文章编号: 1001- 9081(2005)12- 2872- 03

ESR-Tree 一种多维对象的动态索引方法

徐 焕, 林坤辉
(厦门大学 软件学院, 福建 厦门 361005)
(jploxh@ sina com)

摘 要: 在研究 SR-tree(Sphere/Rectangle- tree)和 X-tree(eXtended node tree)的结构与性能的基础上, 针对 SR-tree分裂算法的不足, 改进了分裂算法, 结合两者的优点, 设计了一种新的多维索引结构 ESR-tree(Extended SR-tree)。实验表明, 随着数据量和维数的增多, ESR-tree的性能要优于 SR-tree和 X-tree。

关键词: 多维索引; 超球体; 超矩形; 超级节点
中图分类号: TP311 134 3 文献标识码: A

ESR-Tree: a dynamic index for multi-dimensional objects

XU Huan, LIN Kun-hui
(College of Software, Xiamen University, Fujian Xiamen 361005, China)

Abstract With study on the structure and performance of SR-tree(Sphere/Rectangle- tree) and X-tree(eXtended node tree), the split algorithm was improved to make up for the shortage of the SR-tree algorithm. A new multi-dimensional indexing structure ESR-tree(Extended SR-tree) was designed by combining the advantages of the both. With the increase of data amount and dimensions, experiments show that the performance of ESR-tree is much better than that of SR-tree and X-tree.

Keywords multi-dimensional; super-sphere; super-rectangle; super-node

1 ESR-tree的设计思想

采用不同的特征空间度量定义, 包络类型和数据切分方法, 可以得到不同类型的索引结构[1]。SR-tree(Sphere/Rectangle- tree)[2,6]在SS-tree[3]的基础上引入了超矩形, 采用超矩形和超球形结合的方法构造索引, 减少了区域重叠, 但是SR-tree和R* -tree[4]一样, 也没有避免区域重叠。另一方面, 除了在插入和删除操作后对包络的更新方法应同时更新超矩形和超球体外, SR-tree的插入、删除以及分裂算法完全取自SS-tree, 超矩形信息在选择子树和决定分裂策略时不起任何作用; 相反, 对分裂策略起决定作用的超球体, 却由于球体本身的几何特性, 不大容易产生无重叠的分裂。针对 SR-tree这一特点, 利用X-tree(eXtended node tree)中超节点减少重叠的思想[5], 设计了一种新的索引结构 ESR-tree(Extended SR-tree)。利用 X-tree的设计思想来扩展 SR-tree, 并改进了插入和分裂算法, 在 ESR-tree中通过引入超级节点, 使得新的索引结构充分利用了 SR-tree和 X-tree两者的优点, 弥补了 SR-tree的不足, 减少了区域重叠, 尽可能地产生无重叠分裂, 有效提高了多维索引的性能。

2 ESR-Tree的基本概念和主要算法

2 1 基本概念

定义 1 n维空间中的矩形 R可以由它的主对角线的两个顶点来定义:

$R = (L, U)$

在这里 $L = [l_1, l_2, \cdots, l_n]$, $U = [u_1, u_2, \cdots, u_n]$, 并且对于任何 $i \in [1, n]$都有 $l_i \leqslant u_i$。

定义 2 n维空间中的球体 S可以用它的球心和半径来定义:

$S = (C, r)$

这里 $C = [C_1, C_2, \cdots, C_n]$是球心坐标, r是球的半径。

定义 3 一个节点的超矩形包络 MBR定义为包围该节点的所有子树中的点的最小边界矩形。

定义 4 一个节点的超球体包络定义为包围该节点的所有子树中的点的超球体, 超球体包络是按下面的计算公式计算得到的, 它并不是最小边界球体。

中心坐标 $X = (X_1, X_2, \cdots, X_n)$的计算方法:

$$\frac{\sum_{k=1}^{m} C_k \cdot X_i * C_k \cdot w}{\sum_{k=1}^{m} C_k \cdot w}$$

$i = 1, 2, \cdots, n$, 代表了数据空间的维数;

$C_k \cdot X_i$代表第 k个孩子在第 i维上的中心坐标;

$C_k \cdot w$代表第 k个孩子的权重, 也就是对应子树中的点的数量, 子树只是一个叶子节点时, 将叶子节点中的点看作是一个退化了的球体。这里 $k = 1, 2, \cdots, m$是子树的序号。

半径 r的计算方法:

$r = \min(ds, dr)$

$ds = \max(d(S_k, C))$

$dr = \max(d(R_k, C))$, C是上面计算出来的球体的中心点。

定义 5 一个节点的包络定义为该节点的超球体包络和超矩形包络的相交区域:

$E = MBR \cap MBS$

2 2 节点结构

ESR-tree的结构同 X-tree的结构相似, 它的总体结构图

收稿日期: 2005- 06- 28; 修订日期: 2005- 09- 16 基金项目: "985工程"智能化国防信息安全技术资助项目

作者简介: 徐焕(1975-), 女, 河南禹州人, 硕士研究生, 主要研究方向: 网络多媒体、数据库应用; 林坤辉(1961-), 男, 福建惠安人, 副教授, 主要研究方向: 网络多媒体、数据库应用。

如图 1所示。它也由三类节点组成, 即叶子节点, 非叶子节点以及非叶子超节点。叶子节点的结构为:

$L: (E_1, E_2, \cdots, E_n)$

$E_i: (point, dataitem) (i = 1, 2, \cdots, n)$

其中 *point*是多维空间中的点 (特征向量), *dataitem* 是同该点相关的数据, 例如可以是该高维空间的点所代表的图像的文件名或者图像对象的对象标识 OID。

非叶子节点的结构为:

$N: (C_1, C_2, \cdots, C_n)$

$C_i: (S, R, w, child\_pointer, split\_history, nodesize) (i = 1, 2, \cdots, n)$

其中, *child_pointer*是指向子树的指针; *S*是 *child_pointer* 所指向的子树中所有点的包络球体, *R* 是子树中所有点的包络矩形; *w* 是子树中被索引点的数量; *split_history*是用于表示该入口项分裂历史的位向量, 该位向量的第 *i*位为 1, 表明该入口项曾经以第 *i*维为分裂维进行过分裂; *nodesize*该节点所占的外存页面数多少, *nodesize* > 1则说明该节点是超节点。

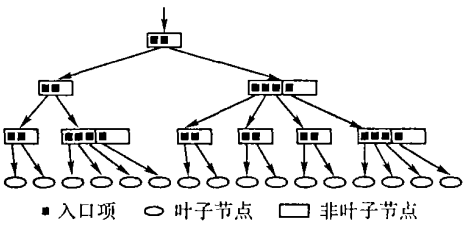

图 1　ESR-tree的总体结构

2.3　ESR-tree的插入算法和分裂算法

插入算法和分裂算法是 ESR-tree树中最重要的算法, 它们最终决定了 ESR-tree的结构, 从而决定了 ESR-tree的性能。ESR-tree采用基于"中心点"的插入算法: 当插入新数据时, 在所有子树中选择"中心点"距离新数据点最近的那棵子树作为新数据点要插入的子树, 递归进行直到找到适当的叶子节点。如果在插入过程中没有分裂, 则逆序更新访问路径上的超矩形和超球体包络即可。

由于强制重插策略能够对树的结构进行动态重组, 它对性能的改善已经在 R*-tree和 SR-tree中得到了证明。所以同 X-tree直接进行分裂相比, SR-tree在节点的入口项数目上溢时, 首先采用强制重插策略, 将部分入口项删除并重新插入到树中。如果该节点上已经进行过强制重插, 则按照 SS-tree的分裂策略进行分裂 (称之为 sphere-split)。由于球体的几何特性, 这种以提高空间利用率为目的的分裂策略极易导致包络区域之间较多的重叠。由于 ESR-tree中的包络是矩形包络和球形包络的相交区域, 所以, 如果超矩形包络存在无重叠的分裂, 则整个包络也一定存在无重叠的分裂。当 sphere-split 产生较多的重叠时, 则可以利用超矩形包络的信息进行无重叠 rectangle-split。但是, rectangle-split可能导致分裂后的节点中入口项数目下溢, 使树结构恶化, 同时降低了空间利用率。出现这种情况时, 停止正常分裂, 生成一个超节点; 如果被分裂节点已经是超节点, 则超节点所占的块数即 nodesize加 1。算法的形式如下:

算法 1: chooseSubtree

输入: 新插入的点 point

输出: 被选择的入口项在节点中的序号 (也就是子树的序号)

步骤:

1)计算 point同节点中的每一个入口项的中心点的距离, 假定距离最小的入口项为 entry[follow];

2)对该入口项中的 weight(子树中多维点的个数)值加 1;

3)扩张超矩形, 使新的矩形能包含 point;

4)按加权平均的方式计算新的球体的中心点; 扩张超球体, 新球体的半径是中心点到所有子树的超矩形的最大距离和到所有子树的超球体的最大距离中的最小值;

5)返回距离最小的入口项在节点中的序号 follow.

该选择子树算法的优点是, 在选择子树的过程中对包络球体和包络矩形进行更新, 在实现上比较直观容易。但这并不是包络更新算法的全部, 当插入导致子节点分裂时, 入口项本身的包络需要重新更新。

算法 2: sphere-split

输入: 待分裂的节点;

输出: 分裂所在的维, 最佳分裂方案;

步骤:

1) 设定 minCount 的值: minCount = (maxCount * MINFANOUT) /100; maxCount为节点中入口项的最大个数, MINFANOUT为节点扇出;

2) 计算分裂所在的维: 对每一维, 计算所有入口项中超球体的中心点在该维坐标的方差, 方差最大的维 dim 作为分裂所在的维[6];

3) 选择最佳分裂方案:

①在分裂所在的维, 按照中心点在该维坐标从小到大的顺序对所有的入口项进行排序;

②对所有的 totalEntries - minCount + 1种可能的分裂方案得到的两个新节点, 分别对每一维计算每个入口项的中心点在该维上坐标的方差[6], 然后将 2* *N* 个方差值累加, 这里 *N* 是数据空间的维数。方差之和最小的分裂方案就是最佳的分裂方案。

4)返回 dim 和最佳分裂方案。

算法 3: rectangle-split

输入: 待分裂的节点;

输出: 分裂所在的维, 最佳分裂方案;

步骤:

1)设定 minCount的值: minCount= 1;

2)将所有入口项的 history进行按位逻辑与操作, 如果所得结果 split_vector为 0, 说明不可能有无重叠分裂, 则直接退出, 进行相应的生成超节点的操作;

3)选择最佳的无重叠分裂进行的维: split_vector中位为 1的维都是无重叠分裂的维。对 split_vector中位为 1的每一维:

①将所有入口项中超矩形按矩形在该维的最小坐标值排序, 并对所有的 totalEntries - minCount + 1种可能的分裂方案, 计算分裂后的两个新节点的超矩形包络的边长之和;

②将所有入口项中超矩形按矩形在该维的最大坐标值排序, 并对所有的 totalEntries - minCount + 1种可能的分裂方案, 计算分裂后的两个新节点的超矩形包络的边长之和, 二者之和最小的维 dim就是分裂所在的维;

4)选择最佳的分裂方案:

①对所有入口项中矩形分别按照在 dim维的坐标最大值和最小值排序, 得到两个超矩形序列: low_mbrs和 up_mbrs;

②对所有的 2* (totalEntries - 2* minCount + 1)种可能的分裂方案, 分别计算出 overlap, deadspace, 它们分别代表两种排序方式下相应的两个超矩形包络之间的重叠和死区体积。重叠最小的分裂方案就是最佳分裂方案, 重叠相同时则比较死区体积的大小;

5)返回 dim 和最佳分裂方案。

上面的分裂算法中, 对分裂维的选择要求所有的入口项很容易地分开, 而对分裂方案的选择则使两个新节点中的入口项分别聚簇在一起。rectangle_split中 minCount为 1, 也就是没有入口项个数的下限要求, 这是产生无重叠分裂的要求,

也是导致树结构不平衡的根源。

算法 4: split

输入: 待分裂的节点;

输出: 如果分裂了,则返回指向新的节点的指针, 同时设置分裂标志位。否则生成超级节点;

步骤:

1)进行拓扑分裂 sphere-split;

2)判断新节点的超矩形包络之间的重叠, 如果重叠小于预定的最大值, 则转到 5), 否则转到 3);

3)进行无重叠分裂 rectangles-split;

4)如果生成的两个节点的扇出小于最小空间利用率的要求, 则生成超节点, 返回 false, 否则转到 5);

5)按照选定的分裂维和分裂方案进行分裂, 生成两个新节点, 更新入口项的分裂历史并返回 true。

算法 5: insert

输入: 一个新的叶子节点入口项

输出: 如果新的叶子节点入口项的插入导致了该节点的分裂, 则返回 SPLIT标志和指向新节点的指针, 以及分裂所进行的维; 如果该节点变成了超节点, 则返回 SUPER NODE, 其他情况返回 NONE。

步骤:

1)调用 chooseSubtree;

2)对选择出的子树(相应的入口项为 follow)调用 insert;

3)如果 2)返回的是 SUPERNODE或者 NONE, 则直接返回 NONE, 否则转到 4);

4)更新入口项 follow中的超矩形包络, 超球形包络以及 weight, 并更新入口项的分裂历史;

5)为 2)生成的新节点构造一个新的入口项, 并将该入口项加到节点中。如果该节点中入口项的数目已经达到 maxCount, 则调用 overflowTreatment算法进行节点的溢出处理。否则简单的将新的入口项插入到该节点中, 并返回 NONE。

算法 6: reinsert

步骤:

1)计算 reinsertCount= (maxCount+ 1)* reinsertFactor(重插因子);

2)计算该节点的中心点, 并计算每个入口项的中心点到节点的中心点之间的距离, 然后按距离从大到小的顺序排序;

3)选择排序后距离最大的 reinsertCount个入口项重新插入到树中。

算法 7: overflowTreatment

如果该节点在插入一个点过程中第一次调用 overflowTreatment, 则调用算法 reinsert, 否则调用 split。

3 测试结果及性能分析

3.1 测试条件

测试评价一个多维索引结构的性能需要一个合理的大规模测试数据集。对图像检索系统而言, 一个好的测试数据集必须满足的条件是: 1)规模足够大, 以测试检索系统中高维索引结构的可扩展性; 2)图像内容丰富, 以测试图像特征的有效性和系统的总体性能。为了测试 MPEG-7标准的有效性, MPEG-7专家组提供了一组测试集, 包括音频、静止图像和视频三个部分。其中, 静止图像部分包括 7000幅从各种渠道得到的各种格式的静止图像。但是由于该数据集的规模不够, 不能很好地测试我们的索引结构的可扩展性。我们选用了 http://www.stlrag.com/~berchtol/papers/founew.normiert.gz上提供的特征数据集。该数据集曾用作 X-tree的测试数据集, 它的特点是特征向量是图像特征的傅立叶系数表示, 因此可以将特征向量的后若干维简单地去掉, 仍然能够得到另一个有效的特征向量。该特征向量集的最大维数是 32维, 因此可以从中得到 1到 32维中任意维数的特征向量。

3.2 测试环境

本文中所做的所有测试使用的软硬件环境如下:

1)硬件环境: PⅣ 1.2GHz CPU, 256MB SDRAM 内存, 80GB硬盘, IBM 兼容机。

2)操作系统平台: Microsoft Windows XP。

3)编程环境: Microsoft VC++ 6.0编译器。

3.3 测试方案

每个数据集上随机选出 10个点, 作为 10个不同查询的查询输入点。每个查询执行 10遍, 查找出同查询输入点距离最近的 8个点。由于所测试的索引结构都是基于外存的索引结构, 因此内存和外存之间的 I/O操作是影响查询性能的最主要的因素。而每次查询执行之后, 查找路径上的节点都被读进了内存中, 这样下次查询时, 就减少了外存到内存的读操作, 这样得到的查询响应时间就是所谓的热结果, 但是这种热结果反应不出基于外存的索引结构的性能特点, 因此只使用冷结果作为评价算法性能的依据: 每次查询进行之后, 都有相应的清理内存的动作。

3.4 测试结果分析

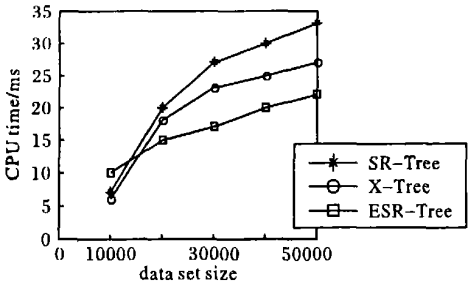

图 2 CPU 时间随数据集大小变化曲线

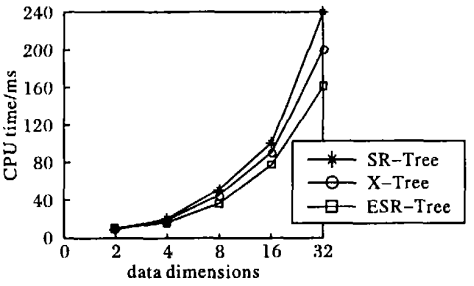

图 3 CPU 时间随着维数变化曲线

图 3显示了数据集大小一定的情况下(包含了 5万个多维空间中的点), 索引结构的性能随着维数的变化而变化的曲线图。由图可以看出, 三种索引结构在数据空间维数较低(2维、4维)的时候表现出了几乎完全相同的性能。随着维数的增加, ESR-tree和 X-tree体现出了明显的性能优越性。在维数为 32维的时候, ESR-tree的查询响应时间分别是 SR-tree的 67.78%和 X-tree的 87.65%; 而 X-tree的查询响应时间是 SR-tree的 77.34%。图 2~图 5的性能曲线说明, 在 I/O次数大致相同的情况下, ESR-tree需要更多的 CPU 时间。这是因为 ESR-tree有更复杂的节点结构。当数据集很大的时候(5万), ESR-tree的性能优越性很明显, ESR-tree的查询响应时间大概是 SR-tree的 66.8%, 是 X-tree的 85.5%。正如文献[8]中分析, X-tree的性能要优于 SR-tree, 但是我们的测

(下转第 2878页)

适合于搜索有适当广泛性并且精度要求较高的查询。

3 结语

召回率和精度是 Web搜索引擎的主要衡量指标,而 Web信息的超文本组织形式,网页的半结构化表现方式以及动态网页技术的广泛应用,使得 Web上网页的数量剧增,如何提高检索精度成了搜索引擎面临的关键问题之一。

随着网络技术的不断发展,使越来越多不同类型和格式的信息能够在 Web上传输,Web上出现了大量对搜索引擎不可见的信息资源。如何在大量异质异构的 Web资源中找到用户最需要的信息,向搜索引擎的检索模式提出了新的挑战。

参考文献:

[1] PAGE L, BRIN S, MOTWANI R, *et al*. The PageRank citation ranking: Bringing order to the Web[R]. Technical report, Stanford University, Stanford, CA, 1998.

[2] KLEINBERG JM. Authoritative sources in a hyperlinked environment [J]. Journal of the ACM, 1999, 46(5): 604-632.

[3] WEN-SYAN LI, K. SELÇÀUK CANDAN, QUOC VU, *et al*. Query Relaxation by Structure and Semantics for Retrieval of Logical Web Documents[J]. IEEE Transactions on Konwledge and Data Engineering, 2002, 14(4): 768-792.

[4] KEISHI TAJIMA, KENJI HATANO, TAKESHI MATSUKURA, *et al*. Discovery and retrieval of logical information units in Web[A]. Proceedings of the 1999 ACM Digital Libraries Workshop on Organizing Web Space[C]. Berkeley, CA, USA, 1999.

[5] NECIP FAZIL AYAN, WEN-SYAN LI, OKAN KOLAK. Automating extraction of logical domains in a Web site[J]. In International Journal of Data and Knowledge Engineering, Elsevier Science, 2002, 43(2): 179-205.

[6] WEN-SYAN LI, NECIP FAZIL AYAN, OKAN KOLAK, *et al*. Constructing multigranular and topic-focused Web site maps[A]. Proceedings of the 10th World-Wide Web Conference[C]. HongKong, China, 2001.

[7] KAO HY, LIN SH, HO JM, *et al*. Entropy-Based Link Analysis for Mining Web Informative Structures[A]. Proceedings of the ACM 11th International Conference[C]. 2002.

[8] CAI D, HE XF, WEN JR, *et al*. Block-level Link Analysis[Z]. Microsoft Research paper, 2004.

[9] Nadav Eiron IBM Almaden Research Center Untangling compound documents on the Web[A]. Conference on Hypertext and Hypermedia archive Proceedings of the fourteenth ACM 2003[C]. 2003.

[10] TAJIMA K, MIZUUCHI Y, KITAGAWA M, *et al*. Cut as a Querying Unit for WWW, Netnews, E-mail[A]. In Proceedings of the 9th ACM Conference on Hypertext and Hypermedia (HYPERTEXT'98)[C]. 1998, 235-244.

[11] HENZINGER MR, BHARAT K. Improved algorithms for topic distillation in a hyperlinked environment[A]. Proceedings of the 21st International ACM SIGIR Conference on Research and Development in IR[C]. 1998.

[12] LEMPEL R, MORAN S. The Stochastic Approach for Link-Structure Analysis(SALSA) and the TKC Effect[A]. Porceedings of 9th International World Wide Web Conferrence[C]. 2000.

[13] MIZUUCHI Y, TAJIMA K. Finding context paths for Web pages [A]. Proceedings of the Tenth ACM Conference on Hypertext and Hypermedia[C]. 1999. 13-22.

(上接第 2874页)

试结果表明并没有数量级上的变化。X-tree的查询响应时间大概是 SR-tree的响应时间的 72.27%。由于 SR-tree的各个子树的包络区域之间有较大的重叠,而 ESR-tree的各个子树的包络区域之间的重叠则较少或者没有,重叠所造成的对多条路径的查找操作就会带来更多的 I/O次数,所以 ESR-Tree需要更少的 I/O操作。

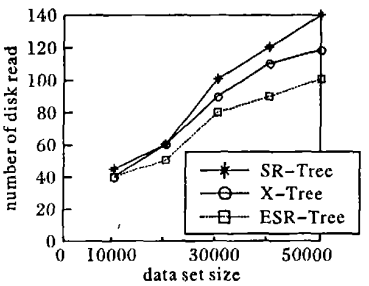

图 4 I/O次数随数据集大小变化曲线

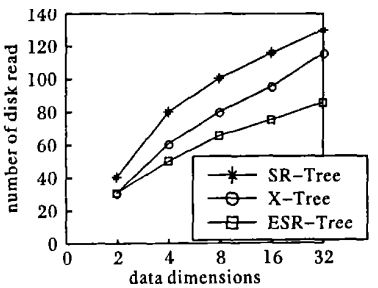

图 5 I/O次数随数据维数变化曲线

ESR-tree采用基于中心点的选择子树策略这种策略同 X-tree以及 R* -tree所采用的通过比较重叠区域的变化和超矩形体积的变化来选择适当的子树相比,需要更少的计算。另一方面,ESR-tree采用的是基于中心点的方差的节点分裂策略,这种策略同 X-tree采用的根据超矩形在两种排序方式下的分裂后的边长,重叠大小和死区体积来决定分裂策略相比,也有更少的计算次数。

通过实验分析,ESR-Tree的性能比 X-Tree和 SR-Tree有所提高。数据集越大 ESR-Tree的优越性越明显。

参考文献:

[1] WHITE DA, JAIN R. Similarity Indexing: Algorithms and Performance[J]. Proceedings of SPIE Storage and Retrieval for Image and Video Databases[C]. San Jose, CA, 1996, Vol 2670. 62-73.

[2] KATAYAMA N, SATOH S. The SR-tree: an index structure for high-dimensional nearest neighbor queries[A]. Proceedings of the 1997 ACM SIGMOD Intl Conference On Management of Data[C]. 1997.

[3] WHITE DA, JAIN R. Similarity indexing with the SS-tree[A]. Proceedings of the 12th Intl Conference on Data Engineering [C]. 1996. 516-523.

[4] BECKMANN N, *et al*. R* -tree: An efficient and robust access method for points and rectangles[A]. Proceedings of the 1990 ACM SIGMOD Intl Conference On Management of Data[C]. 1990.

[5] BERCHTOLD S, KEIM DA, KRIEGEL HP. The X-tree: an index structure for high-dimensional data[A]. Proceedings Of the 22nd VLDB Conference[C]. 1996. 28-39.

[6] KATAYAMA N. SHIN'ICHI SATOH. The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries[DB/OL]. http://www.informatik.unitrier.de/~ley/db/conf/sigmod/, 2002-02-02.

[7] LIN KI, JAGADISH HV, FALOUTSOS C. The TV-tree: An index structure for High-Dimensional Data[J]. VLDB Journal, 1994, 3(4): 517-542.

[8] BOHM C, BERCHTOLD S, KEIM DA. Searching in high-dimensional spaces: Index structures for improving performance of multimedia databases[J]. ACM Computing Surveys, 2001, 33(3): 322-373.