计算机科学
COMPUTER SCIENCE

标签约束图上的*k*步可达性查询

杜明, 邢瑞萍, 周军锋, 谭玉婷

引用本文

杜明, 邢瑞萍, 周军锋, 谭玉婷.标签约束图上的*k*步可达性查询[J]. 计算机科学, 2022, 49(12): 283-292.

DU Ming, XING Rui-ping, ZHOU Jun-feng, TAN Yu-ting. *k*-step Reachability Query Processing on Label-constrained Graph [J]. Computer Science, 2022, 49(12): 283-292.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.211000077

标签约束图上的 k 步可达性查询

杜 明 邢瑞萍 周军锋 谭玉婷
东华大学计算机科学与技术学院 上海 201620
(duming@dhu.edu.cn)

摘 要 标签约束图上的 k 步可达性查询问题,回答了在一个标签约束图上两点之间是否存在一条长度不大于 k 的路径并且这条路径上的标签都在用户给定的标签集中的问题。标签约束图上的 k 步可达性查询问题在现实中有着广泛的应用,然而现有算法无法直接回答这个问题。因此,首先提出 LK2H 算法。LK2H 算法主要包括构建索引和查询两个步骤。第一步是给图上的所有顶点构建一组包含 k 和标签信息的 2-Hop 索引,第二步是基于构建好的索引进行查询。在查询时,为了尽可能地为用户返回更多的信息,LK2H 算法优化了一类不可达查询的返回结果:当用户无法明确所有的标签类型,不能给出完整的标签约束,进而导致查询结果为不可达时,将完整的标签集返回给用户。其次,提出优化算法 LK2H+。LK2H+算法通过构建部分顶点的 2-Hop 索引进一步缩减索引大小和索引的构建时间,并基于构建好的索引进行查询。查询时,需要对顶点按照是否构建了索引进行分类讨论。最后,基于 15 个真实数据集进行测试。实验结果表明,LK2H 算法和 LK2H+算法都可以高效地解决标签约束图上的 k 步可达性查询问题。

关键词:标签约束图;k 步可达性查询;2-Hop 索引;顶点覆盖;图论
中图法分类号 TP301

k-step Reachability Query Processing on Label-constrained Graph

DU Ming,XING Rui-ping,ZHOU Jun-feng and TAN Yu-ting
School of Computer Science and Technology,Donghua University,Shanghai 201620,China

Abstract The k-step reachability query processing on label-constrained graph is used to answer whether there is a path with a length not greater than k between two points and the labels on this path are in the specified label set. The k-step reachability query processing on label-constrained graph is widely used in reality,but there is no relevant algorithm to answer it. Therefore,the LK2H algorithm is proposed firstly. LK2H algorithm mainly consists of two steps. The first step is to build a pair of 2-Hop indexes containing k and label information for all vertices on the graph,and the second step is querying based on the built index. In order to return information as much as possible to the user,LK2H optimizes the results of a type of unreachable query:when the user cannot specify all the label types,and cannot give full label constraints resulting in unreachable query results,the complete label set will return to the user. Secondly,an optimization algorithm,LK2H+,is proposed. LK2H+ algorithm further reduces the index size and time of construction by building a 2-Hop index for part of vertices,and queries based on the built index. Queries require a discussion of whether query vertices are indexed or not. Finally,the test is conducted based on 15 real-world datasets. Experiment results show that both LK2H and LK2H+ algorithms can solve k-step reachability query processing on label constraint graphs efficiently and quickly.

Keywords Label-constrained graph,k-step reachability query,2-hop index,Vertex cover,Graph theory

1 引言

给定有向图中的一个源顶点 u 和目标顶点 v,可达性查询用于回答是否存在一条路径使得 u 能够到达 v 的问题。可达性查询是图数据处理中的一个基本操作,被广泛应用于社交网络[1]、通信网络[2]、知识图谱[3]等,并得到了研究者们的广泛关注[4-12]。

传统的可达性查询仅考虑两点是否可达,对可达路径没有任何限制。但实际应用中,用户的查询需求可能需要附加更多的限制条件才能得到满足。考虑到实际图中的边上可能附带标签信息,用户不但需要检测是否存在满足特定标签限制的路径,而且需要了解路径的长度是否满足要求。

到稿日期:2021-10-11 返修日期:2022-06-07
基金项目:上海市自然科学基金(20ZR1402700);国家自然科学基金(61472339,61572421,61272124)
This work was supported by the Natural Science Foundation of Shanghai,China(20ZR1402700) and National Natural Science Foundation of China (61472339,61572421,61272124).
通信作者:邢瑞萍(1017033346@qq.com)

例如在现实生活中，警方在处理团伙犯罪案件时往往会调查犯罪人员的社会关系网络[13]。查找的方向往往是与嫌疑人有亲密关系的人，比如嫌疑人的朋友、家人。同时，还会调查两个嫌疑人是否通过一些有特定关系的人员联系在一起，从而分析整个犯罪团伙的运作机制。对于与嫌疑人关系非常疏远的人员，警方可以不进行调查，从而减少办案成本。犯罪人员的社会关系网络就是一个标签约束图，而判断是否与嫌疑人关系紧密就相当于给定一个 k 值约束，这种查询就是标签约束图上的 k 步可达性查询问题。

现有研究成果中，标签约束可达和 k 步可达是由不同研究人员提出并作为两个独立问题进行研究的。对于标签约束，研究人员提出了标签约束的可达性查询方法[14-15]。标签约束的可达性查询旨在回答是否存在一条从源顶点 u 出发到目标顶点 v 结束的有向路径并且路径上的标签都包含在给定的标签集中的问题。k 步可达性查询[16-17]用于回答在给定有向图中的两点间是否存在长度不大于 k 的有向路径的问题。但是，当回答标签约束图上的 k 步可达查询时，现有的两类方法均无法处理。原因在于现有的 k 步可达索引没有标签信息，而标签约束可达性查询方法所建立的索引中不包含长度信息。即使两种方法都返回可达的结果，由于两点之间可能存在多条路径，系统也并不知道满足 k 步约束的路径一定同时满足标签约束。

同时，现有各种标签约束可达和 k 步可达查询方法在处理用户查询时，除了可达与否外，没有提供任何可供用户参考的有用信息。继续考虑以上警方办案的例子：由于调查人员可能并不清楚嫌疑人社交关系网络中各种具体的关系标签，如果在查询时不能给定完整的约束条件，现有方法会直接返回否定的查询结果。发生这种情况时，调查人员可能希望进一步了解缺失的关系到底是什么，从而有助于后续的分析处理。

针对上述问题，我们提出通过构建包含标签和 k 值的 2-Hop 索引来解决标签约束图上的 k 步可达性查询问题。2-Hop 标签索引的基本思想是在图中选择一些顶点做双向 BFS，这些被挑选出来的顶点被称为 hop 点。双向 BFS 的作用是给每个顶点预先创建一组出标签和入标签，这一组标签里保存着部分顶点与该顶点的最短路径信息。同时，针对现有可达性查询方法返回信息少的问题，我们的方法在判定不可达时，会明确到底是什么原因导致的不可达，并会根据不同情况返回不同信息。具体来说，当 k 不满足约束条件时，无论标签是否满足约束条件，都只返回“不可达”；当 k 满足约束条件时，如果存在一条可达路径上的标签集是查询标签集的超集，那么除了返回“不可达”，还要返回这个标签超集，以便用户了解两点之间的具体标签信息，在后续处理中可以提交合适的查询。本文的具体贡献如下。

(1)提出了基于 2-Hop 标签索引的 LK2H 算法来回答标签约束图上的 k 步可达性查询问题。

(2)优化了不可达查询的返回结果。对于一个给定的查询，如果查询结果因为标签不满足约束而不可达时，将可达路径的标签集返回，从而让用户知道缺少了哪些标签信息，以便在后续工作中做进一步的查询和分析。

(3)提出了一种基于顶点覆盖集的优化算法 LK2H+。LK2H+算法通过构建部分顶点的 2-Hop 标签索引改善了 LK2H 算法存在的索引创建时间长、索引占用空间大的问题。

(4)在 15 个真实数据集上对本文的方法进行了实验和比较，结果证明本文提出的两个算法都可以高效地解决标签约束图上的 k 步可达性查询问题。

本文第 2 节对相关工作进行介绍；第 3 节介绍 LK2H 算法的基本思想和算法过程；第 4 节介绍 LK2H+算法的基本思想和算法过程；第 5 节给出实验结果和分析；最后对全文进行总结。

2 相关工作

2.1 问题定义

给定一个标签约束的有向图 $G=(V,E,\zeta,\lambda)$，其中 $V=\{v_1,v_2,\cdots,v_n\}$ 是图中顶点的集合，$E=\{(u,v,l) \mid u,v\in V, l\in\zeta\}$ 是图中边的集合，ζ 是初始不为空的标签集合。图中任意边 $e=(u,v,l)\in E$，表示图 G 上存在一条由顶点 u 指向顶点 v 的有向边，这条边上的标签为 l。λ 是一个映射函数，边集 E 中的每一条边与标签集 ζ 中的标签一一对应，即 $\lambda(\langle u,v,l\rangle)=l$。为了叙述方便，在没有歧义的情况下，后续介绍中将标签约束图简称为图。

对于图上的每个顶点 u，$out_G(u)=\{v|(u,v)\in E\}$ 表示顶点 u 的出邻居集合，$in_G(u)=\{v|(v,u)\in E\}$ 表示顶点 u 的入邻居集合；$deg_{\text{out}}(u)=|out_G(u)|$ 表示顶点 u 的出度，$deg_{\text{in}}(u)=|in_G(u)|$ 表示顶点 u 的入度。

定义 1(路径) 图 G 上的一条路径 P 是一个由点和边组成的序列 $\langle v_0,e_0,v_1,\cdots,v_{p-1},e_k,v_k\rangle$，其中 $k>0$，$v_i\in V$，$e_i\in E$。

定义 2(标签路径[14]) 给定一个标签集 L，如果路径 P 上每一条边的标签都属于 L，即 $\lambda(e_i)\in L$，则称路径 P 为 L-路径或一条标签路径。

定义 3(顶点覆盖集[18]) 有向图 G 的一个顶点覆盖集 C 是该图顶点集合 V 的一个子集，并且对于任意一条边 $(u,v)\in E$，其必然满足 $\{u,v\}\cap C\neq\emptyset$。

定义 4(最小顶点覆盖集[18]) 图 G 的所有顶点覆盖集中集合最小的顶点覆盖集被称为图 G 的最小顶点覆盖集。

定义 5(可达性[19]) 在图 G 中有两个顶点 $u,v\in E$，顶点 u 可以到达顶点 v，当且仅当 u 等于 v 或者存在一条边 $(u,w)\in E$ 且顶点 w 可以到达顶点 v。

定义 6(k 步可达性[20]) 在有向图 G 中，如果顶点 u 与顶点 v 之间存在一条长度不大于 k 的有向路径，则称顶点 u k 步可达顶点 v，否则称顶点 u k 步不可达顶点 v。

问题定义：给定一个标签约束图 G 和一个查询 $Q(u,v,L,k)$，在图 G 上查找是否存在一条从顶点 u 到顶点 v 的 L-路径 P，使得顶点 u k 步可达顶点 v。如果存在这样一条路径，则称查询 Q 为可达查询；否则称其为不可达查询。

2.2 相关算法

虽然目前没有算法能够直接解决标签约束图上的 k 步可达性查询问题，但标签约束图上的可达性查询问题和有向图上的 k 步可达性查询问题都取得了大量研究成果，下面分别对其进行介绍。

2.2.1　标签约束的可达性查询相关算法

标签约束图的可达性查询问题目前已得到很深入的研究，并取得了大量研究成果。下面主要介绍基于地标索引的LI+算法和基于2-Hop索引的P2H算法。

文献[15]提出了当前比较流行的基于地标的索引方法LI+算法。LI+算法的主要思想是通过BFS逐个构建每个地标的部分地标索引，当插入一个新的索引项时，LI+将新的索引项与已经存在的索引项进行比较，然后保留标签较小的索引项。当处理查询时，可以通过创建好的索引来回答查询。然而这种索引的可扩展性较差，最坏情况下与仅使用BFS遍历的表现持平。

文献[14]提出了基于2-Hop标签索引的P2H算法。P2H给图 G 上的每一个顶点 $u\in V$ 通过正向和反向的BFS分别创建一对2-Hop标签索引，该索引由一组入标签 $L_{\text{in}}=\{(h_1,l_1),\cdots,(h_i,l_i)\}$ 和一组出标签 $L_{\text{out}}=\{(h_1,l_1),\cdots,(h_j,l_j)\}$ 组成。其中，L_{in} 标签索引表示可以到达 u 顶点的hop点及标签集，L_{out} 标签索引表示可以从顶点 u 到达的hop点及到该hop点的标签集，索引中的每一项 (h,l) 表示hop点 h 和标签集 l。在创建索引的过程中，P2H用到了3种剪枝方法：1）如果顶点 v 已经被计算过，那么跳过顶点 v；2）如果即将插入的索引项能由已经存在的索引项导出，则不插入这个索引项；3）如果已经存在的索引项能由即将插入的新索引项所支配，则删除旧的索引项。当回答查询 $u\xrightarrow{L}v$ 时，分别遍历 u 的出标签和 v 的入标签，如果某一索引项的hop点相同，则再查看索引中的 l 是否是 L 的子集。P2H算法可以实现大图上的索引创建和查询，但该算法给每一个顶点创建一个2-Hop标签索引，因此它的索引创建时间较长，且创建好的索引也较大。另外，为了减小索引大小，在创建索引的过程中会有许多剪枝操作，导致当查询因为标签集不满足要求而不可达时，P2H算法只能返回不可达而无法给查询者提供更多的可用信息。

此外，还有很多其他方法可以解决标签约束图上的可达性查询问题。比如，文献[21]直接使用BFS/DFS或在数据图上创建完整的传递闭包；文献[22]使用边缘切割来回答标签约束图上最短路径的问题；文献[23]通过随机游走在标签约束图上回答简单查询；文献[24]通过图的生成树或生成森林构建压缩传递闭包来回答查询。

2.2.2　k 步可达性查询相关算法

目前对 k 步可达性查询问题已有比较深入的研究，很多相关算法被提出。对于已有的众多算法，本文着重介绍由文献[20]提出的K-reach算法和文献[17]提出的PLL算法。

K-reach算法的基本思想是首先求解有向图 G 的最小顶点覆盖集 C，然后在 C 上创建传递闭包。这个传递闭包只记录集合 C 中的顶点之间的可达信息。当回答一个 k 步可达查询 $u\xrightarrow{k}v$ 时，按照查询顶点类型的不同可分为以下3种查询情况：1）顶点 u 和顶点 v 都是最小顶点覆盖集 C 中的元素，此时直接通过创建好的传递闭包查询；2）顶点 u 或顶点 v 不是覆盖集 C 中的元素，此时如果顶点 u 是覆盖集 C 中的元素，则根据 u 和 v 的入邻居的传递闭包，做 $u\xrightarrow{k-1}v$ 查询，否则根据 v 和 u 的出邻居的传递闭包做 $u\xrightarrow{k-1}v$ 查询；3）顶点 u 和顶点 v 都不是覆盖集 C 中的元素，此时根据 u 出邻居的传递闭包和 v 入邻居的传递闭包做 $u\xrightarrow{k-2}v$ 查询。

PLL算法的基本思想是为图 G 中的每一个顶点 $u\in V$，通过正向BFS和反向BFS分别创建一对2-Hop标签：$L_{\text{in}}=\{(h_1,l_1),\cdots,(h_i,l_i)\}$ 和 $L_{\text{out}}=\{(h_1,l_1),\cdots,(h_j,l_j)\}$。其中，$L_{\text{in}}$ 标签索引表示可以到达 u 顶点的hop点及标签集，L_{out} 标签索引表示可以从顶点 u 到达的hop点及到该hop点的标签集，索引中的每一项 (h,d) 表示hop点 h 和 u 之间的最短距离。PLL算法在创建索引的过程中使用了剪枝策略，具体的做法是：对于每一个即将插入的索引项 (h_n,d_n)，查询是否有已经存在的索引项能够回答 $u\xrightarrow{d_n}h_n$，如果可以回答，则不插入 (h_n,d_n)，否则插入。对于查询，PLL算法将 k 步可达性查询问题转换成查询顶点对之间是否存在一条不大于 k 的最短路径问题。

3　LK2H算法

相比使用BFS/DFS[25-26]在图上直接遍历或计算所有顶点对之间的传递闭包[13]来判断可达性，2-Hop标签索引不但能够保存所有顶点对之间的可达信息，还可以通过剪枝删除不必要的索引项，从而缩减索引大小和构建索引的时间。因此，可以考虑通过创建2-Hop标签索引来解决标签约束图上的可达性查询问题。

LK2H算法的基本思想是在标签约束图上构建包含标签集和步长 k 的2-Hop标签索引，然后在此索引的基础上进行查询，并在查询的过程中保存部分可达路径的标签集，用来优化不可达查询的返回信息。

LK2H算法分为索引构建和基于索引的查询两个主要步骤。

3.1　索引构建

LK2H算法构建索引的基本步骤是：将图 G 中的所有顶点按照顶点度的大小降序排序；然后对每一个顶点 u 按照顺序执行正向和反向的剪枝BFS，构建包含标签和 k 的2-Hop索引。算法1给出了构建索引的伪代码描述。

算法1　LK2H构建索引算法

输入：G

输出：L_n

1. for k=1,2,…,n do
2.　$L_k^{\wedge}\leftarrow$ PrunedBFS(G,v_k,L_{k-1}) /*正向BFS*/
3.　$L_k\leftarrow$ PrunedBFS(G^T,v_k,$L_k^{\wedge}$) /*反向BFS*/
4. return L_n

函数 PrunedBFS

输入：(G,v_k,L_{k-1})

输出：L_k

1. 初始化：Q←{(v_k,0)} /*Q为保存索引项的队列*/
2. 初始化：$L_k\leftarrow L_{k-1}$ /* L_k为新一轮构建的索引*/
3. while Q≠Ø do
4.　v ← pop(Q)
5.　for every edge e in neighbour$_G$(v) do

```
6.      if e 已经被处理过 then
7.          Continue
8.      l_n ← e.label + v.label
        /* l_n为点 v_k到点 e 的标签集 */
9.      k_n ← v.k + 1 /* k_n为点 v_k到点 e 的距离 */
10.     if (h,l_n,k) 被 (h,l_e,k)∈L_k支配 then
11.         Continue
12.     Q.insert({e,l_n,k_n})
13. return L_k
```

构建索引时最重要的一步是每一个顶点进行双向剪枝BFS。剪枝BFS指在顶点进行BFS的过程中使用一些剪枝规则，从而缩小BFS遍历的范围，减少需要插入的索引项。LK2H算法在索引构建过程中使用了以下两个剪枝规则：

(1)如果顶点 u 在BFS遍历过程中遇到已经处理过的顶点 v，则不通过顶点 v 继续向下遍历；

(2)如果一个新的索引项 I 被已经存在的索引项支配，则不插入这个新的索引项。即当索引项 I 中的hop点和标签集与已存在的某个索引项相同但 k 值比该索引项中的 k 值大时，不对 I 进行插入操作。

接下来通过例1对LK2H算法的索引构建过程和其中的两个剪枝规则进行解释。

例1 图1给出了一个标签约束图索引构建的部分过程。图1(a)－图1(e)是正向剪枝BFS搜索过程，图1(f)－图1(j)是反向剪枝BFS搜索过程，其中深灰色顶点表示被处理的顶点，浅灰色顶点代表被访问到的顶点，白色顶点代表未被访问到的顶点。在构建索引之前，首先将所有顶点按照度的大小降序排序，度相同的按顶点序号降序排序，图1中顶点的顺序为{5,2,6,4,7,3,1}。排序后，所有顶点依次进行剪枝BFS操作，每一次BFS都需要依据前一轮生成的索引(算法1第2,3行)。以入标签的构建过程为例，首先进行剪枝BFS的是顶点5，图上的所有顶点都可以被访问到，每访问一个顶点就将索引项插入该点的索引中。注意，从顶点5到顶点7有两条路径，在索引项(5,b,1)已经存在于顶点7的入标签索引的情况下，从顶点5通过顶点6到达顶点7所产生的索引项(5,b,2)不会被插入到顶点7的索引中(规则2)。原因是索引项(5,b,2)与索引项(5,b,1)的hop点、标签都相同，并且索引项(5,b,2)中的 k 值大于1。顶点5 BFS结束后，对顶点2进行剪枝BFS。当访问到顶点5时，由于顶点5已经被处理过，因此停止遍历(规则1)。处理顶点4时，由于顶点4只有一条出边指向已经处理过的顶点5，并且访问到顶点5时停止，因此不对索引产生影响。处理顶点7时，由于顶点7出度为0，因此不会对索引产生影响。顶点1和顶点3剪枝BFS遍历的情况分别与顶点4和顶点7的情况相同，因此图1没有给出遍历过程。出标签的整体构建过程与入标签的构建过程相同。最终生成的索引如表1所列。

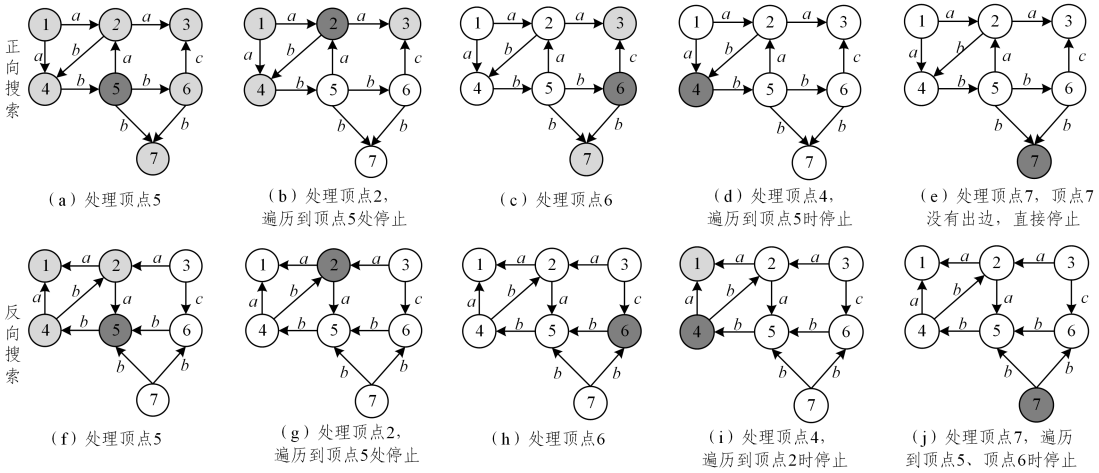

图1 LK2H算法的索引构建过程

Fig. 1 Index construction process of LK2H algorithm

表1 图1示例生成的索引

Table 1 Index of Fig. 1

ID	入标签索引	出标签索引
1	(5,a,2)(2,a,1)	(5,ab,2)(4,a,1)
2	(5,a,1)	(5,ab,3)(5,b,2)
3	(5,cd,2)(5,a,2)(6,c,1)(2,a,1)	—
4	(5,ab,2)(5,a,3)(2,b,1)(2,a,2)	(5,b,1)
5	—	—
6	(5,c,1)	—
7	(5,b,1)(6,b,1)	—

表1列出了每个顶点所对应的入标签和出标签索引。第1轮是由顶点5做剪枝BFS，在正向遍历结束后，索引项(5,a,2),(5,a,1),{(5,cd,2),(5,a,2)},{(5,ab,2),(5,a,3)},(5,c,1),(5,b,1)分别被插入到顶点1,2,3,4,6和7的入标签索引中。在反向遍历结束后，索引项(5,ab,2),{(5,ab,3)(5,b,2)}和(5,b,1)分别被插入到顶点1,2和4的出标签索引中。第4轮处理顶点4，索引项(4,a,1)被插入到顶点1的出标签索引中。其余顶点的处理都不对索引产生影响。

定理1 对于任意第 $r(0\leqslant r\leqslant n)$ 轮双向剪枝BFS，和任意一对标签集为 l 的顶点对 s 和 t，每一条通过顶点 $v_i(i\leqslant r)$ 的 L－路径都被存储在索引 L_r 中。

证明：当 $r=0$ 时，定理显然成立。假设从第0到第 $r-1$ 轮这个定理都成立，接下来证明第 r 轮也成立。利用反证法，假设存在一条包含顶点 $v_i(i\leqslant r)$ 的 L－路径不在 L_r 中。如果 $i<r$，那么一定存在 $L_i\subset L_r$，与假设不符，因此 $i=r$。当 $i=r$ 时，接下来要对 v_i 进行剪枝BFS遍历，将包含 v_i 的所有 L－

路径在第 r 轮插入 L_r，与假设不符。因此，定理 1 得证。

下面是索引构建过程的时空复杂度分析。对于图中的 n 个顶点，每一个顶点最多会存 $n-1$ 个 hop 点和 $2^{|\zeta|}$ 个不同的标签集，因此空间复杂度为 $O(n^2 2^{|\zeta|})$。对每一次剪枝 BFS，到达一个顶点的标签不同的路径数最多是 $2^{|\zeta|}$ 个，并且对于访问到的每一条边，对于标签检查的最坏时间复杂度为 $O(n*2^{|\zeta|})$，因此构建索引的时间复杂度为 $O(m*n^2*2^{2*|\zeta|})$。

3.2　查询算法

查询算法的基本思想是通过查看出发顶点的出标签索引和目标顶点的入标签索引中是否存在 hop 点相同且满足查询约束的索引项来判断是否可达。

给定一个图 G 上的 k 步可达性查询 $Q(u,v,L,k)$ 和一个构建好的 2-Hop 标签索引，查询算法的基本步骤是：1）同时遍历顶点 u 的出标签索引和顶点 v 的入标签索引，查找相同的 hop 点；2）判断通过相同 hop 点的路径是否满足 L 和 k 的查询约束。算法 2 给出了查询算法的伪代码描述。

算法 2　LK2H 查询算法

```
输入：(u,v,l,k)
输出：TRUE/FALSE
1. if 顶点 u 和 v 是同一个顶点 then
2.    return TRUE
3. for every index entry I_i in L_out[u] do
   /* 遍历顶点 u 的出标签索引 */
4.    for hop 点与 I_i 中 hop 点相同的 I_j in L_in[v] do
      /* 遍历顶点 v 的入索引 */
5.       if 距离满足约束 then
6.          if 标签满足约束 then
7.             return TRUE
8.          if l⊆I_i.l+I_j.l then
            /* 给定标签是可达路径标签集的子集 */
9.             记录 I_i.l+I_j.l
10.            return FALSE
11. return FALSE
```

对于不同的查询项，由于它们的出发顶点、目标顶点、k 和 L 约束不同，因此会返回不同的查询结果。在查询中，有一种可达的情况和 3 种不可达的情况。

可达的情况是：u 的出标签索引和顶点 v 的入标签索引中存在相同的 hop 点，并且通过该 hop 点的路径距离小于 k，路径上的标签是 L 的子集。对于可达的情况，会直接返回给用户“可达”。

不可达的情况是：1）u 的出标签索引和顶点 v 的入标签索引中不存在相同的 hop 点；2）存在相同的 hop 点，但通过 hop 点的路径长度大于 k；3）存在相同的 hop 点，路径长度小于或等于 k，但这条可达路径上的标签集不是 L 的子集。为了优化查询返回结果，在 LK2H 算法的查询算法中，将不可达的第三种情况进一步分成了两类：1）可达路径的标签集不是 L 的超集；2）可达路径的标签集是 L 的超集。对于标签集是 L 的超集的情况，先将这个超集记录下来并继续在索引中进行查询，如果最终没有找到满足条件的可达路径，则将这个超集返回给用户。

下面通过例 2 对查询的具体步骤和不同查询返回的结果进行解释。

例 2　对于图 1 中的图 G 和表 1 中的索引，给出 3 个查询项 $(4,6,abc,2)$，$(5,4,b,3)$，$(7,2,cd,3)$。对于查询 $(4,6,abc,2)$，遍历顶点 4 的出标签和顶点 6 的入标签（算法 2 第 3、第 4 行），可以发现它们都以顶点 5 作为 hop 点，并且距离等于 2，满足 k 值约束（算法 2 第 5 行），同时路径上的标签集 ab 也是 abc 的一个子集（算法 2 第 6 行），因此查询 $(4,6,abc,2)$ 的返回结果为可达。对于查询 $(5,4,b,3)$，4 的入标签中有两个索引项的 hop 点是顶点 5，且 k 满足约束。虽然两个索引项都不满足标签约束，但由于索引项 $(5,ab,2)$ 中的标签集是给定标签集 b 的一个超集（第 8 行），因此除了返回“不可达”，还要返回“ab”这个标签集。对于查询 $(7,2,cd,3)$，由于顶点 7 的出标签为空，因此返回结果为不可达。

下面分析查询算法的时间复杂度。由于索引项中的 hop 点按照顶点编号升序排序，因此遍历索引项的时间复杂度为 $O(n+m)$，其中 n 为出标签索引的 hop 点个数，m 为入标签索引的 hop 点个数。虽然一个 hop 点可能存在多条路径，它们对应不同的距离和标签集，但只会增加判断约束是否满足条件的次数，判断 k 值是否满足约束可以在 $O(1)$ 时间内完成，判断标签是否满足约束的时间取决于标签的长度 $|\zeta|$。因此，最终查询算法的时间复杂度为 $O((n+m)\times|\zeta|)$。

4　LK2H+ 算法

LK2H 算法对每一个顶点都构建一组 2-Hop 索引，导致索引的构建时间较长，占用空间较大。因此，本文对 LK2H 算法进行优化，提出了 LK2H+ 算法。

LK2H+ 算法的基本思想是对图中的部分顶点构建包含标签和 k 值的 2-Hop 标签索引，然后在此索引的基础上进行查询。本文选取的部分顶点为图的最小顶点覆盖集中的顶点，提取最小顶点覆盖集的算法由文献[27]提出。LK2H+ 算法分为索引构建和基于索引的查询两个主要步骤。

4.1　索引构建

LK2H+ 算法的基本步骤是：1）提取出图 G 中的最小顶点覆盖集；2）对覆盖集中的顶点按度的大小降序排列并按顺序做剪枝 BFS 遍历。剪枝规则与 3.1 节中的剪枝规则相同。算法 3 给出了 LK2H+ 算法构建索引过程的伪代码描述。

算法 3　LK2H+

```
输入：G
输出：L_k
1. C ← computeVertexCover(G)
   /* 求解图 G 的最小顶点覆盖集并赋值给 C */
2. for each v in C do/* 对于集合 C 中的顶点做剪枝 BFS */
3.    L_i^∧ ← PrunedBFS_MVC(G,v,L_{i-1},C)
4.    L_i ← PrunedBFS_MVC(G^T,v,L_i^∧,C)
5. return L_k
函数 PrunedBFS_MVC
输入：(G,v_k,L_{k-1},C)
输出：L_k
1. 初始化：Q←{(v_k,0)} /* Q 为队列 */
```

2. 初始化：$L_k \leftarrow L_{k-1}$
3. while Q≠∅ do
4. v ← pop(Q)
5. if 顶点 v 不在 C 中 then
6. for every edge e in neighbour$_G$(v) do
7. Q. push({e, l_n, k_n})
/ * l_n 和 k_n 为 v_k 到 e 的标签和步长 * /
8. else
9. if e 已被处理 then
10. Continue
11. if (h, l_n, k_n) 被 (h, l_e, k) ∈ L_k 支配 then
12. Continue
13. Q. push({e, l_n, k_n})
14. return L_k

接下来将通过例 3 并结合图 2 来解释 LK2H+算法的索引构建过程。

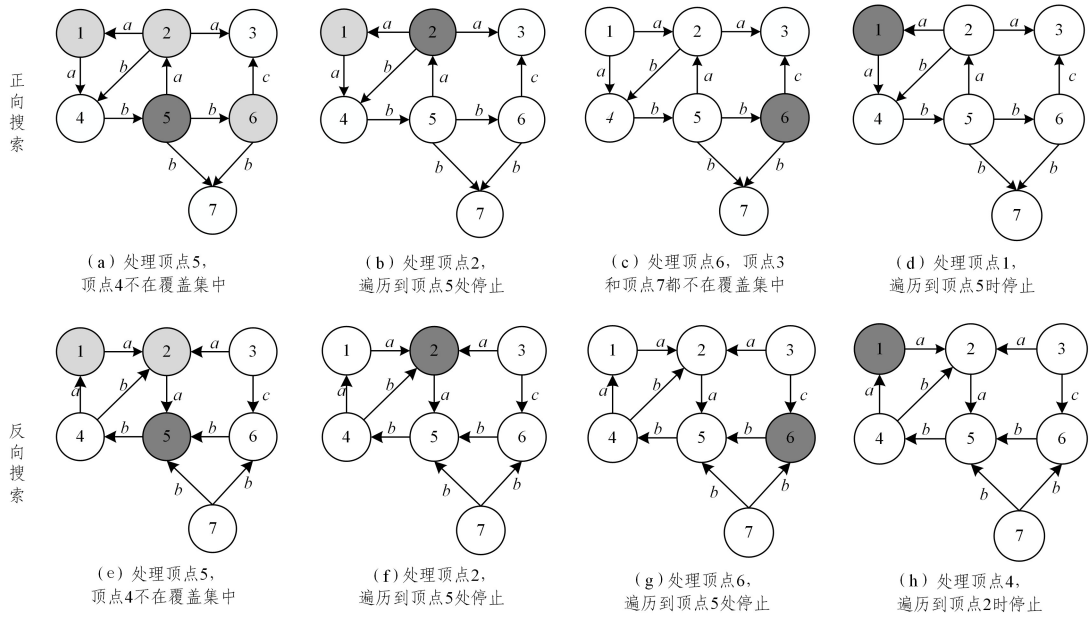

图 2 LK2H+算法的索引构建过程

Fig. 2 Index construction process of LK2H+ algorithm

例 3 图 2 给出了 LK2H+算法构建索引的完整过程。其中，深灰色顶点表示当前处理的顶点，浅灰色顶点表示访问到并且要更新索引的顶点，白色顶点表示未被访问到或者被访问到但不需要更新索引的顶点。在索引构建之前，提取出图上的最小顶点覆盖集并按度的大小降序排列后的结果为{5,2,6,1}(算法 3 第 1 行)，然后依次处理这 4 个顶点。以构建入标签索引的正向剪枝 BFS 搜索过程为例，处理顶点 5 时，可以访问到所有顶点，但只修改在覆盖集 C 中的顶点(函数 *PrunedBFS_MVC*，第 8 行)，访问到不在覆盖集 C 中的顶点 3，4 和 7 时，只把经过它们的路径标签和 k 值记录下来而不插入索引项(函数 *PrunedBFS_MVC*，第 7 行)；处理 2 号顶点时，仅更新顶点 1 的入标签索引，虽然顶点 4 可以被访问到，但是不更新它的索引，访问到顶点 5 时停止；其余顶点的处理与顶点 5、顶点 2 的处理过程相同，出标签的构建过程同理。最终构建好的标签索引如表 2 所列。

表 2 图 2 示例生成的索引

Tabel 2 Index of Fig. 2

ID	入标签索引	出标签索引
1	(5,a,2)(2,a,1)	(5,ab,2)
2	(5,a,1)	(5,b,2)
5	—	—
6	(5,b,1)	—

表 2 列出了 LK2H+算法构建出的 2-Hop 索引。在标签约束图相同的情况下，与表 1 给出的索引相比，表 2 中的索引项数目明显减少。第一轮处理顶点 5，虽然每个顶点都可以访问到，但只修改在最小顶点覆盖集中顶点的索引，即将(5,a,2)，(5,a,1)和(5,b,1)分别插入到顶点 1，2 和 6 的入标签索引中，将标签(5,ab,2)和(5,b,2)分别插入到顶点 1 和顶点 2 的出标签索引中。第 2 轮处理顶点 2，将(2,a,1)插入顶点 1 的入标签索引中。对其余顶点的处理不会使索引发生变化。

下面分析 LK2H+算法的索引构建过程的时空复杂度。由于算法仅对覆盖集中的 c 个顶点构建索引，每一个顶点最多会存 $c-1$ 个点和$2^{|\zeta|}$个不同的标签集，因此空间复杂度为 $O(c^2 2^{|\zeta|})$。求解覆盖集的算法可以在线性时间[27]内完成。对每一次剪枝 BFS，到达一个顶点的标签不同的路径数最多是$2^{|\zeta|}$，由于对覆盖集中的每一个顶点都需要标签检查，故标签检查的最坏时间复杂度为 $O(c * 2^{|\zeta|})$，因此构建索引的时间复杂度为 $O(m * c^2 * 2^{2*|\zeta|})$。

4.2 查询算法

引理 1[27] 如果一个顶点不属于覆盖集，那么它的所有邻居都属于覆盖集。

基于引理 1，LK2H+算法的查询算法的基本思想是将查询的顶点对分为不同的类别：1)在覆盖集中的查询顶点，可以直接使用构造好的索引进行查询；2)对于不在覆盖集中的顶点，将 k 步可达查询转换成 $k-1$ 步或 $k-2$ 步可达性查询。算法 4 给出了查询过程的伪代码描述。

算法 4 LK2H+查询算法

输入：(u,v,k,l,C)

```
输出：TRUE/FALSE
1. if 顶点 u 和 v 都是 C 中的元素 then
2.    return baseQuery(u,v,l,k,∅,∅)
3. if 只有顶点 u 是 C 中的元素 then
4.    for each t∈in_G(v) do /* 查找顶点 v 的入邻居 */
5.       if baseQuery(u,t,l,k−1,λ(t,v),∅) then
6.          return TRUE
7. if 只有顶点 v 是 C 中的元素 then
8.    for each s∈out_G(u) do /* 查找 u 的出邻居 */
9.       if baseQuery(u,t,l,k−1,∅,λ(u,s)) then
10.         return TRUE
          /* 顶点 u 和 v 都不是 C 中的元素 */
11. for each s∈out_G(u) do
12.    for each t∈in_G(v) do
13.       if baseQuery(s,t,l,k−2,λ(t,v),λ(u,s)) then
14.             return TRUE
15. return FALSE
函数 baseQuery
输入：(u,v,l,k,ul,vl)
输出：TRUE/FALSE
1. if 顶点 u 和 v 是同一个顶点 then return TRUE
2. for every index entry I_i in L_out[u] do
3.    for I_j in L_in[v] has same vertex as I_i.vertex do
4.       if 距离满足约束 then
5.          if 标签满足约束 then
6.             return TRUE
7.          if l⊆ I_i.l+I_j.l+ul+vl then
          /* 给定标签是可达路径标签集的子集 */
8.             记录标签集I_i.l+I_j.l+ul+vl
9.             return FALSE
10. return FALSE
```

基于图 2 所示的标签约束图 G，其最小顶点覆盖集是由顶点 5，2，6 和 1 组成的集合，表 2 列出了图 G 上的 2-Hop 标签索引。给定一个可达性查询 $Q(u,v,L,k)$，根据顶点 u 和顶点 v 是否属于覆盖集可以分为以下 4 种情况：

(1) u，v 都属于覆盖集；

(2) u 属于覆盖集，v 不属于覆盖集；

(3) u 不属于覆盖集，v 属于覆盖集；

(4) u，v 都不属于覆盖集。

对于第一种情况，由于顶点 u 和 v 都属于覆盖集，因此可以直接通过构造好的 2-Hop 索引回答查询。例如查询(5，1，a，2)，顶点 5 和顶点 1 都是覆盖集中的顶点，则直接遍历两个顶点的索引(算法 4 第 1、第 2 行)。顶点 1 的入标签索引中有 5 号顶点，并且索引项中的标签集、k 值与查询中的标签集、k 值相同，因此查询结果为可达。

对于第二种情况，由于顶点 v 不属于覆盖集，顶点 v 没有构建索引，因此无法直接使用索引回答。但 v 的入邻居都属于覆盖集，因此可以将问题转化为判断顶点 u 到顶点 v 的入邻居是否是 $k-1$ 步可达来间接回答查询。比如查询(2，8，b，3)，顶点 2 是最小顶点覆盖集中的元素，顶点 8 不是最小顶点覆盖集中的元素，需要访问顶点 8 的两个入边顶点(算法 4 第 4 行)，分别是顶点 5 和顶点 6。而 2 的出标签索引中有 hop 点 5，从顶点 2 到顶点 5 标签为 b，距离为 2，再加上从顶点 5 到顶点 8，标签为 b，距离为 3，都符合查询的约束，故查询(2，8，b，3)可达。

第三种情况与第二种情况类似，可以通过判断顶点 u 的出邻居顶点到顶点 v 的 $k-1$ 步可达来间接回答查询(算法 4 第 8 行)。

对于第四种情况，其解决方法与第二种情况没有本质区别，需要将查询转换成 u 的出邻居顶点到 v 之间的 $k-2$ 步可达查询(算法 4 第 13 行)。

下面分析查询算法的时间复杂度。查询时最坏的情况发生在两个顶点都不在覆盖集中，此时最多进行 $deg_{out}(u)\times deg_{in}(u)$ 次比较，每一次比较花费的时间为 $\bar{n}+\bar{m}$，$\bar{n}$ 和 $\bar{m}$ 分别为顶点 u 出邻居和顶点 v 入邻居的平均标签长度。判断 k 是否满足约束的时间复杂度为 O(1)，判断标签是否满足约束的时间复杂度为 $O(|\zeta|)$。因此总的时间复杂度为 $O(deg_{out}(u)\times deg_{in}(u)\times(\bar{n}+\bar{m})\times|\zeta|)$。

5　实验分析

5.1　实验环境

本文实验中所使用的硬件平台为 11th Gen Intel(R) Core(TM) i5-11300H @ 3.10GHz 3.11GHz CPU，16GB 内存，操作系统为 Windows 10。本文实验使用 C++语言实现。

实验部分首先介绍了实验所用的图数据集的来源、大小以及相应查询集的生成过程；其次，基于 15 个数据集对比和分析了两种算法的性能，其中包括索引大小、索引构建时间和基于索引的查询时间；最后展示了 k 递增时 LK2H+算法查询时间的变化，并对其结果进行了分析。

5.2　数据集

本文使用 15 个数据集进行测试，如表 3 所列。这些数据集均来自斯坦福大型网络数据集(snap.stanford.edu/data/)，且都是有向图。在这 15 个有向图的每条边上随机生成一个标签，使其成为标签约束图，每个图数据集的标签总数给定为 8。表 3 中第 1 列是数据集的名称，第 2 列和第 3 列分别对应每个数据集的顶点数和边数，本文将顶点数大于 10 万的数据集称为大数据集，否则称为小数据集。本次实验共使用 5 个小数据集和 10 个大数据集。每个查询集中包含 100 万个查询项，其中有 50 万可达查询和 50 万不可达查询。

表 3　数据集表

Table 3　Datasets

数据集	$\|V\|$	$\|E\|$
amaze	3710	3600
go	6793	13361
pubmed	9000	40082
citeseer	10720	44258
human	38811	39576
Web_Google	371764	517805
Soc_LiveJournall	971232	1024140
10go_uniprot	469526	3476397
05citeseerx	1457057	3002252
uniprot22m	1595444	1595442
WikiTalk	2281879	2311570
cit-Patents	3774768	16518947
go_uniprot	6967956	34769339
govwild	8022880	23652610
uniprot100m	16087295	16087293

可达查询的生成步骤如下：随机选择一个起始点 u 和终点 v，将 v 初始化为 u，并设置一个集合 l 用来存储路径上的标签，l 初始为空。当顶点 v 存在出邻居时，随机选择一个邻居并将它设置为顶点 v，将边上的标签加入集合 l。重复上述步骤，直到 v 不存在出邻居或已经执行了 k 次。将 (u,v,k,l) 作为一个可达查询项加入查询集。

不可达查询的生成步骤如下：随机选择一个源顶点 u 和目标顶点 v，并设置集合 l 用来存储一条路径上的标签。从顶点 u 开始 BFS 遍历，在遍历的过程中将标签加入集合 l。如果顶点 v 在 u 的 k 步传递闭包中，则随机将标签集中的一个标签修改成与原来不同的标签，然后将 (u,v,l,k) 作为一个查询项加入查询集。如果顶点 v 不在 u 的 k 步传递闭包中，则直接将 (u,v,l,k) 加入查询集，重复上述操作直到生成 50 万个不可达查询。

5.3 性能分析

5.3.1 索引大小对比与分析

表 4 对比了 15 个数据集分别对应的用 LK2H 算法和 LK2H+算法求解的索引大小，－表示构建索引的时间超过 6h，暂时没有实验结果。从表 4 中可以看到，由于 LK2H+算法只构建了图上部分顶点的索引，并且在构建索引的过程中会跳过不属于最小顶点覆盖集中的顶点，因此 LK2H+算法构建出来的索引占用空间整体比 LK2H 算法的索引所占空间小很多。例如图 10go_uniprot-，不使用 LK2H+算法优化的索引大小比优化后的索引大小大了近 20 倍。

表 4 索引大小

Table 4 Index size

（单位：MB）

数据集	LK2H	LK2H+
amaze	0.108	0.540
go	1.470	0.890
pubmed	9.570	1.265
citeseer	7.170	1.528
human	2.010	1.113
Web_Google	22.900	7.120
Soc_LiveJournall	49.830	23.010
10go_uniprot	181.290	9.670
05citeseerx	886.890	614.720
uniprot22m	126.400	29.180
WikiTalk	133.200	61.900
cit-Patents	389.140	176.350
go_uniprot	—	131.910
govwild	—	249.260
uniprot100m	—	272.690

5.3.2 索引创建时间对比与分析

表 5 列出了 15 个数据集分别对应的用 LK2H 算法和 LK2H+算法创建索引的时间，－表示运行时间超过 6h，暂时没有实验结果。LK2H 算法需要对图上的每个顶点都进行剪枝 BFS 操作；而 LK2H+算法仅需要对最小顶点覆盖集中的顶点进行剪枝 BFS 操作，访问到不属于覆盖集的顶点时也不需要插入索引项，因此也减少了剪枝的时间，故使用 LK2H+算法创建索引的时间会比 LK2H 算法更短。并且由于 LK2H+算法只选取部分顶点进行索引的创建，LK2H 算法需要构造全部顶点的索引，因此对于许多顶点数达百万级的大图，LK2H 算法无法在可接受的时间内运行出结果，但 LK2H+算法可以运行出来。这说明 LK2H+算法比 LK2H 算法更具可扩展性。

表 5 索引时间

Table 5 Index time

（单位：s）

数据集	LK2H	LK2H+
amaze	0.103	0.018
go	0.463	0.175
pubmed	4.785	0.300
citeseer	3.896	0.706
human	5.197	0.216
Web_Google	506.700	81.600
Soc_LiveJournall	3255.747	215.100
10go_uniprot	870.600	34.900
05citeseerx	7965.380	947.250
uniprot22m	7882.770	228.100
WikiTalk	18580.100	139.510
cit-Patents	16532.000	7145.910
go_uniprot	—	545.200
govwild	—	1867.260
uniprot100m	—	5698.254

5.3.3 查询时间对比与分析

表 6 列出了各个数据集分别对应的查询时间，－表示由于索引构建时间超过 6h，暂时没有索引构建结果，导致无法进行查询。数据显示：总体上 LK2H+算法所用的查询时间比 LK2H 算法所用的时间稍长。这是因为 LK2H 算法在创建完索引后，所有的顶点都可以通过这个索引直接进行查询，然而 LK2H+算法所构建的索引只有当查询顶点对都是最小顶点覆盖集里的顶点时才可以直接使用索引，其他情况下都要去访问查询顶点的邻居顶点。在最坏的情况下，需要遍历两个查询点的所有邻居，因此 LK2H+算法的查询时间会比 LK2H 算法的查询时间更长，但其查询时间还是在可接受的范围内。

表 6 查询时间

Table 6 Query time

（单位：ms）

数据集	LK2H	LK2H+
amaze	5	7
go	6	7
pubmed	6	12
citeseer	9	23
human	11	26
Web_Google	6	32
Soc_LiveJournall	14	52
10go_uniprot	81	297
05citeseerx	509	843
uniprot22m	48	69
WikiTalk	24	119
cit-Patents	215	699
go_uniprot	—	175
govwild	—	135
uniprot100m	—	247

5.3.4 不同 k 对查询时间影响的对比

表 7 列出了当 k 分别等于 2，6 和 10 时 15 个数据集的查询时间变化。索引使用 LK2H+算法进行构建，除 k 以外的查询信息不变。实验结果显示查询时间随着 k 值的增大而

减少。这是由于在查询过程中，只有判定为可达才会在不遍历完全部 hop 点的情况下停止查询。当 k 较小时，相当于对查询条件的要求更高，步长小于或等于 k 的可达路径更少，从而需要遍历更多的 hop 点才能确定是否可达。当 k 增大时，相当于查询条件放宽，步长小于或等于 k 的可达路径更多，因此更容易遇到满足条件的 hop 点，所有可达的查询项能够更快地返回结果。除此之外，还可以发现 k 发生变化时查询时间的变化不大，这说明 LK2H+算法在查询时有较好的稳定性。

表 7　查询时间

Table 7　Query time

（单位：ms）

数据集	$k=2$	$k=6$	$k=10$
amaze	13	8	6
go	27	9	7
pubmed	58	32	27
citeseer	46	19	10
human	33	28	25
Web_Google	67	39	32
Soc_LiveJournall	88	52	50
10go_uniprot	459	367	312
05citeseerx	952	865	843
uniprot22m	104	79	72
WikiTalk	178	139	116
cit-Patents	985	759	703
go_uniprot	262	196	181
govwild	262	164	139
uniprot100m	285	287	249

通过两种算法的性能对比实验可以发现，LK2H 算法和优化的 LK2H+算法各有优劣。在现实生活中，处理顶点数为百万级以上的标签约束图时，可以使用 LK2H+算法。这是由于大图本身就包含很多信息，如果将其全部预处理并保存下来需要很多时间和空间，这有时是无法接受的。LK2H+算法虽然牺牲了一点查询时间，但在构建索引时节省的时间和空间是巨大的，且查询时间与 LK2H 算法相差不大。而对于小图来说，可以选择 LK2H 算法，这是由于即使把小图上所有可达路径的信息预存下来，所需要的时间和空间消耗与存储部分信息所需要的时空消耗也相差不大，并且能快速地获得查询结果。

结束语　本文提出了标签约束图上的 k 步可达性查询问题，并针对此问题首先提出了 LK2H 算法，该算法包括构建索引和查询两个部分。LK2H 算法通过构建包含标签和距离的 2-Hop 索引，使得索引能够记录全部可达和约束信息。在 LK2H 算法的查询过程中，如果一个查询因为缺少标签而不可达，则在返回否定结果的同时将完整的标签集返回给用户，以便在后续工作中做进一步的查询和分析。另外，本文提出了 LK2H+算法。LK2H+算法在构建索引时只考虑图中最小顶点覆盖集中的顶点，从而能够进一步缩减索引构建时间和索引大小。基于 15 个真实数据集的实验结果表明，LK2H 算法和 LK2H+算法都能够有效解决标签约束图上的 k 步可达性查询问题。构建索引时，LK2H+算法能够更快地构建一个更小的索引；查询时，LK2H 算法用时更短。两种算法各有优劣。实验还表明 k 值能够影响查询时间，即 k 值越大，查询时间越短。

标签约束图上的 k 步可达性查询仍存在一些挑战，比如现在图的规模越来越大，随着新型事物的产生标签类别变得更多，稠密图上索引的构建速度较慢等等。因此，未来的研究方向如下：

(1)能够在上亿规模的标签约束图上进行查询。随着社会发展，各种网络尤其是社交网络中顶点的数量和标签数量剧增，这些数据量大的网络更需要先进的算法进行查询。因此，在超大图上的可达性查询是一个重要的研究方向。

(2)进一步缩减大图上预处理的时间和所占用的空间。对于一些超大图来说，查询之前的预处理往往会花费很多的时间，得到的处理结果也会占用很多空间。这会带来很多额外的开销。

(3)进一步提出动态标签约束图上的可达性查询问题。目前的算法只能解决静态的标签约束图可达性查询问题，但现实世界是不断变化的，图变化时查询结果也会发生相应的变化。因此，研究动态的标签约束图的可达性查询问题是必要的。

参 考 文 献

[1] KUMAR R, ALEXANDER T, FALOUTSOS C, et al. Social Networks: Looking ahead [C]// Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Las Vegas, USA: ACM, 2008: 1060-1060.

[2] STANN F, HEIDEMANN J. RMST: Reliable data transport in sensor networks[C]// Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications. Anchorage, AK, USA: IEEE, 2003: 102-112.

[3] WOOD P T. Query languages for graph databases[J]. ACM Sigmod, 2012, 41(1): 50-60.

[4] CHEN X, LAI L, QIN L, et al. Structsim: Querying structural node similarity at billion scale[C]// Proceedings of the IEEE 36th International Conference on Data Engineering (ICDE). Dallas, TX, USA: IEEE, 2020: 1950-1953.

[5] CHEN Y J, CHEN Y B. An efficient algorithm for answering graph reachability queries[C]// Proceedings of the IEEE 24th International Conference on Data Engineering. Cancun, Mexico: IEEE, 2008: 893-902.

[6] CHENG J, HUANG S L, WU H H, et al. TF-Label: a topological-folding labeling scheme for reachability querying in a large graph[C]// Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. New York, USA: ACM, 2013: 193-204.

[7] FANG Y, YANG Y, ZHANG W, et al. Effective and efficient community search over large heterogeneous information networks[J]. PVLDB, 2020, 13(6): 2093-2107.

[8] WANG K, LIN X, QIN L, et al. Efficient bitruss decomposition for large-scale bipartite graphs[C]// Proceedings of the IEEE 36th International Conference on Data Engineering (ICDE). Dallas, TX, USA: IEEE, 2020: 661-672.

[9] SCHAIK S J, MOOR O D. A memory efficient reachability data structure through bit vector compression[C]// Proceedings of the 2011 ACM SIGMOD International Conference on Manage-

ment of Data. Athens, Greece: ACM, 2011: 913-924.
[10] IMANE H, YAHIAOUI S, BENDJOUDI A, et al. Reachability in Big Graphs: A Distributed Indexing and Querying Approach [J]. Information Sciences, 2021, 573: 541-561.
[11] WEI H, YU J X, LU C, et al. Reachability querying: An independent permutation labeling approach[J]. VLDB, 2018, 27(1): 1-26.
[12] YU J X, CHENG J F. Graph reachability queries: A survey [M]// Managing and Mining Graph Data. Boston, MA: Springer, 2010: 181-215.
[13] CHOUDHARY P. A survey on social network analysis for counterterrorism[J]. International Journal of Computer Applications, 2015, 112(9): 24-29.
[14] PENG Y, ZHANG Y, LIN X, et al. Answering Billion-Scale Label-Constrained Reachability Queries within Microsecond[J]. Proceedings of the VLDB Endowment, 2020, 13(6): 812-825.
[15] VALSTAR L D J, FLETCHER G H L, YOSHIDA Y. Landmark indexing for evaluation of label-constrained reachability queries[C]// Proceedings of the 2017 ACM International Conference on Management of Data. Chicago, Illinois, USA: ACM, 2017: 345-358.
[16] XIAN T, CHEN Z Y, LI K, et al. Efficient computation of the transitive closure size[J]. Cluster Computing, 2019, 22(3): 6517-6527.
[17] YANO Y, AKIBA T, IWATA Y, et al. Fast and scalable reachability queries on graphs by pruned labeling with landmarks and paths[C]// Proceedings of the 22nd ACM International Conference on Information & Knowledge Management. San Francisco, CA, USA: ACM, 2013: 1601-1606.
[18] BOSTJAN B, FRANTISEK K, JAN K, et al. Minimum k-path vertex cover[J]. Discrete Applied Mathematics, 2011, 159(12): 1189-1195.
[19] VELOSO R, JUNIOR W M, CERF L P, et al. Reachability queries in very large graphs: A fast refined online search approach [C]// Proceeding of the 17th International Conference on Extending Database Technology. Athens, Greece: EDBT, 2014: 511-522.
[20] CHENG J, SHANG Z, CHENG H, et al. K-reach: who is in your small world[J]. Proceedings of the VLDB Endowment, 2012, 5(11): 1292-1303.
[21] JIN R M, HONG H, WANG H X, et al. Computing label-constraint reachability in graph databases [C]// Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. Indianapolis, Indiana, USA: ACM, 2010: 123-134.
[22] HASSAN M S, AREF W G, ALY A M. Graph indexing for shortest-path finding over dynamic sub-graphs[C]// Proceedings of the 2016 International Conference on Management of Data. San Francisco, CA, USA: ACM, 2016: 1183-1197.
[23] SARISHT W, PRASAD A, RANU S, et al. Efficiently answering regular simple path queries on large labeled network[C]// Proceedings of the 2019 International Conference on Management of Data. 2019: 1463-1480.
[24] CHEN Y J, SINGH G. Graph Indexing for Efficient Evaluation of Label-constrained Reachability Queries[J]. ACM Transactions on Database Systems, 2021, 46(2): 1-50.
[25] BEAMER S, ASANOVIC K, PATTERSON D. Direction-optimizing Breadth-first search[J]. Scientific Programming, 2013, 21(3/4): 137-148.
[26] STEIER D M, ANDERSON A P. Depth-First Search [M]// Algorithm Synthesis: A Comparative Study. US: Springer, 1989.
[27] WANG X, LIN K, DU M, et al. Efficient k-Hop Reachability Queries Processing on Directed Graphs[D]. Shanghai: Donghua University, 2020.

DU Ming, born in 1975, Ph.D, associate professor. His main research interests include natural language processing, information service, and data analysis.

XING Rui-ping, born in 1997, postgraduate. Her main research interests include graph reachability and so on.

(责任编辑:柯颖)