一种基于树分解的图上点区间编码方法及应用

陈子轩[1] 何震瀛[2] 荆一楠[2]
[1](复旦大学软件学院 上海 201203)
[2](复旦大学计算机科学技术学院 上海 201203)

摘 要 根据图上节点所在位置与邻居节点特征,可以使用不同策略为每个图上节点进行区间编码,基于区间编码,许多在大型图上的应用如知识图谱查询、智能问答等的处理可以加速或得到准确性上的提升。针对此种情况,提出一种基于树分解算法的图上点区间编码方法,并在大型知识图谱上通过智能问答歧义消除的应用验证该方法的有效性。实验结果表明,该方法能够有效地表达出图上节点的位置特征,并帮助智能问答中的实体消除歧义。

关键词 知识图谱 树分解 区间编码

中图分类号 TP3 **文献标志码** A DOI:10.3969/j.issn.1000-386x.2022.03.041

A RANGE ENCODING APPROACH BASED ON TREE DECOMPOSITION AND ITS APPLICATIONS

Chen Zixuan[1] He Zhenying[2] Jing Yi'nan[2]
[1](*School of Software*, *Fudan University*, *Shanghai 201203*, *China*)
[2](*School of Computer Science*, *Fudan University*, *Shanghai 201203*, *China*)

Abstract Based on the position and neighbors of one vertex in a graph, there exist different strategies which are able to conduct range encoding for each vertex in the graph. With range codes, plenty of applications over large graphs such as query processing and question answering over knowledge graphs can be accelerated or processed more accurately. In view of this, we propose a range encoding approach based on tree decomposition algorithm. We verified the effectiveness of the proposed approach by conducting experiments of disambiguation during question answering over large knowledge graph. Experimental results show that the proposed approach can present the position features of vertexes effectively, and help disambiguation during question answering.

Keywords Knowlede graph Tree decomposition Range encoding

0 引 言

近年来,大规模数据图开始在各种各样的应用中扮演重要的角色。许多大型知识图谱如 YAGO[1]、Freebase[2]、Probase[3]等均被构建出来,为许多应用提供知识支持。随着越来越多的大规模数据图在应用中被使用到,快速地对这些图中所包含的数据信息进行处理与分析成为了一件十分有意义的事情。

考虑到在大型数据图上,与某个数据点相关的其他数据其实是非常局限在某个范围内的,因此,若能将每个数据点在数据图上的位置特征(该点在图上的位置、邻居数据点范围等)表示出来,则可以在处理具体应用任务时,只考虑某个范围内的相关数据点,而不是在大型数据图的全图上进行检索,从而可以节省下大量时间,甚至加强图上应用的准确性。

为成功刻画每个数据点的位置特征,本文提出一种基于树分解算法(本文中使用的树分解算法来源于 Wei[4]的工作)的图上节点区间编码方法,用来为图上每个节点(在本文中存在两种节点,分别为图上节点

收稿日期:2020-01-23。国家重点研发计划项目(2018YFB1004404)。**陈子轩**,硕士生,主研领域:数据库。**何震瀛**,副教授。**荆一楠**,副教授。

即数据点与树上节点,在可能存在混淆时文中区别使用,避免歧义理解;在不存在混淆的情况下,仅称节点)赋予一个区间编码,由开始编码与结束编码组成。

对于一个给定的数据图,可以使用树分解算法将这幅图分解成一棵特殊的分解树。这棵分解树上的所有节点均为一个由图上节点组成的集合且具有以下性质:(1) 对于图上的每个节点,包含它的树上节点之间在树上保持连续,可以形成一棵节点子树;(2) 对于在图上连接在一起的两个点,一定会至少共同出现在树上的同一个节点一次。而因为以上两点性质,对于任意两个在数据图上由边连接在一起的数据点,其在分解树上对应的节点子树的根节点一定会存在互相之间的祖先-后代关系。

基于这棵分解树,可以为图上所有节点赋予一个区间编码,由开始编码与结束编码组成,其中开始编码为此图上节点对应的节点子树的根节点在树上的先序遍历序号,而结束编码为此图上节点的开始编码加上其对应的节点子树的树上后代节点数。对于两个图上数据点,若其对应的节点子树的根节点互相之间存在祖先-后代关系,则意味着这两个数据点的区间编码会存在互相包含的关系。因此,若图上两个数据点之间有边,则其区间编码必定存在互相包含关系;反之,若两个数据点的区间编码之间不存在互相包含关系,则这两个数据点在数据图上相互之间不可能有边。

基于本文提出的区间编码方法,数据图上的每一个数据点均有一个对应的区间编码,用来表达其在图上的相关范围,从而刻画出此节点在图上的位置信息与邻居信息。利用每个节点的区间编码,许多在大型数据图上的应用可以得到加速,甚至得到更高的效果保证。举例来说,对于在知识图谱上的查询处理,利用每个节点的区间编码可以快速判断两个数据点之间是否有可能存在一条边,从而加快在查询处理中 JOIN 操作的速度。除此之外,对于基于知识图谱的智能问答系统,使用区间编码可以快速进行一部分的实体消歧,提高整个智能问答的效率,对于在智能问答上的应用,本文后面的实验部分通过实验验证了本文方法的有效性与实用性。

本文的贡献可以总结如下:

(1) 本文提出一种基于树分解的图上点区间编码方法,用来表示图上节点的位置特征。

(2) 本文提出针对 YAGO 数据集的问答问题 100 句,并使用这些问题进行了消歧实验,证明了本文提出的区间编码在实际应用中的有效性。

1　图上的树分解

1.1　基本定义

树分解是一种将一幅图映射到一棵树上的图上算法,通过这种算法,一些图上的特定问题可以得到加速解决。树分解的概念最早由 Robertson 等[5]提出。

定义 1(树分解)　对于一幅图 $G=(V,E)$,其中:V 为图中所有点的集合;E 为图中所有边的集合。其对应的一个树分解 T_G 具体表示为$(\{X_i \mid i\in I\},T)$,其中:I 为一组连续整数,数量对应分解树上节点的数量;X_i 为 V 的一个子集,对应分解树上第 i 个节点中包含的图上数据点;T 为满足以下条件的一棵树(我们将其称为分解树):(1) $\bigcup_{i\in I}X_i=V$;(2) 对于任一对数据点 $(v,u)\in E$,在树 T 上必存在一个节点,使得其上的图上节点集合 X_i 满足 $v,u\in X_i$;(3) 对于任一数据点 v,其对应的节点子树(参见定义 2)一定存在。

定义 2(节点子树)　对于一幅图 $G=(V,E)$ 及在该图上的一个树分解 $T_G=(\{X_i \mid i\in I\},T)$,定义 T_v 为所有包含点 v 的 X_i 及这些 X_i 之间在分解树上的连接关系所组成的数据结构,若 T_v 能够形成一棵树结构,则称其为 v 对应的节点子树。

根据以上定义,可以得出以下重要引理,该引理为本文方法的基础。

引理 1　对于一幅图 $G=(V,E)$ 及在该图上的一个树分解 $T_G=(\{X_i \mid i\in I\},T)$,$v$ 与 u 为 V 中的两个点,令 R_v、R_u 分别为 v 与 u 在分解树 T 上的对应的节点子树的根节点,若$(v,u)\in E$,则 R_v 与 R_u 在分解树 T 上必定存在祖先-后代关系。

对于引理 1 的证明如下:

根据定义 1 中分解树 T 的性质 2 可知,若对于两点 v、u,满足$(v,u)\in E$,则在树 T 上必存在一个节点使得其上对应的图上节点集合 X_i 满足 $v,u\in X_i$。而根据定义 1 中分解树 T 的性质 2 与定义 2 可知,该树上节点必定同时存在于点 v 对应的节点子树 T_v 与点 u 对应的节点子树 T_u 上,因此该节点同时是点 v 对应的节点子树根节点 R_v 与点 u 对应的节点子树根节点 R_u 的后代节点。而因为在一棵分解树上,一个节点至多只能存在一个父亲节点,因此 R_v 与 R_u 之间必定存在祖先-后代关系。引理 1 证毕。

1.2　树分解算法

首先需要指出的是,对于一幅数据图,存在着不止一种满足定义 1 的树分解,本文旨在提出结合树分解算法和区间编码算法的可能与应用前景,对于各种树

分解孰优孰劣,在本文中不予探讨。本工作中采用的树分解方法来源于 Wei[4] 的工作,该工作通过将一幅图分解至一棵分解树上,利用分解树的特性实现了高效地图上最短路径查询,是一次使用树分解算法的极有意义的应用。

算法 1 介绍了本文中使用的树分解算法的具体内容。图 1 为该算法过程的一个示例,可以参考图 1 理解算法 1 的内容。算法 1 可分为两个部分,即从数据图将数据点分解至分解栈中(图 1(a)到图 1(b))与根据分解栈建立分解树(图 1(b)到图 1(c))。

算法 1 树分解算法

输入:数据图 G。

输出:分解树 T_G。

1) 新建分解栈 S;

2) while 数据图上所剩的点未形成完全图 do:

3) 找到图上具有最小度的点 v;

4) 得出点 v 的邻居节点集合 $U=\{u_1,u_2,\cdots,u_n\}$;

5) 将 U 中所有点两两连接起来;

6) 将集合 $\{v,u_1,u_2,\cdots,u_n\}$ 压入栈 S 中;

7) 删除点 v 及与点 v 相连的所有边;

8) 将图上所有剩余点组成的集合压入栈 S 中;

9) 从分解栈中的栈顶点集出栈,并使其作为分解树 T_G 的根节点,开始构建分解树;

10) while 分解栈 S 非空 do:

11) 从栈顶将第 i 个分解栈中的节点集合 $X_i=\{v_i,u_{i1},u_{i2},\cdots,u_{im}\}$ 出栈;

12) 找到满足包含 $\{u_{i1},u_{i2},\cdots,u_{im}\}$ 的最大的 $k(k<i)$ 及其对应的 X_k;

13) 将 X_i 作为 X_k 对应的分解树 T_G 上节点的子节点。

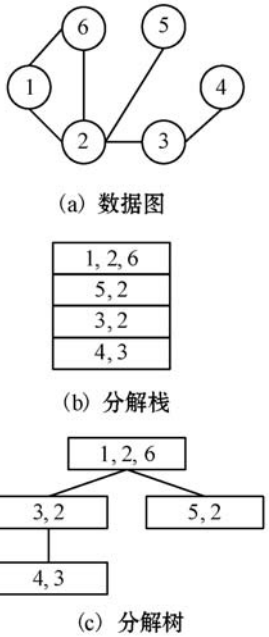


图 1 图上的树分解过程示例

首先介绍图上的分解过程。第一步需要初始化一个分解栈 S,用来承装每次被分解产生出来的数据点集合(行 1)。在算法 1 中分解过程设置为持续进行直至图上剩余点共同构成一个完全图(行 2),值得注意的是,此条件并非分解算法的必要内容,分解过程的进度同样可以设置阈值控制,如设置当数据图上所剩数据点个数小于某个阈值时停止分解或当数据图中所剩数据点的平均度数高于某个阈值时停止分解。设置阈值控制分解进度的好处在于树分解算法本身的时间代价是非常高的,同时,随着分解过程的进行,数据图变得更加稠密,使得每次分解的时间代价逐步提高,因此,通过设置阈值,可以在保持所需要的分解成果的前提下,减少树分解过程的时间消耗。对于每次分解,首先找到图上度最小的点 v,对于度相同的点,则根据预先为该点设置的节点编号大小来决定分解顺序(行 3)。树分解算法在该次分解中将点 v 分解掉,也将其及与其相连的所有边删除,并将点 v 与其所有邻居节点组成的数据点集合压入栈中(行 6 - 行 7)。但仅仅删除掉点 v 有可能破坏原数据图中存在的数据信息。举例说明,若图上存在三个点 v、u、w,三个点满足关系 v 与 u、w 分别相连,而 u 与 w 之间无连接关系,直接删除 v 可能导致 u 和 w 两点之间的连通性消失,从而破坏了原数据图中包含的数据信息。因此,在删除点 v 的同时,需要将点 v 的所有邻居节点两两连接在一起,以保持原图中所有的连通性与位置信息被保存下来(行 4 - 行 5)。最后,将图上所有剩余点组成的点集压入栈中(行 8)。

下面介绍如何由一个分解完成的分解栈构建一棵满足要求的分解树。首先,将图中剩余的所有点合成一个数据点集,使其作为分解树的根节点,也将分解栈栈顶的点集作为分解树的根节点(行 9)。随后,每次从分解栈中出栈一个数据点集 X_i,找到包含该点集除了第一个数据点(即在该次分解中被分解掉的那个数据点)以外的所有点的最近出栈的点集 X_k,令 X_i 作为 X_k 的子节点。以此方式不断进行,构建出整棵分解树(行 10 - 行 13)。

图 1 为一个完整的图上树分解过程示例,参考图 1 可以更好地理解算法 1 的过程。首先,原始数据图中有 6 个点与 6 条边,目标是将该图分解成为一棵分解树。第一步是将该图分解入栈,在第一次分解过程中找到当时图上度最小的点 4,将点 4 与其邻居 3 组成的点集入栈,因邻居个数为 1,所以不需要在邻居中添加边;继续此过程,最终当图上剩下 1、2、6 三个点时形

成完全图,将点 1、2、6 组成的点集压入分解栈中。然后则是构建分解树的过程。首先将栈顶的点集{1,2,6}出栈,使其作为分解树的根节点。之后继续在分解栈中执行出栈操作,点集{5,2}被出栈,需要找到包含{2}的最近被出栈的节点,即根节点,因此{5,2}作为根节点的子节点;继续此过程,构建出如图 1 所示的分解树。

在此示例中回想引理 1,图 1(a)中的所有边上的两个数据点在图 1(c)中的分解树中完全满足引理 1。举例来说,点 3 与点 4 在图 1(a)中互为邻居,在图 1(c)的分解树中,点 4 对应的节点子树的根节点为点 3 对应的节点子树的根节点的子节点。

2　区间编码

2.1　基本定义

区间编码方法[6]是在许多数据管理问题中都被使用到的热门方法。此方法被广泛应用在 XML 文档的查询当中[7-9],使用区间编码方法可以有效地表示出 XML 文档中的嵌套关系,并通过比较两个 XML 文档中两个节点的区间包含关系快速判断此两个节点的结构关系。

在通过图上的树分解方法将一幅数据图分解成一棵分解树之后,一幅数据图被以树的形式表示出来。XML 数据因其特殊的树型结构特征,使得区间编码方法可以在其上得到有效利用;同样地,当我们用一棵分解树来表示一幅数据图之后,区间编码方法同样可以使用在这样的一棵分解树上,去表示数据图上的位置信息与连接信息。因此,本文在通过树分解算法将一幅数据图分解为一棵分解树之后,进一步采用区间编码方法为分解树编码,再从树上编码中得到每个数据图上的数据点相对应的区间编码。本文希望将引理 1 的性质通过区间编码量化出来,以达到若两个数据点互为邻居,其区间编码互相包含。

下面给出本工作中对于树上编码与图中节点编码的定义。

定义 3(树编码)　对于一棵树 T,对于树 T 上的每个节点 X,均有一组一一对应的区间编码($start$,end),其中:$start$ 为该树上节点对应的起始编码;end 为该树上节点对应的结束编码。每个树上节点对应的区间编码称作该节点的树编码。

定义 4(数据点区间编码)　对于一幅数据图 $G=(V,E)$,对于图上的每个数据点 $v \in V$,均有该点对应的区间编码($start$,end),其中:$start$ 为该数据点对应的起始编码;end 为该数据点对应的结束编码。此区间编码称作该数据点的数据点区间编码。需要注意的是,此区间编码与数据点并非一一对应关系,即不同数据点可能对应着相同的一段区间编码。

2.2　区间编码算法

算法 2 介绍了本文中使用的区间编码算法的过程。图 2 为该算法过程的一个示例,在此示例中使用的分解树对应图 1 示例中产生的分解树,可以参考图 2 理解算法 2 的内容。算法 2 可分为两个部分,即树编码过程(图 2(a))与数据点区间编码过程(图 2(b))。

算法 2　区间编码算法

输入:分解树 T_G。

输出:数据图 G 上编码点集 V^*(即在点集 V 的基础上对 V 中每个数据点 v 均补充上了其数据点区间编码信息)。

1) 对分解树 T_G 进行先序遍历,记录每个树上节点的先序遍历序号及每个树上节点的后代节点数量;

2) foreach 分解树上节点 X do

3)　　$X.start$ = 该树上节点的先序遍历序号;

4)　　$X.end = X.start$ + 该树上节点的后代节点数;

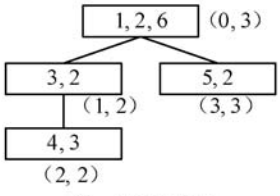
5) foreach 不在树上根节点 X_R 中的数据点 v do

6)　　将 v 的区间编码设置为其节点子树的根节点对应的树编码;

7) foreach 在树上根节点 X_R 中的数据点 u do

8)　　将 u 的开始区间编码设置为 0;

9)　　将 u 的结束区间编码设置为其节点子树中除了根节点(此处所说的根节点既是其节点子树的根节点同时也是整棵分解树的根节点)以外的其他节点的树编码中的最大结束区间编码,需要注意的是若 u 的节点子树只包含根节点一个节点,那么其结束区间编码设置为 0。



(a) 分解树编码

1	(0,0)
2	(0,3)
3	(1,2)
4	(2,2)
5	(3,3)
6	(0,0)

(b) 数据点编码结果

图 2　区间编码过程示例

首先,需要对分解树上的每个树上节点进行区间编码。具体编码方法为,将分解树上每个节点的开始节点编码设置为其在该分解树上的先序遍历序号,其

结束节点编码设置为该节点的开始节点编码加上该节点的后代节点数量(行 1 - 行 4)。

下一步根据树编码来对数据图上的每个数据点进行区间编码。对于所有未在分解树的根节点中出现的数据点,将其区间编码设置为其节点子树的根节点对应的树编码(行 5 - 行 6)。而对于所有出现在树上根节点中的数据点,则需要特殊考虑,首先将它们的开始区间编码均设置为 0,那么无论其结束区间编码为何值,这些根节点中的数据点的区间编码均会互相包含(行 8)。下一步还需满足对于根节点中的数据点,它们的区间编码需要包含其节点子树中所有节点的区间编码,因此,将这些在根节点中出现的数据点的结束区间编码设置为其节点子树中除了根节点以外的其他节点的树编码中的最大结束区间编码。同时需要注意的是,若某数据点出现在根节点中,但同时它又只出现在根节点中,那么将其结束区间编码也设置为 0,因为该数据点的区间不需要包含任何不在根节点中的数据点的区间(行 9)。

图 2 为一个完整的区间编码过程示例,首先对于分解树进行编码,举例来说,对于包含{3,2}的该树上节点,其先序遍历序号为 1,有一个后代节点,因此其树编码为(1,2)。下一步对每个数据点进行区间编码,对于不在根节点上的数据点,如数据点 5,其区间编码设定为对应节点子树根节点的树编码,即(3,3);对于在根节点上出现的数据点,若该数据点只出现在根节点中,如数据点 1,将其区间编码设置为(0,0);若该数据点在分解树上不止出现一次,则将其开始区间编码设置为 0,结束区间编码设置为其节点子树中除根节点外的其他节点中的最大结束编码,如数据点 2,其在分解树中除根节点外还出现在其他节点中,其中最大的结束编码出现在包含{5,2}的该树上节点的树编码(3,3),因此将其区间编码设置为(0,3)。

基于以上区间编码方法,提出引理 2。

引理 2 对于一幅图 $G=(V,E)$ 及在该图上的一个树分解 $T_G=(\{X_i \mid i\in I\},T)$,$v$ 与 u 为 V 中的两个点,若$(v,u)\in E$,则 v 与 u 的区间编码必定存在包含关系。

引理 2 的证明如下。

根据引理 1,可知数据点 v 与 u 的节点子树的根节点必定存在祖先-后代关系。若 v 与 u 的节点子树的根节点为同一个树上节点(在本文的分解树中尽可能是同为分解树的根节点),则 v 与 u 的区间编码开始编码均为 0,结束编码无论为何,v 与 u 的区间编码均会存在包含关系;若 v 与 u 的节点子树的根节点为不同节点,v 的节点子树根节点为 u 的节点子树根节点祖先节点时,根据算法 2,可知 v 的区间编码必定包含 u 的区间编码;u 的节点子树根节点为 v 的节点子树根节点祖先节点时同理可以证明。引理 2 证毕。

完成区间编码后,每个图上节点均有了一个对应的区间编码,且对于互为邻居的两个图上节点,其区间编码必存在包含关系。此性质可以被使用在许多大型数据图上的应用中,如知识图谱查询、基于知识库的智能问答等应用。

3 实 验

3.1 实验设置

在智能问答应用中,基于知识图谱的实体歧义消除得到了广泛应用[10-11]。为验证本文方法的有效性,本文使用真实知识图谱数据集 YAGO[4],并人工构建了在其上适用的 100 句智能问答问题。YAGO 数据集是一个大型知识库,本文中使用的是其第一版的数据,共包含 1 281 万数据点及连接它们的 1 901 万条边。

实验对问句中可能出现歧义的实体利用区间编码进行歧义消除,并计算消歧率。具体消歧方法为:利用问句中出现的其他确定实体对应的数据点的区间编码去与可能出现歧义的实体的候选数据点的区间编码进行比对,删除掉肯定不符合条件(即与确定数据点的区间编码不存在区间包含关系)的数据点,从而进行一部分的消歧工作。需要注意的是,采用本文提出的区间编码方法,并不保证能够完全将歧义消除,但本文方法因其快速比较区间编码的能力,比在自然语言处理中使用的歧义消除方法效率要高得多,因此可以帮助消歧工作提高效率。

为比较本文方法的歧义消除能力,在实验中我们设计了另一组基线方法,在该方法中,随机为所有图上数据点赋予一个序号值,对于某个数据点,其区间编码设置为其邻居数据点的最小序号值至最大序号值的范围区间。判断两个数据点是否可能互为邻居的方法则是分别观察这两个数据点的序号是否落在另一个数据点的区间编码范围内。

3.2 实验结果

在实验中,统计了包括歧义消除率与有效消除问题数等多种体现区间编码方法表现的数据。对于单个问题,其消歧率定义为消除掉的歧义数据点与所有歧义数据点的比值,若对于一个问题,所有歧义干扰项均

被消除,只剩下一个对应的数据点,则称所用方法在此问题上完成完美消歧;若对于一个问题,能够消除一部分歧义数据点,则称所用方法在此问题上形成部分消歧;所有部分消歧与完美消歧合称为有效消歧。

表 1 展示了歧义消除率与歧义消除问题数的实验结果。其中,总消歧率表示的是对于所有 100 个问题的平均消歧率(包括完美消歧问题、无效消歧问题、部分消歧问题);有效消歧率表示的是对于有效消歧的问题的消歧率。可以看出,本文方法表现远超基线方法且有着不错的消歧率。同时,从表中歧义消除问题数的数据可以看出无论是对于完美消歧问题数还是有效消歧问题数,本文方法的表现都远超基线方法,且对于超过 60% 的问题均能形成有效消歧。

表 1　歧义消除实验结果

方法	总消歧率/%	有效消歧率/%	完美消歧数	有效消歧数
本文	43.30	69.9	8	62
基线	9.79	23.3	0	42

综合上述数据,可以看出本文方法表现远高于基线方法,且在实际的歧义消除应用中有着不错的表现。通过本文中的实验,初步证明了本文方法能够在基于大型数据图的实际应用中发挥提高效率的作用。

3.3　消歧应用

下面用一个示例来说明本文方法是如何帮助进行消歧的。

以下问句是一个需要进行实体消歧的问句:

"复旦大学在浦东新区的那个校区的面积是多大?"

对于这个问句,简单的实体识别技术会从句子中提取出"复旦大学"这一实体,但是复旦大学存在着多个校区,即对应着多个歧义项。从句中可以得知需要问的这个实体在数据图上是与"浦东新区"该点互为邻居,因此,此处可以使用"浦东新区"该实体的区间编码与"复旦大学"各个校区所代表的实体的区间编码利用引理 2 进行快速比对,从而可以帮助消歧。需要注意的是,这种消歧方法大多时候并不保证能够完成消歧,可能仍需要使用传统自然语言的消歧方法对剩余歧义项进行消除,但是使用本文方法消除每一个歧义项的效率远高于传统消歧方法,因此只要能够形成有效消歧,应用本文提出的区间编码就能加速歧义消除的过程。

4　结　语

本文提出一种基于树分解的图上节点区间编码方法,利用此方法为图上每个点添加表示位置特征的区间编码属性,从而提供了将许多在大型图上的应用问题转化到一定范围的局部内进行解决的可能。本文在真实数据集上进行了智能问答中实体消歧的实验,通过实验结果验证了本文方法的有效性与实用性。当然本文旨在提出此区间编码的方法,抛砖引玉,并没有深入探讨其在各种应用场景下的使用前景与方法,希望未来能够有更多的工作一起探索。

参　考　文　献

[1] Fabian M S, Gjergji K, Gerhard W. YAGO: A core of semantic knowledge[C]//16th International Conference on World Wide Web,2007:697 - 706.

[2] Kurt D B, Robert P C, Patrick T. Freebase: A shared database of structured general human knowledge[C]//22nd AAAI Conference on Artificial Intelligence, 2007: 1962 - 1963.

[3] Wu W, Li H, Wang H, et al. Probase: A probabilistic taxonomy for text understanding[C]//2012 ACM SIGMOD International Conference on Management of Data,2012: 481 - 492.

[4] Wei F. TEDI: Efficient shortest path query answering on graphs[C]//2010 ACM SIGMOD International Conference on Management of Data,2010:99 - 110.

[5] Robertson N, Seymour P D. Graph minors. III. Planar tree-width[J]. Journal of Combinatorial Theory, Series B,1984, 36(1):49 - 64.

[6] Dietz P F. Maintaining order in a linked list[C]//14th Annual ACM Symposium on Theory of Computing,1982: 122 - 127.

[7] 孟小峰. XML 数据管理[M]. 北京:清华大学出版社, 2009.

[8] 张换香,张晓琳,刘立新,等. 基于 Map-Reduce 的 XML 区间编码方案[J]. 计算机应用与软件,2015,32(12):257 - 260.

[9] Zhang C, Naughton J, DeWitt D, et al. On supporting containment queries in relational database management systems [C]//ACM SIGMOD International Conference on Management of Data,2001:425 - 436.

[10] Wang F, Wu W, Li Z, et al. Named entity disambiguation for questions in community question answering[J]. Knowledge-Based Systems,2017,126:68 - 77.

[11] Kandasamy S,Cherukuri A K. Query expansion using named entity disambiguation for a question-answering system[J]. Concurrency and Computation: Practice and Experience, 2020,32(4):e5119.