RDF 图的 Top-k 最短路径查询

章登义,吴文李,欧阳黜霏
(武汉大学计算机学院,湖北武汉 430072)

摘 要: 最短路径查询是图数据管理与复杂关系挖掘的基本操作之一.本文针对资源描述框架图上的 top-k 最短路径查询,构造基于组件的索引,并在该索引的基础上实现查询的响应.查询优化阶段,针对查询效率问题,提出频繁路径以及结构剪枝策略,并给出有效性证明.实验表明,本文方法准确返回 top-k 最短路径并提高 92%的查询速率.索引构造时间相比已有方法,提高约 56%.同时,索引所占空间仅为原始数据大小的 1~1.2 倍.

关键词: 资源描述框架; 最短路径查询; 图数据库; top-k; 查询处理
中图分类号: TP301 **文献标识码:** A **文章编号:** 0372-2112 (2015)08-1531-07
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2015.08.010

Top-k Shortest-Path Query on RDF Graphs

ZHANG Deng-yi, WU Wen-li, OUYANG Chu-fei
(*School of Computer, Wuhan University, Wuhan, Hubei* 430072, *China*)

Abstract: Finding the shortest path in a graph is a fundamental operation that allows complex relationships discovering in a graph database. This paper considers top-k shortest-path queries on RDF(Resource Description Framework) graphs. To solve this problem, we provide a component-based index for path queries and present an approach to answer the top-k shortest-path query. To efficiently process queries, we propose frequent path strategy and structural pruning. For frequent path strategy, we precompute the top-k shortest-paths between frequent entities obtained from the frequent paths. For the structural pruning, we find out the termination condition for queries by considering the structural information of component-based index and derive two lemmas to guide this pruning mechanism. Experiments results show that the query runtime is improved by 92% using our answering approach and the construction time of our index for RDF graphs is speeded up by 56%. Meanwhile, the size of this index is only 0~0.2 times larger than that of the raw graphs.

Key words: resource description framework(RDF) graph; shortest path query; graph database; top-k; query processing

1 引言

随着语义网的发展,越来越多的数据以 RDF(Resource Description Framework)三元组的形式存储,即〈资源,属性,描述〉(〈subject, property, object〉).从图的角度上看,资源(subject)作为图的顶点,属性(property)是连接各个顶点的边,而三元组的最后一个对象(object),既可以表示资源的值同时也能作为图中的顶点.这种图的表示方式简单灵活,日益成为描述大数据集的重要手段.

目前,在 RDF 图上做路径查询的研究不多,对普通图进行路径查询的研究有文献[1~9].文献[1]研究了代谢网络中两种化学物质之间的最短路径查询.文献[2]面向大型社交网络,表明了最短路径查询在资源数据管理中的作用.文献[3]阐述了最短路径查询在道路网中的重要性.文献[4~8]针对普通图最短路径查询提出了各自的方法并取得了较好的查询效果.然而,它们都忽略 top-k 查询结果的重要性.

随着查询技术的发展,top-k 查询日益成为数据分析与复杂关系挖掘的重要手段.文献[10~13]分别在精确查询、最优查询、多样查询以及近似查询中搜索 top-k 结果,以满足用户的多样化需求.这些查询均为普通图而设计,无法在 RDF 图数据上有效执行.现阶段,为 RDF 图查询而构造的索引有 RDF-3X[14]、GRIN[15]以及文献[16].这些方法执行选择查询效率较高,但执行路径查询效率甚低.另外,已有资源系统[17]逐步考虑海量数据的存储管理以及对数据动态变化的适应.然而,对海量动态数据的实时处理仍是各大查询框架的一大挑战.

针对于此,本文在 RDF 图上考虑 top-k 最短路径的

收稿日期:2014-04-15;修回日期:2014-09-27;责任编辑:覃怀银

查询问题.通过分析可知,RDF 图数据的具体特征为:RDF 图数据库包含有限个独立图,且图的每个顶点非等价存在,其中部分顶点是路径查询的资源对象,部分顶点仅仅是资源对象的值,值围绕资源对象呈组件状.本文利用这些特征提供的信息为路径查询构造基于组件的索引,在索引的基础上实现 top-k 最短路径的查询.同时,针对海量动态数据库产生的需求,在大数据上完成对自身框架的测试并探讨了该框架在动态数据库中的可用性.

2 基于组件的索引

本文将 RDF 图描述成从 subject 到 object 的有向图.该有向图的具体形式如图 1 所示.图中,存在两种顶点:一种顶点与其他顶点有连接并形成关系网,如 Bob、Lily、Susan 和 Saralowndes,记为 V_R;另一种顶点仅仅作为图中某顶点的属性值,与其他顶点不关联,如 Panos、Zoe、1946-07-01 等,记为 V_P.

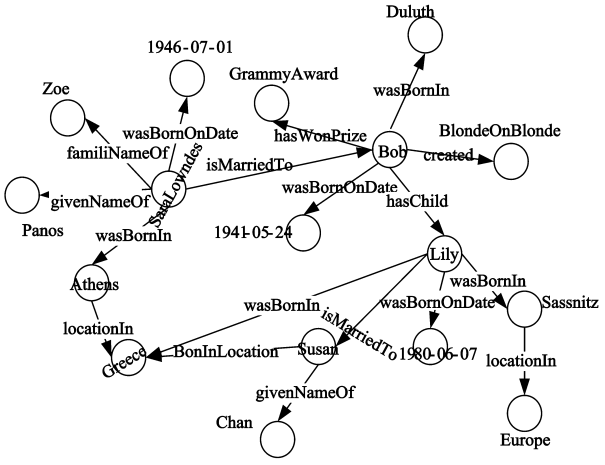


图1 RDF数据

2.1 索引的构造

本节剖析组件索引的构造并简单介绍索引的存储.其中,索引的构造过程包括四个步骤:组件划分、去环、节点压缩和成树.

首先,划分 RDF 图并构建组件.本文将属性顶点 V_P向关联顶点 V_R聚拢并形成基数为 $n=|V_R|$的组件集 $\{C_1, C_2, \cdots, C_n\}$.集合中的元素满足 $0<i,j<n+1, i\neq j, C_i \cap C_j = \varnothing$.图 1 的划分结果如图 2 所示.

其次,去环,即消除图的无向环.图 2 中的路径 $C_1 \rightarrow C_3 \rightarrow C_4$, $C_1 \rightarrow C_2 \rightarrow C_6 \rightarrow C_4$和 $C_1 \rightarrow C_2 \rightarrow C_6 \rightarrow C_5 \rightarrow C_4$可形成无向环.本文将 C_4这种节点进行冗余,即,去环.图 2 去环后如图 3(a)所示.

第三步,节点压缩.本文对出度为 1 的节点向上压缩,允许一个索引节点包含多个组件.图 3(b)是 3(a)经过压缩后的输出.图 3(b)中,矩形表示表示包含多个组件的节点,其余节点用圆形表示.

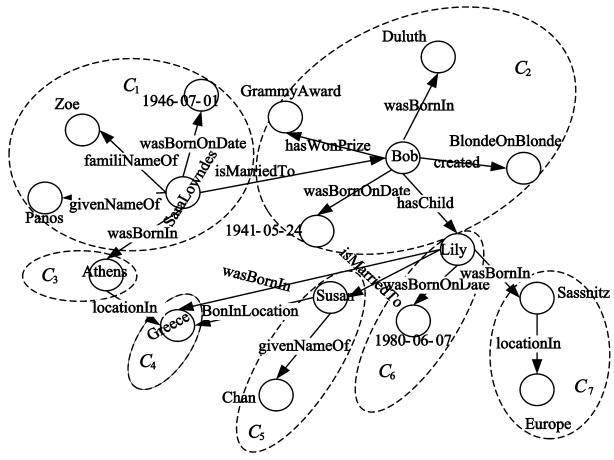


图2 组件划分

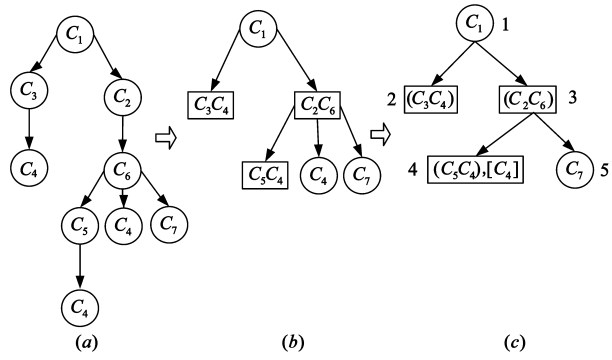


图3 去环压缩节点并成树

构造索引的最后一步是形成树.本文对图 3(b)进行自上而下的节点合并,并形成二叉树.至此,图 1 的索引树构造完毕,如图 3(c)所示.构造索引树的开销为 $O(n)$.

索引存储是构造过程的延续.本文的索引存储分为两部分,一部分是包含真实信息的字符串字典,另一部分是包含组件关联信息的索引树.

2.2 索引的路径

令有向图 $G_i=(C,E)$,其中,C 为组件集,E 是组件之间的边集.组件 $C_a, C_b \in C$,当且仅当存在路径 $p=(C_a, C_1, \cdots, C_n, C_b)$,且$(C_a, C_1)$,$(C_j, C_{j+1})(1\leqslant j\leqslant n)$ $(C_n, C_b)\in E$,则 C_a到 C_b存在路径,记为 $C_a \rightarrow C_b$.本节主要在索引树上考虑 $C_a \rightarrow C_b$的路径问题.

定义 1 非压缩树节点 X_{uz}的定义如下:

$$\left\{ C_j \;\middle|\; \begin{matrix} C_j \in X_{uz}, C_j \in G_i, |X_{uz}|=1 \\ 0<i,j<n+1 \end{matrix} \right\}$$

图 3(c)中的根节点 $X_1=\{C_1 | (v_{'Saralowndes'}, v_{'Zoe'}), (v_{'Saralowndes'}, v_{'Panos'}), (v_{'Saralowndes'}, v_{'1946\text{-}07\text{-}01'})\}$和叶子节点 $X_5=\{C_7 | (v_{'Susan'}, v_{'Chan'})\}$均为非压缩节点.

定义 2 压缩树节点 X_z的定义如下:

$$\left\{[C_m]\cup(C_lC_t\cdots C_r)\left|\begin{array}{c}[C_m]\in X_z,(C_lC_t\cdots C_r)\in X_z,C_m\in G_i\\(C_lC_t\cdots C_r)\in G_i,|X_z|>1\\0<m,l,t,r<n+1\end{array}\right.\right\}$$

压缩节点具体形式如图 3(c)的 $X_4=\{(C_5\,C_4),[C_4]\}$.

定义 3　非压缩 X_{uz}中的路径定义如下:

$$\left\{X_p\rightarrow C_m\left|\begin{array}{c}X_p\rightarrow C_m\in G_i,C_m\in X_{uz},0<m<n+1\\X_p\text{ 是 }X_{uz}\text{的父节点}\end{array}\right.\right\}$$

索引中的非压缩节点只包含一个组件.因此,X_{uz}只存在一条由父节点到子节点的路径.具体见图 3(c)中的 $X_3\rightarrow C_7$.

定义 4　压缩节点 X_z的父类节点为X_p,X_z在索引树中的路径定义如下:

$$\left\{(X_p\rightarrow C_l\rightarrow C_t\cdots\rightarrow C_r)\cup(X_p\rightarrow C_s)\right.$$

$$\left|\begin{array}{c}(X_p\rightarrow C_l\rightarrow C_t\cdots\rightarrow C_r)\in G_i,(X_p\rightarrow C_s)\in G_i,(C_lC_t\cdots C_r)\in X_z,\\[C_s]\in X_z,0<l,t,r,s<n+1,X_p\text{ 是 }X_z\text{ 的父节点}\end{array}\right\}$$

压缩节点路径具体见图 3(c)中 $X_4=\{(X_3\rightarrow C_5\rightarrow C_4),(X_3\rightarrow C_4)\}$.

定义 5　有向图 $G_i=(C,E)$,$v_a\in C_a\in X_a$且$v_b\in C_b\in X_b$,v_a和v_b是 RDF 图顶点,C_a和C_b是组件,X_a和X_b是树节点.索引树中 v_a到v_b的路径定义为X_a到C_b的路径.其中,包含一系列的非压缩节点路径和压缩节点路径.

图 1 中,顶点 Zoe 到 Lily 的路径为('Zoe','Saralowndes','Bob','Lily').该路径在索引树中被描述为 $X_1\rightarrow C_2\rightarrow C_6$,而不是 $C_1\rightarrow C_2\rightarrow C_6$.因此,为表明 X_1中哪个组件与 $C_2\rightarrow C_6$相连,以获取只包含组件的路径,本文做了以下约定.

定义 6　规则 1 的定义为,树节点中元素的位置与其父类元素位置一致,那么这两元素可形成父类元素到子类元素的路径.

3　top-k 最短路径的响应

本节的响应过程为:(1)RDF 图顶点到组件的转换;(2)搜索遍历;(3)组件到 RDF 图顶点的转换((1)的逆过程).

3.1　RDF 图顶点到组件的转换

RDF 图路径查询主要为了获取 $v_a\rightarrow v_b$的路径$\{v_a,v_1,\cdots,v_n,v_b\}$.但是,由于索引存储的是组件信息,因此,已知起点(和终点)时,查询计划首先将已知顶点转换成组件,然后才开始对索引树进行遍历.转换过程在索引字典中进行.

字典表按照组件宽度优先顺序排放,组件内部的顺序为 V_R顶点、V_R的对应边、V_P顶点、V_P的对应边、下一 V_R顶点、下一 V_P的对应边….针对图 1 所构造的字典如表 1 所示.

表 1　索引中的字典表

字典表		
字符串		组件号
值	ID	C
Saralowndes	1	1
Zoe	2	1'
…	…	…
Duluth	13	2
- -	8	2'
BlondeOnBlonde	15	2'
…	…	…

查询时,图顶点到组件的转换过程如算法 1 所示.对于查询 $v_{\text{'Saralowndes'}},\left\{\begin{array}{l}\text{givenNameOf = 'Panos'}\\\text{familyNameOf = 'Zoe'}\end{array}\right\}\rightarrow v_{\text{'Greece'}}$,$k=2$,本文首先将 $v_{\text{'Saralowndes'}},\left\{\begin{array}{l}\text{givenNameOf = 'Panos'}\\\text{familyNameOf = 'Zoe'}\end{array}\right\}$转换成 C_1.然后,确定 Greece 在组件 C_4中.

算法 1　字典扫描算法

输入:dictionary 和 Query←{v_a|conditions}→v_b|conditions}}//conditions 中包含了 subject,property 或 object 的已知信息

输出:C_a和[C_b] //C_b是不定项,当 v_b|conditions}不为空时,才返回

开始

1. $C_a\leftarrow\varnothing$, $C_V\leftarrow\varnothing$, $t\leftarrow\varnothing$, $C_T\leftarrow\varnothing$
2. if (Query.v_a.a 是字符串)　then//字符串已知
3. $C_V\leftarrow$dictionary.getCaya(a)
4. end if
5. if(C_V为空)　then //考虑字符串 a 没给出的情况
6. $C_V\leftarrow$dictionary.getCaByCondition(v_a.conditions[$t-1$]))
7. end if
8. $t\leftarrow$Query.v_a.conditions.size
9. while $t>0$ do
10. $C_T\leftarrow$dictionary.getCaByCondition(v_a.conditions[$t-1$]))
11. //每个 conditions 中包含两字符串如 givenNameOf = 'Panos'
12. //包含了 Panos 和 givenNameOf,后者的地址为前者地址加 1
13. if (C_V.contain(C_T)) then
14. $C_V[0]\leftarrow C_T$
15. C_V[others]←∅ // others>0
16. else　终止循环
17. end if
18. t 减 1
19. end while
20. $C_a\leftarrow C_V[0]$ //若 v_b.conditions 不为空同理可返回 C_b

结束

3.2 搜索遍历

搜索遍历分为两步,首先选择出包含起点 C_a 的索引树,其次对该树进行遍历.为快速选择出包含起点 C_a 的树,本文在字典中增加一个哈希表,哈希函数如式(1)所示.它标明了每棵树所包含的组件号范围.其中,1,2,3…对应 T_{Gi} 中的 i.因此,选择索引树的开销为 $O(1)$.

$$H(a)\begin{cases}1, & 0<a<8\\ 2, & 7<a<27\\ 3, & 26<a<54\\ \cdots\end{cases}\cdots\cdots \quad (1)$$

索引树的遍历过程自上而下扫描 C_a 到 C_b 的所有路径,返回 v_a 到 v_b 之间的 top-k 最短路径.该过程由两部分组成,一是树的遍历,二是树节点的解压以及路径的形成.其具体实现如算法 2 所示.

算法 2 树的遍历算法

输入: C_a, C_b//起始和终点组件,若 C_b 不确定,则一直扫描到叶子节点

输出: 前 k 条最短路径 p

开始

1. $Q\leftarrow C_a$, $C_{cur}\leftarrow C_a$, $X_{pre}\leftarrow\varnothing$
2. while Q 不为空 do
3. $C\leftarrow Q$.dequeue
4. if (($C==C_b$) ‖ ($C_b==\varnothing$ && C 是叶子节点)) then
5. $p\leftarrow$ 基于定义 3,4,5 以及规则 1 推导出 C_a 到 C 的所有路径
6. end if
7. if X_{pre} 为空 then
8. $Q\leftarrow X_{Ca}$.leftAndRightNodes
9. $X_{pre}\leftarrow X_{cur}$
10. $C_{cur}\leftarrow\varnothing$
11. end if
12. 从 X_{pre} 中删除 C
13. $C_{cur}\leftarrow$ 与 C_a 形成路径的下一元素
14. end while
15. $p\leftarrow$ 前 k 条最短路径

结束

4 查询优化

为尽可能减少扫描节点的个数,本文采用了预存储以及剪枝策略.前者可将响应时间减低为 $O(1)$,后者可将扫描所需时间 $O(n)$ 中的 n 降到最低.

4.1 频繁路径策略

本策略只对紧密关联的属性集进行预处理.假设频繁出现的线性路径 $prop_1, prop_2, \cdots, prop_n$ 的属性集是紧密关联的.本文使用 RDF-3X 的频繁路径挖掘算法(Frequent Path Mining algorithm,FPM)迅速找出频繁线性路径的属性集.该算法率先找出长度为 1 的频繁路径,不断对其首尾进行频繁路径扩展,直至无法继续更新为止.本文取该频繁线性路径的始发点和终点作为预存储的顶点对,计算顶点对之间的 top-k 最短路径并存储于索引中,以供查询响应之用.当查询的始发点和终点在预存储中命中时,直接返回结果并停止对索引树的访问.

4.2 剪枝策略

结构剪枝旨在设定终止条件,以避免返回 top-k 结果前对所有 X_i 全面解压、扫描和计算.通过观察组件关联图发现,图中频繁出现多条路径同时指向某一组件的情况.如图 3 所示,路径 $C_1\rightarrow C_2\rightarrow C_6\rightarrow C_5\rightarrow C_4$, $C_1\rightarrow C_2\rightarrow C_6\rightarrow C_4$ 和 $C_1\rightarrow C_3\rightarrow C_4$ 共享组件 C_4.而数据库中,C_4 后跟着一定数额的路径,即这 3 条路径共享从 C_4 开始的子图.

查询 C_1 到共享子图中某一组件的 top-2 最短路径.当算法遍历到 X_4 和 X_5 时,中间结果集中有 4 条路径,分别为 $C_1\rightarrow C_2\rightarrow C_6\rightarrow C_5\rightarrow C_4$, $C_1\rightarrow C_2\rightarrow C_6\rightarrow C_4$ 和 $C_1\rightarrow C_3\rightarrow C_4$ 以及不共享 C_4 的 $C_1\rightarrow C_2\rightarrow C_6\rightarrow C_7$.此时,剪枝策略终止路径 $C_1\rightarrow C_2\rightarrow C_6\rightarrow C_5\rightarrow C_4$ 继续参与 top-2 查询.结论依据见引理 1 和引理 2.

引理 1 从 C_a 出发的路径 $p_1, p_2, \cdots, p_n$ 同时指向组件 C_i,那么从 C_a 到 C_b 的每条 top-k 最短路径中一定包含 C_i.

证明 假设 top-k 最短路径不包含 C_i,那么至少 $\exists p_j, 0<j<n+1$ 不包含 C_i.命题中可知 $p_1, p_2, \cdots, p_n$ 同时指向组件 C_i.假设与之矛盾,因此,假设不成立.引理 1 成立.

引理 2 从 C_a 出发的路径 $p_1, p_2, \cdots, p_n$ 同时指向组件 C_i,那么从 C_a 到 C_b 的 top-k 最短路径中,若 $n>k$,有 $p_1, p_2, \cdots, p_k$ 均包含从 C_i 到 C_b 的最短路径.

证明 从 C_a 出发的路径 $p_1, p_2, \cdots, p_n$ 同时指向组件 C_i,若 $n>k$,从引理 1 中可知,C_a 到 C_b 的 top-k 最短路径 $p_1, p_2, \cdots, p_k$ 均包含 C_i,那么任意 $p_j(0<j<n+1)$ 的长度包含两个部分,一部分是从 C_a 到 C_i 的长度,另一部分是从 C_i 到 C_b 的长度,前者已知,要使 p_j 长度最短,只需取 C_i 到 C_b 的最短长度.因此,引理 2 成立.

5 实验结果与评价

本文工作的评价主要包括两个方面—索引、top-k 最短路径查询.索引的评价指标包含创建耗时、大小.top-k 查询的指标有查询响应时间、优化效果以及海量数据的影响.

5.1 实验设置

我们用 C 语言实现了 Jena 的 API 以及本文的技术,并为 top-k 查询提供语法分析器.实验所用数据来自著名的数据基准-YAGO.同时,本文设计了一系列针对 YAGO 数据集的查询语句,具体如表 2 所示.最后一

个参数是比较对象,分别为 TEDI 和 RDF-3X.此外,本文　　方法记为 CBI.

表 2　查询语句

语句	起点	起点查询条件	终点	终点查询条件
Q_1	$v_{'Saralowndes'}$	familyNameOf = 'Zoe' givenNameOf = 'Panos'	$v_{'America'}$	– –
Q_2	$v_{'Bannen'}$	hasChild = 'Orpheus' acteIn = 'Braveheart' familyNameOf = 'Lan'	$v_{'Cosmo'}$	acteIn = 'Citizen' familyNameOf = 'Lee'
Q_3	$v_{'Saralowndes'}$	familyNameOf = 'Zoe' givenNameOf = 'Panos'	$v_{'Bindusara'}$	– –
Q_4	$v_{'Rising'}$	endedOnDate = '1916 – 04 – 03' diedIn = 'London' graduatedFrom = 'GeorgetownUniversity'	$v_{'America'}$	– –
Q_5	$v_{'Rising'}$	diedIn = 'London' endedOnDate = '1916 – 04 – 30' graduatedFrom = 'GeorgetownUniversity'	$v_{'Brazil'}$	– –
Q_6	$v_{'Cosmo'}$	diedIn = 'Citizen' familyNameOf = 'Lee'	v	– –
Q_7	$v_{'Rising'}$	endedOnDate = '1916 – 04 – 30' diedIn = 'London' graduatedFrom = 'GeorgetownUniversity'	v	– –
Q_8	$v_{'Saralowndes'}$	familyNameOf = 'Zoe' givenNameOf = 'Panos'	v	– –
Q_9	$v_{'Alber\_Speer'}$	hasChild = ' Nissen' hasChild = ' Schramm'	v	– –

5.2　索引评价

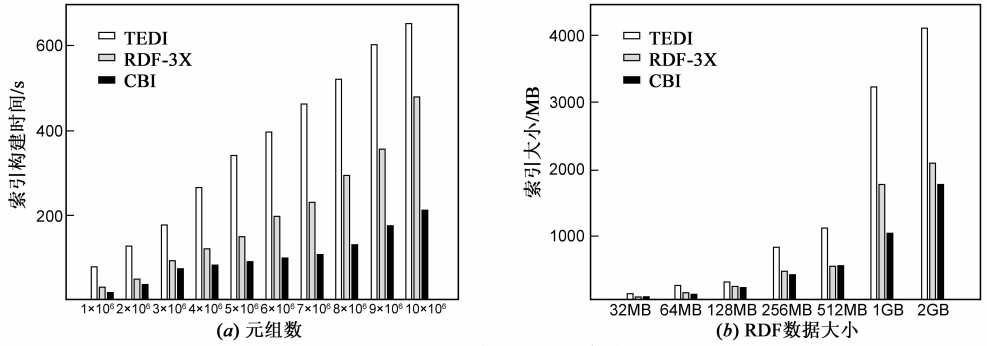
本节评价本文索引 CBI 的创建时间和占用空间.实验结果分别如图 4(a)和 4(b)所示.

5.2.1　索引创建时间

图 4(a)可见,CBI 在创建速率上比 RDF-3X 高 41%,比 TEDI 高 71%.这是因为 CBI 的构建单位为组件,而 TEDI 和 RDF-3X 的构建单位为顶点.过小的索引单位带来的过多的处理量,大大增加了 TEDI 和 RDF-3X 的创建时间.同时,构造 TEDI 的过程中冗余了大量顶点,因此,TEDI 耗时最长.

5.2.2　索引大小

图 4(b)可见,三种索引所占空间,TEDI 最大,RDF-3X 次之,CBI 最小.由于 TEDI 内部包含大量顶点的冗余,因此,该索引大小为原数据大小的 2 ~ 5 倍,甚至更高.RDF-3X 的压缩效果较好,索引所占空间明显比 TEDI 少.CBI 基于组件而构造,降低了原始图的复杂度,同时采用 RDF-3X 的压缩办法对预存储内容进行压缩,因此,CBI 索引大小仅为原数据的 1 ~ 1.2 倍.



(a) 元组数　　(b) RDF数据大小

图4　索引的对比实验

5.3　top-k 响应时间评价

5.3.1　查询响应效率评价

本节对 k 随机取值并测试系统对不同查询的响应效率.实验结果如和图 5(a)所示.图 5(b)是 CBI 的具体情况.

图 5(a)可见,RDF-3X 的响应时间远远长于 CBI 的.因为 CBI 在确定了查询所涉及的索引树后,仅需对树上的节点进行解压、扫描和剪枝,大大降低访问索引的次数.同时,组件的内容是它在字典中的地址.在返回最终结果时,组件与字符串的转换只需通过地址直接存取.因此,其查询响应高效.图 5(b)中,Q_1到 Q_9返回的最短路径长度依次递增,但查询时间不骤增.

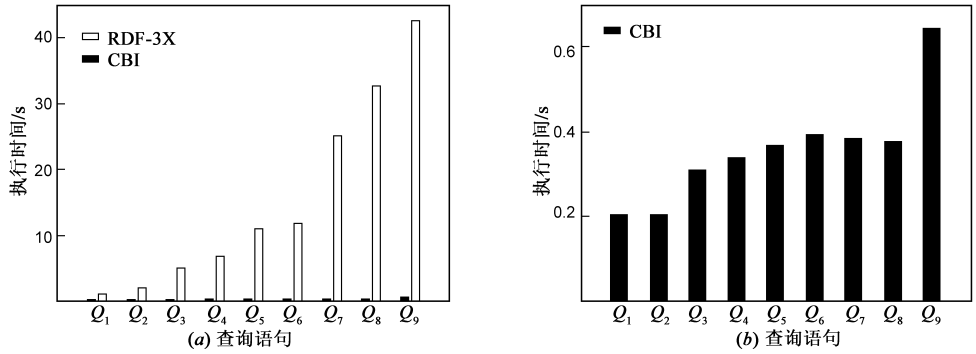


图5 k取随机值时响应时间对比实验

5.3.2 优化策略的效果

为测试优化策略的效果,我们将 Q_1到 Q_9分别在优化前的 CBI、只加入频繁路径策略的 CBI 以及只加入剪枝策略的 CBI 上运行.执行结果如表 3 所示.

表 3 可见,使用频繁路径策略后,只有 Q_3、Q_8、Q_9没有得到优化.其他查询都可从预存储中命中并直接取得结果.对于剪枝策略,优化效果在 9 条查询语句中均有所反映,约提高 12.5% 到 33% 的查询速率.

5.3.4 数据量对响应时间的影响

本节实验的数据量不断增大至 50GB.实验结果如图 6 所示.

随着数据量的增大,CBI 的响应时间呈平稳状态,RDF-3X 的变化较大.虽然 RDF-3X 可高效响应选择查询,但进行路径查询时,因涉及过多的连接,效率低下.同时,CBI 主要为路径查询而设计,RDF-3X 则不然.因此,进行 top-k 最短路径查询时,CBI 优势明显.

表 3 优化效果对比

CBI	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9
前	312	345	476	487	498	512	578	615	778
频繁路径策略(ms)									
后	202	217	476	321	333	354	356	615	778
剪枝策略(ms)									
后	273	284	347	356	382	399	398	412	573

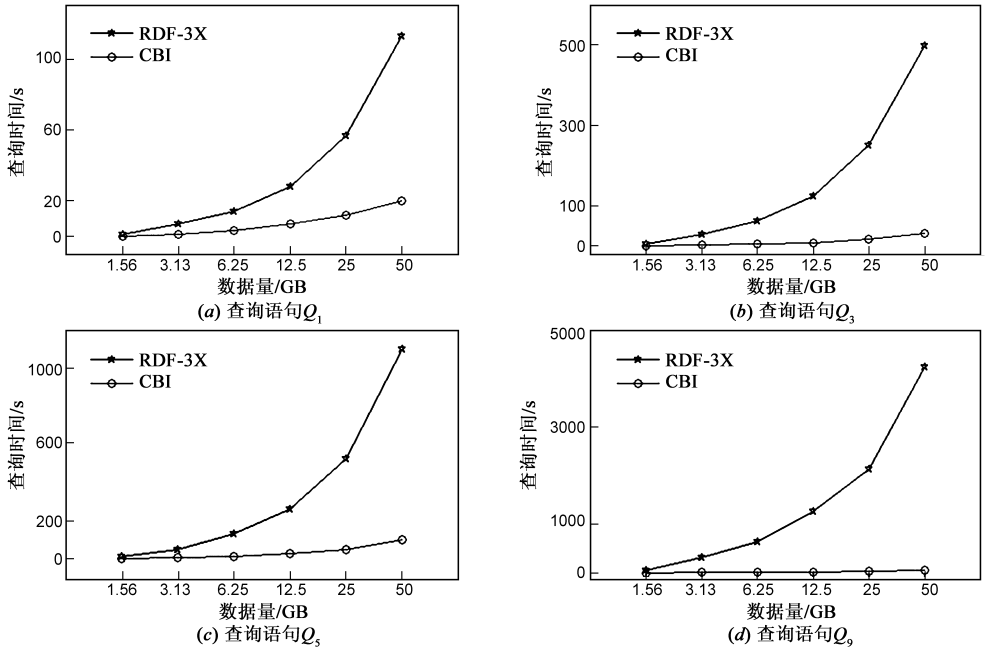


图6 查询语句 Q_1、Q_3、Q_5、Q_9随数据量递增的效率变化

6 总结及展望

本文在 RDF 图上解决 top-k 最短路径查询的问题.首先,本文设计了基于组件的索引.然后,在索引的基础上,实现 top-k 最短路径查询的响应过程.查询优化阶段,本文提出频繁路径策略以及结构剪枝策略.频繁路径策略引入了频繁路径挖掘算法,高效获取频繁路

径并压缩至索引中.结构剪枝策略分析了查询的响应过程,挖掘出终止条件以提高查询效率.最后,通过实验评价本文方法.

目前,本文工作均在静态数据库中完成.而 RDF 动态数据库的 top-k 最短路径查询要解决的主要问题是如何权衡数据的更新与查询.通过添加增量索引并推迟数据的实时更新操作.在数据库处于闲暇时,批量合并增量索引与原索引,完成更新操作.因此,本文框架在动态数据库中是可用的.其具体实践是后续工作的重点内容.

参考文献

[1] Syed A R, Dietmar S. Observing local and global properties of metabolic pathways: 'load points' and 'choke points' in the metabolic networks[J]. Bioinformatics, 2006, 22(14): 1767 – 1774.

[2] Stefano B, et al. Complex networks: structure and dynamics [J]. Physics Reports, 2006, 424(1): 175 – 308.

[3] Takuya A, et al. Shortest-path queries for complex networks: exploiting low tree-width outside the core[A]. International Conference on Extending DB Technology [C]. London: Springer, 2012. 144 – 155.

[4] Zhigang W, et al. Shortest path computation over disk-resident large graphs based on extended bulk synchronous parallel methods[A]. Database Systems for Advanced Applications [C]. London: Springer, 2013. 1 – 15.

[5] Mohamed S, et al. Horton + : a distributed system for processing declarative reachability queries over partitioned graphs[J]. The Proceedings of the VLDB Endowment, 2013, 6(14): 1918 – 1929.

[6] Fang W. TEDI: efficient shortest path query answering on graphs[A]. ACM Conference on Management of Data[C]. New York: ACM, 2010. 99 – 110.

[7] Yanghua X, et al. Efficiently indexing shortest paths by exploiting symmetry in graphs[A]. International Conference on Extending DB Technology [C]. London: Springer, 2009. 24 – 26.

[8] Takuya A. Pruned labeling algorithms: fast, exact, dynamic, simple and general indexing scheme for shortest-path queries [A]. International World Wide Web Conferences[C]. London: Springer, 2014. 1339 – 1340.

[9] Lingkun W, et al. Shortest path and distance queries on road networks: an experimental evaluation[J]. The Proceedings of the VLDB Endowment, 2012, 5(5): 406 – 417.

[10] Jiefeng C, et al. Top-k graph pattern matching over large graphs[A]. IEEE International Conference on Data Engineering[C]. Washington: IEEE Press, 2013. 1033 – 1044.

[11] Qian W, et al. Finding top-k profitable products[A]. IEEE International Conference on Data Engineering[C]. Washington: IEEE Press, 2011. 1055 – 1066.

[12] Lu Q, et al. Diversifying top-k results[J]. The Proceedings of the VLDB Endowment, 2013, 5(11): 1124 – 1135.

[13] Yuanyuan Z, et al. Finding top-k similar graphs in graph databases[A]. International Conference on Extending DB Technology[C]. London: Springer, 2012. 456 – 467.

[14] Thomas N, Gerhard W. The RDF-3X engine for scalable management of RDF data[J]. VLDB Journal, 2010, 19(1): 91 – 149.

[15] Octavian U, et al. GRIN: A graph based RDF index[A]. AAAI Conference on Artificial Intelligence [C]. California: AAAI Press, 2007. 1465 – 1470.

[16] Mihaela A B, et al. Building an efficient RDF store over a relational database[A]. ACM Conference on Management of Data[C]. New York: ACM, 2013. 121 – 132.

[17] Takuya A, et al. Dynamic and historical shortest-path distance queries on large evolving networks by pruned landmark labeling[A]. International World Wide Web Conferences[C]. London: Springer, 2014. 237 – 248.

作者简介

章登义　男,1965 年出生于湖北省荆州市.现为武汉大学计算机学院教授、博士生导师.
E-mail: dyzhangwhu@163.com

吴文李(通信作者)　女,1986 年出生于广东省湛江市.现为武汉大学博士研究生.主要研究方向为语义网数据挖掘.
E-mail: huihuigou@whu.edu.cn

欧阳黜霏　男,1988 年出生于湖北省荆州市.现为武汉大学博士研究生,从事数据库、数据挖掘方面的研究工作.
E-mail: whuxiaoouou@whu.edu.cn