基于强连通分量的最短环计数索引

杨迎, 周军锋, 杜明

引用本文

杨迎, 周军锋, 杜明. 基于强连通分量的最短环计数索引[J]. 计算机科学, 2025, 52(4): 169-176.

YANG Ying, ZHOU Junfeng, DU Ming. Index for Counting the Shortest Cycle Based on Strongly Connected Components [J]. Computer Science, 2025, 52(4): 169-176.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.240200105

基于强连通分量的最短环计数索引

杨 迎 周军锋 杜 明
东华大学计算机科学与技术学院 上海 201600
(2212623@mail.dhu.edu.cn)

摘 要 最短环计数是图分析的一种基本模式。经过某个顶点的最短环计数指经过该顶点且长度最短的环的数目。在现实生活中,最短环计数应用十分广泛,如欺诈交易检测、罪犯预筛选以及文件共享优化等。针对现有方法索引空间较大、查询效率较低等问题,研究如何在原始图上构建最短环计数索引,提出了一种针对最短环计数且无需进行图转换操作的 STC 索引(Trough Shortest Cycle Counting Index)。该索引根据最短环的特征对其进行分类,针对不同类型的最短环分别构建不同的索引信息,能够直接基于原始图构造索引,并且在保证索引规模不扩大、索引构造时间不增加的前提下,进一步提升查询效率。此外,根据环与强连通分量的特殊关系,提出了基于强连通分量的索引策略,通过在强连通分量内部构造最短环计数索引,可以进一步提升索引构造效率,有效减小索引规模,提升查询效率。在 10 个真实数据集上进行了实验。实验结果验证了所提出的 STC 索引的高效性,以及基于强连通分量的策略可以有效减小索引空间,提升索引构造以及查询效率。

关键词:图分析;最短环;最短环计数;2-hop 索引;强连通分量
中图分类号 TP301

Index for Counting the Shortest Cycle Based on Strongly Connected Components

YANG Ying,ZHOU Junfeng and DU Ming
School of Computer Science and Technology,Donghua University,Shanghai 201600,China

Abstract The shortest cycle counting is a basic pattern of graph analysis. The shortest cycle counting through a certain vertex refers to the number of shortest cycles passing through the vertex. In real life,the shortest cycle counting has a wide range of applications,such as fraudulent transaction detection,criminal pre-screening,and file sharing optimization. In response to the problems of large indexing space and low query efficiency of existing methods,this paper studies the construction of shortest cycle counting indexing on the original graph,and proposes a trough shortest cycle counting index(STC index) that does not require graph transformation operations. The index classifies the shortest cycles according to their characteristics,constructs different indexing information for different types of shortest cycles,and can directly construct indexes based on the original graph,and further improve query efficiency while ensuring that the indexing size does not expand and the indexing construction time does not increase. In addition,based on the special relationship between cycles and strongly connected components,this paper proposes an indexing strategy based on strongly connected components. By constructing shortest cycle counting indexing within strongly connected components,it can further improve the efficiency of indexing construction,effectively reduce the index size,and improve query efficiency. Experiments on 10 real datasets verify the efficiency of the proposed STC index and the strategy based on strongly connected components,which can effectively reduce the indexing space and improve the efficiency of indexing construction and querying.

Keywords Graph analysis,Shortest cycle,Shortest cycle counting,2-hop index,Strongly connected components

1 引言

随着信息技术的飞速发展,越来越多的应用领域以图的形式来对现实世界中实体之间的复杂关系进行建模和表示[1]。对社交网络、电子商务网络以及电子支付网络等动态变化的大型网络进行分析,有助于深入了解底层实体之间的复杂交互情况[2-6]。最短环计数是图分析的重要基础。经过某个顶点的最短环计数指经过该顶点且长度最短的环的数目。最短环计数在现实生活中应用十分广泛。在交易网络中,最短环计数可用于欺诈交易检测,有效发现潜在的欺诈群体[7-10]。通过顶点的最短环数量越多,个人参与洗钱活动的可能性就越大,对违规洗钱的人的预筛选标准就包括其所在

到稿日期:2024-02-27 返修日期:2024-06-26
基金项目:国家自然科学基金(62372101,61873337,62272097)
This work was supported by the National Natural Science Foundation of China(62372101,61873337,62272097).
通信作者:周军锋(zhoujf@dhu.edu.cn)

的最短环数目[7,11]。在点对点文件共享网络[12]中，最短环计数可以优化文件共享效率。具体来说，网络中的顶点代表主机，而边表示主机之间的交互。对于文件请求或传输来说，通过主机的文件共享回路标志着文件共享活动的结束。通常，该操作会选择跳数最少的有效路由。如果经过主机的最短环较长且较多，则可能需要代理服务器来降低传输成本。文献[13]介绍了 P2P 文件共享的索引服务器优化问题，将具有大量最短环的机器作为首选服务器。

现有的用于求解最短环计数问题的方法主要分为 3 类：1)基于直接搜索遍历的算法，如 BFS，BI-BFS 算法；2)运用现有的最短路径计数算法来求解最短环计数问题，将一个最短环计数问题转化为多个最短路径求解问题[14-17]；3)Feng 等提出的专门回答最短环计数的算法 CSC[1]，该算法的主要思想是构建原始图对应的转换图，并在转换图上构建基于最短路径计数的 2-hop 索引来加快最短环计数的求解速度。给定查询顶点 u，经过 u 的最短环数目可通过检查 u 的出顶点标签和 u 的入顶点标签来确定二者之间的最短路径数目，该数目即为经过 u 的最短环计数。该过程只需要执行一次基于 2-hop 索引的查询操作，因此该算法相较于前两类算法查询效率较高。然而，该算法需要提前对原始图进行图转换操作，并且在索引构造过程中需要对 $2*n$ 个顶点构造索引，其中 n 为原始图的顶点数目，因此在索引构造过程中对内存要求更高。此外，现有算法不论是基于直接搜索遍历的算法还是基于构建索引的算法，都是在全图上进行操作的，都会面临索引构造时间长、索引空间开销大、查询时间过长等诸多问题。因此，构建一个规模更小且构造和查询过程更为高效的索引十分必要。

本文对最短环计数查询问题进行研究，提出了更为高效的索引解决方案。具体贡献如下：

1)提出了针对最短环计数的 STC 索引，该索引根据最短环的特征将其进行分类，针对不同类型的最短环分别构建不同的索引信息，能够直接基于原始图构造索引，并且在保证索引规模不扩大、索引构造时间不增加的前提下，进一步提升查询效率。

2)提出了基于强连通分量的索引策略，该策略根据环和强连通分量的特殊关系，在强连通分量内部构造最短环计数索引，可以进一步提升索引构造效率，有效减小索引规模，提升查询效率。

3)在 10 个真实数据集上了进行实验，结果表明，与现有方法相比，本文方法的索引规模更小，索引构造和查询效率更高。

2 背景知识和相关工作

2.1 背景知识

给定有向图 $G(V,E)$，其中 V 表示图 G 的顶点集，$E\subseteq V\times V$ 表示图 G 的边集，n 表示图 G 的顶点数目，m 表示图 G 的边数。$e=\langle u,v\rangle$ 表示从顶点 u 到顶点 v 的一条有向边。对 $\forall u\in V$，定义 $in(u)=\{v|\langle v,u\rangle\in E\}$ 为顶点 u 的入邻居顶点集或前驱顶点集，$out(u)=\{v|\langle u,v\rangle\in E\}$ 为顶点 u 的出邻居顶点集或后继顶点集。定义顶点 u 的入度为该顶点的入邻居顶点集大小，即 $|in(u)|$，出度为该顶点的出邻居顶点集大小，即 $|out(u)|$。按照度数对图 G 中的顶点降序排序，得到总体排序 O。当两个顶点度数相等时，则根据顶点 ID 确定一个全局唯一的顺序。在后续讨论中，若顶点 u 的排序高于顶点 v，则 u 位于 v 的左边，用 $u<v$ 表示。定义从顶点 u 出发到顶点 v 的一条有向路径为 $p(u,v)=\langle u=v_0,v_1,\cdots,v_{k-1},v_k=v\rangle$，$p(u,v)$ 的长度等于该路径中边的数目。若一条路径上途经的所有顶点均不重复，则称该路径为简单路径。对于顶点 u 到顶点 v 的一条路径 $p(u,v)$，若端点 u 或 v 为该路径上排序最高的顶点，则称该路径为槽路径。如果顶点 u 到顶点 v 之间的一条路径 $p(u,v)$ 的长度小于等于顶点 u 和 v 之间任何其他路径的长度，则称路径 $p(u,v)$ 为顶点 u 和 v 之间的一条最短路径，用 $sp(u,v)$ 表示，长度记为 $sd(u,v)$。若最短路径 $p(u,v)$ 为槽路径，则称该路径为槽最短路径。$SP(u,v)$ 表示顶点 u 和 v 之间所有最短路径的集合；$SPCnt(u,v)$ 表示顶点 u 和 v 之间最短路径的数量，即 $SPCnt(u,v)=|SP(u,v)|$。对于有向图 G，若 G 中存在一条以 u 为起点、以 v 为终点的有向路径，并且存在一条以 v 为起点、以 v 为终点的有向路径，则顶点 u 和 v 是强连通的。如果有向图 G 的任意两个顶点都具有强连通性，则称 G 是一个强连通图。有向图的极大强连通子图称为强连通分量(Strongly Connected Component，SCC)。

环(Cycle)是起始点和结束点相同的非空路径。环的长度等于环中边的数目。c_v 表示经过顶点 v 的环，长度为记为 $len(v)$。只有起始顶点和结束顶点相同的简单路径被称为简单环。如果通过顶点 v 的某个环，其长度不大于通过 v 的任何其他环的长度，则称其为顶点 v 的一条最短环(Shortest Cycle，SC)。通过顶点 v 的最短环数目记为 $SCCnt(v)$。表 1 列出了后文中涉及的常用符号。

问题定义：给定有向图 G 和任意顶点 v，返回 v 的最短环计数 $SCCnt(v)$。

表 1 本文所用的符号及其意义

Table 1 Symbols in this paper and their meanings

符号	意义
$G=(V,E)$	顶点集为 V、边集为 E 的有向图 G
$sp(u,v)$	顶点 u 到顶点 v 之间的一条最短路径
$sd(u,v)$	顶点 u 到顶点 v 之间的最短距离
$SP(u,v)$	顶点 u 到顶点 v 之间的所有最短路径的集合
$SPCnt(u,v)$	顶点 u 到顶点 v 之间的最短路径的数量
$SCCnt(u)$	经过顶点 u 的最短环的数量

2.2 相关工作

2.2.1 最短环计数

现有方法中，最短环计数问题是基于最短路径计数问题进行求解的。根据最短路径计数算法的应用场景，可以将最短环计数的方法分为两类：一类是基于原始图的最短路径计数问题进行求解；另一类是基于转换图的最短路径计数问题进行求解。下面分别进行讨论。

1)该类方法[14-17]直接在原始图上构建基于最短路径计数的 2-hop 索引，将一个最短环计数查询问题转化为多个最短路径计数查询问题。

2)该类方法[1]是在转换图上构建基于最短路径计数的 2-

hop 索引，对于一个最短环计数查询问题只需要进行一次最短路径计数查询操作，因此相较于第一类算法其查询效率更高。该方法的具体操作是：首先将原图中的所有顶点一分为二，一个顶点（入顶点）连接入邻居，另一个顶点（出顶点）连接出邻居，同时在这两个顶点之间添加一条边。以此构造原图对应的转换图。然后，在转换图上构建针对最短路径计数的 2-hop 索引。给定查询点 u，通过检查 u 的出顶点标签和 u 的入顶点标签来确定二者之间的最短路径长度 d 以及数目 c，则经过顶点 u 的最短环长度为 $(d+1)/2$，计数为 c。可以看出，该过程只需要执行一次最短路径查询操作，即可获得最短环计数，因而查询效率较高。然而，由于图转换操作将图的规模几乎扩大为原来的两倍，因此在索引构建过程中对内存空间的要求更高。

2.2.2　最短路径计数

现有方法关于最短路径计数问题的求解都是基于 2-hop 索引策略。文献[14]提出了精确最短路径覆盖约束（ESPC），可以覆盖任意两点之间的最短路径，保证准确计算出最短路径的数量，并且提出了在无向图上计算最短路径数量的 HP-SPC 算法。为了解决 HP-SPC 算法索引空间大、构建时间长等问题，文献[15]提出了基于树状索引回答最短路径计数的算法，该算法可以根据用户给定的参数 d 将图分解为稀疏连通的树部分和密集连通的核心部分，并分别构建树状索引和类似于 HP-SPC 的索引结构。为了解决 HP-SPC 算法索引构建时间长的问题，文献[16]提出了支持并行的最短路径计数算法 PSPC；文献[17]提出了在无向图上支持动态维护最短路径计数算法 DSPC，该算法通过只定位受影响的顶点来支持高效的增边和删边操作。

上述方法都可以用来求解最短环计数问题，即可以采取上述第一类方法的主要思想，也可以采用第二类方法的主要思想。若采用第一类方法的主要思想，则只需要在原始图上构建基于最短路径计数的 2-hop 索引。这种情况下，一个最短环计数查询需转化为多个最短路径计数查询问题，因此查询效率较低。若采用第二类方法的主要思想，则首先需要将原始图进行转换，在转换图上构建基于最短路径计数的 2-hop 索引。由于图转换操作使得图的规模变大，从而在索引构建过程中对内存空间要求更高，因此，建立既能满足高效查询，又能够保证高效构建的索引至关重要。

3　基于强连通分量的 STC 索引策略

本章介绍一种新的用于最短环计数的索引——STC 索引。该索引能够在原始图上高效构建，避免了 CSC 算法的图转换操作。相比 CSC 算法，STC 的索引空间更小，索引构建时间更短，并且仍然保持高效的查询效率。

3.1　STC 索引策略

虽然文献[14]中提出了精确最短路径覆盖约束，并且提出了用于回答最短路径计数的索引 HP-SPC，但该索引在用于回答最短环计数时，可能存在丢解的问题。

例 1　对图 1 中 G_1 的顶点按照度数降序排序，得到顶点排序为 $v_0<v_1<v_2<v_3<v_4<v_5<v_6<v_7<v_8<v_9$。对该图构造 HP-SPC 索引，对应于表 2 中的中间两列。该索引为图 G_1 中的每个顶点 v 赋予一对标签，分别为 $L_{\text{in}}[v]$ 和 $L_{\text{out}}[v]$。其标签项是由形如 $(h,sd(h,v),\sigma_{h,v})$ 的三元组构成，其中顶点 h 是 v 的 hub 顶点，$sd(h,v)$ 和 $\sigma_{h,v}$ 分别表示顶点 h 和 v 之间的槽最短路径长度和计数。$L_{\text{in}}[v]$ 标签记录其 hub 顶点到 v 的最短路径距离和计数信息，$L_{\text{out}}[v]$ 标签记录顶点 v 到其 hub 顶点的最短路径距离和计数信息。当计算经过顶点 v_1 的最短环计数时，使用式(1)搜索 $L_{\text{in}}[v_1]$ 和 $L_{\text{out}}[v_1]$ 标签中除自身之外经过公共顶点的标签项，即 $L_{\text{in}}[v_1]=\{(0,1,1)\}$ 和 $L_{\text{out}}[v_1]=\{(0,4,2)\}$，通过入标签项(0,1,1)和出标签项(0,4,2)得到经过 v_1 的最短环长度为 $1+4=5$，数量为 $1*2=2$，仅能覆盖两个最短环，分别为 $c_1=\langle v_1,v_7,v_2,v_3,v_0,v_1\rangle$，$c_2=\langle v_1,v_7,v_2,v_4,v_0,v_1\rangle$。但是实际上经过顶点 v_1 的最短环有 3 个，长度为 5。也就是说，并没有将环 $c_3=\langle v_1,v_7,v_2,v_3,v_6,v_1\rangle$ 计算在内。

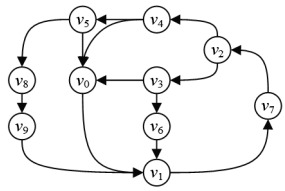

图 1　有向图 G_1

Fig. 1　Directed graph G_1

表 2　图 1 的 STC 索引

Table 2　STC index of Fig. 1

v	$L_{\text{in}}[v]$	$L_{\text{out}}[v]$	$L_{\text{STC}}[v]$
0	$(v_0,0,1)$	$(v_0,0,1)$	(5,2)
1	$(v_0,1,1)(v_1\ 0,1)$	$(v_0,4,2)(v_1\ 0,1)$	(5,1)
2	$(v_0,3,1)(v_1,2,1)(v_2,0,1)$	$(v_0,2,2)(v_1,3,1)$ $(v_2,0,1)$	(∞,0)
3	$(v_0,4,1)(v_1,3,1)(v_2,1,1)$ $(v_3,0,1)$	$(v_0,1,1)(v_1,2,1)$ $(v_3,0,1)$	(∞,0)
4	$(v_0,4,1)(v_1,3,1)(v_2,1,1)$ $(v_4,0,1)$	$(v_0,1,1)(v_4,0,1)$	(∞,0)
5	$(v_0,5,1)(v_1,4,1)(v_2,2,1)$ $(v_4,1,1)(v_5,0,1)$	$(v_0,1,1)(v_5,0,1)$	(∞,0)
6	$(v_0,5,1)(v_1,4,1)(v_2,2,1)$ $(v_3,1,1)(v_6,0,1)$	$(v_0,5,2)(v_1,1,1)$ $(v_6,0,1)$	(∞,0)
7	$(v_0,2,1)(v_1,1,1)(v_7,0,1)$	$(v_0,3,2)(v_1,4,1)$ $(v_2,1,1)(v_7,0,1)$	(∞,0)
8	$(v_0,6,1)(v_1,5,1)(v_2,3,1)$ $(v_4,2,1)(v_5,1,1)(v_8,0,1)$	$(v_0,6,2)(v_1,2,1)$ $(v_8,0,1)$	(∞,0)
9	$(v_0,7,1)(v_1,6,1)(v_2,4,1)$ $(v_4,3,1)(v_5,2,1)(v_8,1,1)$ $(v_9,0,1)$	$(v_0,5,2)(v_1,1,1)$ $(v_9,0,1)$	(∞,0)

造成以上问题的原因在于：环中顶点 v_1 恰好为该环上排序最高的顶点，即 hub 顶点。若考虑将顶点 v_1 自身作为 hub 顶点，则通过式(1)和式(2)计算得到的最短环长度和计数均为 0；若不考虑将 v_1 自身作为 hub 顶点，因为不存在其他 hub 顶点 w 能够表示该环，因此无法覆盖该环。对于 HP-SPC 标签来说，如果路径的起点和终点相同，路径长度即为 0，计数也为 0。但是对于环来说却并非如此。因此用于最短路径计数的 HP-SPC 索引并不能完全覆盖所有的最短环，故无法直接用于最短环计数问题。

$$H=\{h\mid \arg\min_{h\in L_{\text{out}}(u)\cap L_{\text{in}}(u),h\neq u}\{sd(u,h)+sd(h,u)\}\tag{1}$$

$$NSCCnt(u)=\sum_{h\in H}\sigma(u,h)\cdot\sigma(h,u)\tag{2}$$

针对上述问题，本文给出两种不同类型的环定义。

定义 1(槽环) 给定某个顶点 $u\in G$，存在一个经过顶点 u 的环 $c=\langle u,w_1,w_2,\cdots,w_k,u\rangle$。若该环上其余顶点的排序均低于顶点 u，即对任意 $i\in[1,k]$，$u<w_i$，则称该环为经过顶点 u 的槽环(Trough Cycle，TC)。

定义 2(槽最短环) 给定某个顶点 $u\in G$，若存在一个经过顶点 u 的槽环且该槽环是该顶点的最短环，则称该环为经过顶点 u 的槽最短环(Trough Shortest Cycle，TSC)。

例 2 继续例 1 的讨论。对于图 1 来说，经过顶点 v_1 的环有 5 个，分别为 $c_1=\langle v_1,v_7,v_2,v_4,v_0,v_1\rangle$，$c_2=\langle v_1,v_7,v_2,v_3,v_0,v_1\rangle$，$c_3=\langle v_1,v_7,v_2,v_4,v_5,v_0,v_1\rangle$，$c_4=\langle v_1,v_7,v_2,v_3,v_6,v_1\rangle$，$c_5=\langle v_1,v_7,v_2,v_4,v_5,v_8,v_9,v_1\rangle$。根据例 1 可知，$v_0$ 排序高于 v_1，因此环 c_1，c_2 以及 c_3 是非槽环。因为 v_1 是环 c_4 和 c_5 上排序最高的顶点，所以 c_4 和 c_5 是槽环。结合环的长度来看，因为 $len(c_1)=len(c_2)=len(c_4)=5$，$len(c_3)=6$，$len(c_5)=7$，所以 c_1，c_2 和 c_4 为经过顶点 v_1 的最短环，其中 c_1 和 c_2 是经过 v_1 的非槽最短环，而 c_4 为经过 v_1 的槽最短环。

假设经过顶点 u 所在环上的顶点集为 S，S 中除顶点 u 之外排序最高的顶点为 w。如果顶点 w 的排序高于顶点 u，即 $w<u$，则该环为非槽环；反之，如果顶点 w 的排序低于顶点 u，则表明该环上除 u 之外的所有顶点的排序均小于 u，即该环为槽环。由此可以发现，对于任意顶点 u，经过 u 的(最短)环只有两种，分别为槽(最短)环和非槽(最短)环。因此，若要计算经过某个顶点的最短环数量，则必须计算经过该顶点上的槽最短环数量和非槽最短环数量的总和，以此保证计数的正确性。

对于一个经过顶点 u 的非槽环 c_u，该环上的顶点集为 S，即 $u\in S$。根据对非槽环的定义可知，该环上一定存在至少 1 个排序高于顶点 u 的顶点 w，其中 $w\in S$。假设顶点 w 为该环上排序最高的顶点，即 $w<u$，那么顶点 w 将该环拆分为两条路径，分别为路径 $p_1=\langle u,\cdots,w\rangle$ 和路径 $p_2=\langle w,\cdots,u\rangle$，即 $c_u=p_1\oplus p_2$。假设路径 p_1 和 p_2 上经过的顶点集分别为 S_1，S_2，因为 w 是 S 中排序最高的顶点，且 $S_1\cup S_2=S$，所以 w 也分别是 S_1 和 S_2 中排序最高的点。根据槽路径的定义可知，因为 w 为 p_1 和 p_2 路径上排序最高的顶点，所以 p_1 和 p_2 均为槽路径。因此，对于任意一个非槽最短环，一定是由两个槽最短路径拼接而成。

基于上述发现，本文提出一种新的基于 2-hop 的最短环计数索引——STC 索引。该索引可以保证覆盖所有的槽最短环，从而覆盖经过给定顶点的所有最短环，正确回答经过给定顶点的最短环计数问题。图 1 中 G_1 对应的 STC 索引如表 2 所列，该索引共分为 3 部分，分别为 L_{in} 标签、L_{out} 标签以及 L_{STC} 标签，对应于表 2 中的第 2—4 列。其中 L_{in} 标签和 L_{out} 标签可以精确覆盖所有的非槽最短环，而 L_{STC} 标签可以精确覆盖所有的槽最短环。L_{STC} 标签由形为 (d_{stc},c_{stc}) 的二元组组成，其中 d_{stc} 和 c_{stc} 分别表示经过顶点 v 的最短槽环长度和数目。

例 3 继续例 1 的讨论，经过顶点 v_1 的最短环共有 3 个，长度为 5。其中，$L_{in}[v_1]$ 标签和 $L_{out}[v_1]$ 标签可以覆盖 $c_1=\langle v_1,v_7,v_2,v_3,v_0,v_1\rangle$ 和 $c_2=\langle v_1,v_7,v_2,v_4,v_0,v_1\rangle$ 两个非槽最短环，$L_{STC}[v_1]$ 标签可以覆盖槽最短环 $c_3=\langle v_1,v_7,v_2,v_3,v_6,v_1\rangle$。

3.2 基于强连通分量的 STC 索引策略

根据强联通分量的定义可知，强连通分量实际上是由若干个环组成。根据这一特性，有如下结论。

定理 1 对于任意顶点 $u\in V$，经过顶点 u 的所有环上的顶点一定与顶点 u 处于同一个强连通分量中。

证明：假设顶点 u 所在的强连通分量编号为 $Scc\_id[u]$，并且顶点 v 存在经过该顶点的环，环上的其余顶点集为 W。假设存在一个 v 所在环上的顶点 t，即 $t\in W$，但是顶点 t 与 v 不属于同一个强连通分量，即 $Scc\_id[t]\neq Scc\_id[v]$。由于顶点 t 在顶点 v 所在的环上，根据环的定义，则存在两条路径 $p_1=\langle v,\cdots,t\rangle$ 和 $p_2=\langle t,\cdots,v\rangle$。根据强连通性的定义，说明顶点 v 和顶点 t 是强连通的，处于同一个强连通分量中，即 $Scc\_id[t]=Scc\_id[v]$，与假设矛盾。因此对于任意顶点 v，经过 v 的所有环上的点一定与 v 处于同一个强连通分量中。证毕。

定理 2 对于任意的顶点 v，若对 v 构造索引的过程中只在 v 所在的强连通分量中进行 BFS 搜索，则最终得到的标签仍然满足最短环计数的正确性。

证明：对于最短环计数的索引，其本质上是最短路径的 2-hop 标签，因此，对于最短环计数问题的求解方法也是基于最短路径计数的求解方法。对于任意的顶点 v，经过 v 的最短环可以分为两类：第一类是该环上存在一个顶点 w 的排序高于该环上其余顶点；第二类是 v 即为该环上排序最高的顶点。下文将对两类情况分别加以证明。

关于第一类情况，假设存在一个顶点 w 既在 v 的 L_{in} 标签中，又在 L_{out} 标签中，且 w 不处于 v 所在的强连通分量中，即 $Scc\_id[w]\neq Scc\_id[v]$。因为 w 既在 v 的 L_{in} 标签中，又在 L_{out} 标签中，所以存在两条最短路径 $p_1=\langle v,\cdots,w\rangle$ 和 $p_2=\langle w,\cdots,v\rangle$，二者拼接形成环。因此根据定理 1，$w$ 与 v 处于同一个强连通分量中，即 $Scc\_id[w]=Scc\_id[v]$，与假设矛盾。若 w 仅在 v 的 L_{in} 或 L_{out} 标签中，则根据 v 的标签无法查询到由 $\langle v,\cdots,w\rangle$ 和 $\langle w,\cdots,v\rangle$ 两条最短路径拼接而成的环。所以从 v 的标签中删除 w 并不影响最短环计数的正确性。因此，若 w 在 v 所在的强连通分量之外，则在索引构建过程中无需将 w 加入 v 的标签中，即从 w 开始 BFS 搜索时，只需搜索 w 所在的强连通分量的顶点即可。

关于第二类情况，v 即为该环上排序最高的顶点，所以对于该环上 v 的前驱顶点 t_1，其 L_{in} 和 L_{out} 标签均存在 v。因此存在最短路径 $\langle v,\cdots,t_1\rangle$ 和 $\langle t_1,v\rangle$，二者拼接形成环。假设存在顶点 t_2 是 v 的入邻居顶点，且 $Scc\_id[t_2]\neq Scc\_id[v]$，$v$ 在 t_2 的 L_{out} 标签(L_{in} 标签)中，那么 v 不可能也在 t_2 的 L_{in} 标签(L_{out} 标签)中，否则就会存在两条最短路径 $\langle v,\cdots,t_2\rangle$ 和 $\langle t_2,v\rangle$，二者拼接形成环。根据定理 1，t_2 与 v 处于同一个强连通分量中，即 $Scc\_id[t_2]=Scc\_id[v]$，与假设矛盾。因此，若 t_2 在 v 所在的强连通分量之外，在索引构建过程中无需将 v 加入 t_2 的标签中，即从 v 开始 BFS 搜索时，只需搜索 v 所在的强连通分量的顶点即可。证毕。

因此，由定理 1 和定理 2 可知，为了更高效地构建索引，

只需在强连通分量内部区域进行 BFS 搜索，构建 STC 索引，而无需在全图区域构建索引。

例 4　如图 2 所示，在图 G_2 中，若直接对该图构造 STC 索引，则需要从每个顶点出发在全图区域内进行 BFS 遍历操作。假设图 G_2 中的顶点排序为 $v_0<v_1<v_2<v_3<v_4<v_5<v_6<v_7<v_8<v_9<v_{10}<v_{11}<v_{12}<v_{13}<v_{14}<v_{15}<v_{16}<v_{17}$，其中顶点 v_0 的排序最高，v_{17} 的排序最低。那么，顶点 v_0，v_1，v_2，v_3，v_4，v_5，v_6，v_7，v_8，v_9 均在顶点 v_{11} 的 L_{in} 标签中，但是实际上这些顶点与 v_{11} 并不属于同一个强连通分量，经过 v_{11} 的环并不经过这些顶点。因此，不将这些顶点添加到 v_{11} 的 L_{in} 标签中，仍然能够保持查询结果的正确性。此外，由于这些顶点不放入 v_{11} 的 L_{in} 标签中，因此 $L_{in}[v_{11}]$ 标签只包括其自身，可以显著减小索引空间，提高索引查询效率。

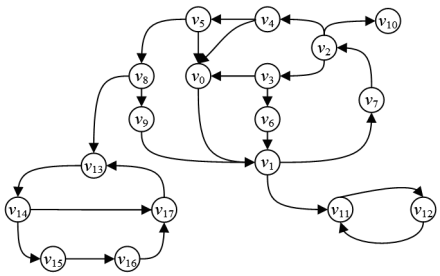

图 2　有向图 G_2

Fig. 2　Directed graph G_2

4　查询处理

基于 STC 索引，计算经过顶点 u 的最短环计数 $SCCnt(u)$，具体计算式包括 3.1 节中的式(1)和式(2)以及式(3)和式(4)。

$$TSCCnt(u)=\{c_{stc}\mid L_{STC}(u)=(d_{stc},c_{stc})\} \tag{3}$$

$$SCCnt(u)=\begin{cases}NSCCnt(u), & \text{if } sd(u,h)+sd(,h,u)<d_{stc}\\ TSCCnt(u), & \text{if } sd(u,h)+sd(,h,u)>d_{stc}\\ NSCCnt(u)+TSCCnt(u), & \text{otherwise}\end{cases} \tag{4}$$

通过式(1)找到顶点 u 的 $L_{out}[u]$和 $L_{in}[u]$标签中的公共 hub 顶点 h，h 位于经过 u 的非槽最短环上，且是该环上排序最高的顶点。通过 hub 顶点 h，可以将该环拆分为两条槽最短路径相连接，从而通过式(2)计算得到经过 u 的最短的非槽环计数 $NSCCnt(u)$。需要注意的是，hub 顶点不能是 u 自身，即 $h\neq u$。这是因为对于 2-hop 的路径标签来说，u 到其自身的距离为 0，计数也为 0，这显然是错误的。然后，在式(3)中直接查找 $L_{STC}(u)$得到经过 u 的最短槽环计数 $TSCCnt(u)$。

由于最短的非槽环不一定是经过该点的最短环，最短的槽环也不一定是最短环，因此需要比较计算得到的槽最短环长度和非槽最短环长度。式(4)通过比较经过 u 的最短的非槽环长度 $sd(u,h)+sd(h,u)$和槽最短环长度 d_{stc}，来确定经过该点的最短环长度和计数。具体操作可以分为以下 3 种情况。

1)若 $sd(u,h)+sd(h,u)<d_{stc}$，则表明经过 u 的最短环均为非槽最短环，这种情况只需统计非槽最短环计数 $NSCCnt(u)$即可。

2)若 $sd(u,h)+sd(h,u)>d_{stc}$，则表明经过 u 的最短环均为槽最短环，此时只需统计槽最短环计数 $TSCCnt(u)$即可。

3)若以上两种情况均不成立，说明 $sd(u,h)+sd(h,u)=d_{stc}$，即经过 u 的最短环既包括非槽最短环也包括槽最短环，此时需要累加非槽最短环计数 $NSCCnt(u)$和槽最短环计数 $TSCCnt(u)$以得到最终结果。

算法 1　STC Query Process

Input: The Label L_{in}, L_{out} and L_{stc} of nodes in G, node u

Output: The Shortest Cycle Distance and Counting of node u

1. $\delta\leftarrow\infty$; $\sigma\leftarrow 0$;
2. foreach h$\in L_{out}[u]\cap L_{in}[u]$ do
3.　　if h=u then break;
4.　　if sd(u,h)+sd(h,u)$<\delta$ then
5.　　$\delta\leftarrow$d(u,h)+d(h,u);
6.　　$\sigma\leftarrow$c(u,h) * c(h,u) ;
7.　　else if sd(u,h)+sd(h,u)$=\delta$ then
8.　　$\sigma\leftarrow\sigma$+c(u,h) * c(h,u);
9. if $\delta>$d$_{stc}$, where (d$_{stc}$, c$_{stc}$)$\in L_{stc}[u]$ then
10.　$\delta\leftarrow$d$_{stc}$; $\sigma\leftarrow$c$_{stc}$;
11. else if $\delta=$d$_{stc}$, where (d$_{stc}$, c$_{stc}$)$\in L_{stc}[u]$ then
12.　$\sigma\leftarrow\sigma$+c$_{stc}$;
13. Return (δ,σ)

算法 1 具体描述了基于 STC 索引的查询算法。给定查询顶点 u，该算法首先通过 $L_{out}[u]$和 $L_{in}[u]$标签计算经过 u 的最短的非槽环长度 δ 和计数 σ(第 2－8 行)，然后将其与 $L_{STC}[u]=(d_{stc},c_{stc})$进行比较，从而计算出经过 u 的最短环长度和计数(第 9－12 行)。对于 $L_{out}[u]$和 $L_{in}[u]$标签中的公共 hub 顶点 h，如果 $h=u$，则终止查询(第 2－3 行)。此时说明不存在经过 u 的最短的非槽环，则当前的最短非槽环长度值 δ 和计数值 σ 仍为初始值 ∞ 和 0。若经过 u 的最短环均为槽最短环，则只需要将最短槽环的长度值 d_{stc} 和计数值 c_{stc} 赋值给 δ 和 σ 即可(第 9－10 行)。若经过 u 的最短的非槽环长度值 δ 等于最短槽环长度值 d_{stc}，则表明经过 u 的最短环既包括非槽最短环也包括槽最短环，仅更新计数值 σ 即可(第 11－12 行)。

例 5　假设要查询经过顶点 v_1 的最短环数量，首先在 $L_{in}[v_1]$和 $L_{out}[v_1]$中查找除 v_1 之外的公共 hub 顶点$\{v_0\}$，计算得到经过 v_1 的最短非槽环长度为 $sd(v_1,v_0)+sd(v_0,v_1)=4+1=5$，数量为 $\sigma_{v_1,v_0}+\sigma_{v_0,v_1}=2*1=2$。然后通过查询 $L_{STC}[v_1]=(5,1)$，得到经过 v_1 的最短槽环长度 $d_{stc}(v_1)=5$ 且 $d_{stc}(v_1)=sd(v_1,v_0)+sd(v_0,v_1)$，因此经过 v_1 的最短环数目 $SCCnt(v_1)=2+1=3$，最短环长度为 5，且经过 v_1 的最短环包括两个非槽最短环和一个槽最短环。

根据式(1)－式(4)可知，算法 1 求解给定顶点的最短环计数的时间复杂度为 $O(|L_{in}[v]|+|L_{out}[v]|)$。

此外，若存在一个强连通分量内部只有一个顶点时，则不存在经过该顶点的最短环；若存在一个强连通分量内部只有两个顶点 u，v 时，则经过 u 或 v 的最短环长度为 2，计数为 1。基于该特性，无需访问索引即可直接得到结果，有效加速查询。

5 索引构建

STC 索引的构造过程本质上是通过从某个顶点 u 开始进行 BFS 遍历和反向 BFS 遍历，访问与 u 位于同一个强连通分量且排序低于 u 的顶点 v，并记录 u 和 v 之间的最短路径长度和最短路径计数，即记录槽最短路径的长度和数目。需要注意的是，在遍历过程中对起始 u 访问两次，以记录最短槽环。

算法 2 展示了 STC 索引的构造算法，具体包括以下步骤：初始时所有标签均为空(第 1－3 行)。将图中顶点按照度数降序排序，对于每个处理顶点 u，从 u 开始在其所在的强连通分量内部进行正向 BFS 遍历(第 4 行)。当访问到顶点 v，首先基于现有标签判断 u 到 v 的最短距离 d，若 $D[v]>d$，则表明当前搜索到的路径并非顶点 u 到 v 的最短路径，因此不需要将 u 添加到 v 的入标签中，并且停止继续遍历(第 11 行)。若 $D[v]<d$，则表明 u 到 v 的所有最短路径均为槽最短路径，因此将 u 作为 v 的标准 hub 顶点添加到 v 的 $L_{in}^{c}[v]$ 标签中(第 12 行)。若 $D[v]=d$，则表明 u 到 v 之间的最短路径并非都是槽最短路径，即存在位于 u 和 v 最短路径上的中间节点 t，且 t 的排序高于 u 和 v。因此将 u 作为 v 的非标准 hub 顶点添加到 v 的 $L_{in}^{nc}[v]$ 标签中(第 13 行)。当再次访问到起始顶点 u，则表明形成了从顶点 u 出发回到自身的环路，并且该环为槽环。注意，此时不再从 u 开始继续遍历(第 22 行)。根据现有的槽环长度 d_{stc} 和 $D[v]+1$ 进行比较，更新最短槽环长度和计数(第 16－21 行)。对于 L_{out} 标签构造过程和 L_{in} 标签构造过程类似，只需要从 u 开进行反向 BFS 操作即可。需要注意的是，在反向 BFS 遍历过程中，不需要二次访问顶点 u，即不需要第 16－22 行的操作，否则将出现最短槽环重复计数的情况。

算法 2 STC Index Construction

Input: Graph G(V, E)

Output: The Label L_{in}, L_{out} and L_{stc} of nodes in G

1. foreach u∈V do
2. $L_{in}^{c}[u]\leftarrow\emptyset$; $L_{in}^{nc}[u]\leftarrow\emptyset$; $L_{out}^{c}[u]\leftarrow\emptyset$; $L_{out}^{nc}[u]\leftarrow\emptyset$;
3. $L_{stc}[u]\leftarrow(\infty,0)$; $D[u]\leftarrow\infty$; $C[u]\leftarrow0$;
4. foreach u∈V in descending order do
5. //forward construction L_{in} Label
6. $Q\leftarrow\emptyset$; $R\leftarrow\emptyset$;
7. $D[u]\leftarrow0$; $C[u]\leftarrow1$; Q.enqueue(u); $R\leftarrow R\cup\{u\}$;
8. While Q is not empty do
9. $v\leftarrow$Q.dequeue();
10. $d\leftarrow\min_{u'\in L_{out}^{c}[u]\cap L_{in}^{c}[v]} sd(u,u')+sd(u',v)$;
11. if $D[v]>d$ then continue;
12. if $d>D[v]$ then append(u, D[v], C[v]) to $L_{in}^{c}[v]$;
13. else append (u, D[v], C[v]) to $L_{in}^{nc}[v]$;
14. foreach $w\in nbr_{out}(v)$ and w, v in the same SCC do
15. if w<u the continue;
16. else if u=w then
17. $(d_{stc}, c_{stc})\leftarrow L_{stc}[u]$;
18. if $d_{stc}>D[v]+1$ then
19. $L_{stc}[u]=(D[v]+1, C[v])$;
20. else if $d_{stc}=D[v]+1$ then
21. $L_{stc}[u]\leftarrow(d_{stc}, c_{stc}+C[v])$;
22. continue;
23. if $D[w]=\infty$ then
24. $D[w]\leftarrow D[v]+1$; $C[w]\leftarrow C[v]$;
25. Q.enqueue(w); $R\leftarrow R\cup\{w\}$;
26. else if $D[w]=D[v]+1$ then
27. $C[w]\leftarrow C[w]+C[v]$
28. foreach v∈R do
29. $D[v]\leftarrow\infty$; $C[v]\leftarrow0$;
30. //backward construction L_{out} Label
31. foreach v∈V do
32. $L_{in}[v]\leftarrow L_{in}^{c}[v]\cup L_{in}^{nc}[v]$; $L_{out}[v]\leftarrow L_{out}^{c}[v]\cup L_{out}^{nc}[v]$;

例 6 表 2 即为对图 1 中的 G_1 构建的 STC 索引。其中第二、三列分别为索引的 L_{in} 标签和 L_{out} 标签，第四列为标签 L_{STC}。继续例 1 中对图 G_1 顶点的排序。由于顶点 v_0 的排序最高，因此首先从该点开始进行正向 BFS 遍历。因为 v_0 可以访问到所有顶点，并且这些顶点排序均低于 v_0，所以 v_0 到这些顶点的最短路径均为槽最短路径，将 v_0 作为标准 hub 顶点添加到这些顶点的 L_{in} 标签中。当 v_0 再次访问到自身时，得到经过 v_0 的最短槽环长度和计数，因此将标签项(5, 2)添加到 $L_{STC}[v_0]$ 中，并且停止继续遍历。当从顶点 v_4 开始遍历时，由于 v_0 的排序高于 v_4，因此不访问 v_0。由于 v_5，v_8 和 v_9 的排序均低于 v_4，并且 v_4 到这些顶点的最短路径均为槽最短路径，因此将 v_4 作为标准 hub 顶点添加到这些顶点的 L_{in}^{c} 标签中。当从顶点 v_1 开始反向遍历访问 v_7 时，通过比较现有标签 $L_{out}[v_7]$ 和 $L_{in}[v_1]$，可知 v_7 到 v_1 的最短路径距离 $sd(v_7, v_1)=\min\{sd(v_7, v_0)+sd(v_0, v_1)\}=4$。因为 $D[v_7]$ 也为 4，说明找到了一条 v_7 到 v_1 的最短路径。又因为 $sd(v_7, v_1)=D[v_7]=4$，说明 v_7 到 v_1 之间还存在途经 v_0 的非槽最短路径，因此将标签项 $(v_1, 4, 1)$ 添加到 $L_{out}^{nc}[v_7]$ 中。

6 实验

6.1 实验环境

本文实验所使用的硬件配置为 11th Gen Intel(R) Core™ i5-1135G7 @2.4GHz，运行内存为 16GB，操作系统为 CentOS，所有算法均用 C++实现。

6.2 数据集

实验所用数据集来自 SNAP[1] 和 NetRep[2] 的 10 个网络，其中 citeeseer 来自 NetRep。P04，P24，P30 以及 P31 为 P2P 网络，EP，SD08，SD09 以及 SEP 为社交网络，ST 为 Web 网络，CS 为引文网络。数据集的具体信息如表 3 所列。所有的图都是有向图，并且没有自环和重边。

[1] https://snap.stanford.edu

[2] https://networkrepository.com/index.php

表 3　数据集统计信息

Table 3　Dataset statistics

数据集	描述	$\|V\|$	$\|E\|$
P04	p2p-Gnutella04	10 876	39 994
P24	p2p-Gnutella24	26 518	65 369
P30	p2p-Gnutella30	36 682	88 328
P31	p2p-Gnutella31	62 586	147 892
EP	soc-Epinions1	75 879	508 837
SD08	soc-sign-Slashdot0811	77 357	516 575
SD09	soc-sign-Slashdot0902	81 871	545 671
SEP	soc-sign-epinions	131 828	841 372
SF	web-Stanford	281 903	2 312 497
CS	citeseer	384 413	1 751 463

6.3　实验对比算法

本文实验中的对比算法主要分为两类，分别为基于全图的算法和基于强连通分量的算法。其中，基于全图的算法主要包括 4 种：1）BFS 算法，即从查询顶点出发开始 BFS 遍历，当回到查询顶点时，得到经过该顶点的最短环长度和数量；2）HP-SPC 算法[14]；3）CSC 算法[1]；4）本文提出的 STC 算法。其中，HP-SPC 和 CSC 算法的源代码是通过与原作者联系取得的。

基于强连通分量的算法通过对全图进行强连通分量分解操作，在强连通分量内部遍历，或者构建索引。该类方法主要包括 4 种，分别为 BFS_SCC 算法、HP-SPC_SCC 算法、CSC_SCC 算法和 STC_SCC 算法。

在实验过程中，我们比较了回答最短环计数问题索引的构建时间、索引构建大小以及查询时间。

6.4　性能比较与分析

1）索引构建时间。表 4 列出了索引构造时间对比。从表中可以发现，对于基于全图的 3 种算法，CSC 算法的索引构造时间最长。本文提出的 STC 算法的索引构造时间和 HP-SPC 算法差别不大，这是因为本文提出的 STC 索引中的 L_{STC} 标签可以在 L_{in} 和 L_{out} 标签的构造过程中同时构造，因此整体的索引构造时间与 HP-SPC 几乎相同。进一步，引入 SCC 后，HP-SPC 的索引构建时间平均是 HP-SPC_SCC 的 1.93 倍，CSC 的索引构造时间是 CSC_SCC 的 1.83 倍，STC 的索引构造时间平均是 STC_SCC 的 1.97 倍。由此可知，SCC 可加速索引构建。

表 4　索引构造时间

Table 4　Index construction time

(s)

数据集	HP-SPC	CSC	STC	HP-SPC_SCC	CSC_SCC	STC_SCC
P04	2.92	3.20	2.77	1.37	1.50	1.18
P24	11.32	12.58	10.45	3.57	4.26	3.54
P30	15.25	17.66	16.56	6.59	7.23	6.06
P31	45.06	49.67	44.41	16.88	19.69	16.70
EP	64.26	75.20	68.29	45.17	58.15	43.96
SD08	50.56	55.92	49.13	29.66	37.25	30.89
SD09	52.20	60.98	52.85	34.92	42.74	33.52
SEP	154.42	162.55	144.41	94.36	133.44	93.51
SF	57.83	80.12	58.62	38.63	71.02	39.51
CS	50.07	58.81	50.17	41.30	34.32	41.28

2）索引大小。表 5 展示了索引大小的比较。可以看出，HP-SPC，CSC 以及 STC 算法三者的索引规模几乎相同。原因是影响这 3 个算法索引规模的主要因素是 L_{in} 和 L_{out} 标签大小。具体来说，HP-SPC 索引由 L_{in} 和 L_{out} 标签构成，而 STC 索引除了 L_{in} 和 L_{out} 标签，还在每个顶点上维护槽最短环信息，该信息空间代价为 $O(n)$，其中 n 为顶点数目，因此可以忽略不计。对于 CSC 算法来说，虽然在图转换过程中将顶点一分为二，原始图规模扩大为原来的两倍，但是由于分解后的两个顶点只需要分别维护 L_{in} 和 L_{out} 标签即可，因此 CSC 算法的索引规模与 HP-SPC 相近。进一步，引入 SCC 后，HP-SPC 的索引规模是 HP-SPC_SCC 的 3.5 倍，CSC 的索引规模是 CSC_SCC 的 3.3 倍，STC 的索引规模是 STC_SCC 的 3.4 倍。由此可知，基于强连通分量策略构建索引可以有效减小索引空间。这是因为对于图中任意顶点 u，其只需要存储和它自身在同一个强连通分量中的 hub 顶点作为标签项，而不需要存储 u 所在强连通分量之外的 hub 顶点，因此可以有效减少 u 的标签项数目，进而减小索引空间。

表 5　索引大小

Table 5　Index size

(MB)

数据集	HP-SPC	CSC	STC	HP-SPC_SCC	CSC_SCC	STC_SCC
P04	31.470	31.56	31.55	17.01	17.10	17.06
P24	103.650	103.86	103.85	33.82	34.03	33.92
P30	145.690	145.98	145.97	51.81	52.09	51.95
P31	336.520	337.00	336.99	117.22	117.71	117.46
EP	558.534	559.21	559.11	369.41	370.10	369.70
SD08	416.820	417.48	417.41	226.06	226.69	226.32
SD09	442.260	442.96	442.89	229.83	230.52	230.14
SEP	1053.810	1054.93	1054.81	619.34	620.47	619.85
SF	532.790	535.82	534.94	352.87	355.89	353.94
CS	437.110	440.10	438.58	27.52	30.51	28.99

3）查询时间。表 6 列出了 BFS，HP-SPC，CSC 和 STC 算法以及它们分别基于强连通分量的算法回答最短环计数的查询时间比较。从表中可以看出，BFS 算法的查询时间要普遍慢于基于索引的算法，差距在 10～1000 倍之间。对于基于全图的 3 种算法，CSC 算法的查询时间平均比 HP-SPC 算法少 75%。这是因为对于一个给定的最短环查询，HP-SPC 算法需要执行多次最短路径计数查询，而 CSC 算法只需要执行一次。和 CSC 相比，STC 算法的查询时间平均增加约 10%。此外，可以发现，引入 SCC 后，BFS_SCC 算法的查询速度比 BFS 快 7 倍；HP-SPC 的查询时间是 HP-SPC_SCC 的 2 倍；CSC 的查询时间是 CSC_SCC 的 1.2 倍；STC 的查询时间是 STC_SCC 的 1.2 倍。由此得出，基于强连通分量策略可以提升查询效率。从表 6 中还可以发现，在 P30，SD09 等数据集上，STC_SCC 相较于 STC 算法的查询时间反而有所增加，造成该情况的主要原因是在使用 STC_SCC 算法进行查询时，将算法选择写在了 for 循环内部，导致当查询 n 个顶点的最短环计数时，需要判断 n 次选择了何种算法，而在使用不加 SCC 优化的 STC 算法进行查询时，先判断选择何种算法，再对 n 个顶点进行查询，只统计对这 n 个顶点的查询时间。由于 STC_SCC 算法在查询过程中需要多判断 $4*n$ 次算法选择，因此 STC_SCC 算法的查询时间反而比 STC 算法长。

表 6 查询时间

Table 6 Query time

(ms)

数据集	BFS	BFS_SCC	HP-SPC	HP-SPC_SCC	CSC	CSC_SCC	STC	STC_SCC
P04	865.16	376.67	43.50	32.56	18.98	18.92	16.15	15.02
P24	2693.95	773.22	92.02	57.21	32.14	30.92	33.57	30.58
P30	5134.81	2003.39	111.24	97.34	43.11	45.62	43.78	57.93
P31	19051.00	6334.18	247.78	166.09	99.52	95.56	98.48	94.06
EP	131871.00	16774.90	1699.64	1318.22	289.88	256.11	313.11	224.72
SD08	70961.70	18086.30	673.48	603.53	147.44	162.46	143.57	144.81
SD09	73833.00	21780.10	690.83	755.31	150.52	242.66	144.37	188.84
SEP	387276.00	58089.10	2685.98	2547.30	405.36	422.92	443.02	382.85
SF	133304.00	481739.00	1505.63	1102.20	310.67	314.18	284.01	229.00
CS	1060920.00	24782.10	294.45	37.12	85.57	24.06	62.43	23.44

结束语 本文研究原始图上最短环计数的处理问题。针对现有方法存在索引空间较大、查询效率较低等问题，提出了新的基于 2-hop 索引的最短环计数算法。该算法基于最短路径和最短环之间的特殊关系，将最短环分为非槽最短环和槽最短环两种，通过在原图上构建索引，分别统计这两种最短环计数。进一步，根据强连通分量实际上是由若干个环组成这一特殊关系，提出了基于强连通分量的索引策略，通过在强连通分量内部构造最短环计数索引，可以进一步提升索引构造效率，减小索引规模，提升查询效率。在 10 个真实数据集上进行了实验，结果验证了本文方法的高效性，以及基于强连通分量策略可以有效减小索引空间，提升索引构造以及查询效率。

参 考 文 献

[1] FENG Q, PENG Y, ZHANG W, et al. Towards Real-Time Counting Shortest Cycles on Dynamic Graphs: A Hub Labeling Approach[C]// 2022 IEEE 38th International Conference on Data Engineering(ICDE). IEEE, 2022: 512-524.

[2] 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.

[3] LAI Z, PENG Y, YANG S, et al. PEFP: Efficient k-hop Constrained s-t Simple Path Enumeration on FPGA[C]// 2021 IEEE 37th International Conference on Data Engineering(ICDE). IEEE, 2021: 1320-1331.

[4] PENG Y, LIN X, ZHANG Y, et al. Efficient Hop-constrained s-t Simple Path Enumeration[J]. The VLDB Journal, 2021, 30(5): 799-823.

[5] PENG Y, ZHAO W, ZHANG W, et al. DLQ: A System for Label-Constrained Reachability Queries on Dynamic Graphs[C]// Proceedings of the 30th ACM International Conference on Information & Knowledge Management. 2021: 4764-4768.

[6] LI C, KYNG R, YANG P. Maximilian Probst Gutenberg: Almost-Linear Time Algorithms for Incremental Graphs: Cycle Detection, SCCs, s-t Shortest Path, and Minimum-Cost Flow [J]. arXiv: 2311.18295, 2023.

[7] QIU X, CEN W, QIAN Z, et al. Real-time constrained cycle detection in large dynamic graphs[J]. Proceedings of the VLDB Endowment, 2018, 11(12): 1876-1888.

[8] PENG Y, ZHANG Y, LIN X, et al. Towards bridging theory and practice: hop-constrained s-t simple path enumeration[J]. Proceedings of the VLDB Endowment, 2019, 13(4): 463-476.

[9] BODAGHI A, TEIMOURPOUR B. Automobile Insurance Fraud Detection Using Social Network Analysis[M]// Applications of Data Management and Analysis, Lecture Notes in Social Networks. 2018: 11-16.

[10] HAJDU L, KRÉSZ M. Temporal Network Analytics for Fraud Detection in the Banking Sector[C]// International Conference on Theory and Practice of Digital Libraries. 2020: 145-157.

[11] YUE D, WU X, WANG Y, et al. A Review of Data Mining-Based Financial Fraud Detection Research[C]// 2007 International Conference on Wireless Communications, Networking and Mobile Computing. IEEE, 2007: 5519-5522.

[12] SCHLOSSER M, CONDIE T, KAMVAR S. Simulating a file-sharing p2p network[C]// 1st Workshop on Semantics in Grid and P2P Networks. Stanford InfoLab, 2003.

[13] OHTA C, GE Z, GUO Y, et al. Index-server optimization for P2P file sharing in mobile ad hoc networks[J]. Ite Technical Report, 2004, 28(493): 960-966.

[14] ZHANG Y, YU J X. Hub Labeling for Shortest Path Counting [C]// Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. 2020: 1813-1828.

[15] WANG Y Q, YUAN L, CHEN Z, et al. Towards Efficient Shortest Path Counting on Billion-Scale Graphs[C]// 2023 IEEE 39th International Conference on Data Engineering (ICDE). IEEE, 2023: 2579-2592.

[16] PENG Y, YU J, WANG S B. PSPC: Efficient Parallel Shortest Path Counting on Large-Scale Graphs[C]// 2023 IEEE 39th International Conference on Data Engineering (ICDE). IEEE, 2023: 896-908.

[17] FENG Q S, PENG Y, ZHANG W J, et al. DSPC: Efficiently Answering Shortest Path Counting on Dynamic Graphs[J]. arXiv: 2307.05918, 2023.

YANG Ying, born in 1999, postgraduate. Her main research interests include reachability and shortest paths and so on.

ZHOU Junfeng, born in 1977, Ph.D, professor, Ph.D supervisor, is a member of CCF(No. 16083M). His main research interests include information retrieval technology, semi-structured data, query processing and optimization of graphic data.

(责任编辑：杨雪敏)