文章编号：2095-2163(2021)02-0023-07 中图分类号：TP301.6 文献标志码：A

基于有向无环图的区间覆盖率求解算法

杜 明，庞建成，周军锋
（东华大学 计算机科学与技术学院，上海 201620）

摘 要： 给定一个有向无环图，回答可达性查询是图的基本操作之一。虽然很多方法使用树区间来加速可达查询的处理速度，但并不明确使用多少个区间比较合适。本文提出一种快速计算区间覆盖率的算法，该方法通过使用有效的剪枝策略来支持高效的覆盖率计算。基于所得到的区间覆盖率，可针对不同数据图确定合适的区间个数，以便在加速查询处理的同时，降低索引规模。基于多个真实数据集的实验验证了本文提出方法的高效性。

关键词： 有向无环图；可达性查询处理；区间覆盖率

Algorithm for solving interval coverage based on directed acyclic graph

DU Ming, PANG Jiancheng, ZHOU Junfeng
(School of Computer Science and Technology, Donghua University, Shanghai 201620, China)

【Abstract】 Given a directed acyclic graph, answering the reachability query is one of the basic operations of the graph. Although many methods use tree intervals to speed up the processing speed of reachable queries, it is not clear how many intervals are appropriate. This paper proposes an algorithm to quickly calculate the coverage of multiple intervals. This method supports efficient coverage calculation by using an effective pruning strategy. Based on the obtained interval coverage, an appropriate number of intervals can be determined for different data graphs, so as to reduce the index scale while speeding up query processing. Experiments based on multiple real data sets verify the efficiency of the method proposed in this paper.

【Key words】 directed acyclic graph; reachability queries processing; interval coverage

0 引 言

给定有向无环图（Directed Acyclic Graph，DAG），以及图中任意两个顶点 u、v，可达性查询 $u?\rightarrow v$ 用于回答从顶点 u 出发是否存在一条路径可以到达顶点 v。可达性查询处理是图数据管理与分析的基本操作之一，一直以来都是研究者广泛关注的热点问题[1-10]。在实际应用中，可达性查询被广泛应用到社交网络、通信与传感器网络、生物网络、可扩展标记语言数据、资源描述框架数据等领域，用于检测两点间是否存在特定关系。

为了加速回答可达性查询，已有的很多方法使用生成树区间来提升可达查询的处理效率。具体来说，给定图 G，这些方法先是构造 G 的生成树 T，接着基于 T 为每个顶点 u 设定一个区间 $I_u=[r_x,r_u]$，该区间覆盖了 u 在生成树 T 上的所有后代。对于给定查询 $u?\rightarrow v$，如果 v 是 u 在生成树上的后代，则可通过区间的包含关系在 $O(1)$ 时间内判断出 u 可达 v。已有方法的实验结果显示，基于生成树区间可以有效降低查询处理时的搜索时间，加速可达查询的处理速度。例如，有向无环图 G 如图 1(a)所示，G 的生成树如图 1(b)所示。其中，实线边表示树边，虚线表示非树边。根据树区间的特性，给定查询 $v_1?\rightarrow v_6$，由于 $I_6=[4,5]\subset[2,5]=I_1$，可以得出 v_1 可达 v_6 的结论，表示为 $v_1\rightarrow v_6$。

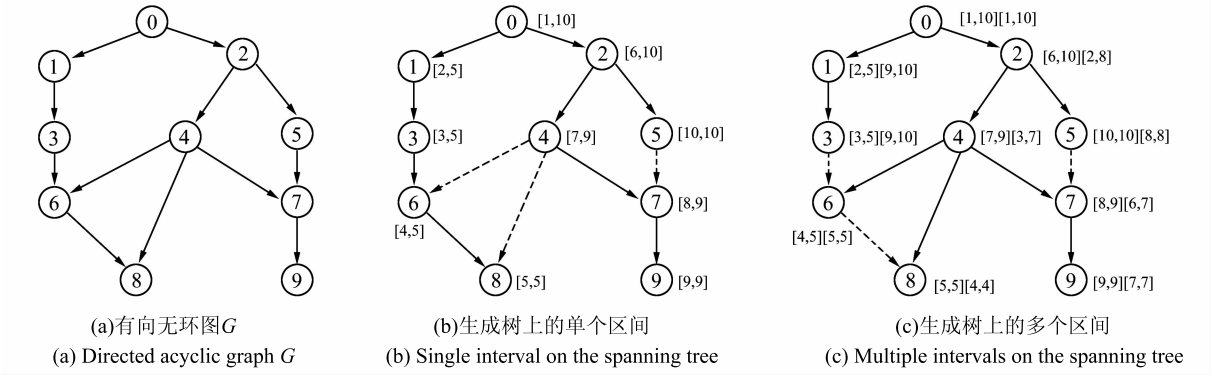
需要注意生成树区间不能完全覆盖 DAG 中节点之间的可达性关系，例如根据图 1(a)，v_4 可达 v_6，但根据图 1(b) 的区间显示可知，$I_6=[4,5]\not\subset[7,9]=I_4$，这时不能说明 v_4 不可达 v_6。对该问题，现有工作常使用多个生成树区间来扩大其覆盖率，从而增强查询处理时的剪枝能力。例如，图 1(c) 为每个点设定了 2 个区间。[4,5](v_6 的第一个区间) $\not\subset$ [7,9](v_4 的第一个区间)，但是[5,5](v_6 的第二个区间) $\subset$ [3,7](v_4 的第二个区间)，由第二个区间可知 $v_4\rightarrow v_6$。

基金项目： 国家重点研发计划(2017YFB0309800)。

作者简介： 杜 明(1975-)，男，博士，副教授，主要研究方向：自然语言处理、信息查询、数据分析；庞建成(1995-)，男，硕士研究生，主要研究方向：图的可达查询覆盖率研究；周军锋(1977-)，男，博士，教授，博士生导师，主要研究方向：大图数据的查询处理技术、推荐系统关键技术。

通讯作者： 周军锋 Email：zhoujf@dhu.edu.cn

收稿日期： 2020-11-17



(a)有向无环图G (b)生成树上的单个区间 (c)生成树上的多个区间

(a) Directed acyclic graph G (b) Single interval on the spanning tree (c) Multiple intervals on the spanning tree

图 1 生成树区间示例

Fig. 1 Example of spanning tree interval

虽然使用多个区间可以回答更多的可达性查询,但是现在的方法并不明确使用多少个区间比较合适。一方面,使用多个区间可以扩大覆盖率,从而增强区间标签的剪枝能力;另一方面,使用多个区间也意味着较大的索引和较长的查询响应时间。因此使用区间标签来加速可达查询处理时,一个关键问题是:应该建立多少个区间标签,从而在查询时间、索引大小以及索引构建时间之间获得平衡。显然,该问题的解决依赖于高效获知区间标签的覆盖率,如此即可根据系统的硬件环境限制,选择合适的区间个数,进而加速查询响应的效率。

然而,覆盖率的计算并非易事,原因在于不同生成树区间所覆盖的后代顶点有重复,多个区间覆盖率并非简单的累加和,需要在计算过程中去除重复部分。针对该问题,本文首次提出一种快速计算区间覆盖率的算法,称 k – DFSIC(k – Depth First Search Interval Coverage),其中 k 表示生成树区间的个数。该方法在求解覆盖率时通过使用有效的剪枝策略来支持高效的覆盖率计算,基于所得到的区间覆盖率,用户可针对不同数据图确定合适的区间个数来获得最佳的性能体验。

本文的其余部分安排如下。第 1 节介绍相关工作,第 2 节提出求区间覆盖率的算法,第 3 节展示实验结果,第 4 节总结全文。

1 相关工作

1.1 问题定义

虽然可达性查询针对的是一般有向图,但是现有方法可以通过压缩其中的强连通分量将其转换为有向无环图来处理,因此,本文假设输入的都是有向无环图。以下介绍区间覆盖率的定义及问题的定义。

定义 1 区间覆盖率 给定有向无环图及其生成树上的多个区间,区间覆盖率表示仅通过顶点的区间能回答的可达对个数占有向无环图中所有可达对个数的比例。

例如图 1(a)中有向无环图 G 的所有可达对的数目为 28,图 1(b)中所用一个生成树区间能够回答的可达对个数是 22 个,则使用一个区间时,其区间覆盖率为 22/28,图 1(c)中使用 2 个生成树区间能够回答的可达对个数是 26 个,则其区间覆盖率为 26/28。

问题定义:给定有向无环图及其对应的多个生成树区间,返回这些生成树的区间覆盖率。

1.2 相关算法

虽然区间覆盖率可用于协助用户针对不同图来确定合适的区间个数,但关于区间覆盖率的计算问题还未见到任何研究,本文首次对区间覆盖率的问题进行研究,提出一种快速计算区间覆盖率的算法。这里,研究中仅讨论使用区间的可达性查询处理算法,包括 GRAIL[11]、FERRARI[5]、FELINE[12]三个算法。对此可做阐释分述如下。

GRAIL 算法基于生成树为每个点附加多个区间,其中一半区间是包含所有图中后代的区间,用于判断不可达,另外一半区间是包含生成树中后代的区间,用于判断可达。由于区间个数会影响索引大小及查询效率,本次研究在进行实验时,基于数据图的稀疏程度来设定区间个数。对于稀疏图(度小于 2),使用 2 个区间,其它图使用 5 个区间。该方法是一种基于经验的直觉,区间增多时,不一定能增强查询处理能力。

FELINE 用 x 和 y 表示顶点的 2 个拓扑号,用 I 表示顶点的区间标记。在判断顶点 u 是否可达顶点 v 的时候,如果顶点的拓扑号满足 $x_u > x_v$ 或者 $y_u > y_v$,则可知顶点 u 不可达顶点 v;如果 $x_u < x_v$ 并且

$y_u < y_v$，并且满足 $I_v \subset I_u$，则顶点 u 可达顶点 v；如果用 2 个拓扑号和区间标记判断不出顶点对的可达性，就还需要利用深度优先遍历或者广度优先遍历的方法进行判断。该方法仅使用一个区间来协助查询剪枝，不能根据区间的覆盖率来增强剪枝效果。

FERRARI 根据内存的限制情况来确定区间个数，由于区间之间有重叠，因此即使内存够大，可支持多个区间，但却仍然难以保证查询效率。

以上方法虽然都使用区间来加快可达性查询的处理效率，但是这些方法都不能确定应该使用多少区间来回答才合适，针对该问题，本文提出一种快速计算区间覆盖率的方法，以便针对不同数据图确定合适的区间个数，从而提升查询效率并减小索引规模。

2　求区间覆盖率的算法

2.1　基础算法

给定有向无环图 G 及其生成树上的 k 个区间，求区间覆盖率的基本思想是：首先找到图 G 中所有的可达对；然后判断这些可达对是否能通过区间回答，能回答说明该可达对能够通过区间判断，不能回答则说明该可达对不能通过区间判断；最后求出可以通过区间回答的可达对个数占图 G 中所有可达对个数的比例，即可得到使用 k 个生成树区间的覆盖率。

算法 1 展示了基础算法求区间覆盖率的伪代码，具体如下。

算法 1　基础算法

输入： $G=(V,E)$，k 个生成树区间

输出： 区间覆盖率 *icr*

```
1   nRch ← 0; nUnRch ← 0;
2   foreach reachable pair(u,v) of G do
3     foreach i in [ I,k ] do
4       if I[v][i] ⊂ I[u][i] then
5         nRch ++
6       else
7         nUnRch ++
8     icr ← nRch/(nRch + nUnRch)
```

算法中，第 1 行设置 2 个变量 *nRch* 和 *nUnRch*，分别表示可以通过生成树区间回答的可达对个数和不能通过生成树区间回答的可达对个数，并将其初始值设为 0；第 2 ~ 7 行针对图 G 中所有的可达对，依次判断其是否能通过给定的 k 个区间回答，如果可以回答，*nRch* ++，如果不能回答，则 *nUnRch* ++；第 8 行通过 *nRch* 的值除以（*nRch* + *nUnRch*）的值求出使用 k 个生成树区间的覆盖率。

给定有向无环图 G 见图 1(a)。首先找到图 G 中所有的可达对，详见表 1，总共有 28 个；然后判断这些可达对能否通过给定的区间回答。研究可知，图 1(b) 为图 G 中每个节点给定了一个生成树区间，[8,9](v_7 的区间) ⊂ [7,9](v_4 的区间)，说明 $v_4 \rightarrow v_7$ 可以通过该区间回答，则通过一个区间能回答的可达对个数 *nRch* ++，而[4,5](v_6 的区间) ⊄ [7,9](v_4 的区间)，说明 $v_4 \rightarrow v_6$ 不能通过该区间回答，则不能通过一个区间回答的可达对个数 *nUnRch* ++，当图 G 中所有可达对判断结束后，可以得到 *nRch* 为 22，*nUnRch* 为 6，故通过计算可得使用一个区间的覆盖率是 22/(22 + 6)，即 22/28。研究中的图 1(c) 又为 G 中每个节点给定了 2 个生成树区间，[5,5](v_6 的第二个区间) ⊂ [3,7](v_4 的第二个区间)，说明 $v_4 \rightarrow v_6$ 可以通过该区间回答，则通过 2 个区间能回答的可达对个数 *nRch* ++，而[6,7](v_7 的第二个区间) ⊄ [8,8](v_5 的第二个区间)，说明 $v_5 \rightarrow v_7$ 不能通过该区间回答，则通过 2 个区间不能回答的可达对个数 *nUnRch* ++；当图 G 中所有可达对判断结束之后，可得 *nRch* 为 26，*nUnRch* 为 2，则通过计算可得使用 2 个区间的覆盖率是 26/(26 + 2)，即 26/28。

表 1　G 中的可达对

Tab. 1　Reachable pairs of G

顶点编号	G 中可达的顶点集
0	{1,2,3,4,5,6,7,8,9}
1	{3,6,8}
2	{4,5,6,7,8,9}
3	{6,8}
4	{6,7,8,9}
5	{7,9}
6	{8}
7	{9}
8	∅
9	∅

基础算法若要枚举有向无环图 G 中所有的可达对，需要的时间为 $O(kn(n+m))$，其中 n 为 G 的顶点个数，m 为 G 的边数。判断每个可达对能否通过生成树区间回答，可以在 $O(k)$ 时间内完成，故基础算法的时间复杂度是 $O(kn(n+m))$。

2.2　k –DFSIC 算法

由于基础算法在计算过程中需要枚举有向无环

图中所有的可达对,时间复杂度太大,在实践中无法处理大图。针对该问题,本文提出一种快速计算区间覆盖率的 k - DFSIC 算法。给定有向无环图 G 及其 k 个生成树区间,求区间覆盖率之前,为图中每个节点设定一个计数器,表示该节点通过区间能回答的可达对个数,初始值为第一个区间的长度。求区间覆盖率的基本思想是:当求解顶点 u 的第 i 个区间新增的可达查询数量时,对于第 i 个区间中的所有生成树后代点,即检视 u 与其之间的可达性是否可以通过前 $i-1$ 个区间来判断。如果可以,则 u 的计数器不变,否则说明该可达对无法通过前 $i-1$ 个区间判断,则 u 的计数器加 1。当所有顶点处理结束后,将顶点的计数器累加,即可得到所有节点通过区间能回答的可达对个数。该值除以图 G 中所有可达对个数,就是使用 k 个区间的覆盖率。

算法 2 展示了 k - DFSIC 算法求区间覆盖率的伪代码,具体如下。

算法 2 k-DFSIC

输入: $G=(V,E)$,k 个生成树区间

输出: 区间覆盖率 icr

```
1   nTC ← the size of transitive closure of G
2   nRCH ← 0
3   foreach( u ∈ V ) do u. nRch ← I[u][1]的区间长度
4   foreach (i in [2,k]) do
5       foreach (u ∈ V) do
6           foreach ( v in I[u][i] ) do
7               flag = TRUE
8               foreach( j in [1,i - 1] ) do
9                   if I[v][j] ⊂ I[u][i] then
10                      flag = FALSE
11                      Break
12              if ( flag = TRUE) then u. nRch ++
13  foreach( u ∈ V ) do nRCH + = u. nRch
    icr ← nRCH/nTC
```

算法中,第 1 行设置了一个变量 nTC 表示有向无环图 G 的传递闭包的大小;第 2 行设置了一个变量 $nRCH$ 表示图 G 中所有节点通过区间能回答的可达对个数,并设其初始值为 0;第 3 行为图 G 中每个节点 u 给定一个计数器表示该节点通过区间能回答的可达对个数 $nRch$,其初始值为节点 u 的第一个区间长度;第 4 ~ 12 行在求解节点 u 通过 $i(2 \leqslant i \leqslant k)$ 个区间新增的可达查询数量时,对于第 i 个区间中的所有生成树上后代节点 v,设置一个变量 $flag$ 表示 u 和 v 之间的可达性是否可以通过前 $i-1$ 个区间来判断,其初始值设为 $TRUE$,如果 u 和 v 之间的可达性可以通过前 $i-1$ 个区间来判断,则 $flag$ 值为 $FALSE$,判断后如果 $flag$ 值为 $TRUE$,说明该可达对无法通过前 $i-1$ 个区间判断,则 $nRch$ 加 1;第 13 ~ 14 行将图 G 中每个节点的计数器 $nRch$ 加和即所有节点通过区间能回答的可达对个数 $nRCH$,用 $nRCH$ 的值除以图 G 中所有可达对的个数 nTC,即可得到使用 k 个区间的覆盖率。

例如,图 1(a) 所示有向无环图 G 中所有可达对共有 28 个,参见表 1,即图 G 的传递闭包的大小 nTC 为 28。

由图 1(b) 可知,为图 G 中每个节点给定了一个区间,并为每个节点给定一个计数器表示该节点通过一个区间能回答的可达对个数 $nRch$ 为该节点的区间长度,比如说 v_4 的区间是[7,9],则 v_4 通过一个区间能回答的可达对个数 $nRch$ 为 v_4 的区间长度 2,将每个节点的 $nRch$ 值加和可以得到所有节点通过一个区间能回答的可达对个数 $nRCH$ 为 22,则使用一个区间的覆盖率是 22/28。

由图 1(c) 可知,为图 G 中每个节点给定了 2 个区间,并为每个节点给定一个计数器表示该节点通过 2 个区间能回答的可达对个数 $nRch$ 首先设置为该节点的第一个区间的长度,对于生成树上每个节点来说,要找到该节点到其后代节点的所有可达查询,参见表 2,并检查这些查询能否通过前 $i-1$ 个区间回答来更新 $nRch$ 的值,比如 v_4 节点,v_4 的第一个区间是[7,9],则 v_4 通过 2 个区间能回答的可达对个数 $nRch$ 首先设置为 v_4 的第一个区间的长度 2,然后依次检查 $v_4 \rightarrow v_6$,$v_4 \rightarrow v_7$,$v_4 \rightarrow v_8$,$v_4 \rightarrow v_9$ 能否通过前 $i-1$ 个区间回答来更新 $nRch$ 的值,可以发现 $v_4 \rightarrow v_6$ 不可以通过第一个区间回答,但是可以通过第二个区间回答,则 $nRch++$,同理 $v_4 \rightarrow v_8$ 也不可以通过第一个区间回答,但是可以通过第二个区间回答,则 $nRch++$,而 $v_4 \rightarrow v_7$,$v_4 \rightarrow v_9$ 已经可以通过第一个区间回答了,则 $nRch$ 值不变,当所有查询判断结束之后,可以得到 v_4 通过 2 个区间能回答的可达对个数 $nRch$ 为 4;将每个节点的 $nRch$ 值加和可以得到所有节点通过 2 个区间能回答的可达对个数 $nRCH$ 为 26,则使用 2 个区间的覆盖率是 26/28。

k - DFSIC 算法需要在有向无环图 G 上找到其生成树上的每个节点的后代节点。和基础算法不同,基础算法需要枚举图上的后代,而 k - $DFSIC$ 只需要枚举树上的后代,该值等于树上每个顶点的高

度之和。假设顶点的平均高度是 h，则求解可达对个数 $nRCH$ 的代价是 $O(k(n+m)+khn)$，其中 n 为 G 的顶点个数，m 为 G 的边数；同时，求传递闭包的时间代价为 $O(wm)$[13]，这里 w 是求传递闭包大小过程中，处理每个点时平均处理代价。因此，k - DFSIC 的时间复杂度是 $O(k(n+m)+khn+wm)$。

表 2　生成树上的可达对

Tab. 2　Reachable pairs on the spanning tree

顶点编号	给定 2 个区间时在生成树上可达的顶点集
0	{1,2,3,4,5,6,7,8,9}
1	{3}
2	{4,5,6,7,8,9}
3	Ø
4	{6,7,8,9}
5	Ø
6	Ø
7	{9}
8	Ø
9	Ø

3　实验分析

3.1　实验环境

实验所用的硬件平台是 Intel Core i5-4460 主频为 3.20 GHz 的 CPU，4 GB 的 RAM 内存，操作系统为 Ubuntu 8.3.0，并使用 gcc 8.3.0 进行编译，以上算法均采用 C++语言实现。

3.2　数据集

本文中使用的 12 个数据集见表 3。这些数据集都广泛地出现在图的可达查询研究中[4-5,9,14-16]，这些数据集都是有向无环图，表 3 中标注了每个数据集的顶点数 $|V|$ 以及边数 $|E|$。

表 3　数据集

Tab. 3　Datasets

数据集	$\|V\|$	$\|E\|$
human	38 811	39 576
anthra	12 499	13 104
agrocyc	12 684	13 408
xmark	6 080	7 025
mtbrv	9 602	10 245
nama	5 605	6 537
uni100m	16 087 295	16 087 293
WikiTalk	2 281 879	2 311 570
dbpedia	3 365 623	7 989 191
govwild	8 022 880	23 652 610
twitter	18 121 168	18 359 487
web-uk	22 753 644	38 184 039

3.3　索引大小

k-DFSIC 算法求区间覆盖率使用不同区间时的索引大小见表 4。由表 4 可以看出随着区间个数的增加，索引大小呈线性增加。

表 4　索引大小

Tab. 4　Index size　　MB

数据集	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$
human	0.29	0.59	0.89	1.18	1.48
anthra	0.10	0.19	0.29	0.38	0.48
agrocyc	0.10	0.19	0.29	0.39	0.48
xmark	0.05	0.09	0.14	0.19	0.23
mtbrv	0.07	0.15	0.22	0.29	0.37
nasa	0.04	0.09	0.13	0.17	0.21
uni100m	122.74	245.47	368.21	490.95	613.68
WikiTalk	17.41	34.82	52.23	69.64	87.05
dbpedia	25.68	51.36	77.03	102.71	128.39
govwild	61.21	122.42	183.63	244.84	306.05
twitter	138.25	276.51	414.76	553.01	691.27
web-uk	173.60	347.19	520.79	694.39	867.98

3.4　覆盖率求解时间

表 5 比较了基础算法和 k - DFSIC 算法求区间覆盖率所需的时间。由表 5 可以发现 k - DFSIC 算法与基础算法相比，在使用相同区间个数的情况下，所需的时间要少得多，并且基础算法在大图当中运行时间太长，而 k - DFSIC 算法则只需很少的时间，表 5 中，“-”表示基础算法运行时间超过了 2 h。

3.5　区间覆盖率

表 6 展示了本文实验所得到的区间覆盖率。由表 6 发现随着区间个数的不断增加，覆盖率虽然也不断增加，但是有的数据集覆盖率增加值比较小，比如 twitter 数据集；而有的数据集覆盖率增加值比较大，比如 xmark 数据集，使用 5 个区间时的覆盖率是使用 1 个区间时覆盖率的 3 倍多。

3.6　实验结论

首先，本文提出的区间覆盖率计算算法可高效求解区间覆盖率，在实际中需要了解区间覆盖率的情况下，可通过使用本文提出的算法进行高效求解；其次，通过本文实验所得到的区间覆盖率，可以发现有些数据集只需要使用 2 个区间来回答可达性查询就比较合适了，比如 human 数据集，而有的数据集使用 5 个区间比较合适，比如 xmark 数据集。假设内存足够的情况下，考虑到查询效率，针对不同数据集回答可达性查询合适的区间个数见表 7；如果内存不足以使用多个区间，则需要用户根据实际情况确定合适的区间个数。

表 5 计算区间覆盖率的运行时间

Tab. 5 Running time of computing interval coverage rate ms

数据集	$k=1$		$k=2$		$k=3$		$k=4$		$k=5$	
	基础算法	k - DFSIC	基础算法	k - DFSIC	基础算法	k - DFSIC	基础算法	k - DFSIC	基础算法	k - DFSIC
human	3 973.61	5.29	3 865.27	23.42	3 974.09	23.44	3 912.71	23.66	3 859.38	23.78
anthra	438.37	2.48	427.78	8.19	446.19	8.24	431.87	8.97	427.45	8.96
agrocyc	444.51	2.98	443.85	8.59	461.64	8.76	441.87	8.79	461.73	9.03
xmark	198.76	1.21	202.17	4.86	214.35	5.19	205.25	5.76	214.71	5.61
mtbrv	270.04	1.36	263.46	5.88	276.86	6.01	267.46	6.05	273.26	6.62
nasa	117.33	1.29	116.24	4.62	120.22	5.35	117.86	5.69	119.19	5.99
uni100m	–	4 585.67	–	7 199.26	–	7 186.87	–	7 211.65	–	7 223.42
WikiTalk	–	312.37	–	850.46	–	868.52	–	875.19	–	890.74
dbpedia	–	2 541.88	–	3 272.14	–	3 291.96	–	3 328.17	–	3 365.14
govwild	–	5 038.32	–	6 921.92	–	6 938.48	–	6 942.28	–	6 948.32
twitter	–	3 588.31	–	7 568.73	–	7 687.18	–	7 811.46	–	7 936.26
web-uk	–	4 596.89	–	10 718.81	–	10 840.42	–	10 991.85	–	11 157.28

表 6 区间覆盖率

Tab. 6 Interval coverage

数据集	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$
human	0.646 500 00	0.651 300 00	0.655 300 00	0.656 100 00	0.656 700 00
anthra	0.471 700 00	0.480 200 00	0.483 300 00	0.485 300 00	0.489 200 00
agrocyc	0.413 900 00	0.422 900 00	0.430 500 00	0.430 800 00	0.431 200 00
xmark	0.080 000 00	0.130 000 00	0.160 000 00	0.210 000 00	0.270 000 00
mtbrv	0.381 200 00	0.384 400 00	0.389 500 00	0.391 600 00	0.394 800 00
nasa	0.290 000 00	0.320 000 00	0.340 000 00	0.370 000 00	0.390 000 00
uni100m	0.082 700 00	0.083 200 00	0.083 600 00	0.084 200 00	0.084 400 00
WikiTalk	0.000 090 00	0.000 160 00	0.000 210 00	0.000 260 00	0.000 320 00
dbpedia	0.000 009 93	0.000 011 30	0.000 011 80	0.000 013 60	0.000 014 50
govwild	0.001 700 00	0.001 900 00	0.002 300 00	0.002 500 00	0.002 700 00
twitter	0.000 001 20	0.000 001 80	0.000 002 40	0.000 003 10	0.000 003 60
web-uk	0.000 000 70	0.000 000 80	0.000 000 90	0.000 001 10	0.000 001 30

表 7 建议的区间个数

Tab. 7 Suggested number of intervals

数据集	建议的区间个数	数据集	建议的区间个数
human	2	uni100m	2
anthra	2	WikiTalk	5
agrocyc	2	dbpedia	2
xmark	5	govwild	2
mtbrv	2	twitter	5
nasa	4	web-uk	5

4 结束语

本文针对已有算法回答可达性查询时使用树区间来加快处理速度、但是并不明确使用多少个区间比较合适的问题,首次提出了一种快速计算区间覆盖率的算法。实验结果表明,本文提出的算法可高效计算区间覆盖率。基于所得到的区间覆盖率,用户可结合实际应用环境的限制确定可达查询处理过程中应该使用的区间数量,从而获得最佳的性能体验。

参考文献

[1] AGRAWAL R, BORGIDA A, JAGADISH H V, et al. Efficient management of transitive relationships in large data and knowledge bases[J]. ACM SIGMOD Record, 1989, 18(2): 253-262.

[2] CHENG J, HUANG S, WU 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.

(下转第 34 页)