文章编号: 1007-4619(2006)06-0854-10

GML空间数据查询与索引机制研究

兰小机[1], 闾国年[2], 刘德儿[1]
(1. 江西理工大学 建筑与测绘工程学院, 江西 赣州 341000;
2. 南京师范大学 地理信息科学江苏省重点实验室, 江苏 南京 210097)

摘 要: 由于传统 GIS数据模型的差异,导致空间数据难以集成与共享。各 GIS软件厂商及第三方软件厂商提出了利用空间数据转换的解决方案,但是它还是不能很好地解决空间数据集成与共享存在的问题。地理标记语言 GML的出现,为 GIS空间数据建模、集成与共享提供了统一的标准与框架。GML已经成为事实上的空间数据编码、传输、存储和发布的国际标准,大量 GML格式的空间数据开始涌现。如何有效地存储管理 GML空间数据,已经成为 GIS研究的热点问题。本文结合 XML数据库技术和传统的空间数据库技术,对 GML空间数据的查询、索引进行了深入的研究。以 XML标准查询语言 XQuery为基础,提出了 XQuery空间扩展的内容,开发了 GML空间数据查询语言,实现了 GML空间数据的本原查询;结合 XML文档编码和传统的空间数据索引,提出了基于空间索引的 GML一体化索引机制,并以 R树索引为例,对一体化索引的查询处理性能进行了实验分析。实验结果表明,本文提出的基于空间索引的 GML一体化索引机制是可行的、高效的。

关键词: GML; GML查询; GML索引; 空间索引
中图分类号: P208 **文献标识码**: A

Research on Query and Index for GML Spatial Data

LAN Xiao-ji[1], LU Guo-nian[2], LIU De-er[1]
(1. Jiangxi University of Science and Technology, Jiangxi Ganzhou 341000, China;
2. Jiangsu Provincial Key Lab of GIS Science, Nanjing Normal University, Jiangsu Nanjing 210097, China)

Abstract: The differences among traditional GIS data models result in difficulty of spatial data integrating and sharing. Commercial GIS vendors and third parties suggested the data conversion solutions. These methods cannot solve the problem perfectly. Geography Markup Language(GML) established by OGC provides standard and framework for spatial data modeling, integrating and sharing. GML has become the defacto international standard for spatial data encoding, transmitting, storing and distributing; more and more spatial data has been stored in GML format. The issue of how to manage the spatial data in GML format efficiently has become the hot one in GIS research. Combining XML database technology and traditional spatial database technology, this paper conducts a deep research on querying and indexing GML data. The spatial extensions to XQuery are suggested. Consequently one GML query language based on XQuery is developed, and native query over GML data is realized. By integrating XML document encoding techniques and traditional spatial index methods, one unified indexing model based on spatial index for GML data is suggested. Taking R-Tree as an example, the performance of the unified indexing model is tested and analyzed through a set of experiments. Experimental results show that the unified indexing model is feasible and efficient.

Key words: GML; GML query; GML index; spatial index

收稿日期: 2005-05-15; **修订日期**: 2005-09-13

基金项目: 国家自然科学基金项目(编号: 40401045),江西省教育厅科技研究项目(编号:赣教技字[2006]195号),地理信息科学江苏省重点实验室开放基金项目(编号: JK20050302)

作者简介: 兰小机(1965—),男,副教授。1994年 6月于武汉测绘科技大学获工程测量硕士学位,2005年 6月于南京师范大学获地图学与地理信息系统博士学位。主要研究方向为 GML空间数据库、空间数据共享与互操作等,已发表学术论文 20余篇。E-mail: landcom8835@163.com。

1 引　言

在 GIS发展过程中,由于数据模型的差异,导致不同格式的空间数据不能无损地共享,多源异构空间数据不能很好地集成,二进制格式的空间数据需要专门的 GIS软件才能编辑修改和应用,且各 GIS软件的功能都是针对其自身的数据格式而设计的,针对这些问题,各 GIS软件厂商及第三方软件厂商提出了利用空间数据转换的解决方案,由于各 GIS软件采用的数据模型不同,定义的几何基元不一致,导致空间数据由一种格式转换到另一种格式时损失部分信息,再转回时无法复原,因此这种解决方案仍不能很好地解决以上问题。地理标记语言 GML(Geography Markup Language)[1]的出现,为 GIS空间数据建模、集成与共享提供了统一的标准与框架。GML是开放地理信息系统协会 OGC(Open Geospatial Consortium)制定的基于 XML的中立于任何厂商、任何平台的地理信息编码标准,用于地理信息的传输、存储和发布。GML已经成为事实上的空间数据编码、传输、存储和发布的国际标准。如果使用 GML来存储管理空间数据,即各 GIS软件开发商都使用 GML作为其数据模型和文件格式,那么以上所有问题就可以彻底得到解决,不需要任何转换,所有 GIS的空间数据都可以有效地集成与共享。但是,如何有效地存储、查询和检索 GML格式的空间数据,还有很多不确定性和问题需要研究,需要从理论和技术上寻求新的可行途径,构造新一代空间数据库和基于新一代空间数据库的 GIS。

由于 GML半结构化的层次树状数据模型与传统的关系数据模型之间存在的重大差别,传统的关系数据库技术并不适合 GML空间数据的存储、查询和索引。XML数据库技术的发展为 GML空间数据的存储管理奠定了一定的基础。目前已经涌现了一批专门用来存储管理 XML数据的数据库系统[2],这些系统能够很好地实现 XML数据的存储管理、查询检索等功能,为 XML数据的存储管理提供了有效的解决方案。但是 GML并不等同于 XML,GML技术与 XML技术的区别正如 GIS技术与 MIS技术的区别,GML空间数据库与 XML数据库的区别正如空间数据库与一般数据库的区别。所以,需要在 XML数据库技术的基础上,研究 GML空间数据的存储管理等问题。

对 XML查询的研究,国内外学者提出了多种 XML查询语言[3-6],并对 XML的查询优化进行了广泛的研究。为了统一 XML查询语言,W3C于 1999年 9月正式成立了 XQuery工作组,先后颁布了多个 XQuery草案,仍在不断的修订和完善之中[7]。目前,数据库业界的三大主流厂商 Oracle、IBM和 Microsoft都已经在各自的产品中提供了对 XQuery规范的支持,大多数本原 XML数据库,如 Tamino、eXist、Saxon也不同程度地支持 XQuery规范,XQuery将成为 XML查询语言的标准。对 XML索引的研究,国内外学者提出了多种 XML结构索引方案,比较著名的有:DataGuides[8],T-index[9],Index Fabric[10],A(k)-Index[11]等。

国内外对 GML查询、索引的研究较少,Corcoles等人在文献[12]中提出了基于 SQL的 GML空间查询语言,由于 XML数据模型与传统的关系模型之间的重大差别,扩展 SQL以支持 GML查询并不是 GML查询语言的最有效方案,当用户需要进行 GML查询或其他处理时,在 GML与关系数据库之间进行来回转换要耗费相当多的处理时间,这会降低数据的处理速度。此外,这种处理方式与 XML查询语言的标准 XQuery不相符。Vatsavai在文献[13]中比较了几种 XML查询语言,并提出了对 XQuery语言进行扩展以支持 GML查询的设想,但文中并没有涉及如何实现与 GML索引等更深层次的问题。由武汉大学和复旦大学承担的国家 863项目"基于移动 Agent和 GML的分布式空间信息集成研究与实现(编号:2002AA135340)"和南京师范大学承担的国家自然科学基金项目"GML空间数据存储索引机制研究(编号:40401045)",正在对 GML存储管理、查询、索引、集成等问题进行研究。

本文结合 XML数据库技术和传统的空间数据库技术,对 GML空间数据的查询、索引进行了深入的研究。以 XML标准查询语言 XQuery为基础,提出了 XQuery空间扩展的内容,开发了 GML空间数据查询语言;结合 XML文档编码和传统的空间数据索引,提出了基于空间索引的 GML一体化索引机制,并对一体化索引的查询处理性能进行了实验分析。

2 基于 XQuery的 GML查询语言

传统的关系数据库查询语言 SQL是针对平面的二维关系数据而设计的,并不适合 XML/GML半结构化数据的查询;商品化 GIS软件的查询系统都是针对自身的封闭数据模型和数据结构而设计的,只能查询自身的空间数据而无法查询其他 GIS系统

的空间数据。扩展 XML查询语言 XQuery是 GML查询实现的最佳选择。首先 XQuery是 W3C制定的专门针对 XML的查询语言规范,目前,主流数据库厂商和大多数本原 XML数据库厂商都已在各自的产品中提供了对 XQuery规范的支持,并且在互联网上可以找到一些开放源代码的 XQuery实现如 XQEngine[14]和 eXist[15]等。其次,GML本身是基于 XML的,是 XML在空间数据领域的一个应用,XML的所有技术如存储、查询、索引等都可以应用于 GML。第三,GML数据查询可以分为三类:(1)仅包含空间属性和空间关系的查询;(2)仅包含非空间数据的查询;(3)包含空间和非空间属性的混合查询。其中仅包含非空间数据的查询可以直接使用 XQuery提供的查询功能来完成,当然空间查询功能需要对 XQuery进行扩展,使其支持空间数据类型和空间操作算子。第四,从零开始开发 GML查询语言,技术难度大、成本高、周期长,很多底层的技术问题都需要自己解决。

本文研究开发的 GML查询语言以 XQuery引擎为基础,对其进行扩展,增加空间数据类型及空间操作算子,实现对 GML空间数据的查询。空间数据类型及空间操作算子以 OGC简单要素规范 SFS(Simple Features Specification For SQL)[16]中定义的基本空间数据类型(Spatial Data Types)和空间操作算子(Spatial Operators)为基础。OGC的 SFS中定义的基本空间数据类型只限于 2维、线性几何对象(Linear Geometries),GML 2中的空间数据类型与 SFS中定义的数据类型一致,但 GML 3中的空间数据类型已超出 SFS中定义的数据类型,除了简单要素之外,还要支持 3维、非线性几何对象(Nonlinear Geometries),因此需要对 SFS中的空间数据类型进行扩展,使其支持 GML 3。

OGC的 SFS中定义的空间操作算子包括基本操作、空间关系运算和空间分析操作三大类[9],后两类大部分操作是基于 DE-9IM的。在本文研究开发中,基本上实现了这些空间操作算子。

3 GML一体化索引的基本思想

GML数据是基于 XML格式的文本数据,它既具有普通 XML数据的特征,又具有空间数据的特点,既包括普通的属性数据,也包括空间定位数据。对 GML数据的索引也必须考虑这些特点,要结合普通的 XML索引和传统的空间数据索引。为了提高 XML数据的查询处理效率,可以通过 XML编码和结构索引两种途径实现。XML文档通过前序 后序(preorder postorder)或(start end)或(order size)编码之后,XML文档中的元素、属性就转换为二维空间中的点,既然是空间数据,自然就可以使用空间索引来加速其查询检索。基于这点考虑,本文提出基于空间索引的 GML一体化索引的思想,即属性数据、空间定位数据都使用传统的空间索引。本文研究中的空间索引使用 R树索引。

3.1 基于扩展的前序 后序 GML文档编码

文献[17, 18]提出利用前序 后序(preorder postorder)对 XML文档树中的结点进行编码。对于树中的任意两个节点 u和 v,如果 preorder(u)<preorder(v),且 postorder(u)>postorder(v),则 u是 v的祖先节点。

这种编码方法的优点是能够在常数复杂度的时间内实现任意两个结点间祖先 后代关系的判断。缺点是缺乏灵活性,当一个新的结点插入时,许多结点的前序、后序值都要重新遍历计算。针对这种前序 后序编码存在的缺陷,本文提出扩展的前序 后序编码方案,基本思想是相邻的前序(或后序)留有一定的间隔,即假如相邻前序(或后序)为 i j,j=i+k,k≥1。k=1时为普通的前序 后序编码。k为常整数或由 GML应用模式 maxOccurs属性或 GML文档的统计信息确定。

对于 XML/GML文档中的任一节点 v,其编码形式为(DocID(v), preorder(v), postorder(v), level(v), Type(v)),其中 DocID为 XML/GML的文档编号,preorder postorder为节点 v的前序、后序遍历值,level为节点 v所在的层,Type为节点 v的类型(元素节点、属性节点或文本节点等)。这种编码同样具有普通的前序 后序编码的优点,而且解决了由于新节点插入而重新计算节点的前序 后序遍历值问题。

3.2 XPath的关键轴和前序 后序平面分割特性

XPath是 XQuery的重要组成部分,是 XML/GML文档树导航查询的基本工具,支持丰富的路径查询功能。XPath的轴共 13个,其中 ancestor descendant following preceding及 self五轴合称为关键轴,它们将一个 XML/GML文档(忽略属性和命名空间节点)划分成了互不重叠的五部分(四个文档区域,一个上下文节点),而且它们组合在一起则包

含了这个 XML/GML文档的所有节点,即对于一个 XML/GML文档 D的任一节点 v,有如下关系式:

$$D=\{v/ancestor\}\cup\{v/descendant\}\cup\{v/preceding\}\cup\{v/following\}\cup\{v\}$$

如果一个 XML/GML文档采用扩展的前序后序编码方案进行编码,得到各节点的编码值(preorder, postorder),它们对应于前序后序平面(以下简称 preorder/postorder平面)中的点。图 1为一简单的 XML/GML文档各节点(preorder, postorder)分布图。从图中容易看出:对于 XML/GML文档的任一节点 v,在 preorder/postorder平面中过点 v分别作 preorder和 postorder轴的垂线,将 preorder/postorder平面分为四个子区域,其中:左上角区域 R包含 v的所有祖先节点;右下角区域 U包含 v的所有子孙节点;左下角区域 T包含 v的所有前轴节点;右上角区域 S包含 v的所有后轴节点。

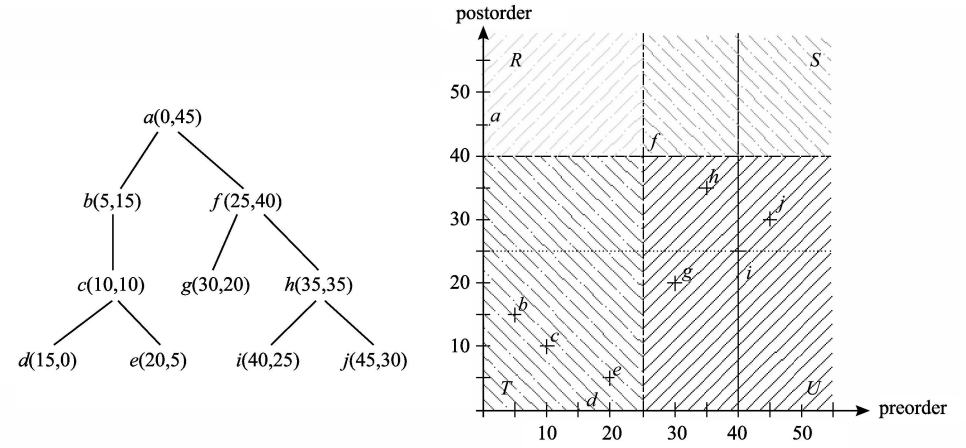

图 1　前序后序平面分割特性

Fig 1　Preorder/postorder node distribution in the resulting preorder/post order plane

利用以上特性,XML/GML中的路径查询处理就转换为 preorder/postorder平面中的一系列的窗口查询(对于一个给定的 XML/GML文档,其前序、后序值必定有边界,即最大值)。例如,要查询检索图 1中 f节点的所有子孙节点就转化为 preorder/postorder平面中的右下角窗口 U的窗口查询。这类查询处理自然可以使用空间索引(如 R树)来提高其查询处理效率。

3.3　GML数据 R树索引的创建

GML文档树通过扩展的前序后序编码映射为 preorder/postorder平面中的一系列的点。为了加速 XML/GML文档中路径查询,可以利用 R树对 XML/GML文档树结点建立索引。

GML规范没有强行规定每个地理要素一定要有一个 ID和 MBR,而这些对 R树的创建和应用是必需的。对 GML空间数据创建 R树索引的基本思路是,首先利用 SAX解析技术(和 DOM相比,SAX解析效率非常高,根据本文的实验,SAX读取一个 20M的文件在两秒内即能完成,而 DOM要解析一个 20M的 GML文档几乎不可能),逐要素读取 GML文档,并自动对每一要素进行编号 FID,同时根据要素的坐标数据自动生成要素的最小外包矩形 MBR,然后调用 R树生成算法建立 R树索引。当对 GML文档进行查询的时候,先通过 R树索引找到符合条件的要素 FID,然后再到 GML数据文件或数据库中提取 FID要素的相关数据。

对于 GML属性数据,首先对 GML文档中的属性数据进行前序后序编码,获得每一节点 v的编码数据(DocID(v), FID(v), preorder(v), postorder(v), level(v), Type(v)),其中 FID(v)为这一属性数据对应的要素的编码 FID,然后根据这些编码数据即可建立 R树索引。

4　基于一体化索引的 GML路径查询处理

GML文档经过扩展的前序、后序编码后,GML中的路径查询处理就转化为 preorder/postorder平面中的一系列的窗口查询,见表 1。例如,要查询检索图 1中 f节点的所有子孙节点,就转化为 preorder/postorder平面中的右下角窗口((preorder(v), Max), [0, postorder(v)))的窗口查询,其中(preorder(v), Max)为该窗口 preorder坐标的最小、最大值,[0,

postorder(v))为 postorder坐标的最小、最大值。

preorder/postorder平面中的点被关键轴分为四个子区域,分别对应四个关键轴,也即四个关键轴的查询处理结果对应于上下文节点所分割的四个子区域中的节点。其他轴的查询处理可以在关键轴查询处理基础上进行扩展(表 1)。

表 1 XPath轴对应的查询窗口 window(α; v)

Table 1 XPath axes and their corresponding query windows window(α; v)

XPath轴 α	查询窗口 window(α; v)			
	preorder取值范围	postorder取值范围	层	类型
child	(preorder(v), Max)	[0, postorder(v))	Level(v)+1	元素
descendant	(preorder(v), Max)	[0, postorder(v))	*	元素
descendant-or-self	[preorder(v), Max)	[0, postorder(v)]	*	元素
parent	[0, preorder(v))	(postorder(v), Max)	Level(v)-1	元素
ancester	[0, preorder(v))	(postorder(v), Max)	*	元素
ancester-or-self	[0, preorder(v)]	[postorder(v), Max)	*	元素
following	(preorder(v), Max)	(postorder(v), Max)	*	元素
preceding	[0, preorder(v))	[0, postorder(v))	*	元素
following-sibiling	(preorder(v), Max)	(postorder(v), Max)	Level(v)	元素
preceding	[0, preorder(v))	[0, postorder(v))	Level(v)	元素
attribute	(preorder(v), Max)	[0, postorder(v))	*	属性

注:表中 Max为文档中节点编号的边界值,* 为可以不考虑。

基于 R树索引的窗口查询相对比较简单,只要从 R树的根节点开始,依次往下检测各节点的 MBR与查询窗口是否满足查询条件(包含),若条件成立,则继续往下搜索、检测,直到 R树的叶节点;若条件不成立,该节点以下的子节点就不用再检测了。

5 GML查询索引系统的实现

本研究的所有开发都是用 Java语言在 JBuilder X环境下完成的。在 GML查询索引系统 GMLQEngine实现过程中,首先针对 GML空间数据的特点,对 JTS[19]提供的空间数据类型和空间操作算子进行了修改、扩充;同时根据本文提出的 GML一体化索引的思想,开发了 GML索引模块;在此基础上,对开放源代码 XQuery引擎 XQEngine[14]进行了扩展,增加了空间数据类型和空间操作算子及空间索引功能,使其支持 GML空间数据查询。GML数据为基于 XML编码的文本数据,查询结果为 GML数据的子集,即为文本数据。在 GIS应用中,一般都要将查询结果以图形的方式显示给用户。本研究开发中,将 GML格式的查询结果转换为 SVG格式,然后以图形方式显示给用户;或直接采用本系统开发的内部图形系统显示。

GML查询索引系统 GMLQEngine的主界面如图 2所示,将查询、结果显示、原文件显示及图形显示等都集成在一个图形界面下,这样可以使用户不必在不同的窗口之间来回切换。除了菜单条和工具条外,主要包括四个区:GML查询语句编辑区、GML查询结果文本显示区、GML查询结果图形显示区和

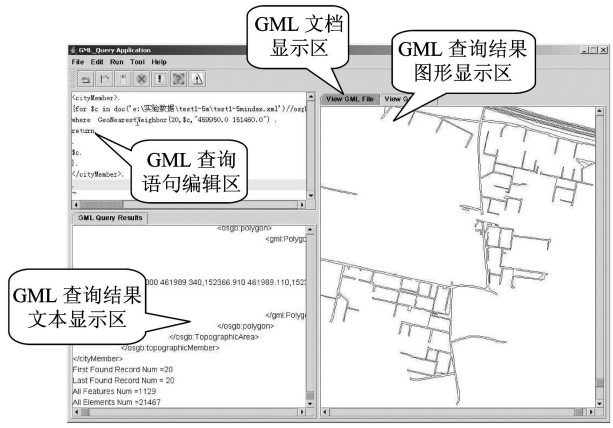

图 2 GML查询系统主界面

Fig 2 Main user interface of GML query system

GML文档显示区。查询语句编辑区是用户输入、编辑查询语句的工作区,除了键盘输入外,还可以调入外部以文件保存的查询语句。GML文档显示区以GML格式显示要查询的 GML数据,帮助用户书写查询语句。在本系统中,对查询结果的处理提供了三种方式:XML/GML的格式、SVG格式(将查询结果转换为 SVG格式)和直接利用本系统提供的图形功能。GML查询结果文本显示区以 GML格式显示GML的查询结果。GML查询结果图形显示区利用 Java 2维提供几何图形矢量绘图函数来实现图形的自动绘制。

GML查询语言 GMLQEngine完全遵循 XQuery语法,只是在 XQuery语法的基础上增加了空间关系运算和空间分析运算。以下为一典型的GMLQEngine查询语句:

```
For $ c in doc('d:\GMLIndexQuery\test10m\test10m.gml')//topographicMember
where intersects($c,"461400 152700 461420 152720") and $c//theme="Road"
return $c
```

在这一查询语句中,空间数据以 GML格式存储在 test10m.gml文件中。这一语句是要查询test10m.gml文档中所有满足条件的 topographicMember元素,这里的条件包括两部分:空间条件和非空间条件。空间条件是和矩形窗口(461400 152700 461420 152720)相交,非空间条件是元素 topographicMember的后代元素 theme的值为“Road”,同时满足这两个条件的所有 topographicMember元素为这一语句的查询结果。查询结果首先仍以 GML格式表示。

该查询语句的处理步骤如下:

(1)将 Where子句中查询条件分解为一个空间条件部分和一个非空间条件部分。空间条件部分包括空间关系运算和(或)空间分析运算,非空间条件部分不包括空间数据、空间谓词和空间算子,仅包括原始非空间属性。这里 where子句分解为空间条件部分 intersects($c,"461400 152700 461420 152720")和非空间条件 theme="Road"。即该查询语句相当于分解为两个新的查询语句:

① 非空间条件查询:

```
For $c in doc('d:\GMLIndexQuery\test10m\test10m.gml')//topographicMember
where $c//theme="Road" return $c
```

② 空间条件查询:

```
For $c where intersects($c,"461400 152700 461420 152720") return $c
```

或

① 空间条件查询:

```
For $c in doc('d:\GMLIndexQuery\test10m\test10m.gml')//topographicMember
where intersects($c,"461400 152700 461420 152720") return $c
```

② 非空间条件查询:

```
For $c where $c//theme="Road" return $c
```

即有两种查询处理策略:①先执行非空间查询后执行空间查询;②先执行空间查询后执行非空间查询。一般来说,空间查询的代价高于非空间查询的代价。所以,一般先执行非空间查询后执行空间查询。

(2)执行非空间条件查询;

(3)在非空间条件查询结果的基础上,根据查询条件中涉及空间运算,执行空间查询,得到最终的查询结果。

6　实验结果与性能分析

6.1　实验环境与数据

为了对本文提出的 GML一体化索引性能进行测试,本研究首先从国外某网站上下载了 15个 1M—25M不等的 GML文档,然后对这些文档进行了编辑、修改和合并,得到了本次实验所需的 10个GML文档,文件大小由 0.5M逐渐变化到 50M,各文档所包含的几何对象个数和节点个数见表 2。

表 2　实验使用的 GML文档大小及所含的节点个数、几何对象个数

Table 2　GML document sizes and its numbers of nodes and geometries used in experiments

文档大小/M	几何对象个数	节点个数
0.5	198	4367
1.0	589	11698
1.5	1129	21467
2.5	2089	38757
5.0	4610	84337
10.0	9467	172094
15.0	13903	269828
20.0	20060	363476
25.0	24742	470621
50.0	49322	894940

这些 GML文档内容包括以下 9个主题:roads, tracks and paths, land, buildings, water, rail, height, heritage, structures, administrative boundaries。其几何类型有 Point, LineString, LinearRing, Box, Polygon

和 MultiPoint、MultiLineString、MultiPolygon。

实验中共设计了两大类、6个查询:空间查询(Q_1—Q_3)和空间、属性相结合的混合查询(Q_4—Q_6)。6个查询的查询语句见表 3。空间查询包括三类典型的空间查询:包含、相交和 k邻近查询。

实验中使用的软件为本课题组用 Java开发的 GML空间数据查询索引系统 GMLQEngine。实验中的每个查询对每个不同大小文档都运行 10次,实验结果中的运行时间是取 10次的平均时间。实验中,空间数据、属性数据(前序后序编码后的)都采用 R树索引。实验平台是 Mobile intel 4 COMPAQ笔记本电脑,主频 2.4GHz,内存 512M,硬盘 40G。操作系统为 Microsoft Windows 2000 Advanced Server。

表 3 实验中的查询样例

Table 3 Query samples used in following experiments

编号	查询语句	查询类型
Q_1	for $c in doc('GML文档')//GMLQLFeatureMember where Containment($c, "461800 152000 461900 152510") return $c	空间包含查询
Q_2	for $c in doc('GML文档')//GMLQLFeatureMember where Intersection($c, "461800 152000 461900 152510") return $c	空间相交查询
Q_3	for $c in doc('GML文档')//GMLQLFeatureMember where nearestNeighbor(k, $c, "空间对象") return $c	空间 k邻近查询
Q_4	for $c in doc('GML文档')//GMLQLFeatureMember $t in $c//GMLQLtheme where $t="Land" and Containment($c, "461800 152000 461900 152510") return $c	混合查询
Q_5	for $c in doc('GML文档')//GMLQLFeatureMember $t in $c//GMLQLtheme where $t="Land" and Intersection($c, "461800 152000 461900 152510") return $c	混合查询
Q_6	for $c in doc('GML文档')//GMLQLFeatureMember $t in $c//GMLQLtheme where $t="Land" and nearestNeighbor(k, $c, "空间对象") return $c	混合查询

6.2 实验结果与性能分析

GML空间查询表达为 GMLQEngine中的 where条件,即在 where子句中增加表达空间查询的操作算子。空间包含、相交和 k邻近查询的结果分别见表 4、表 5和表 6。对于空间包含查询(Q_1),有空间索引和无空间索引的查询相比,查询效率(时间比)都超过 27,并且随着文件大小的增加,查询效率升高,查询时间比越大,当文件大小为 20M的时候,时间比达到 791.25。当文件大小超过 20M以后,没有空间索引已经无法进行空间查询了。这也充分说明了空间索引的作用。其他空间查询,有类似的结论。

对于 K-NN查询,如果没有索引,则要计算所有几何对象的 MinDist和 MinMaxDist,并要对所有的 MinDist和 MinMaxDist进行大小比较、排序,计算工作量相当大。所以本研究中没有考虑无索引的 K-NN查询,而只考虑了基于 R树索引的 K-NN查询,实验结果见表 6。从表中可以看出,K-NN查询的效率是相当高的,从一个大小为 20M的 GML文档中检索与某点最近的 10个几何对象,仅需 320ms。K-NN查询,随着文件大小的增加,检索时间略有增加;对于同一 GML文档,1-NN,5-NN,10-NN的查询时间也略有增加。

GML混合查询是包含空间条件和属性条件的联合查询,它表达为 GMLQEngine中的 where条件,即在 where子句中既有空间操作算子又有属性条件。空间条件包含、条件相交和条件 k邻近查询的结果分别见表 7、表 8和表 9。

将表 4(Q_1)与表 7(Q_4)对照比较,在有空间索引的情况下,空间包含查询的效率大大高于条件包含查询的效率,如图 3所示;而在没有索引的情况下,空间包含查询的效率略高于条件包含查询的效

表 4　包含查询效率比较(Query: Q_1)

Table 4　Containment query processing time with and without spatial index(Query: Q_1)

文档大小 /M	0.5	1	1.5	2.5	5	10	15	20	25	50
无索引查询时间 /ms	836	1467	1840	2062	2756	6098	11308	175657	内存溢出错误	内存溢出错误
R树索引查询时间 /ms	30	40	55	70	90	110	175	222	243	271
时间比	27.86	36.68	33.45	29.46	30.62	55.44	64.62	791.25		
查询结果个数	23	26	26	36	28	36	36	29	29	87

表 5　相交查询效率比较(Query: Q_2)

Table 5　Intersection query processing time with and without spatial index (Query: Q_2)

文档大小 /M	0.5	1	1.5	2.5	5	10	15	20	25	50
无索引查询时间 /ms	994	1499	1582	2199	3429	6532	13420	内存溢出错误	内存溢出错误	内存溢出错误
R树索引查询时间 /ms	160	175	246	280	320	441	561	850	962	1352
时间比	6.21	8.57	6.33	7.85	10.72	14.81	23.92			
候选几何对象个数	26	52	43	80	67	77	96	109	109	109
最终结果个数	9	9	19	12	22	64	25	105	105	27

表 6　K 邻近查询效率比较(Query: Q_3)

Table 6　K nearest neighbors query processing time with spatial index (Query: Q_3)

文档大小 /M	0.5	1	1.5	2.5	5	10	15	20	25	50
R树索引查询时间 /ms										
1-NN	40	50	70	90	120	140	180	220	内存溢出错误	内存溢出错误
5-NN	60	80	90	120	150	180	220	280	内存溢出错误	内存溢出错误
10-NN	90	100	125	140	170	210	260	320	内存溢出错误	内存溢出错误

表 7　包含条件查询效率比较(Query: Q_4)

Table 7　Condition based containment query processing time with and without spatial index (Query: Q_4)

文档大小 /M	0.5	1	1.5	2.5	5	10	15	20	25	50
无索引查询时间 /ms	1250	1653	1882	2194	3343	6290	12456	内存溢出错误	内存溢出错误	内存溢出错误
R树索引查询时间 /ms	120	145	156	180	291	523	951	1182	2268	3354
时间比	10.42	11.40	12.06	12.19	11.48	12.03	11.91			
候选对象个数	23	29	29	36	28	36	36	29	29	87
查询结果个数	19	20	20	5	7	27	31	25	25	25

表 8　相交条件查询效率比较(Query: Q_5)

Table 8　Condition based intersection query processing time with and without spatial index (Query: Q_5)

文档大小 /M	0.5	1	1.5	2.5	5	10	15	20	25	50
无索引查询时间 /ms	1442	1725	1943	2253	3755	6719	13747	内存溢出错误	内存溢出错误	内存溢出错误
R树索引查询时间 /ms	411	482	528	631	756	952	1112	1382	2764	4575
时间比	3.51	3.58	3.68	3.75	4.96	7.06	12.36			
候选对象个数	26	26	26	30	34	38	38	39	39	172
最终结果个数	7	10	10	13	21	23	23	23	23	35

表 9 带条件的 k邻近查询效率比较(Query: Q_6)

Table 9 Condition based k nearest neighbors query processing time with spatial index(Query: Q_6)

文档大小 /M	0.5	1	1.5	2.5	5	10	15	20	25	50
R树索引查询时间 /ms										
1-NN	60	80	110	140	170	210	240	280	内存溢出错误	内存溢出错误
5-NN	120	150	180	210	260	300	340	370	内存溢出错误	内存溢出错误
10-NN	220	260	280	290	310	330	380	520	内存溢出错误	内存溢出错误

率,如图 4所示。对于 K-NN查询,附加属性条件后,查询时间和无属性条件的 K-NN查询相比,略有增加,其对比如图 5所示,NNC表示附加属性条件的 K-NN查询。

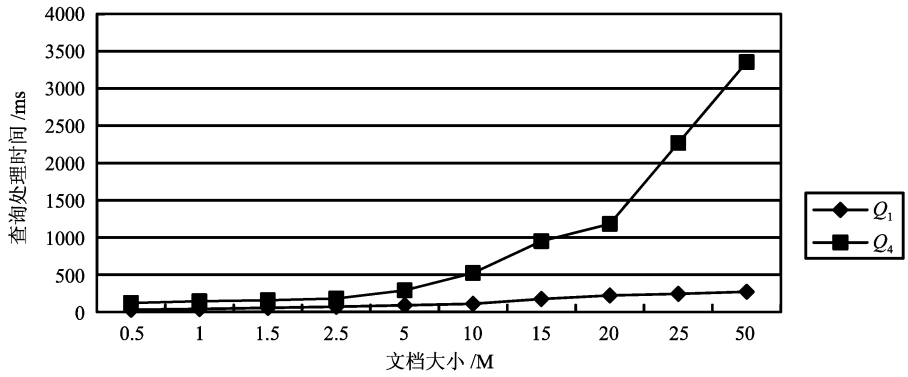

图 3 Q_1、Q_4查询时间比较(有索引)

Fig 3 Comparison of query processing time of Q_1 and Q_4 (with spatial index)

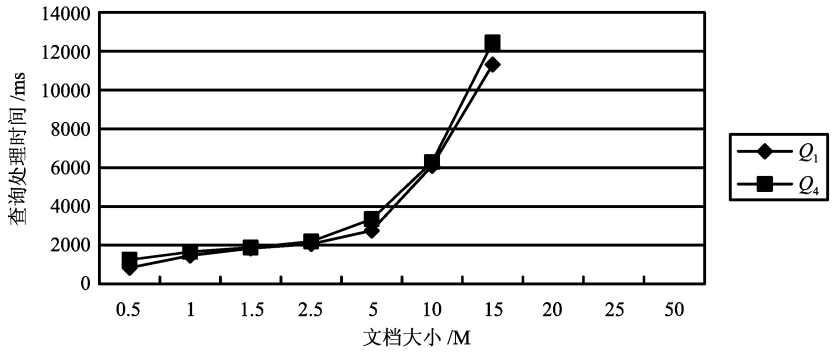

图 4 Q_1、Q_4查询时间比较(无索引)

Fig 4 Comparison of query processing time of Q_1 and Q_4 (without spatial index)

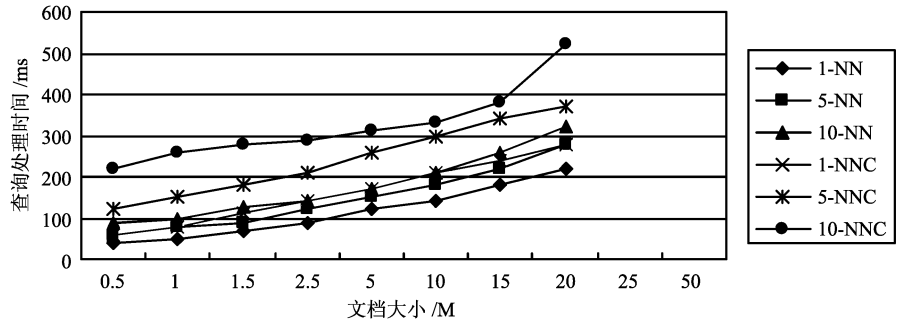

图 5 Q_3、Q_6查询时间比较

Fig 5 Comparison of query processing time of Q_3 and Q_6

以上实验结果表明:本文提出的基于空间索引的 GML一体化索引机制是可行的、高效的。

7 结　论

GML空间数据是基于 XML的文本格式的空间数据,本文以 XML标准查询语言 XQuery为基础,提出了 XQuery空间扩展的内容,开发了 GML空间数据查询语言,实现了 GML空间数据的本原查询;同时提出了基于空间索引的 GML一体化索引机制,并以 R树索引为例,对一体化索引的查询处理性能进行了实验分析。实验结果表明,本文提出的 GML空间数据查询索引机制是可行的、高效的。本文的研究为 GML空间数据查询和索引提供了新的思路、新的解决方案,为 GML空间数据的广泛应用奠定了一定的基础,同时进一步充实了空间数据库的理论体系,对空间数据的集成与共享有重要的实际意义。

参 考 文 献 (References)

[1] OpenGIS Geography Markup Language (GML) Implementation Specification 3.0 [S]. Open GIS Consortium, 2003.

[2] XML Database Products: Native XML Databases [N]. http://www.rpbourret.com/xml/ProdsNative.htm, 2005.

[3] Abiteboul S, Quass D, McHugh J, et al. The Lorel Query Language for Semistructured Data [J]. International Journal on Digital Libraries, 1997, 1(1): 68—88.

[4] Deutsch A, Fernandez M F, et al. A Query Language for XML [J]. Computer Networks, 1999, 31(11—16): 1155—1169.

[5] Robie J, Lapp J, Schach D. XML Query Language (XQL) [A]. In Proceedings of the Query Language Workshop [C]. 1998.

[6] Chamberlin D, Robie J, Florescu D. Quilt: An XML Query Language for Heterogeneous Data Source [A]. WebDB2000 [C]. 2000.

[7] XML Query [N]. http://www.w3.org/XML/Query, 2005.

[8] Jason McHugh, Jennifer Widom, Serge Abiteboul, et al. Indexing Semistructured Data [N]. http://www-db.stanford.edu/lore/pubs/semiindexing98.pdf, 1998.

[9] Tova Milo, Dan Suciu. Index Structures for Path Expressions [A]. Intl. Conf. on Database Theory [C]. 1997.

[10] Brian F. Cooper, Neal Sample, Michael J. Franklin, et al. A Fast Index for Semi-structured Data [A]. Proceedings of the 27th VLDB Conference [C]. Roma, Italy, 2001.

[11] Raghav Kaushik, Pradeep Shenoy, Philip Bohannon, et al. Exploiting Local Similarity for Indexing Paths in Graph-Structured Data [A]. Proceedings of the 18th International Conference on Data Engineering [C]. 2002.

[12] Corcoles J E, Gonzalez P. A Specification of a Spatial Query Language over GML [A]. In Proceedings of the Ninth ACM International Symposium on Advances in Geographic Information Systems [C]. ACMGIS: ACM Press, 2001.

[13] Ranga Raju Vatsavai: GML-QL: A Spatial Query Language Specification for GML [N]. http://www.cobblestoneconcepts.com/ucgis2summer2002/vatsavai/vatsavai.htm, 2002.

[14] XQEngine [N]. http://xqengine.sourceforge.net/, 2005.

[15] eXist [N]. http://exist.sourceforge.net/, 2005.

[16] OpenGIS Simple Feature Specification For SQL Revision 1.1 [S]. Open GIS Consortium, 1999.

[17] Torsten Grust: Accelerating XPath Location Steps [A]. In Proc. of the 21st Int'l ACM SIGMOD Conference on Management of Data [C]. Madison, USA, 2002.

[18] Torsten Grust, Maurice Van Keulen, Jens Teubner: Accelerating XPath Evaluation in Any RDBMS [J]. ACM Transactions on Database Systems, 2004, 29(1): 91—131.

[19] JTS Topology Suite [N]. http://www.vividsolutions.com/JTS/JTSHome.htm, 2004.