文章编号: 1001- 2486(2003) 06- 0005- 05

基于 DJI 分步实现的联机空间距离查询处理*

肖 予钦, 张 巨, 陈 荦, 景 宁
(国防科技大学电子科学与工程学院, 湖南 长沙 410073)

摘 要: 综合考虑了查询处理时的计算费用和存储费用, 提出了距离连接索引(Distance-associated Join Indices, DJI) 的分步实现方法, 以高效地支持联机空间距离查询。该方法采用分阶段计算 DJI 的策略, 根据用户的查询条件动态地计算 DJI 的一部分以支持查询。实验研究证明, 与传统方法相比, DJI 的分步实现方法在性能上具有较大优势。

关键词: 联机空间距离查询; 距离连接索引; 空间数据挖掘; 空间数据库
中图分类号: TP392 文献标识码: A

Online Spatial Distance Queries Processing Based on the Multi-step Implementation of DJI

XIAO Yu-qin, ZHANG Ju, CHEN Luo, JING Ning
(College of Electronic Science and Engineering, National Univ. of Defense Technology, Changsha 410073, China)

Abstract: Taking both computing and storage cost for query processing into consideration, we propose the multi-step implementation algorithm of distance-associated join indices (DJI) to efficiently support online spatial distance queries. The algorithm computes DJI in a stepwise strategy in which only a part of join indices is computed dynamically according to the user's query condition. Experiment shows that the proposed multi-step implementation algorithm of DJI has considerable performance advantage over the traditional methods.

Key words: online spatial distance queries; distance-associated join indices; spatial data mining; spatial databases

近年来, 空间距离查询在空间数据挖掘、空间数据分析等信息处理领域越来越受到广泛的重视。这类查询描述了空间对象之间的度量关系, 对于理解空间数据以及揭示空间数据和非空间数据之间隐含的关系具有非常重要的作用。同时, 这些空间信息处理领域对空间距离查询处理的特殊需求又产生了新的研究问题。首先, 空间数据挖掘研究的往往是主题对象及其邻域中对象之间的关系, 因此需要执行一个带有最大距离约束的空间距离查询。其次, 在联机决策支持环境中执行空间距离查询时, 用户需要动态修改最大距离值, 以便在和计算机的不断交互中得到令人满意的最终结果。因此, 如何高效地处理最大距离约束动态变化的空间距离查询就成为提高联机分析挖掘性能的一个至关重要的问题。

处理空间距离查询需要执行距离连接操作, 但是目前针对距离连接处理所提出的方法却很难处理动态的空间距离查询[1~3]。处理空间距离查询的另一种方法就是利用距离连接索引(Distance-associated Join Indices, DJI)[4]。虽然距离连接索引支持动态的空间距离查询, 但是由于它存储了所有对象之间的距离信息, 因而实际上是计算了两组空间对象的笛卡尔积。对于数据量较大的两组空间对象而言, 它们的笛卡尔积中所包含的数据量不是一个小数目, 存储这样大量的数据需要非常大的空间, 这就使得这种方法的性能受到影响[5]。本文将综合考虑查询处理时的计算费用和存储费用, 研究联机空间距离查询的高效处理方法。

* 收稿日期: 2003- 09- 14
基金项目: 国防科技大学预研基金资助项目(JC02- 04- 018)
作者简介: 肖予钦(1975—), 女, 博士生。

1 距离连接索引的分步实现算法

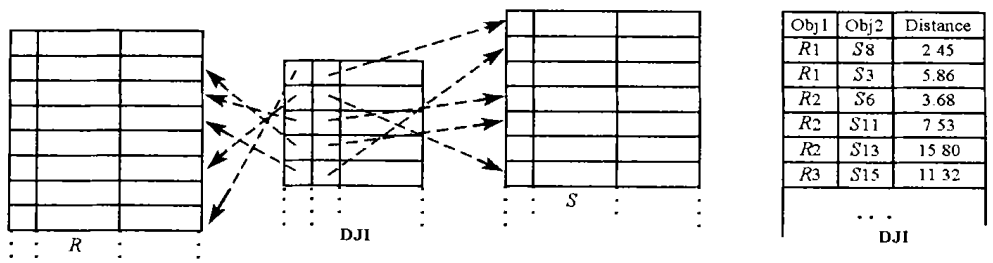
1.1 距离连接索引

距离连接索引存储了两个空间对象的标识符以及它们之间的距离信息(见图 1)。对两个输入的空间数据集 R 和 S, 它们之间的 DJI 可以表示为

$$DJI = \{< O_i,\ O_j,\ d_{ij} > \mid O_i \in R,\ O_j \in S,\ d_{ij} = d(O_i,\ O_j)\}$$

其中, d 表示任意的距离函数。

通过预先计算出空间对象间的距离信息, DJI 将两个空间数据集之间的连接操作转化为对 DJI 文件的访问, 避免了在联机查询时执行复杂的几何计算, 节省了连接操作的执行时间和 I/O 访问次数。但是, 由于 DJI 计算了两个空间数据集的笛卡尔积, 如果每个空间数据集都包含 5 万条记录, 则它们的笛卡尔积将包含 25 亿条记录, 如此大量的数据对存储和访问都会带来问题。因此, 利用这种基本的 DJI 进行查询处理并不实际。



Obj1	Obj2	Distance
R1	S8	2.45
R1	S3	5.86
R2	S6	3.68
R2	S11	7.53
R2	S13	15.80
R3	S15	11.32
...		

图 1 距离连接索引 DJI

Fig. 1 Distance-associated join indices, DJI

1.2 分步实现算法

在联机空间距离查询中, 那些距离值较小的对象偶将会经常被访问到, 而那些距离值较大的对象偶将很少甚至不会被访问到。从这个认识入手, 可以将距离值较小的对象之间的距离信息预先计算并存储起来, 而根据用户的查询条件动态地计算 DJI 的其他部分来支持查询。

算法首先选取一个合理的最大距离值 M_0, 计算出所有距离在 $[0,\ M_0]$ 范围内的距离连接索引 DJI_0, 即 $DJI_0 = \{< O_i,\ O_j,\ d_{ij} > \mid 0 \leqslant d_{ij} \leqslant M_0\}$, 然后用这个 DJI_0 来支持后继的用户查询。M_0 最开始先取为估计值, 然后在算法处理过程中自动实现更新。有关 M_0 的初始估计, 我们将在第 3 节进行讨论。

记联机查询中用户给出的动态最大距离值为 $M_i (i \geqslant 1)$。我们采用链表数据结构 $QList$ 来对查询距离值进行管理并维护它们的一个升序, $QList$ 首先只包含 0 和 M_0。每当用户给出一个新的查询距离值 M_i, 将 M_i 插入到 $QList$ 中的相应位置。如果 M_i 不大于当前 $QList$ 中的最大值, 则对已计算出的 DJI 进行选取就能给出查询结果; 否则, 计算距离在 M_i 和当前 $QList$ 中最大值范围之间的对象偶, 所有已计算出的对象偶将共同给出查询结果。上述的计算过程也称为分步计算, 对应 M_i 所得的计算结果记为 DJI_i。通过这种分步实现 DJI 的方法, 既能避免冗余的距离计算, 又能减少过多的存储空间, 从而以较低的代价满足了用户的动态查询需求。图 2 给出了算法的伪代码。

1.3 M_0 的自动更新

链表数据结构 $QList$ 除了用来管理 M_i, 还维护了用来更新 M_0 的统计信息。M_0 的更新基于以下认识: 每调用一次程序, 若在 $[0,\ M_0]$ 范围内的 M_i 出现的总次数小于在该范围外 M_i 出现的总次数, 则说明超过半数的查询要访问距离值在 $[0,\ M_0]$ 范围以外的对象之间的距离, 因此有必要更新 M_0。但是, M_0 的更新不应该以单次执行程序的结果为依据, 因为单次执行程序的结果可能带有同一个用户的个人偏见。我们采用另一个链表数据结构 $CList$ 来存储所有用来更新 M_0 的候选值。如果 M_0 需要更新, 则将所有比 M_0 大的 M_i 中的最小者输入链表 $CList$ 中; 否则, 将 M_0 输入链表 $CList$ 中。当 $CList$ 中大于 M_0 的数的个数超过 M_0 的个数时, 则将 M_0 修改为这些数的算术平均值, 进而将描述距离在 M_0 到该值范围内的所有对象之间的距离信息计算并存储到 DJI_0 中, 并用更新后的 DJI_0 支持后继的

查询。

Algorithm MDJI(R, S: R _ Node, M_i(i ≥0))

```
01 Output ← an empty set
02 QList, CList ← empty query list and candidate list
03 EnList(QList, {0, M0})
04 while(there is Mi which has not been processed) do
05   MinDist ← 0
06   MaxDist ← the largest element of QList
07   EnList(QList, {Mi})
08   if Mi ≤MaxDist then
09     DJIi ← an empty set
10     Output ← select from the computed DJI all records < Oi, Oj, dij> satisfying
                    0≤dij ≤Mi
11   else
12     MinDist ← MaxDist
13     MaxDist ← Mi
14     DJIi ← CompDJI(R, S, MinDist, MaxDist)
15     Output ← ∪_{j=0}^{i} DJIj
16   end if
17 end while
18 if(# (elements in QList larger than M0) > |QList|/2) then
19   EnList(CList, {Mk}), Mk is the first element in QList which is larger than M0
20 else
21   EnList(CList, {M0})
22 end if
23 if(# (M0 in CList) < |CList|/2) then
24   replace M0 by the arithmetic mean of all candidates not equal to M0 and update DJI0 correspondingly
25 end if
```

图 2 DJI 的分步实现算法

Fig. 2 Multi-step implementation algorithm of DJI

2 距离在*(MinDist, MaxDist]* 范围内的 *DJI* 的计算

在以上提出的 *DJI* 的分步实现算法中, 关键是要实现过程 CompDJI*(R, S, MinDist, MaxDist)*, 即计算出距离在*(MinDist, MaxDist]* 范围内的 *R* 和 *S* 中的对象偶及其距离信息, 这里 *MinDist*, *MaxDist* 分别表示最小距离值和最大距离值。

我们采用 R 树作为两个输入空间数据集 *R* 和 *S* 的空间索引结构, 以下不区分 *R* 和 *S* 及其相应的 R 树表示。基于 R 树索引的 *DJI* 的计算, 实际上是对 *R* 和 *S* 执行了一个距离连接操作, 不同的只是输出结果中多了描述两个空间对象间距离信息的项。目前对距离连接处理研究最多的是带有最大距离约束的距离连接问题, 即找出所有距离在一个最大值范围之内的对象偶[1, 6]。解决这类问题的关键是通过对目录矩形之间的距离进行约束, 来限制遍历 R 树时的搜索空间以得到最终结果。但是, 当距离约束包含最小值时, 通过约束目录矩形之间的距离并不能限制搜索空间, 因此需要研究新的解决办法。为了在遍历 R 树时限制搜索空间, 我们需要给出两个对象之间距离的上界。下面, 我们首先定义对象之间的最大距离函数和最小距离函数。

2.1 最大距离函数和最小距离函数

通常距离函数是基于点之间的距离度量而定义的, 如 $d(p_1, p_2)$, p_1 和 p_2 表示点对象, d 可以代表欧氏距离或者曼哈顿距离。给定对象 o_1, o_2, 定义它们的最大、最小距离为

$$d_{\max}(o_1, o_2) = \max_{p_1 \in \partial o_1,\ p_2 \in \partial o_2} d(p_1, p_2) \quad (1)$$

$$d_{\min}(o_1, o_2) = \min_{p_1 \in \partial o_1,\ p_2 \in \partial o_2} d(p_1, p_2) \quad (2)$$

其中, ∂ 表示对象的边界。这里, 点对象的边界就是该点本身, 线对象的边界就是该线本身, 面对象的边界就是围成该面的边。称满足(1)式的函数 $d_{\max}$ 为最大距离函数, 满足(2)式的函数 $d_{\min}$ 为最小距离函数。实际上, 最小距离函数 $d_{\min}$ 就是通常所用的距离函数 d, 即 $d = d_{\min}$。显然, 有性质 $d_{\min}(o_1, o_2) \leqslant$

$d_{max}(o_1, o_2)$成立。

2.2 利用 d_{max}和 d_{min}限制搜索空间

以下讨论如何在遍历R树时,利用距离约束进行剪枝以限制搜索空间,使距离计算只在一小部分对象之间执行。我们要利用R树结构的一个重要特征,即目录矩形构成了相应子树中所有项的矩形的最小包围矩形(Minimum Bounding Rectangle, MBR)。假设 E_R 和 E_S 是 R 和 S 中两个非叶节点中项的矩形,若它们相应子节点中有项的矩形 I_R 和 I_S,由于 E_R, E_S 分别包含 I_R, I_S,故有

$$d_{max}(E_R, E_S) \geqslant d_{max}(I_R, I_S), \quad d_{min}(E_R, E_S) \leqslant d_{min}(I_R, I_S) \tag{3}$$

因此,若要求对象 o_1, o_2 之间的距离限制在$(MinDist, MaxDist]$范围内的话,由(3)式可知,只有当 E_R 和 E_S 满足 $d_{max}(E_R, E_S) > MinDist$ 和 $d_{min}(E_R, E_S) \leqslant MaxDist$ 时,它们子节点中的项才有可能满足距离范围约束。根据这一结论,可以得到如下算法:

Step1 从根节点开始向下遍历R树,验证目录节点是否满足距离约束条件;

Step2 只对那些满足约束条件的节点的子节点进行筛选,得到下一级需要验证约束条件的两组节点;

Step3 重复Step2,直到叶节点;

Step4 只对那些满足约束条件的MBRs才计算相应对象之间的距离,所有距离在$(MinDist, MaxDist]$范围内的对象偶构成最终的结果集。

在上述算法中,函数 d_{max}和 d_{min}被同时用来验证距离约束条件。由于计算矩形之间的距离要比计算对象之间的距离节约很多费用,因此利用这种剪枝策略可以极大地提高处理性能。

3 对 M_0 初始值的一个合理估计

M_0 的初始值给出了需要预计算的对象之间距离的一个最大约束值。由于那些距离较近的对象之间的距离信息会经常被查询到,因此最好将所有对象与其最邻近对象间的距离计算并存储起来,故 M_0 的初始值可以取所有对象与其最邻近对象之间的距离值中的最大者。

以下讨论如何估计所有对象与其最邻近对象之间距离值中的最大者。假定两个空间数据集 R 和 S 中的数据是均匀分布的,$|R|$、$|S|$ 分别表示 R 和 S 中的对象数目,则与 R 中一个对象的距离小于等于 D 的 S 中的对象数目可近似估计为 $|S| \times \frac{\pi \times D^2}{area(R \cap S)}$,因此与 R 中所有对象满足距离小于等于 D 的 S 中的对象数目为 $|R| \times |S| \times \frac{\pi \times D^2}{area(R \cap S)}$。由于 DJI_0 存储了 R 中所有对象与其最邻近对象之间的距离信息,因此 $|R| \times |S| \times \frac{\pi \times D^2}{area(R \cap S)} = |R|$,则初始 M_0 的估计值为:

$$M_0 = \sqrt{\rho} \tag{4}$$

其中,$\rho = \frac{area(R \cap S)}{\pi \times |S|}$。当数据不是均匀分布时,在一个小区域内可能得到很多距离较近的对象偶,这时(4)式给出的对 M_0 的估计偏小。由于我们在处理联机距离查询时采用了 DJI 的分步实现策略和 M_0 的自动更新,因此该值作为 M_0 的初始估计是合理的。

4 实验设计及结果分析

前面我们综合考虑了查询处理时的计算费用和存储费用,提出了DJI的分步实现方法来处理联机空间距离查询。本节将考虑三种方法,比较它们处理联机空间距离查询时的性能。这三种方法是:(1) R-tree,即没有任何DJI来支持查询,只能在运行时利用R-tree来执行多次距离连接;(2) MDJI,即采用DJI的分步实现方法来处理查询;(3) BDJI,即采用基本的DJI来支持查询。

实验中的测试代码用C语言编写,测试环境是Celeron 1.3GHz CPU, 256MB SDRAM, 100Mbps PCI总线和5400RPM IDE硬盘,运行的操作系统是Windows 2000 Server。实验数据取自US Bureau of the Census发布的TIGER/Line文件[7],它是测试空间连接处理算法性能的标准数据集。我们选取美

国 District of Columbia 地区的水系(321 个线对象) 和街区(5665 个面对象) 作为两个输入数据集。

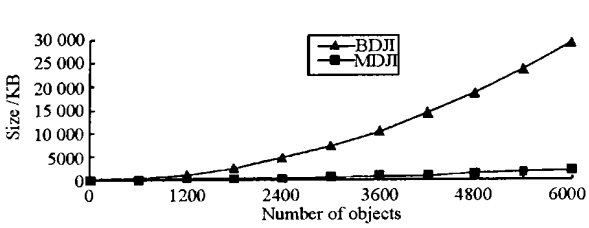


图 3 存储空间
Fig. 3 Storage space

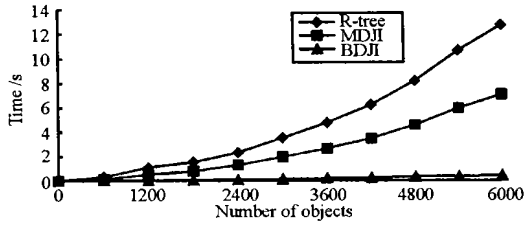


图 4 执行时间
Fig. 4 Execution time

我们仿真了 M_0= 2km, M_1= 1km, M_2= 6km 和 M_3= 4km 时的查询。图 3 和图 4 分别给出了三种方法处理上述查询所需的存储空间和执行时间。其中, 图 3 中 R-tree 方法所需的存储空间为 0, 图 4 中 R-tree方法所需的执行时间是算法执行三次的累计值。

从图 3 可以看出, MDJI 的存储空间需求远远小于 BDJI, 并且随着对象数目的增长, 所需的存储空间呈线性缓慢增长, 而不是像 BDJI 那样呈爆炸性增长。另外, 从图 4 可以看出, MDJI 所需的查询处理时间比 R-tree 几乎少 50% 。因此, 可以得出以下结论: MDJI 在存储和计算费用之间实现了较好的平衡, 与传统的 R-tree 方法和基本的 DJI 方法相比, 在性能上具有较大的优势。

5 结束语

在本文中, 我们综合考虑了查询处理时的计算费用和存储费用, 提出了距离连接索引的分步实现方法, 以高效地支持联机空间距离查询。该方法采用分阶段计算距离连接索引的策略, 根据用户的查询条件动态地计算距离连接索引的一部分以支持查询。算法采用一个链表数据结构来维护所有已计算出的连接索引文件的距离范围值。当处理距离查询时, 首先访问该链表中的数据, 以确定是否需要进行进一步的计算或者选取相关的索引文件回答用户的查询。通过这种方法, 既避免了冗余的距离计算, 又能满足用户的动态查询, 实现了存储和计算费用之间的平衡。通过实验证明, 与传统方法相比, 距离连接索引的分步实现方法在性能上具有较大的优势。

目前, 作为一种高效的空间数据挖掘的支持技术, 距离连接索引的分步实现方法被用于实现空间数据挖掘和空间数据库紧密集成的研究当中。今后, 我们将研究如何基于这种方法高效地发现空间数据库中隐含的知识, 以帮助人们科学地制定决策。

参 考 文 献:

[1] Rotem D. Spatial Join Indices [C]. Proc. 7th Int. Conf. Data Engineering, Kobe, Japan, 1991: 500– 509.

[2] Hjaltason G R, Samet H. Incremental Distance Join Algorithms for Spatial Databases [C]. Proc. 1998 ACM SIGMOD Int. Conf. Management of Data, Seattle, WA, 1998: 237– 248.

[3] Shin H, Moon B, Lee S. Adaptive Multi-stage Distance Join Processing [C]. Proc. 2000 ACM SIGMOD Int. Conf. Management of Data, Dallas, TX, 2000: 343– 354.

[4] Lu Wei, Han Jiawei. Distance-associated Join Indices for Spatial Range Search [C]. Proc. 8th Int. Conf. Data Engineering, Tempe, Arizona, 1992: 284– 292.

[5] Yeh Tsinshu. Spot: Distance Based Join Indices for Spatial Data [C]. Proc. 7th ACM Symposium on Advances in Geographic Information Systems, Kansas City, Mousa, 1999: 103– 109.

[6] Shafer J C, Agrawal R. Parallel Algorithms for High-dimensional Proximity Joins [C]. Proc. 23rd Int. Conf. Very Large Data Bases, Athens, Greece, 1997: 176– 185.

[7] US Bureau of the Census. Census 2000 TIGER/ Line Files[EB]. http://www.census.gov.