第33卷 第15期 计 算 机 工 程 2007年8月
Vol.33 No.15 Computer Engineering August 2007
·博士论文· 文章编号：1000—3428(2007)15—0049—03 文献标识码：A 中图分类号：TP391

Z树：一个高维度的数据索引结构

张 强，赵 政
(天津大学电信学院，天津 300072)

摘 要：Z树能够高效地处理对高维度数据集的矩形区域查询和最邻近搜索。它按照节点的形状变化量优化数据的插入位置，使节点形状趋于合理。文章给出了一个新的无重叠分裂算法，减少超级节点的产生。引入了动态剪枝和重新插入策略，压缩超级节点的数量和体积。提出了矩形节点的球形化方法和最优子树搜索算法。实验表明Z树的矩形区域查询和最邻近搜索的效率远远高于X树和SR树。
关键词：索引；高维度数据；矩形区域查询；最近邻域搜索

Z Tree: An Index Structure for High-dimensional Data

ZHANG Qiang, ZHAO Zheng
(School of Electronic Information Engineering, Tianjin University, Tianjin 300072)

【Abstract】The Z Tree supports the searches of rectangle area and the nearest-neighbors (NN) effectively for high-dimensional data sets. The shape variable of nodes is taken into account to optimize the sub-tree's selection for new data insertion. A new overlap-free split algorithm is proposed to avoid the generation of supernodes. A dynamic pruning and reinsertion policy is used to reduce the number and volume of supernodes. A novel method is introduced to convert a rectangle tree to a sphere tree to speed up the NN search. A new efficient algorithm of the NN search is proposed based on the optimization of search order among sub-trees. The experiments show that the Z Tree is more efficient than X Tree and SR Tree for high-dimensional data.
【Key words】index; high-dimensional data; rectangle area search; nearest-neighbor search

1 概述

在实际工作中高维度数据的应用越来越普遍，如数据仓库中关系的数量较普通数据库大幅增加；多媒体数据集中每一个点都是一个高维度向量；从DNA中得到的基因序列以及字符串匹配应用中的字符串向量都是高维度的。如何将数据组织起来以实现高速查询，对这些应用至关重要。

高维度数据的基本查询方式有两种：矩形区域查询和球形区域查询。矩形查询Q_R：$\{(x_1,...x_i,...,x_m)|(\min_i \le x_i \le \max_i)\}$，其中$\min_i$和$\max_i$是矩形的边界。球形查询$Q_S$：$\{p \mid d(p-q) \le \varepsilon\}$，其中，$q$是区域查询的中心；$\varepsilon$是半径。令$l_i=(\min_i+\max_i)/2$，$\varepsilon_i=(\max_i-\min_i)/2$，那么$Q_R$等价于$\{(x_1,...x_i,...,x_m)| \ |x_i-l_i| \le \varepsilon_i\}$，即一条矩形查询相当于$m$条球形查询。

R树及其变体是最具代表性的矩形树，适合于矩形区域查询。R树是一棵平衡树，使用最小边界矩形(minimum bounding rectangles，MBR)描述节点、节点占用的字节数以及访问时间与维度呈线性关系。它的主要问题是节点之间重叠严重，造成多径搜索，查询效率低。R*树[1]优化了插入算法和节点分裂算法，低维度下节点体积明显缩小。它的主要问题是高维度下重叠严重。X树[2]对R*树作了2点改进：进一步优化分裂算法并使用超级节点SN(super node)来避免节点间的重叠。SN的引入在一定程度上缓解了重叠问题，但随着维度的增高，SN的数目和体积大量增加，严重地影响查询效率。矩形树的共同缺点是kNN(k-nearest-neighbors)搜索效率低。

SS树[3]及其变体是最具代表性的球形树，适合于kNN搜索。SS树使用球心和半径描述节点，节点占用的字节数以及访问时间与维度无关。它的主要问题是节点空间体积过大，节点重叠严重。SS+树采用聚类方法优化分裂算法，但其重叠问题仍然严重。SR树[4]用球形和矩形相交区域表示节点，降低了节点体积，但由于沿袭了SS树的插入和分裂算法，因此节点体积与矩形树相比仍然很大。此外节点结构复杂导致节点的扇出小，并且查询时需要同时处理球形和矩形区域，影响效率。球形树的共同缺点是矩形查询效率低。

2 Z树

本文提出一个新的索引结构Z树来提高矩形查询和kNN搜索的效率。Z树采用X树的矩形节点结构，用4种方法优化其性能：

(1)按照节点形状变化量优化数据的插入位置，使节点形状趋于合理；

(2)给出一个新的无重叠分裂算法；

(3)动态对超级节点剪枝并重新插入；

(4)给出矩形节点的球形化方法和kNN最优子树搜索算法。

2.1 选择子树

R*树和X树在插入数据时，选择加入新数据后MBR体积增加最小的子树进行插入，即$\{MBR \mid \underset{MBR \in N}{Min} \Delta S\}$。规则只考虑了体积增长，忽视了MBR形状变化因素。MBR形状的好坏严重影响索引树的生长和查询。理想的MBR形状应该是匀称的，即在每一个坐标轴上的投影长度相近。不理想的形状是棒状的细长条，它会带来3个问题：

作者简介：张 强(1972－)，男，博士研究生，主研方向：数据挖掘，无线网；赵 政，教授、博士生导师
收稿日期：2006-08-10 **E-mail：**zq8622@yahoo.com

(1)如果存在几根取向不同的棒状 MBR，节点分裂时重叠现象就无法避免；

(2)棒状 MBR 在生长过程中如果沿长轴生长，则会变得更加不匀称；如果沿短轴生长，体积会迅速增大，很容易与其他节点重叠；

(3)文献[4]通过实验证明畸形 MBR 会严重影响查询效率。

Z 树将 MBR 的形状变化量加入到选择子树的规则中，即 $\{MBR \mid \underset{MBRi\in N}{Min}\ \Delta S_i\ (1+\gamma_i)\}$，其中，$\gamma_i$ (-1 ≤ γ_i ≤ 1)是形变系数。

定义 1 一个 D 维度矩形的对角线长度为 L，体积为 V，其形状系数为 $\alpha = L^D/(V*D)$。一个 MBR 插入前的形状系数是 α，插入后的形状系数变为 β，则形变系数为

$\gamma = 1$，$|w(\beta-\alpha)/\alpha| > 1$

$\gamma = w(\beta-\alpha)/\alpha$，$|w(\beta-\alpha)/\alpha| < 1$

其中，w 是形变影响权重。

从定义中知道正方体的形状系数 α=1，α 随着矩形的不匀称性的增长而增长。当形状向匀称方向变化时，γ 为负值。调整 w 可以改变 γ 对规则影响的强弱。

2.2 有效分裂算法

X 树根据分裂历史选择分裂轴的方法在高维度下经常失效。Z 树提出一个新的无重叠分裂算法，它可以发现 X 树无法找到的分裂轴。

定义 2 分裂方案 split 将节点 S 分为 2 个节点 S_1 和 S_2，S_2=S-S_1 且 $S_1 \neq \varphi$，$S_2 \neq \varphi$。

(1) $MBR(S_1) \cap MBR(S_2) = \phi$

(2) $m \leq \| S_1 \| \leq M+1-m$

其中，$\| S_1 \|$表示 S_1 中子节点的数目；M 和 m 是正常节点扇出数的上下限。

称满足(1)的分裂为无重叠分裂，同时满足(1)和(2)的分裂为有效分裂。

引理 1

split(S)是无重叠分裂 <=> ∃轴 d 和点 p∈[a,b]⊂d，使得 $p \notin \bigcup l_i$，其中，l_i 是 MBR_i 在 d 上的投影；$MBR_i \subset S$；a= $\min_{mbr_i \in S} l_i$；b= $\max_{mbr_j \in S} l_j$。

证明

=>(反证法) 假设∀轴 d 和点 p∈[a, b]⊂d，都有 $p \in \bigcup l_i$，∵ $p \in \bigcup l_i$，∴∃ l_k，p∈ l_k，过 p 点作垂直于 d 轴的平面，平面与 mbr_k 相交，split(S)不是无重叠分裂，矛盾。

<= ∵ $p \notin \bigcup l_i$，∴过 p 作垂直 d 轴的平面与 S 中任意 mbr 都不相交，将 S 分成 2 个不相交的部分 S_1 和 S_2。∵ p∈[a,b]，$p \notin \bigcup l_i$，∴ a<p<b；∵ $\min_{mbr_i \in S} l_i$=a，∴∃ mbr_j，a∈project(mbr_j), {q|q∈project(mbr_j), q<p}，同理∃ mbr_k，b∈project(mbr_j), {q|q∈project(mbr_k), q>p}，∴ mbr_j 与 mbr_k 分别位于分裂平面的两侧，∴ $S_1 \neq \varphi$，$S_2 \neq \varphi$，∴split(S)是无重叠分裂。证毕。

算法 1 有效分裂算法

(1) If there is no unprocessed axis, then return false

Else select an unprocessed axis: d

l_i ={project(mbr_i ,d)}

Sort({ l_i })

Get { interval_j } between adjacent l_i

If j=0 then goto step 1

(2) Splitting points p_j =RandomSelectAPointFrom(interval_j)

For every p_j

{ S_1 , S_2 }=split(S, p_j)

If m≤|| S_1 ||≤M+1-m then add { S_1 , S_2 } to SplittingList

(3) If SplittingList.number=0 then goto step 1

Else { $Split_1$, $Split_2$ }= $\underset{SplittingList}{Min}$ (Volume(S_1)-Volume (S_2))

return { $Split_1$, $Split_2$ }

2.3 剪枝和重新插入

索引树是动态生成的，在生长初期数据的插入带有盲目性，导致大量超级节点 SN 的产生。SN 严重破坏了树的层次化结构，主要危害有：

(1)SN 形成后就不再分裂，其 MBR 不断扩张，过大的 MBR 会产生吸引效应，新插入数据进入到 SN 的概率远远大于正常节点，产生恶性循环；

(2)过大的 SN 影响父节点的正常分裂，使其也变为 SN；

(3)SN 在生长过程中与其他节点重叠概率更大；

(4)SN 的查询效率低。

Z 树提出了动态剪枝和重新插入的策略：(1)在树生成过程中剪掉并重新插入体积过大的 SN，即 SN 的空间体积超过其父节点体积的 2/3 或者 SN 中的子节点数目超过所有兄弟节点所包含子节点数目总和的 2 倍；(2)在树生成后剪掉所有没经过剪枝处理的 SN 并重新插入。

2.4 球形化和 kNN 搜索

矩形树 kNN 搜索效率低的主要原因是：(1)矩形树需要处理各个维度，搜索效率至少与维度呈线性关系；(2)储存一个节点需要的字节数较多，受存储页面尺寸的限制，节点的扇出数小于球形树。为了提高 kNN 搜索效率，Z 树提出了矩形树的球形化方法和最优子树搜索算法。

球形化的过程为：

(1)按照球形树的扇出数目计算矩形节点所需要的字节数，这可能导致一个节点占用多个存储页面。但是矩形节点树只是中间过程，最终球形节点只保存球心和半径，占用一个存储页面；

(2)在矩形树生成后，以节点 MBR 的外接球表示该节点；

(3)自顶至下为每个球形节点建立新的指针；

(4)计算并保存每个节点的矩形内接球半径，以辅助最优子树搜索；

(5)释放掉矩形树。

定义 3 对于给定的查询点 p 和参数 k，kNN(p)表示 p 的最近 k 邻域球，$d_p^k = \max_{q \in kNN(p)} \{d(p,q)\}$ 表示 kNN(p)的半径；S_d(p) ={q| $d(p,q) \leq d$}表示以 p 为中心，d 为半径的球；||S|| 表示区域 S 中对象的个数。

定义 4 球 S_r (c)，到点 p 的最大距离 $d_{\max}$ (p, S_r (c))= d(p,c)+r，最小距离 $d_{\min}$ (p, S_r (c)) $= Max(0, d(p,c)-r)$。节点 N 的外接球为 S_e，内切球为 S_i，p 到 N 的最小距离 $d_{\min}$ (p,N)= $d_{\min}$ (p, S_e)，最大距离 $d_{\max}$ (p,N)= $d_{\max}$ (p, S_e)，最优距离 d_{opt} (p,N)= $d_{\min}$ (p, S_i)。

如果 d_p^k 已知，那么 kNN(p)={q| $d(p,q) \leq d_p^k$ }，kNN 搜索转化为球形区域查询。人工确定 d_p^k 很困难，文献[5]给出了矩形树 d_p^k 的确定算法，它执行时间与维度呈正比。引理 2 给出了动态逼近 d_p^k 的方法，效率和维度无关。

引理 2 对于给定的查询点 p 和参数 k，如果 $\| S_r(c) \| \geq k$，则 $d_p^k \leq d_{\max}(p, S_r(c))$。

证明 ∵ $S_r(c) \subseteq S_{d(p,c)+r}(p)$，$\| S_r(c) \| \geq k$，∴ $\| S_{d(p,c)+r}(p) \| \geq k$；∵ ||kNN(p)||=k，kNN(p)与 $S_{d(p,c)+r}(p)$同心，∴ $kNN(p) \subseteq S_{d(p,c)+r}(p)$，∴ $d_p^k \leq d_{\max}(p, S_r(c))$。证毕。

kNN 搜索采用深度优先的原则，搜索中根据节点到查询点的距离剪掉无用的分支；动态调整 d_p^k 的估计值 d_p^{kE}，使之逐步逼近 d_p^k。剪枝规则是：对于内部节点 N，如果 $d_{\min}(p,N) > d_p^{kE}$，则 $N \bigcap kNN(p) = \phi$，应剪掉它。更新规则是 $d_p^{kE} = \text{Min}(d_{\max}(p, S_r(c)))$，其中 $S_r(c)$是搜索路径上的节点。引理 2 已证明 $d_p^{kE} \geq d_p^k$，保证了搜索的正确性。

搜索中满足条件的子树可能不只一个，访问顺序对 d_p^{kE} 的逼近速度和剪枝效率有重大影响。如果先搜索距查询点较近的分支，可以迅速降低 d_p^{kE}，从而剪掉更多的无用分支。图 1 中 $d_{\min}(p,N1) < d_{\min}(p,N2)$，$d_{opt}(p,N1) > d_{opt}(p,N2)$。如果 2 个节点都满足搜索条件，显然先搜索 N2 更为合理。d_{opt} 描述了查询点到节点 MBR 的远近程度，而节点的所有分支都位于 MBR 中，所以使用 d_{opt} 决定搜索顺序比 $d_{\min}$ 更有效。

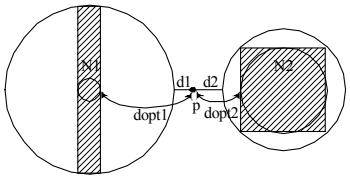

图 1 查询点 p 到节点的距离 $d_{\min}$ 和 d_{opt}

算法 2 kNN 最优子树搜索

(1) d_p^{kE} =Init(root,p)

push all sub-nodes of the root into Stack in decreasing order of the distance d_{opt} between the sub-node and p (the minimum d_{opt} on top)

(2) if Stack is null, then return

else current=pop(Stack)

(3) if $d_{\min}(p, current) > d_p^{kE}$, then goto step2

(4) if current.type is leaf then

for each point q in current

if $d(p,q) < d_p^{kE}$ and buffersize<k

insert q into buffer

bufffersize=bufersize+1

if buffersize=k then $d_p^{kE} = d(p,q)$

else if $d(p,q) < d_p^{kE}$ and buffersize=k

delete $o = \max_{t \in buffer} \{d(p,t)\}$ from buffer

insert q into buffer

$d_p^{kE} = d(p,q)$

(5) else if current.type is internal node then

for each node q in current

if $d_{\max}(p,q) \leq d_p^{kE}$ and $\| q \| \geq k$ then

$d_p^{kE} = d_{\max}(p,q)$

push all sub-node of current into Stack in decreasing order of d_{opt}

(6) goto step(2)

3 实验测试

实验的目的是测试 Z 树的矩形区域查询和 kNN 搜索的效率。计算机为神舟 715 笔记本电脑，内存 512MB，使用 C++ 语言编程。每个数据集为 100 000 个点，由随机数程序产生。

(1)测试矩形区域查询的效率。由于球形索引树在这方面的性能远低于矩形索引树，因此采用 X 树作为对比。图 2 给出了 Z 树和 X 树在搜索过程中查询存储页面的百分数。X 树需要查询的页面数随着维度的增加而快速增长，导致搜索效率的降低。这是因为 X 树中超级节点的数目和体积以及节点间的重叠区域较大，并且随着维度的增高而恶化。Z 树受维度影响较弱，性能超过 X 树。

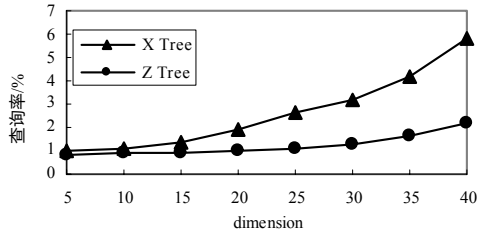

图 2 Z 树和 X 树搜索页面的百分数

(2)测试 kNN 搜索效率，使用球形化的 Z 树和最优子树搜索算法，以 SR 树作为对比组。测试结果如图 3 所示。SR 树执行时间随着维度的增长而迅速增加。Z 树的效率受维度的影响较弱。

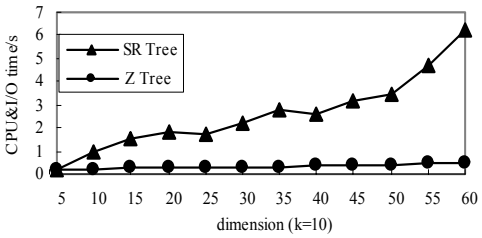

图 3 Z 树和 SR 树 kNN 搜索的执行时间

4 结语

本文提出了一个新的高维度数据索引结构 Z 树，能够高效地处理矩形区域查询和 kNN 搜索。它按照节点形状变化量优化数据的插入位置，使节点形状趋于合理；给出了一个新的无重叠分裂算法，减少超级节点的产生；引入动态剪枝和重新插入策略，压缩超级节点的数量和体积；提出矩形节点的球形化方法和最优子树搜索算法。实验表明 Z 树的矩形区域查询和 kNN 搜索效率远远高于 X 树和 SR 树。

参考文献

1 Beckmann N, Kriegel H P, Schneider R, et al. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles[C] //Proc. of ACM-SIGMOD Int'l Conf. on Management of Data, Atlantic City, USA. 1990: 322-331.

2 Berchtold S, Keim D A, Kriegel H P. The X-Tree: An Index Structure for High-dimensional Data[C]//Proc. of the 22nd Int'l Conf. on Very Large Databases, Mumbai (Bombay), India. 1996: 28-39.

3 White D A, Jain R. Similarity Indexing with the SS-Tree[C]//Proc. of the 12th Int'l Conf. on Data Eng., New Orleans, USA. 1996: 516-523.

4 Katayama N, Satoh S. The SR-Tree: An Index Structure for High-dimensional Nearest Neighbor Queries[C]//Proc. of 1997 ACM SIGMOD Int'l Conf. on Management of Data, Tucson, USA. 1997: 369-380.

5 Roussopoulos N, Kelley S, Vincent F. Nearest Neighbor Queries[C] //Proc. of ACM SIGMOD'95, San., USA. 1995: 71-79.