计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.200900114

基于邻域结构的时态 RDF 模型及索引方法

陈圆圆　严　丽　章哲庆　马宗民
南京航空航天大学计算机科学与技术学院/人工智能学院　南京 211106
(chenyuanyuan@nuaa.edu.cn)

摘　要　资源描述框架(Resource Description Framework,RDF)是 W3C 推荐的一种元数据模型和信息描述规范,已被广泛地应用于各个领域。为了跟踪 RDF 数据随时间的变化,将时态信息引入 RDF 的框架中,随着时态 RDF 数据的快速增长,对时态 RDF 数据的有效管理变得十分必要,构建合理的索引机制能够实现对数据的高效存储和查询。文中提出了一种时态 RDF 数据模型,给出了具体的一维编码方案,实现了简单地表示时态信息,并以较低的开销扩展现有的 RDF 数据模型。在此基础上,提出了基于邻域的二级索引结构。首先利用动态计数过滤器的方法索引的邻域信息,然后利用 B+树索引每个结点相关的全部时态 RDF 数据,同时,可对大规模时态 RDF 数据进行更新。实验结果表明,所提方法相比对比方法在大多数情况下性能提高了 35%左右,具有可扩展性和有效性。

关键词:RDF;时态 RDF;编码;索引结构;动态计数过滤器
中图法分类号　TP399

Temporal RDF Model and Index Method Based on Neighborhood Structure

CHEN Yuan-yuan,YAN Li,ZHANG Zhe-qing and MA Zong-min
College of Computer Science and Technology/College of Artificial Intelligence,Nanjing University of Aeronautics and Astronautics,Nanjing 211106,China

Abstract　Resource description framework (RDF) is a metadata model and information description specification recommended by W3C,which is widely used in various fields. To track changes in RDF data over time,temporal information is introduced into the RDF framework. With the rapid growth of temporal RDF data,effective management of temporal RDF data is necessary. A reasonable index mechanism can achieve efficient storage and query of data. In this paper,we first present a temporal RDF data model. We propose a specific one-dimensional coding scheme,which represent temporal data simply and extend the existing RDF data model with lower overhead. Furthermore,we present its two levels of indexes based on neighborhood structure. The first one uses dynamic counting filter to index the neighborhood information of the node,and the second builds the B+ tree to index the temporal RDF data related to each node. Moreover,large-scale temporal RDF data can be updated. Experimental results show that the proposed method is around 35% better than the comparison method in most cases,and it is scalable and effective.

Keywords　RDF,Temporal RDF,Encoding,Index structure,Dynamic counting filter

1　引言

随着语义 Web[1]和知识工程技术的快速发展,资源描述框架(RDF)[2]作为数据表示的主要形式,具有语义准确且表达灵活的优点,现已在很多领域得到广泛应用。例如,维基百科数据库 DBpedia[3]、生物信息学的药物数据库 DrugBank[4]、全球地理数据库 GeoNames[5]及政府公开数据库 Data.gov[6]等都以 RDF 的形式存储数据。截至 2012 年,链接开放数据[7](LOD)中包含了近 300 个数据集,数据量达到了 320 亿;截至 2020 年 5 月,其已包含 1 260 个数据集及 16 187 个链接。随着 RDF 数据量的快速增长,对大规模 RDF 数据的有效管理变得十分必要。构建合理的索引机制是提升 RDF 查询处理性能的一种有效方法[8-13]。

很多应用领域(如地理信息系统、动态社交网络、环境气象监测系统和城市交通管理系统等)包含大量的时态信息,其中时态数据仅在指定的时间内有效。例如,在环境气象监测系统中表示雾霾出现的时间区间、在城市交通管理系统中表示道路阻塞发生的时间点等。为了有效管理时态数据,不同类型的时态数据模型已经被开发出来。时态关系数据库是最早被使用的时态数据模型[14],现已成为 SQL 2011 中推荐的标准[15],它开启了时态数据处理的规范形式。此外,很多研究集中在一些特定的领域,如开发高效、健壮的时态数据库模型和语言[16]、语义注释信息中的不精确问题和比例问题[17],以及时态数据库的索引技术[18]。

为了使 RDF 能够表示和处理时间语义,有必要对静态 RDF 数据模型进行时态扩展。文献[19-20]最早提出了时态

到稿日期:2020-09-14　返修日期:2020-12-31　本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
通信作者:马宗民(zongminma@nuaa.edu.cn)

RDF 模型，通过定义一组时间谓词和对应的语法语义来表示时态信息。此外，文献[21]提出了多维时态 RDF 模型，文献[22]提出了包含更新次数的时态 RDF 模型，文献[23]对时态 RDF 的推理问题进行了研究。为了有效管理时态 RDF 数据，文献[24-25]针对时态 RDF 数据提出了基于树型结构的索引，文献[26]提出了基于路径结构的索引结构，文献[27]提出了以时间信息为主要索引项的双层索引结构。

应当指出的是，现有的时态 RDF 模型把时态信息处理成 RDF 三元组的标注信息，两者以直接关联的形式进行存储。时态信息通常具有两种表现形式，即时间点和时间区间，因此，为了更直观地表示时间区间，一般采用诸如坐标的形式 (a,b) 直接地存储区间内的时态信息，然而这样的表达形式通常会带来较多的额外开销。此外，大规模时态 RDF 数据的有效查询依赖于索引结构，时态信息本身也对数据更新提出了要求。基于这些方面的考虑，本文提出了一种新的时态 RDF 数据模型，该模型能够将不同形式的时态信息映射成长度固定的整型编码。在此模型的基础上，本文提出了一种基于邻域结构的索引，可有效支持时态 RDF 数据的查询及数据更新。本文的主要贡献如下：

(1)提出了一种时态 RDF 数据模型。该模型通过一维编码的方式压缩存储时态信息，并将其表示成固定长度的整数，以较低的开销对现有的 RDF 数据进行时态扩展。

(2)提出了一种基于邻域边标签的二级索引结构。该结构使用动态技术过滤器的方法对结点的邻域边标签集进行全局索引，使用 B+树结构对时态 RDF 数据进行局部索引。该索引结构具有较高的剪枝能力，能实现高效查询，同时能够轻松地实现数据的更新操作。

(3)基于以上方案，本文提出了两种不同的查询计划，并将本文方案与其他实验方案进行了对比，给出了有效性和可扩展性分析。

本文第 2 节介绍了相关工作；第 3 节对时态 RDF 数据模型及具体的编码方案进行了阐述；第 4 节描述了索引结构，并给出了索引构建过程和更新过程的算法实现；第 5 节阐述了模型的整体架构及查询实现过程；第 6 节进行了实验分析；最后总结全文。

2 相关工作

2.1 RDF 数据

对于经典 RDF 数据的表示，通常采用两种不同的视角：基于基本三元组的方法和基于 RDF 图结构的方法。早期如 Jena[8]，RDF-3X[9]，BitMap[10] 等是基于基本三元组的 RDF 管理系统。Jena[8] 使用属性表的方法存储 RDF 三元组数据，按照谓语划分数据，将具有相同主语的数据存储于数据表的同一行中；RDF-3X[9] 对数据进行压缩存储，把三元组中的主语、谓语和宾语，按照在三元组中的不同位置，进行 6 种可能的冗余索引；BitMap[10] 则利用垂直存储的思想，将数据存储于 3 个组合位图中，其中以主-谓、谓-宾、宾-主的联合值为键，将对应的宾语、主语、谓语作为值存储。基于基本三元组的索引方法通常使用关系模型存储数据，并通过关系操作符来处理 RDQL(RDF Data Query Language)或 SPARQL 查询。这种方法的优点是能够较为简单地组织和存储 RDF 数据，可借助相对成熟的索引机制实现查询；缺点是在查询处理过程中会造成过多的连接操作或产生大量的冗余数据。

基于 RDF 图结构的常见方法包含基于路径索引的 BRAHMS 方法[11]、基于树型索引的 GRIN 方法[12]、基于图模式索引的 RG-index 方法[13] 等。BRAHMS 方法根据 RQL 中的查询表达式进行路径匹配；基于树型索引的 GRIN 方法首先对 RDF 图进行聚类，然后根据聚类中心和聚类半径构造平衡二叉树，最后采用图模式匹配的方法进行查询；RG-index 方法提取 RDF 图中的高频图模式并构建索引，利用关系操作符进一步地过滤无关数据，实现了高效剪枝。基于 RDF 图结构的数据管理方案既存储了 RDF 数据本身又维护了其部分结构信息，因此该方法在查询过程中具有较强的剪枝能力，但是匹配子结构的时间通常较长，索引规模往往较大。

2.2 时态 RDF 数据

现有的时态 RDF 数据模型通常考虑两个维度的时态信息[19]，一种是事务时间，即数据实际存储于模型中的时间；另一种是有效时间，即标记 RDF 三元组在模型世界中有效的时间数据。一般是通过版本控制的方法来捕捉事务时间。由于创建一个新的版本往往会造成大量的时间开销和空间开销，因此仅有少数方法[19-21] 提出了不同数据模型来维护 RDF 数据的版本快照。近年来，大多数方法则是从有效时间这个维度来对 RDF 数据模型进行扩展。

根据时态 RDF 数据的组织和存储策略，通常从两个方面展开对时态 RDF 数据的研究：1)通过一组时间谓词来扩展 RDF 词汇表，实现对时态 RDF 数据的存储和查询；2)直接对三元组的谓词部分或三元组本身添加时间戳，用于表示和管理时态 RDF 数据。对于第一种类型，文献[19-20]最早提出了时态 RDF 模型，通过使用 RDF 词汇表和时态标签来表示时态 RDF 的语法与语义，实现了时态 RDF 数据的表示和查询。文献[28]则进一步提出了时间事实的概念，并提出新的时态、RDF 图模式，为 YAGO 数据集扩展时间信息，同时符合已有的 RDF 抽象模型和语义。文献[21-22]对特殊时态 RDF 模型进行了研究，文献[21]构建了一种多时态 RDF 数据模型，可对至少 3 个独立的时间信息进行处理，用于管理本体的时态信息；文献[22]提出了关于更新次数的时态 RDF 模型，并给出了对应的查询语言 SPARQLt。

对于第二种类型，文献[29]使用四元组 (s,p,o,l) 的形式表示时态 RDF 数据模型 stRDF，利用时间相关的数学公式表示有效时间，并提出了相应的查询语言 stSPARQL。文献[23]给出了时态 RDF 数据在确定推理规则下的查询及评估算法。为了表示更多的注释信息，文献[30]描述了语义 Web 数据表示和推理的通用框架，能够表示时间、真实性或出处等多种注释信息，并定义了一种扩展的 SPARQL 查询语言。

为了提高数据的查询效率，文献[24-27]通过构建索引的方式存储和查询时态 RDF 数据。文献[24]提出了一种树型索引结构 tGrin，将数据物理地存储于关系数据库中，树中根结点映射数据库中的全部数据，并对此给出了一种归一化算法。同样地，文献[25]提出了一种基于命名图的树型索引结构 KeyTree，使用压缩算法构建区间索引结构，能够适应稀疏

型和均匀型的时间间隔，以实现对时间信息的高效查询。文献[27]则提出了一种双层索引结构，利用 K-D 树对二维的时间信息进行全局索引，然后使用组合位图索引存储 RDF 三元组的局部信息。

现有的时态 RDF 数据的管理方法通常需要较多的额外开销来对时态信息进行处理。文献[31]的研究表明，将附加信息编码成整数值，能够使查询得到有效执行，且不会对原始查询方案造成额外的开销。受文献[31-32]的启发，本文提出了一种一维的编码方案，能够对有效时间进行编码，标识两种不同粒度的时间信息，即时间点和时间区间。

3 时态 RDF 数据模型

本节描述了编码后的时态 RDF 数据模型及具体的编码方案，并给出了一些概念层面上的定义。

定义 1（RDF 图） 令 U 表示统一资源标识符（记作 URI），B 表示空白节点，L 表示字面量，则 RDF 三元组可表示为 $(subject, predicate, object) \in (U \cup B) \times U \times (U \cup B \cup L)$。RDF 图即 RDF 三元组的集合，记为 G。

图 1 给出了一个 RDF 数据图示例。其中，三元组的主语（subject）和宾语（object）表示 RDF 图中的结点，谓语（predicate）表示主语指向宾语的有向边。

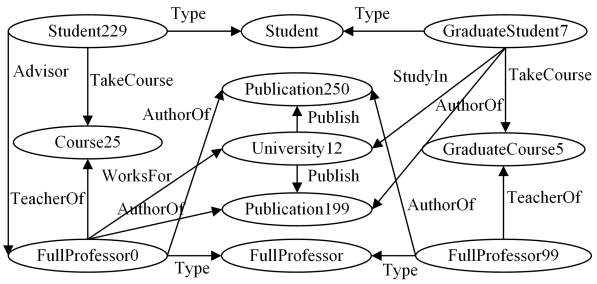


图 1 RDF 图

Fig. 1 RDF graphs

定义 2（SPARQL 查询） 基本 SPARQL 查询 Q 可表示为 *select* ?x *where*{s p o.}。其中，*select* 语句中"?x"表示查询变量 V；*where* 语句中的"s p o"表示一个三元组模式 tp，且有 $s \in (V \cup U \cup B)$，$p \in (V \cup U)$，$o \in (V \cup U \cup B \cup L)$。记查询 $Q=(tp_1, tp_2, \cdots, tp_n)$。同时，*where* 中全部的三元组模式也称作基本图模式（BGP）。

例如，对图 1 所示的 RDF 图执行 SPARQL 查询"是著作 199 的作者且是在大学 12 学习的学生"：

```
select ?x
where {
        ?x rdf:type Student.
        ?x StudyIn University12.
        ?x AuthorOf Publication199.}
```

3.1 时态 RDF 模型表示

遵循文献[19]的建议，本文将时间看作一个离散、线性有序的域，并用简化、抽象的整数形式表示时间点。对于时态数据来说，基本上存在两种不同粒度的时态域：基于时间点的和基于时间区间的。时间区间的边界值由两个有效时间点组成，即开始时间（t_s）和结束时间（t_e）。这里使用基于 N-Quads[33] 的建议语法表示时态 RDF 数据。数据结构分为 4 部分：〈主语〉〈谓语〉〈宾语〉〈上下文〉。其中，〈主语〉〈谓语〉〈宾语〉表示一个 RDF 三元组（如定义 1 所示），元素〈上下文〉用来注释时态数据。

如图 2 所示，初始化时，时态数据有两种表示形式：1）$(s,p,o):[t]$，其中 t 为自然数，表示在时间点 t 时，数据有效；2）$(s,p,o):[t_s,t_e]$，其中 $t_s \leqslant t_e$，$[t_s,t_e]$ 表示从起始时间 t_s 到结束时间 t_e 内，数据有效。其中，(s,p,o) 为标准三元组。本文提出了一种时态 RDF 数据模型，用一个整型的一维编码来注释 RDF 数据。具体语法如定义 3 所示。

```
〈Student229〉〈Type〉〈Student〉
〈Student229〉〈TakeCourse〉〈Course25〉:[2002]
〈Student229〉〈Advisor〉〈FullProfessor0〉:[2000,2004]
〈FullProfessor0〉〈TeacherOf〉〈Course25〉:[2002,2012]
〈FullProfessor0〉〈WorksFor〉〈University12〉:[1996,2020]
〈FullProfessor0〉〈Type〉〈FullProfessor〉
〈FullProfessor0〉〈AuthorOf〉〈Publication199〉
〈University12〉〈Publish〉〈Publication250〉:[2010]
〈University12〉〈Publish〉〈Publication199〉:[2018]
〈GraduateStudent7〉〈Type〉〈Student〉
〈GraduateStudent7〉〈StudyIn〉〈University12〉:[2015,2018]
〈GraduateStudent7〉〈AuthorOf〉〈Publication199〉
〈GraduateStudent7〉〈TakeCourse〉〈GraduateCourse5〉:[2015]
```

图 2 初始时态 RDF 数据表示

Fig. 2 Initial temporal RDF data representation

定义 3（基于编码的时态 RDF 模型） 时态 RDF 模型由三元组 (s, p, o) 与第四个组件时间编码组成，可表示为四元组 (s,p,o,tid)，其中，tid 表示时间编码，且 $tid \in N$。

为了更好地解释这个模型，图 3 给出了图 2 所示的模型示例。

```
〈Student229〉〈Type〉〈Student〉〈0〉
〈Student229〉〈TakeCourse〉〈Course25〉〈130〉
〈Student229〉〈TeacherOf〉〈FullProfessor0〉〈551〉
〈FullProfessor0〉〈TeacherOf〉〈Course25〉〈645〉
〈FullProfessor0〉〈WorksFor〉〈University12〉〈139269〉
〈FullProfessor0〉〈Type〉〈FullProfessor〉〈0〉
〈FullProfessor0〉〈AuthorOf〉〈Publication199〉〈0〉
〈University12〉〈Publish〉〈Publication250〉〈128〉
〈University12〉〈Publish〉〈Publication199〉〈32768〉
〈GraduateStudent7〉〈Type〉〈Student〉〈0〉
〈GraduateStudent7〉〈StudyIn〉〈University12〉〈36865〉
〈GraduateStudent7〉〈AuthorOf〉〈Publication199〉〈0〉
〈GraduateStudent7〉〈TakeCourse〉〈GraduateCourse5〉〈4096〉
(Take 2006 as the initial time,1 year as the unit code)
```

图 3 时态 RDF 模型示例

Fig. 3 Temporal RDF model

图 3 给出了学校、老师、学生和课程的相关信息，以及有效时间的编码表示。结合图 2 可知，(Student229, TakeCourse, Course25):[2002] 表示在 2002 年学生 229 完成课程 25；模型对应编码为 130。(FullProfessor0, WorksFor, University12):[1996,2020] 表示从 1996 年到 2020 年间，教授 0 工作于大学 12 中；模型对应编码为 139269。(GraduateStudent7, Type, Student) 表示研究生 7 一直是学生；模型对应

编码为 0。更具体的编码过程见 3.2 节。

3.2 时态数据编码

本节给出具体的时态 RDF 数据的编码规则。首先，为了清晰起见，将时态信息本身映射成一个整数 t，表示以某个初始时间为起点的第 t 个时间单位，将该初始时间之前的时间信息映射为负整数，初始时间及之后的时间信息映射为正整数。

例如，以 2006 年为初始时间，1 年为时间单位，则 2016 年映射为整数 $t=10$，2000 年映射为整数 $t=-6$。

其次，对时态信息进行编码。考虑它由 k 位的二进制位组成，初始化时，每一位设为 0。其中，第 0 位为类型标志位，标识当前有效时间是时间点还是时间区间，若为时间点，则设为 0；若为时间区间，则设为 1。第 1 位和第 2 位为联合标志位，标识当前时态信息是负数（初始时间之前）还是正数（初始时间及之后），具体映射规则如表 1 所列。为了表明时间区间中 t_s 和 t_e 是初始时间之前还是初始时间之后的时间信息，用两个二进制位表示可能的 4 种情况。对于有效时间为时间点 t 时，用 00 表示初始时间之前的时间点，01 表示初始时间之后的时间点。其余二进制位则用来具体地标识有效时间。时间点[t]有效时，将从低位到高位的第 t 位设为 1；时间区间[t_s, t_e]有效时，将二进制位上的第 t_s 位和 t_e 位均设为 1，最后得到的二进制值即为当前 RDF 实体的时态编码。其中，无时态信息的 RDF 实体编码为 0。

表 1 联合标志位含义

Table 1 Meaning of flag bits

t	t_s	t_e	The second bit	The first bit
+	+	+	0	0
−	−	+	0	1
−	−	+	1	0
−	−	−	1	1

如基于图 3，时间点[−4]标识为 10 000，标志位为 010，故编码为 $2^7+2=130$；时间间隔[−10, 14]标识为 0100 0100 0000 0000，标志位为 101，故编码为 $2^{17}+2^{13}+5=1\,394\,269$；当编码为 645 时，由 $645=2^9+2^7+5$ 可知该 RDF 实体的时态信息为时间间隔[−4, 6]。

由于二进制编码具有唯一性，因此能够实现编码与时态 RDF 实体的双向映射。在大多数场景下，该方案可对初始时间和 k 的大小进行不同的设置，以满足时态数据的编码需求。然而，少数对于时间粒度要求极低的场景，亦可类似地使用该编码方案，得到时态信息相近的时间集合，结合后文提出的邻域索引方案，进一步实现高效查询。

4 索引方法

为了加快时态 RDF 数据的查询效率，结合前文提出的基于编码的时态 RDF 数据模型，本节提出了一个索引方案，并给出了索引实现和索引维护的算法。

定义 4（时态 RDF 图） 时态 RDF 图表示为 RDF 三元组与时间编码 tid 的集合，即时态 RDF 四元组的集合。时态 RDF 图的结点为对应 RDF 图的结点；时态 RDF 图的边为对应 RDF 图的边。

定义 5（d-邻域子图） 令 n 表示 RDF 图中的结点，则距离结点 n、路径长度为 d 的子图即为结点 n 的 d-邻域子图。

例如，图 1 所示的 RDF 图与图 3 所示的时态 RDF 模型中，结点 University12 对应的 1-邻域子图均为{(University12, Publish, Publication250), (University12, Publish, Publication199), (FullProfessor0, WorksFor, University12), (GraduateStudent7, StudyIn, University12)}。

定义 6（邻域边标签集） 令 n 表示时态 RDF 图中的结点，结点 n 的 1-邻域子图中边标签（谓词 P）的集合即为邻域边标签集，记为 $P(n)$。

例如，结点 University12 对应的邻域边标签集为{Publish, WorksFor, StudyIn}；结点 FullProfessor0 对应的邻域边标签集为{Advisor, TeacherOf, WorksFor, AuthorOf, Type}。

4.1 索引方案

4.1.1 索引结构

对经典 RDF 数据模型构建索引时，基于基本三元组的方法的实现过程相对简单，但对无关三元组的过滤性能较低，会产生过多的中间冗余结果。基于 RDF 图结构的方法可从图结构中提取邻域特征、路径特征、树特征或子图特征，并将其作为索引单元。其中，路径特征、树特征或子图特征作为索引单元，通常具有较佳的剪枝能力，但会导致索引量极大并且查询匹配时间较长。由于时态 RDF 图与相应 RDF 图的相关结构相同（如定义 4），结合两种方法的优缺点，本文选择基于 RDF 图的邻域结构作为索引单元。同时，TALE[34] 中的结果也表明，邻域结构具有较高的剪枝能力，索引大小与结点的数量呈线性关系，是个紧凑且过滤性能较高的索引单元。此时，邻域可认为是结点的 d-邻域子图（如定义 5）。

文献[34-35]通过对邻域结构的结点标签集进行索引，研究了大型图的近似查询匹配问题。一方面，本文需要得到时态 RDF 数据的精确查询结果而非近似匹配结果；另一方面，从目前常用的 RDF 数据集[36] 所提供的基准查询测试中可以发现，通常边为查询常量，而结点为查询变量。因此，本文对邻域结构的边标签集进行索引。根据查询结点的邻域边标签集，过滤数据集中不符合要求的结点，从而实现高效的剪枝。

如图 4 所示，该邻域索引为一个二级索引结构。第一部分是一个全局索引，时态 RDF 图中的全部结点维护一个列表，用于保存每个结点对应的邻域边标签集信息。第二部分构建了一个 B+树结构的局部索引，用于存储每个结点的全部时态 RDF 四元组。其中，B+树的叶子结点为时态编码 tid，且存储具有相同编码的 RDF 三元组。由于每个 RDF 实体具有唯一的时态编码，因此可通过时态编码定位查询数据，进一步剪枝无关数据，减少候选结果集的数量。

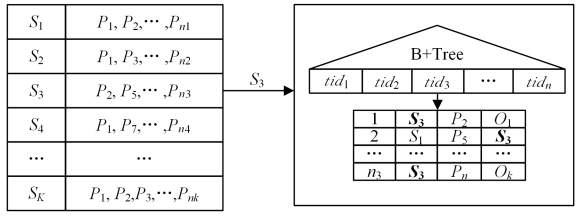


图 4 索引结构

Fig. 4 Index structure

4.1.2 动态计数过滤器

在全局索引结构中，若要对结点的邻域边标签集进行索引，一种简单的方式就是枚举出该结点所有的边标签并直接进行存储。一方面，索引的条目变长会导致较多的空间浪费；另一方面，这种暴力索引的方式会造成索引空间过大。假设时态 RDF 图的结点数为 N，则在最坏情况下，将造成 $O(N^2)$ 的空间成本。本文通过使用动态计数过滤器[37]的方法来压缩存储。

动态计数过滤器(Dynamic Counting Filter，DCF)是一种节省空间的概率性数据结构，可以对数据进行高效的插入、删除和查询操作，因此能够有效判断一个元素是否是集合的子集，并对集合中的元素进行更新操作。动态计数过滤器(见图 5)由向量 ***CBFV***，***OFV***，以及 k 个不同的哈希函数组成。其中，***CBFV*** 和 ***OFV*** 向量均由 m 个计数器组成，***CBFV*** 向量仅处理基本计数且每个计数器长度相同。通常，当长度 $x=4$ 时，即可满足常规需求；而 ***OFV*** 向量则处理溢出计数量，即总计数量与 ***CBFV*** 可处理计数量的差值。且 ***OFV*** 向量中每一项的长度 y 可根据溢出情况动态调整，值为 $\lfloor \log(\max(\text{溢出量}))+1 \rfloor$。假设第 i 个位置 ***CBFV*** 和 ***OFV*** 的值分别为 C_j 和 OF_j，则该位置的总计数值 $V_j=(2^x \times OF_j+C_j)$。

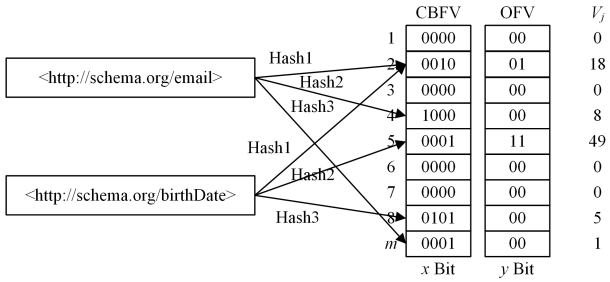


图 5 动态计数过滤器

Fig. 5 Dynamic count filter

初始化时，两个向量值均设为 0；当元素插入到集合时，则使用 k 个哈希函数计算出 k 个散列值，将 ***CBFV*** 向量中对应位置计数器的值增加 1；反之，当删除某个元素时，则将对应位置计数器中的值减少 1。当查询某个元素在集合中是否存在时，根据哈希后得到的 k 个位置，查看对应的 V_1，V_2，…，V_k 的值，若不全为 1，则认为该元素一定不存在于集合中；否则认为该元素可能存在于集合中。该问题即为动态计数过滤器的假阳性问题，表明 DCF 方法可能带来元素的误报问题，但是这个错误率可通过 m，k 及插入元素数目 n 进行控制。由文献[37]可知，当 k，m 的值满足式(1)、式(2)，并假设 n 的值为 RDF 图结点的平均度数 d，此时错误率 p 收敛为0.1%。

$$k=\frac{m}{n}\ln 2$$

$$m=-\frac{n\ln P}{(\ln 2)^2}$$

动态计数过滤器的空间复杂度为 $O(N)$，查询时间复杂度仅为 $O(1)$。基于动态计数过滤器的索引方法不仅可实现对邻域边标签集的轻量存储及高效查询，同时该方法允许我们简单地对数据进行更新。

4.2 索引实现

算法 1 给出了时态 RDF 数据的索引构建过程。首先遍历时态 RDF 数据集中每个 RDF 三元组(见算法 1 中的第 1 行)。同时，捕获三元组中的主语结点和 URI 类型的宾语结点，并为每个结点保存邻域边标签集，且将对应的时态三元组数据集存储于临时列表 *nodeList* 中，而对于字面量类型的宾语结点并不进行额外操作(见算法 1 中的第 2—8 行)。然后，遍历临时列表 *nodeList*，为每个结点构建一个动态计数过滤器，索引邻域边标签集。同时，维护一个指针 $bf.tripleT$，用于映射对应时态三元组在数据库中的 id(见算法 1 中的第 10—16 行)。最后，将当前创建的邻域索引结构写入文件中，并将时态 RDF 四元组集有序地写入数据库中(见算法 1 中的第 17，18 行)。

算法 1 索引构建算法

输入：时态编码 RDF 数据集 S

输出：邻域索引和时态三元组是否成功写入文件和数据库

```
1. for tripleT←1 to S do /* 遍历数据集获得每个时态三元组 */
2. nodeString← getNode(tripleT)
3. predicate←getPredicate(tripleT)
4. node←CreateNode(nodeString)
5. node. setNodeId()
6. node. addNeighbor(predicate)
7. node. addTripleT(tripleT)
8. nodeList. add(node)
9. end
10. for node←nodeList. get(0) to nodeList. size() do/* 遍历所有结
    点，索引邻域边标签集，索引相关时态三元组 */
11. degreeOfNode←node. getNeighbor. size()
12. dcf←CreateDCFilter(degreeOfNode)
13. dcf. tripleT←node. getTripleT()
14. for predicate←node. getNeighbor(). get(0) to node. getNeighbor().
    size() do
15. dcf. add(predicate)
16. end
17. insertIntoDataBase(node. getTripleT())
18. writeIntoFile(dcf)
19. end
```

实际上，该索引的构建过程是将时态 RDF 数据中的边标签部分通过哈希函数映射到位向量中，同时将时态四元组数据本身存储在关系型数据库中。这一过程是简单健壮的，这允许我们能够轻松地对数据进行更新操作。

4.3 索引的维护

本文提出的索引方案可以灵活地对数据进行更新。在进行大量插入和删除操作时，对时态四元组均进行索引构建时的类似操作。算法 2 给出了时态 RDF 数据添加或数据删除的算法实现。首先，遍历需新增或删除的数据集，捕获主语的结点，宾语的结点及边标签(见算法 2 中的第 1—3 行)。然后，搜索结点在索引列表中是否存在，若存在，则将结点对应 DCF 中该边标签所映射计数器的值增加 1 或减少 1；并且新增或删除 DCF 指向的数据集中的数据(见算法 2 中的第 4—7

行);若不存在,则为新的结点构建 DCF 索引,或放弃删除(见算法 2 中的第 8—16 行)。最后,对数据库或索引文件进行相关数据写入或删除(见算法 2 中的第 17—19 行)。数据的修改操作通过先删除指定数据再插入修改后的数据来实现。

算法 2 数据的增加、删除或修改

输入:待更新时态编码 RDF 数据集 S

输出:是否成功更新数据

```
1. for tripleT←1 to S do
2. nodeString←getNode(tripleT)
3. predicate←getPredicate(tripleT)
4. if isExist(nodeString)
5. then dcf←node.getDCFilter()
6. dcf.updateTripleT(tripleT)
7. dcf.update(predicate)
8. else if addData is true
9. then node←CreateNode(nodeString)
10. node.setNodeId()
11. node.addNeighbor(predicate)
12. node.addTripleT(tripleT)
13. nodeList.add(node)
14. dcf←CreateDCFilter()
15. dcf.tripleT←node.getTripleT()
16. dcf.add(predicate)
17. updateDataBase(S)
18. updateFile(dcf)
19. end
```

文献[27]中,由于对 RDF 数据的时态信息构建 K-D 树全局索引,因此数据进行更新时若产生了一个新的时态域,则要对全局索引进行重构,这将造成极大的时间开销和空间开销。一方面,本文提出的索引方案,在索引更新过程中可对时态三元组进行独立的处理;另一方面,基于邻域的索引结构不会因为数据的增多、删除或修改而受到影响。因此,结构是易扩展且稳定的。

5 查询实现

5.1 整体架构

本文旨在提出一个基于编码的时态 RDF 数据模型和支持查询及更新的一个通用索引方案。该模型的整体架构(见图 6)分成 3 部分,第一部分是对数据进行预处理,第二部分是构建索引,第三部分是查询实现。首先,输入数据集为 nt(N-Triple)类型的文件。由于经典 RDF 模型中不包含时态信息,因此需要通过对数据集添加随机整数的方式来表示时态信息。然后,对新的时态 RDF 数据集按照 3.2 节中的规则进行编码处理。数据预处理后,通过动态计数过滤器对结点的邻域部分进行索引,并写入文件系统中进行存储,将每个结点对应的时态 RDF 四元组写入关系型数据库中,并对数据库表的时态编码列构建 B+树索引。最后,查询处理部分以类 SPARQL 查询的形式接受查询输入。实际的初始查询由一个经典 SPARQL 查询和时态信息约束集来定义。首先查询处理器需要对初始查询进行解析,然后根据提出的不同的查询计划执行查询。5.2 节将给出具体的查询计划及算法实现。

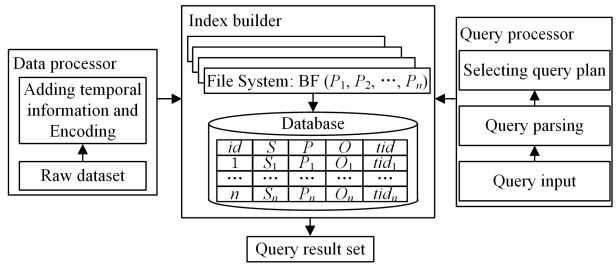


图 6 整体架构

Fig. 6 Overall structure

5.2 查询实现

在对时态 RDF 数据进行查询处理这一阶段中,查询输入是由经典 SPARQL 查询 Q 和时态约束集 T 组成的。处理时态查询包括两个部分:1)由 SPARQL 查询中的基本图模式(BGP)筛选 RDF 数据;2)由时态约束集 T 筛选时态数据。对两种约束执行查询的方式不同,可产生不同的查询计划,并导致不同的性能结果。对此,本文提出两种逻辑查询计划:第一种,交叉过滤查询(见图 7),分别对输入数据执行 *BGP* 约束查询和 T 约束查询,求得两种结果集的交集,得到最终的结果;第二种,二级过滤查询(见图 8),有序执行 *BGP* 约束查询和 T 约束查询,经过二级过滤后得到最终结果。

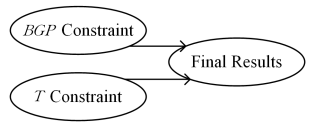


图 7 交叉过滤查询

Fig. 7 Cross-filtered query

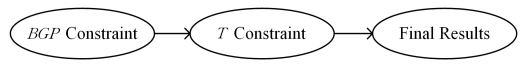


图 8 二级过滤查询

Fig. 8 Two-layer filter query

5.2.1 交叉过滤查询

算法 3 描述了交叉过滤查询计划。首先对查询输入进行解析,以获得三元组模式的集合 *BGP* 及时态信息约束集 T,同时对 T 中的数据进行编码(见算法 3 中的第 1—3 行)。然后,在数据库中分别对两种约束进行查询,得到对应三元组集合,对结果集取交集后,将最终结果连接排序后输出(见算法 3 中的第 4—10 行)。该算法实际并未用到邻域索引部分,因此将其用于对照评估无索引的时态 RDF 模型。

算法 3 交叉过滤查询算法

输入:时态查询输入

输出:查询结果

```
1. bgp←QueryParser.getBGp()
2. tList←QueryParser.getT()
3. tList.enCode()
4. for tp←1 to bgp do
5. resBgp←queryInDataBase(tp)
6. end
```

```
7. for tid←1 to tList. size() do
8. resT←queryInDataBase(tid)
9. end
10. return join(resBgp,resT)
```

5.2.2 二级过滤查询

算法 4 对二级过滤查询计划进行了描述。首先，对查询输入的解析处理分为两步：1）得到三元组模式与时态信息编码的映射列表；2）获取查询结点变量及对应的邻域边标签集（见算法 4 中的第 1－4 行）。然后，将邻域索引读入内存中，当且仅当查询结点的邻域标签集是数据集结点邻域边标签的子集时，将该结点加入候选结点集中（见算法 4 中的第 5－14 行）。最后，查询数据库，找到候选结果集中满足时态编码要求的三元组，将得到的结果进行连接排序后格式化输出（见算法 4 中的第 15，16 行）。接下来，对时间复杂度进行分析，假设时态 RDF 数据中的结点数为 N，结点平均度数为 D，时态编码数量为 M，则遍历邻域索引的复杂度为 $O(N*D)$；在 B＋树索引中，搜索时态信息的复杂度为 $O(\log M)$，因此总的复杂度可表示为 $O(N*D+\log M)$。该算法可对本文提出的索引模型进行综合评估。

算法 4 二级过滤查询算法

```
输入：邻域索引 dcfIndex，时态查询输入 queryString
输出：查询结果
1. bgp←QueryParser. getBGp()
2. ptList←QueryParser. getPT()
3. ptList. enCodeT()
4. valNeighbor←QueryParser. getValNeighbor()
5. for val←1 to valNeighbor. size() do
6. for dcf←1 to dcfIndex do
7.    flag←FALSE
8.    for Neighbor←1 to val. getNeighbor(). size() do
9.       flag←Dcf. search(Neighbor)
10.   end
11.   if flag is true
12.   then res. add(dcf. getTripleId)
13. end
14. end
15. res←queryInDatabase(res,ptList)
16. return postProcess(res)
```

6 实验分析

6.1 实验设置

为验证实验的有效性和可扩展性，我们将本文方法（记为 DCF 索引）与文献[27]提出的方法（记为 KDTree 索引）的性能进行了实验对比分析。其中，文献[27]提出的方法能同时有效处理时间点和时间区间信息，并且是目前对时态 RDF 数据进行索引的最好方法。两种方法都是用 JAVA 代码完成的，并在 JDK 1.8.0 中使用 Eclipse 4.7 和 MySQL 8.0 来实现。同时，在处理器 Intel i5-4210U 2.40 GHz、8 GB RAM 和 Windows 10 操作系统上执行。所有结果均为独立运行 5 次的结果的平均值。

实验采用标准合成数据集 WatDiv[36] 和从维基百科中撷取出的真实数据集 DBpedia[3]，这两种数据集可用来度量在具有不同结构特征的 SPARQL 查询中 RDF 数据管理系统的执行情况。为了评价时态 RDF 索引方案的有效性，可利用数据生成器获得 8 组不同规模的测试数据集，如表 2 中的 DS1－DS8。其中，DS1－DS4 是 WatDiv 测试数据集，DS5－DS8 是 DBpedia 测试数据集。此时，通过生成随机数的方式为三元组添加有效时间点、有效时间区间或不添加时态信息。

表 2 测试数据集及特征

Table 2 Dataset and characteristics

Dataset	Number of triples /10^5	Size of datasets/MB	Number of nodes
DS1	1	14.8	5597
DS2	2	29.0	11165
DS3	3	43.6	16378
DS4	5	70.0	26804
DS5	8	115.6	35020
DS6	10	139.7	41785
DS7	15	213.2	65635
DS8	20	275.2	86950

另外，我们定义 4 种不同类型的查询：1）直线型，由一个或多个连接的三元组模式组成的一条路径查询；2）星型，由两条以上的路径查询组成，并共享一个公共结点；3）雪花型，由两条以上的路径查询组成，但共享两个公共结点；4）复杂型，多条路径查询共享两个以上的公共结点。对此，我们构造 5 种不同查询（见表 3），其中，Q1－Q4 代表在 *BGP* 约束和 *T* 约束下查询，Q5 代表 *BGP* 约束下对时态数据的查询。

表 3 查询及特征

Table 3 Query and characteristics

Query	Type	Number of BGPs	Number of variables
Q1	Linear	2	3
Q2	Star	7	8
Q3	Snowflake	13	10
Q4	Complex	18	14
Q5	Star	7	8

6.2 索引性能分析

本节从索引大小和索引构造时间这两个方面来评估索引性能。图 9 给出了本文索引方案和对比方案在不同数据集上的索引大小。可以发现，与本文索引方案相比，对比方案额外占用约 120％的存储空间。图 10 则给出了两种方案在不同数据集上的索引构建时间。实验结果表明，对比方案的索引构建时间大约比本文索引方案少 23％。

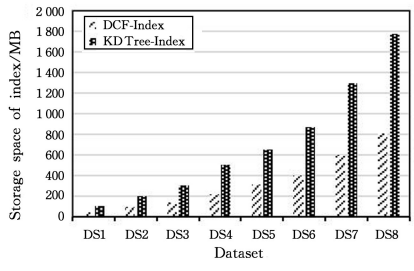


图 9 不同数据集上的索引大小

Fig. 9 Storage space of index on different datasets

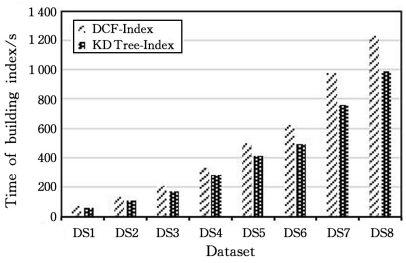


图 10 不同数据集上的索引构建时间

Fig. 10 Time of building index on different datasets

造成索引性能差异的主要原因是两种方案局部索引性能的差异。在索引构建过程中，由于全局索引并非真正地对数据进行存储，其构建过程时间短且占用空间小，因此可忽略这一部分对性能的影响。两种方案的局部索引都是向数据库中写入数据，其中对比方案的局部索引是对 RDF 数据进行位图索引，相当于原数据量的 3 倍存储。为了实现对时态数据的查询，还额外地存储了三元组的二维时态信息。本文方案的局部索引为原数据量的两倍存储，且编码后的有效时间仅为一维整数。另外，由于本文的局部索引构建过程中需要对时态编码列构建 B+树索引，经验证这个过程额外地造成了 47%左右的时间开销。当数据集中的时态信息较少时，此部分索引的构建收益较小；当数据集中时态信息跨度更大且数量更多时，局部索引构建过程中的时间开销是值得的。

从索引的整体性能来说，两种索引的大小和构建时间都与数据集的规模呈线性相关，这表明两种索引方案本身都具有可扩展性。

6.3 查询性能分析

本节从索引加载到内存中的时间和查询响应时间两个角度来评估查询性能。由于在进行时态 RDF 数据查询时，首先要将全局索引结构加载到内存中，这一部分时间通常较长，实际上一次加载完成后可实现多次查询，因此将索引加载到内存的时间区别于查询响应时间，并作为查询性能评估的一个尺度。图 11 给出了不同规模数据集上索引加载到内存中的时间，可以发现这部分时间基本与数据集的规模呈线性关系，相比对比方案，加载时间平均提高了 15%左右。

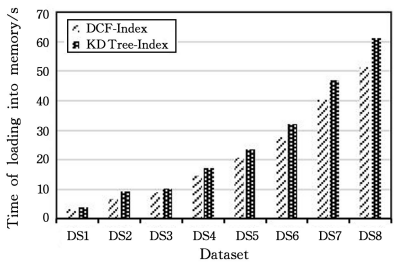


图 11 不同数据集上索引加载到内存的时间

Fig. 11 Load time of indexes on different data sets into memory

图 12 和图 13 分别给出了在数据集 DS3 上 5 种不同查询的查询响应时间和查询 Q2 在不同规模数据集上的查询响应时间。同时，本文对比了 5.2 节中提出的两种查询计划，即交叉过滤查询算法和二次过滤查询算法，并给出了对比实验方案。首先，本文对 DCF 索引-交叉过滤算法和 KDTree 算法进行评估。图 12 所示的实验结果表明，在路径查询 Q1 中，对比方案具有一定查询优势，而在较为复杂的 Q2 查询－Q5 查询中，交叉过滤算法的查询响应时间比对比方案平均少 35% 左右。这主要是由于在路径查询中每个结点的邻域集较小，过滤无关数据的性能较弱；而在较为复杂的查询中，本文索引方案具有较强的过滤能力，极大地减少了候选结果集的数量，因此查询时间显著缩短。其次，交叉过滤算法和对比方案都通过各自的索引进行高效查询，二次过滤查询计划由于仅对时态编码进行 B+树索引，此时可认为是无索引时的对照方案，实验结果表明其查询响应时间是有索引方案的数倍。

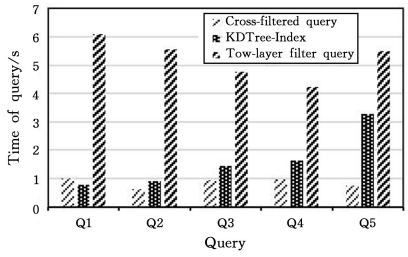


图 12 在数据集 DS3 上的查询响应时间

Fig. 12 Response time of queries on dataset DS3

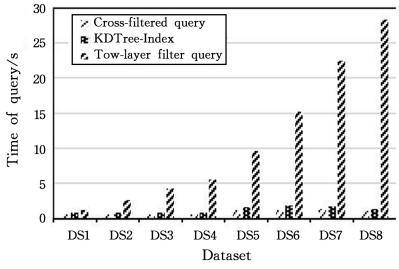


图 13 查询 Q2 在不同数据集上的查询响应时间

Fig. 13 Response time of Q2 on different datasets

上文表明，本文提出的索引方案具有有效性。由于 Q5 与 Q2 为同类型查询，是 *BGP* 约束下对时态数据的查询，此时交叉过滤查询算法仍然保持优良的性能，而 KDTree 索引方案的查询时间却大幅增加，这表明在无时态数据的经典 RDF 模型中，本索引方案仍然具有一定优势。

图 13 中的查询结果表明，随着数据规模的变大，二次过滤查询算法的查询响应时间呈正增长的趋势，而本文索引方案中的交叉过滤算法与对比方案中的查询响应时间均无明显变化，且本文索引方案的查询响应时间相比对比方案少 35% 左右。这表明本文索引方案具有较好的可扩展性和有效性。

另外，如图 14 所示，本节给出了两种方案在执行数据更新时的时间开销对比。

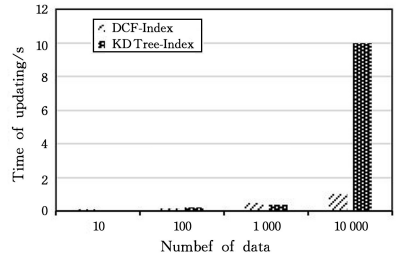


图 14 更新响应时间

Fig. 14 Time of updating

从图 14 可知，当更新数据量较小时，两种方案的更新效率相近。随着更新数据量变大，对比方案的效率明显下降。由于更新数据中包含全新的时态域信息，此时对比方案需要重构索引，因此造成了较大的时间开销。综上，进一步地验证了本文提出的索引结构在更新时具有稳定性。

上文在 WatDiv 和 DBpedia 测试基准上的实验结果表明，本文提出的邻域索引方案在整体性能上优于对比方案，具有有效性和可扩展性。

结束语 为了表示跨时间的 RDF 数据，对现有的 RDF 数据模型进行时态扩展十分必要，而构建合理的索引机制能有效地管理时态 RDF 数据。为了能简单轻量地对时态信息进行处理，本文提出了一种时态 RDF 数据模型，描述了一种一维编码方案，能够将有效时间点和有效时间区间映射成一个整型值。在此基础上，本文提出了一个基于邻域的二级索引结构，首先利用动态计数过滤器方法对结点的邻域边标签集进行索引，然后每个结点映射一个 B+树索引结构，存储包含该结点的所有时态四元组。实验结果显示，本文方法能够简单、有效地表达时态信息，且相比对比方案，在索引构建时能有效节省索引空间，查询性能也提高了 35%左右。需要指出的是，本文为了以较低的存储空间进行存储数据，采取的索引方案是基于 1-邻域子图的索引结构，因此对 RDF 的图特征提取仍存在优化空间。在下一步的工作中，我们希望能够进一步地提高索引的剪枝能力，尝试提取 RDF 图的路径特征或 K-邻域特征，结合时态编码方案，实现对时态 RDF 数据的高效存储和查询。

参 考 文 献

[1] BERNERS-LEE T，HENDLER J，LASSILA O. The semantic Web[J]. Scientific American，2001，284(5)：28-37.

[2] World Wide Web Consortium：RDF/XML Syntax Specification (Revised)[EB/OL]. [2020-05-30]. http://www.w3.org/TR/2004/REC-rdf-syntax-grammer.

[3] AUER S，BIZER C，KOBILAROV G，et al. DBpedia：A nucleus for a Web of open data[J]. The Semantic Web，2007，4825：722-735.

[4] WISHART D S，KNOX C，GUO A C，et al. DrugBank：a comprehensive resource for in silico drug discovery and exploration [J]. Nucleic Acids Research，2006，34：668-672.

[5] WICK M. GeoNames [EB/OL]. [2020-05-30]. https://www.geonames.org/.

[6] Technology Transformation Service. Data.gov [EB/OL]. [2020-05-30]. https://www.data.gov/.

[7] W3C SWEO Community Project. Linking open data on the semantic Web [EB/OL]. [2020-05-30]. https://lod-cloud.net/.

[8] CARROLL J，DICKINSON I，DOLLIN C，et al. Jena：Implementing the semantic Web recommendations[C]// Proceeding of the World Wide Web Conference on Alternate Track Papers and Posters. New York：ACM，2004：74-83.

[9] NEUMANN T，WEIKUM G. The RDF-3X engine for scalable management of RDF data[J]. The VLDB Journal，2010，19(1)：91-113.

[10] MADDURI K，WU K. Massive-scale RDF processing using compressed bitmap indexes[C]// LNCS 6809：Statistical and Scientific Database Management. Berlin：Springer，2011：470-479.

[11] JANIK M，KOCHUT K. BRAHMS：A WorkBench RDF store and high performance memory system for semantic association discovery[C]// Proceeding of the International Semantic Web Conference. Berlin：Springer，2005：431-445.

[12] UDREA O，PUGLIESE A，SUBRAHMANIAN V. Grin：A graph based RDF index[C]// Proceeding of the National Conference on Artificial Intelligence. Palo Alto：AAAI Press，2007：1465-1470.

[13] KIM K，MOON B，KIM H J. RG-index：An RDF graph index for efficient SPARQL query processing[J]. Expert Systems with Applications，2014，41(10)：4596-4607.

[14] JENSEN C. The consensus glossary of temporal database concepts—February 1998 version[C]// LNCS 1399：Temporal Databases：Research and Practice. Berlin：Springer，1997：367-405.

[15] KRISHNA G，MICHELS J. Temporal features in SQL：2011 [J]. SIGMOD Record，2012，41(3)：34-43.

[16] ZAIDI A K. A temporal programmer for time-sensitive modeling of discrete event systems[C]// Systems Man and Cybernetics. Piscataway，NJ：IEEE，2000：2186-2191.

[17] PEUQUET D. Making space for time：Issues in space-time data representation[J]. GeoInformatica，2001，5：11-32.

[18] NORVAG K，NYBO A O. DyST：Dynamic and Scalable Temporal Text Indexing[C]// International Symposium on Temporal Representation and Reasoning. Piscataway，NJ：IEEE，2006：204-211.

[19] CLAUDIO G，CARLOS A H，ALEJANDRO A. Temporal RDF [C]// LNCS 3532：The semantic Web：Research and Applications. Berlin：Springer，2005：93-107.

[20] GUTIERREZ C，HUIRADO C A，VAISMAN A. Introducing time into RDF[J]. IEEE Transactions on Knowledge and Data Engineering，2006，19(2)：207-218.

[21] GRANDI F. Multi-temporal RDF ontology versioning[J]. CEUR Workshop Proceedings，2009，519：1-10.

[22] ZHANG F，WANG K，LI Z Y，et al. Temporal data representation and querying based on RDF[J]. IEEE Access，2019，7：85000-85023.

[23] MOTIK B. Representing and querying validity time in RDF and OWL：A logic-based approach[J]. Journal of Web Semantics，2012，12：3-21.

[24] PUGLIESE A，UDREA O，SUBRAHMANIAN V S. Scaling RDF with time[C]// Proceeding of the 17th Int Conference on World Wide Web. New York：ACM，2008：605-614.

[25] TAPPOLET J，BERNSTEIN A. Applied temporal RDF-efficient temporal querying of RDF data with SPARQL[C]// LNCS 5554：The Semantic Web：Research and Applications. Berlin：Springer，2009：308-322.

[26] YAN L，ZHAO P，MA Z M. Indexing temporal RDF graph[J]. Computing，2019，101(10)：1457-1488.

[27] ZHAO P，YAN L. A methodology for indexing temporal RDF data[J]. Journal of Information Science and Engineering，2019，35(4)：923-934.

[28] WANG Y F, ZHU M J, QU L Z, et al. Timely YAGO: Harvesting, querying, and visualizing temporal knowledge from Wikipedia[C]// Proceeding of the 13th International Conference on Extending Database Technology. New York: ACM, 2010: 697-700.

[29] KOUBARAKIS M, KYZIRAKOS K. Modeling and querying metadata in the semantic sensor Web: The model stRDF and the query language stSPARQL[C]// LNCS 6088: Proceeding of the 7th Extended Semantic Web Conference. Berlin: Springer, 2010: 425-439.

[30] ZIMMERMANN A, LOPES N, POLLERES A, et al. A general framework for representing, reasoning and querying with annotated semantic Web data[J]. Journal of Web Semantics, 2012, 11: 72-95.

[31] LIAGOURIS J, MAMOULIS N, BOUROS P, et al. An effective encoding scheme for spatial RDF data[J]. Proceedings of the VLDB Endowment, 2014, 7(12): 1271-1282.

[32] VLACHOU A, DOULKERIDIS C, GLENIS A, et al. Efficient spatio-temporal RDF query processing in large dynamic knowledge bases[C]// ACM Symp on Applied Computing. New York: ACM, 2019: 439-447.

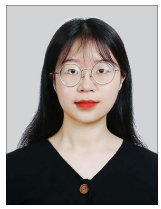
[33] CYGANIAK R, HARTH A, HOGAN A. N-Quads: Enxtending N-Triples with Context [EB/OL]. [2020-05-30]. http://sw.deri.org/2008/07/n-quads/.

[34] TIAN Y Y, JIGNESH M. TALE: A tool for approximate large graph matching[C]// Proceeding of the 2008 IEEE 24th International Conference on Data Engineering. Los Alamitos, CA: IEEE Computer Society, 2008: 963-972.

[35] KHAN A, LI N, YAN X, et al. Neighborhood based fast graph search in large networks[C]// Proceeding of the 2011 ACM SIGMOD International Conference on Management of Data. New York: ACM, 2011: 901-912.

[36] ALUÇ G, HARTIG O, ÖZSU T, et al. Diversified stress testing of RDF data management systems[C]// LNCS 8796: Int. Semantic Web Conf. Berlin: Springer, 2014: 197-212.

[37] AGUILAR-SABORIT J, TRANCOSO P, MUNTES-MULERO V, et al. Dynamic count filters[J]. ACM SIGMOD Record, 2006, 35(1): 26-32.



CHEN Yuan-yuan, born in 1996, postgraduate. Her main research interests include RDF data and the semantic web.

MA Zong-min, born in 1965, Ph.D, professor. His main research interests include databa- ses, the semantic web, knowledge representation and reasoning, information uncertainty.