连通路径,每个中心点 w_i 被赋予一对集合 In 和 Out,分别记录能够到达 w_i 的图中部分顶点和 w_i 能够到达的图中的部分顶点。这样任意两个顶点 u 和 v 之间的可达关系可表达为:顶点 u 是否可以通过某个中心顶点 w_i 到达顶点 v。

利用 2-Hop 覆盖可计算出相应的 2-Hop 标记 $L=\{Lin(u),u,Lout(u)\}$,其中 u 是 G 中的任意一个顶点,$Lin(u)/Lout(u)$ 分别是能够到达 u 或 u 能够到达的中心点的集合。回答顶点 u 是否可达 v 时,只需判断 $Lout(u) \bigcap Lin(v)$ 是否为空,若空则不可达,反之则为可达。

图 1 所示为 2-Hop 覆盖中的两项,其可解释为 $In(w_i)$ 或 $In(w_j)$ 中的顶点能够通过中心点 w_i 或 w_j 到达 $Out(w_i)$ 或 $Out(w_j)$ 中的顶点,例如 v_1 可以通过中心点 w_i 到达 v_{10}。而进一步观察图 1 所示的可达关系,不难发现顶点之间的可达关系被重复记录了,例如 v_4,v_5 既可以通过 w_i,又可以通过 w_j 到达 v_{12},v_{13}。因此,如果记录 w_i 和 w_j 间的可达关系,那么 v_4,v_5 可从 $In(w_j)$ 中删除而不改变图的可达性关系。同理,v_{12},v_{13} 也可从 $Out(w_i)$ 删除。这给我们一个启示,即如果记录下中心点间的可达关系,将能够减少 2-Hop 覆盖所需记录的顶点的数量,从而减少存储开销。

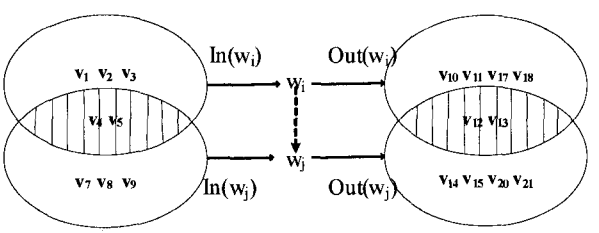

图 1　2-Hop 基本结构与冗余

2-Hop 覆盖中的每个项可以看作是对图的传递闭包的部分覆盖，因而其目标是尽可能大地减少覆盖中各项之间对传递闭包的重复覆盖，但在一般的图中，这种重复覆盖是不可避免的，尤其是在稠密图中将大量存在。另外，即使是 2-Hop 覆盖中的两项间不存在对传递闭包的重复覆盖，例如图 1 中 $In(w_i)$和 $In(w_j)$没有交集时，$Out(w_i)$和 $Out(w_j)$仍然可能会出现交集，而这些处于交集中的点就在 2-Hop 覆盖中重复存储。所以通过记录中心点间的可达关系来减少存储开销是一种有效的方法，这在实验部分也得到了证明。

3.2 X-Hop 存储及其计算

本节将上述观察一般化，通过记录中心点间的可达关系可以对基本 2-Hop 覆盖中的存储项进行精简，从而减少存储消耗。记图 G 的 2-Hop 覆盖中的中心点集合为 W，G_w 表示 W 中顶点所构成的子图，$TC(G_w)$为 G_w 的传递闭包。

定义 2(SOC)　中心点组织结构 *SOC*(structure of center)是一个记录了 2-Hop 覆盖中的中心点间的部分可达关系的集合，也就是 $SOC \subseteq TC(G_w)$。

一个给定的 *SOC* 可以用来对 2-Hop 覆盖中每个中心点的 *In*/*Out* 集合进行精简。如果 *In* 和 *Out* 集合被精简为空集，那么该中心顶点就从当前的 2-Hop 覆盖中删除。每个中心顶点的 *Out* 和 *In* 集合的精简过程如下：$\forall w_i \in W$，其 $Out(w_i)$，$In(w_i)$集合分别精简为 $Out'(w_i)$，$In'(w_i)$，精简公式如下：

$$Out'(w_i)=Out(w_i)-\bigcup_{w_j} Out(w_j)\cap Out(w_i)) \quad (1)$$

式中，w_j 是 *SOC* 中 w_i 的可达顶点。

$$In'(w_i)=In(w_i)-\bigcup_{w_k} (In(w_k)\cap In(w_i)) \quad (2)$$

式中，w_k 是 *SOC* 中可到达 w_i 的顶点。

定义 3(X-Hop 覆盖)　若记精简后的 2-Hop 覆盖集为 $S'(In'(w_i), w_i, Out'(w_i))$，其精简操作采用的中心点组织结构为 *SOC*，那么 S'和 *SOC* 一起构成了 X-Hop 覆盖。

X-Hop 的最终目标就是计算出存储代价最小的 $S'(In'(w_i), w_i, Out'(w_i))+SOC$ 组合来覆盖图的传递闭包。X-Hop 的存储开销由 S'和 *SOC* 这两个部分组成，并且 *SOC* 的选择将会极大地影响 S'的大小，因此找出最优的 *SOC* 是降低总的存储开销的关键所在。

定义 4(精简收益)　对于 2-Hop 覆盖的中心顶点集 W 中的每个中心顶点 w_i，其 $Out(w_i)$，$In(w_i)$集合依据 *SOC* 并按式(1)、式(2)分别精简为 $Out'(w_i)$和 $In'(w_i)$，那么记录了由 *SOC* 所减少的存储开销可以表达为 $B=\sum_{w_i \in W}(|In(w_i)|+|Out(w_i)|-|In'(w_i)|-|Out'(w_i)|)-|SOC|$，这个值就称为精简收益。

若记 $Bout(w_i)=|\bigcup_{w_j}(Out(w_j)\cap Out(w_i))|$，其中 w_j 是 *SOC* 中 w_i 的可达顶点；$Bin(w_i)=|\bigcup_{w_k}(In(w_k)\cap In(w_i))|$，其中 w_k 是 *SOC* 中可到达 w_i 的顶点。精简收益也可以表示为 $B=\sum_{w_i \in W}(Bout(w_i)+Bin(w_i))-|SOC|$。因此，为了增大总的精简收益值 B，必须记录较小的 $|SOC|$，同时保持 $\sum_{w_i \in W}(Bout(w_i)+Bin(w_i))$尽可能大。

在本文中采用记录中心点间的树形可达关系的 *SOC* 来达到上述目标。计算一棵最优的树分解，使得存储开销最小，需要列举出所有的生成树，其计算为 NP 难。下面将给出一个启发式算法来计算树形 *SOC*。

第一步　初始化 *SOC* 为 2-Hop 覆盖中的中心点间的传递闭包(可看作多个子图)。给每条边 e 赋权重为 $h(e)=|In(w_1)\cap In(w_2)|+|Out(w_1)\cap Out(w_2)|$，即边的两个端点 w_1，w_2 相互精简带来的收益值。若子图(均为 *DAG*)不唯一，可添加一个虚拟起始点，并赋相应边权值为 0。

第二步　利用文献[6]等中的生成树算法对 *SOC* 计算一棵最大生成树 T(如图 2 所示)，由于边上的权值是保存该边所表达的可达关系至少能带来多少的存储收益(边的两个顶点相互精简带来的存储收益)，因此不难看出这棵最大生成树能够很好地近似最优树结构。

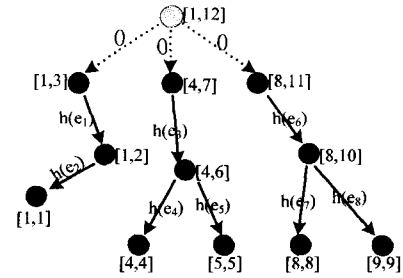

图 2　中心点的最大生成树与顶点标号

第三步　删除 *SOC* 中的不在 T 中的边，这样得到的 *SOC* 就是中心点所构成的图的一棵最大生成树。

完成最大生成树的构造后，对生成树的每个顶点按照后序遍历进行编号，每个顶点记录其编号 j 和以它为根的子树中的最小编号 i。这样对于每一个顶点就得到了一个区间标号$[i,j]$，记为 l，对于任意的两个顶点 w_m，w_n，若 w_m 的区间标号 $w_m.l$ 包含 w_n 的区间标号 $w_n.l$，则 w_m 可到达 w_n，否则 w_m 不可到达 w_n。接下来便可以利用计算出的 *SOC*，采用式(1)、式(2)对 2-Hop 覆盖中的所有中心点的 *Out* 和 *In* 集合进行精简，这样就得到了 X-Hop 覆盖。

定义 5(X-Hop 标记)　对于图 $G(V,E)$及其 X-Hop 覆盖，可以对 G 中的每个顶点 u 赋予一对标签集合，$Lin(u)\subseteq W$，$Lout(u)\subseteq W$，其分别表示能够到达顶点 u 的中心点和 u 能够到达的中心点，其中 W 是 *X-Hop* 覆盖中的中心点集合。G 中所有顶点的标签集合和 X-Hop 覆盖中的 *SOC* 一起构成了 X-Hop 标记。

算法 1　X-Hop(G)

1. 快速地计算 G 的 2-Hop 覆盖；
2. 初始化 *SOC*，并给其中的每条边赋权重；
3. 计算 *SOC* 的最大生成树 T；
4. 对 T 中的顶点计算区间编号；
5. 用 T 精简 2-Hop 的覆盖；
6. 对每个顶点 $v \in G(V)$；
7. 计算其 X-Hop 的标记 $Lin(v)$ 和 $Lout(v)$；
8. 对 $Lin(v)$ 和 $Lout(v)$中的顶点按其标号排序存储；

最后，将整个 X-Hop 存储通过算法 X-Hop(G)详细表示出来。X-Hop(G)首先利用快速算法(参见文献[7])计算出图 G 的 2-Hop 覆盖，接着初始化 *SOC* 为 2-Hop 覆盖中的中心点集的传递闭包，然后计算出 *SOC* 的树形结构来对 2-Hop 覆盖进行精简，最后得到 X-Hop 标记。下面将用 T 来简单表示该树形的 *SOC*，用 L 来表示 X-Hop 标记所有顶点的 $Lin(v)$和 $Lout(u)$集合，那么将有如下结论保证该 X-Hop 标记能够正确回答图上的可达性查询。

定理 1 给定图 G，X-Hop 覆盖能够恢复 G 的传递闭包。简要证明如下：

X-Hop 覆盖由基本的 2-Hop 覆盖计算而来，通过式(1)、式(2)的逆向推导，可以很容易地恢复出 2-Hop 覆盖，即在 X-Hop 覆盖集 S' 的每一项的 $In'(w_i)/Out'(w_i)$ 集中加入 w_i 在 SOC 中的所有前驱/后继中心点的 In/Out 即可，这样得到的覆盖集将包含 2-Hop 覆盖，从而能够覆盖图 G 的传递闭包。

推论 1 给定图 G，算法 X-Hop(G)生成的 X-Hop 标记 $(L+T)$能够覆盖 G 的传递闭包。

X-Hop 标记$(L+T)$能够恢复出 X-Hop 覆盖(以中心点为主键做一个倒排表)，而依据定理 1，X-Hop 覆盖又能够恢复 G 的传递闭包，故上述结论成立。

性质 1 最优 X-Hop 计算出的存储结构将优于最优 2-Hop 生成的存储结构。

从存储项的精简过程中可以看出，X-Hop 标记中的每个顶点所记录的存储项均为 2-Hop 标记所需要记录的存储项的子集，而额外记录的生成树其大小远远小于顶点所记录的存储结构，可以忽略不计。因此总的存储代价将小于 2-Hop 的存储代价。

4 X-Hop 的查询处理

在本小节中，将介绍基于 X-Hop 标记的快速可达性查询算法，其中 4.1 节叙述基本的查询算法，4.2 节叙述利用了 X-Hop 标记特性的改进算法，并进行算法时间复杂度分析。

4.1 基本的查询处理

回答图 G 上的可达性查询的最基本的方法是：首先利用 X-Hop 标记$(L+T)$恢复出基本的 2-Hop 标记，然后再按照基本的 2-Hop 标记中的方式回答两个顶点是否可达。由 X-Hop 标记恢复基本的 2-Hop 标记的过程如下：对于任意一个顶点 x，令

$Lin(x)=L.Lin(x)\cup\{w_j$ 在 T 中的前驱顶点/$w_j\in L.Lin(x)\}$

$Lout(x)=L.Lout(x)\cup\{w_i$ 在 T 中的后继顶点/$w_i\in L.Lout(x)\}$

利用上述公式，对于顶点 u,v 可以得到集合 $Lin(v)$，$Lout(u)$。设 $Lout'(u)$，$Lin'(v)$是 G 的 2-Hop 标记中关于顶点 u,v 的标记集合，显然有 $Lin'(v)\subseteq Lin(v)$，$Lout'(u)\subseteq Lout(u)$。令 $A=Lin(v)\cap Lout(u)$，若 A 为空，则 u 不可达 v，否则 u 可达 v。

按照上述方式来回答顶点 u,v 的可达性，需要访问中心顶点在 T 中的所有前驱或后继顶点，其时间开销较大，同时采用上述公式恢复出来的 $Lin(v)/Lout(u)$集合规模将不小于基本的 2-Hop 标记中对应项的规模，这样进一步造成较高的时间复杂度。然而在 X-Hop 中，并不需要恢复出上述的 $Lin(v)/Lout(u)$。基于这样一个事实：若 $u\rightarrow v$，当且仅当 $\exists w_i\in L.Lout(u)$，$w_j\in L.Lin(v)$且 $w_j.l\subseteq w_i.l$。因此在回答 u 是否可达 v 时，只需要检测是否有 $\exists(w_i,w_j)\in(L.Lout(u)\odot L.Lin(v))$，使得 $wj.l\subseteq wi.l$，其中$\odot$为集合的笛卡尔积运算。

4.2 改进的查询处理

回顾 X-Hop 标记构建中的精简过程可以得出，$L.Lin$ 中记录的任意两个中心顶点 w_i,w_j 在 T 中将不存在祖后关系，即 $w_i.l\not\subset w_j.l$ 且 $w_j.l\not\subset w_i.l$；并且 X-Hop 中采用的中心节点标号方法，对于任意的两个顶点，赋予其的区间不会出现交叠，例如 $w_i.l=(b_i,e_i)$，$w_j.l=(b_j,e_j)$，那么若 $b_i>b_j$，则 $e_i>e_j$。这样，$L.Lin$ 中的每个中心点都可以看成一个独立的区间，所有的顶点可以按照其区间的位置顺序排好，这将得到 $L.Lin$ 集合上的一个全序序列。同理对于 $L.Lout$，也可以对其做一个相同的排序操作。在回答 u 到 v 的可达性查询时，仅仅需要在 $L.Lin(v)$和 $L.Lout(u)$上做一次归并操作，就能判断是否有条件：$\exists w_i\in L.Lout(u)$，$w_j\in L.Lin(v)$，且 $w_j.l\subseteq w_i.l$ 成立。若成立则 u 可达 v，否则不可达，其时间复杂度为 $O(|L.Lin|+|L.Lout|)$。

性质 2 X-Hop 的查询时间将优于 2-Hop 的查询时间。因为对于任意两个顶点 u,v，设 $Lin(v)$，$Lout(u)$为 2-Hop 中顶点 v,u 的存储项，则有 X-Hop 中对应的存储项，$L.Lin(v)\subseteq Lin(v)$，$L.Lout(u)\subseteq Lout(u)$，而利用 2-Hop 回答 u 是否能到 v，需要计算 $Lin(v)\cap Lout(u)$，其时间复杂度为 $O(|Lin|+|Lout|)$，这将高于 X-Hop 的查询时间 $O(|L.Lin|+|L.Lout|)$。

5 实验结果

在本节中，将通过详细的实验对 X-Hop 的有效性进行验证。实验测试了 X-Hop 的存储结构大小和查询时间，并和 2-Hop，3-Hop 方法进行了对比(在文献[8]的主页上，可下载 3-Hop 的源代码和实验数据)，同时为了更加直观地体现 X-Hop 的存储结构对查询效率的影响，实验中加上了与基本的图遍历算法 DFS 和 BFS 的对比。实验采用 C++和 STL 进行编写，并在 Ubuntu 10.04LTS 上测试，测试用的主机 CPU 为 Intel(R) Core(TM)2 CPU 4300@1.80GHz，内存为 2GB。

实验中用到的数据集(图)采用文献[10]中提出的模式生成和文献[8]中用到的实际数据，测试中用到的图都是 DAG。在实验中，测试了图的顶点规模在存储开销和查询时间上和密度对各个算法的影响。

5.1 存储开销

在实验结果中，算法的存储开销将用存储结构中记录的点的数量来表示。2-Hop 方法的存储开销为 $\sum_{u\in G}(|Lin(u)|+|Lout(u)|)$。X-Hop$(L+T)$方法中存储开销的大小为 $\sum_{u\in G}(|L.Lin(u)|+|L.Lout(u)|)+2|T|$。3-Hop 方法的存储开销采用文献[8]中的计量方式。由于基本的 DFS 和 BFS 无需额外存储，因此没有对比。

(1)图的规模对存储开销的影响　在固定图密度(边的数量/顶点的数量=4)的前提下，从图的顶点大小为 2k 开始，逐渐提高到 10k，生成了一组图。在这组图上分别执行 2-Hop，X-Hop 和 3-Hop 算法，其存储开销如图 3 所示。随着图规模的增大，3 个算法的存储开销也都有明显增大，这是因为，随着图顶点数量的增大，图的传递闭包将急剧增大，为覆盖所有传递关系，需要存储更多的顶点。从图中可以看出，X-Hop 的存储开销略小于 3-Hop，而远小于 2-Hop 索引的大小，其差值约为 10%。

(2)图的密度对存储开销的影响　在顶点数量为 2k、密度从 2 至 12 的一组图上分别执行 2-Hop，X-Hop 和 3-Hop 算法，其存储开销如图 4 所示。可以看出，在开始阶段随着图密度的增大 3 个算法的存储开销增长较明显，这是因为在一段区间内，图密度的增加将导致图传递闭包的急剧增大，但随着

密度超过一定值后，传递闭包的增长速度将放缓，因而存储开销的增长速度也将降低。但总的来说，3 个算法都不同程度地压缩了传递闭包，并且 X-Hop 算法的存储开销总是小于 2-Hop 的存储开销，并且随着图密度的增大，两者之间的差别有所增大。与 3-Hop 相比，X-Hop 在图密度为 2 至 10 这个区间内具有明显的压缩优势，并且随着图密度的继续增大，两者的存储开销反而均有所减少，这也说明在图特别稀疏或者特别稠密时，回答可达性查询所需的存储消耗都不会太大，因而算法的有效性应该反映在其对密度处于中间阶段的图的处理上。从这个角度来看，X-Hop 有一定的优势，而现实生活中的图数据，其密度往往服从正态分布，因此对于实际数据集的压缩率，X-Hop 将取得更好的效果，这点从表 1 可以看出。

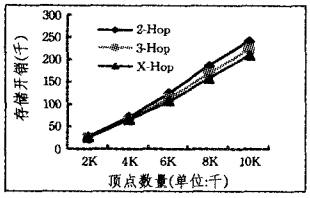

图 3　图规模 VS 存储开销

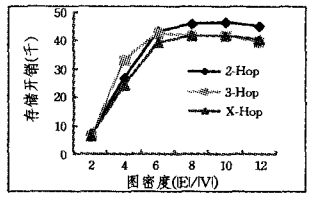

图 4　图密度 VS 存储开销

表 1　实际数据集上的存储开销(个)比较

数据集	图密度	X-Hop	2-Hop	3-Hop
Yago	6.38	15610	17767	22990
Pubmed	4.45	39930	42841	75289
Go	1.97	22382	23458	29753
Citeseer	4.13	36757	41876	53500
Arxiv	11.12	46851	54719	57957

5.2　查询时间

表 2　实际数据集上的查询时间(ms)比较

数据集	X-Hop	2-Hop	3-Hop	DFS	BFS
Yago	0.97	1.89	1.02	241	235
Pubmed	1.47	1.81	3.43	784	772
Go	1.16	1.39	2.55	314	298
Citeseer	1.55	1.83	2.16	793	798
Arxiv	2.62	3.44	6.98	2439	2782

表 3　人造数据集上的查询时间(ms)比较

数据集	X-Hop	2-Hop	3-Hop	DFS	BFS
Graph2k4k	0.97	1.45	1.02	102.1	104.3
Graph2k8k	1.87	3.02	4.01	677.8	656.2
Graph2k12k	2.64	4.51	6.05	2174	2223
Graph2k16k	2.88	4.43	8.06	3621	3634
Graph2k20k	2.77	4.37	12.0	4837	4916
Graph2k24k	2.71	4.06	8.07	6046	6693

为了验证 X-Hop 的查询效率，在图顶点为 4k 密度为 2 到 12 的一组图和多个实际数据集上进行了测试。测试所用的查询，是在所用的每张图上随机生成的 1000 对没有重复的点对。在每张图上对所生成的这 1000 个查询执行 1000 次，最后取其平均值作为最终的查询时间。结果如表 2、表 3 所列，其比基本的 DFS/BFS，X-Hop 的查询效率要高出两个数量级，并且随着图密度的变化，X-Hop 的查询时间的增长也较之小得多。在图较小时，2-Hop 和 X-Hop 的差别不明显，当随着图密度的增大，X-Hop 表现出了更高的效率。相比于 3-Hop，X-Hop 的查询效率明显，这是因为随着图密度的增加，3-Hop 中链的条数会增加，链与链的可达关系更多，在查询时恢复相应的可达性关系需要访问更多的索引项，从而导致时间开销增大；而 X-Hop 由于采用生成树结构，在顶点不变的情况下，其索引大小基本不变，并且从压缩项中恢复点间的可达关系也不需要访问更多的索引项，这使得 X-Hop 能够取得较高的查询效率。

结束语　本文提出了 X-Hop 方法来压缩传递闭包，通过消除 2-Hop 的冗余和优化查询算法解开压缩比率和查询效率的矛盾。在给出理论分析的同时，通过与 2-Hop 的对比实验证明了 X-Hop 在提高存储效率的同时未牺牲查询效率，与 3-Hop 相比，在保证较高压缩率的同时大大提高了查询效率。

参 考 文 献

[1] Agrawal R, Borgida A, Jagadish H V. Efficient management of transitive relationships in large data and knowledge bases[C]// SIGMOD. 1989:253-262

[2] Jin Ruo-ming, Xiang Yang, Ruan Ning, et al. Efficiently answering reachability queries on very large directed graphs[C]// SIGMOD Conference. 2008:595-608

[3] Wang Hai-xun, He Hao, Yang Jun, et al. Dual Labeling: Answering graph reachability queries in constant time[C]// ICDE'06. 2006:75

[4] Cohen E, Halperin E, Kaplan H. Reachability and distance queries via 2-Hop labels[C]// Proceedings of the 13th annual ACM-SIAM Symposium on Discrete algorithms. 2002:937-946

[5] Jagadish H V. A compression technique to materialize transitive closure[J]. ACM Trans. Database Syst., 1990, 15(4):558-598

[6] Kapoor S, Ramesh H. Algorithms for generating all spanning trees of undirected and weighted graphs[J]. SIAM J. Comput., 1995, 24(2)

[7] Cheng J, Yu J X, Lin X, et al. Fast computation of reachability Labeling for large graphs[C]// Proc. of EDBT'06. 2006

[8] Jin Ruo-ming, Xiang Yang, Ruan Ning. 3-Hop: a high-compression indexing scheme for reachability query[C]// Proceedings of the 35th SIGMOD International Conference on Management of Data. 2009:813-826

[9] Yildirim H, Chaoji V, Zaki M J. GRAIL: Scalable Reachability Index for Large Graphs[C]// Proceedings of PVLDB. 2010:276-284

[10] Johnsonbaugh R, Kalin M. A graph generation software package [C]// SIGCSE '91. New York, NY, USA, ACM, 1991:151-154

[11] Gallo G, Grigoriadis M D, Tarjan R E. A fast parametric maximum flow algorithm and applications [J]. SIAMJ. Comput., 1989, 18(1):30-55

[12] Wei Fang. TEDI: efficient shortest path query answering on graphs[C]// Proceedings of the 2010 International Conference on Management of Data. 2010:99-110

[13] Cheng Jie-feng, Yu J X, Lin Xue-min, et al. Fast computing reachability labelings for large graphs with high compression rate[C]// EDBT. 2008:193-204

[14] Cheng Jie-feng, Yu J X, Yu P S. Graph Pattern Matching: A Join/Semijoin Approach [J]. IEEE Transactions on Knowledge and Data Engineering, Sept. 2010

[15] Gallo G, Grigoriadis M D, Tarjan R E. A fast parametric maximum flow algorithm and applications [J]. SIAMJ. Comput., 1989, 18(1):30-55

[16] 冯建华，王钦克，周立柱，等. 半结构数据的存储模型和查询执行[J]. 计算机科学，2002，29(10)：6-10

[17] 葛唯益，程龚，瞿裕忠. 语义 Web 链接结构分析之综述[J]. 计算机科学，2010，37(3)：17-21