度量空间中高维索引结构回顾*)

刘芳洁　董道国　薛向阳

(复旦大学计算机科学与工程系　上海200433)

Review of High Dimensional Index Structures in Metric Spaces

LIU Fang-Jie　DONG Dao-Guo　XUE Xiang-Yang

(Dept. of Computer Science and Engineering, Fudan University, Shanghai 200433)

Abstract　Fast searches and query operations in high dimensional databases require efficient index structures. Among a variety of index structures, the index structures in metric spaces are very useful. They can be used in an extensive field, such as searching for protein molecular chains with certain sequences in Computational Biology and matching a given strings fuzzily in Text Retrieval. In this paper, the features of index structures in metric spaces are analyzed and subsequently a further classification is given to these index structures. Finally, some representative index structures are introduced in detail.

Keywords　Metric space, Index structure, Similarity-based query

1　引言

近年来，高维数据库的应用得到快速的发展，如海量的多媒体数据库、大规模的文本数据以及生物信息学中庞大的DNA数据库等，这些信息一般使用特征抽取等方法映射为高维数据，然后通过计算这些高维数据之间距离实现相似性查询。例如，对于图像数据，往往采用颜色直方图来表征一幅图像，当需要从数据集查找与给定图像相似的图像时，通过计算颜色直方图之间的距离实现查询。在这种背景之下，高维数据索引结构和适用于高维索引结构的近似查询算法得到了人们的极大重视。

在已提出的众多高维索引结构中，有的特定工作在向量空间中，而有的是针对度量空间而设计。在过去的研究中，人们对于向量空间索引结构做了大量的工作，在文[24]中已经对向量空间中索引结构做出了比较全面的回顾。相比较而言，人们对度量空间索引结构的重视程度要小很多。但实际上，向量空间可以被看作是带有坐标信息的度量空间的特例，因此相比之下在广义的度量空间下的索引结构具有更为普遍的适用范围。尤其是在某些领域，向量空间索引结构很难或不再适用，例如：在生物信息学中常需要从庞大的DNA数据库中快速查找具有给定顺序的分子片断；在允许的错误范围内从大规模的文档集中查询和给定字符串相匹配的文档片断等[8]。由于向量空间索引结构无法适用于这些情况，这时都必须借助于度量空间索引结构。

本文将主要对已提出的度量空间索引结构做出回顾，将介绍度量空间以及度量空间索引结构的基本概念，给出在度量空间上索引结构的分类方法，介绍度量空间中一些有代表性的索引结构。

2　度量空间的基本概念

2.1　度量空间

令集合W表示所有对象的集合。令需要查询的所有对象构成的集合表示为U，U是W的有限子集，大小为n，$n=|U|$。可以简单地把U看作存放所有数据的数据库。定义函数

$d: W\times W\rightarrow R$

为U中任意两个对象之间距离大小的一个函数(距离越小，表示两个物体之间的相似程度越大)。函数d具有下列基本性质[8]：

①非负性：$\forall x,y\in W, d(x,y)\geqslant 0$

②对称性：$\forall x,y\in W, d(x,y)=d(y,x)$

③自反性：$\forall x\in W, d(x,x)=0$

④三角不等式性质：$\forall x,y,z\in W, d(x,y)\leqslant d(x,z)+d(z,y)$

⑤在绝大多数情况下满足严格非负性：

$\forall x,y\in W, x\neq y\Rightarrow d(x,y)>0$

满足以上性质的(W,d)对被称为一个度量空间。需要指出的是，在度量空间中，各种高维索引结构将主要利用到性质④，以减少搜索范围，提高查询效率。

当性质⑤不能满足的条件下，这样的空间被称为伪度量空间。但是为了简单起见，在实际的工作中是不考虑伪度量空间的，而只需把所有相互距离为0的对象简单看作一个对象就可以了。这是因为当性质④成立时，可以很容易证明$d(x,y)=0\Rightarrow\forall z, d(x,z)=d(y,z)$。

2.2　基本查询方式

在度量空间中有下面两种基本的查询方式：

①范围查询(RQ：Range Query)。给定查询数据q与查询距离门限r，从数据库中找出所有与q距离小于r的数据：

$$RQ(q,r)=\{o\in DB\mid d(o,q)\leqslant r\}$$

②k-近邻查询(kNNQ：k Nearest-Neighbor Query)。给定查询数据q及正整数k，从数据库中找出距离q最近的k个数据：

$$KNNQ(q,k)=\{o_0\cdots o_{k-1}\in DB\mid \forall e\in DB\setminus\{o_0\cdots o_{k-1}\}, d(o_i,q)\leqslant d(e,q), 0\leqslant i\leqslant k-1\}$$

*)本文得到国家自然科学基金资助项目(60003017, 69935010)、国家863高科技发展计划资助项目(2001AA11 4120)、上海市政府资助项目(01QD14013, 015115044)等资助。刘芳洁　硕士研究生，主要从事高维数据索引结构研究。董道国　博士研究生，主要从事高维数据索引结构研究。薛向阳　教授，博士生导师，主要从事基于内容的多媒体信息检索技术研究。

当 $k=1$ 时，又称为‘最近邻查询’。

其中范围查询又是最基本的查询方式，k-近邻查询可以通过范围查询构造。很显然，任何一种查询都可以通过顺序扫描数据集 W 中的所有数据得到结果，但是计算度量空间中两个对象之间的距离被认为是代价高昂的。我们的目标是希望为整个数据库建立一个较好的索引结构，以减少查询比较的次数。

2.3 向量空间与度量空间中的索引结构的异同

向量空间和度量空间索引结构有着不同的侧重点，但也有相通之处。它们的异同简单总结如下：

①向量空间被看作是带有坐标信息的特殊的度量空间。在一定条件下，这两类空间是可以相互转换的。快速映射算法(FASTMAP)[12]就是一种将度量空间中的对象转换为具有低维坐标的向量空间中的点的算法。而当我们在向量空间中定义好一个距离函数而只取物体之间的距离信息时，就将向量空间转换成了度量空间。因此不难看出，在度量空间中的索引结构比在向量空间中具有更普遍的适用范围，所用到的信息更少，难度也会更大。

②进行相似性查询时，在度量空间中算法执行的主要代价被认为是计算距离函数的 CPU 代价，因此度量空间索引结构主要考虑的是减少距离函数的计算次数。而在向量空间查询的主要代价被认为是读取磁盘的 I/O 代价，向量空间中的索引结构主要考虑的是减少磁盘的 I/O 次数。

③在度量空间索引结构中，为了实现相似性查询，唯一用到的方法就是距离函数的三角不等式性质。而在向量空间中，除了利用距离函数外，更主要的可利用信息是数据的坐标信息。因此，在向量空间中的索引结构上设计查询算法时，比在度量空间中具有更大的灵活性，因为它不仅可以利用距离信息，还可以利用物体的位置(坐标)信息，比如普遍使用的 K-D-tree[3]、R-tree[16]等，它们都广泛使用了数据坐标信息。

3 度量空间中索引结构的分类

在多年的研究中，人们提出了多种适用在度量空间中的索引结构。根据这些索引结构的特点和不同的出发点有以下几种分类方法：

①根据索引创建时数据组织形式，可以分为静态结构类和动态结构类。静态结构类是指用批处理的方式将全部数据构建成索引，不能支持或不能有效地支持动态的插入和删除，如 SA-tree[15]；动态结构是指数据依次插入生成的索引结构，如 M-tree[9]、Slim-tree[21]等。

②根据索引的组织形式，可以分为树形结构类和非树形结构类。树形结构类指索引结构是按照树的形式组织的，如 BK-tree[6]、VP-tree[22]等；非树形结构类是指索引结构不是按照树的形式组织的，如 FQA[7]、AESA[19]等。

③根据索引结构使用的距离定义是离散型还是连续型，可以分为离散距离类和连续距离类。离散距离类的索引结构是指所使用的距离函数是离散的，如 BK-tree、FQ-tree[2]等；连续距离类的索引结构是指所使用的距离函数是连续的，如 VP-tree、M-tree 等。

4 度量空间中代表性的索引结构

限于篇幅的原因，本文无法对所有度量空间索引结构一一介绍，只是选取一些具有代表性的索引结构给予比较详细的描述。

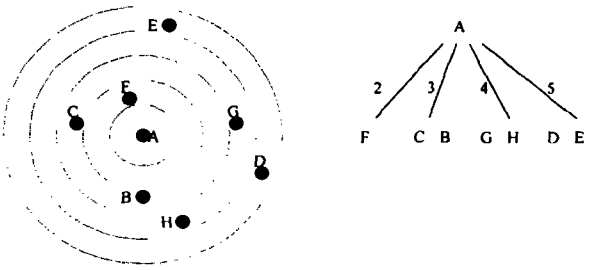
4.1 BK-tree 类

BK-tree 类的索引结构都是针对离散的距离函数而设计的。最早的 BK-tree 类结构是由 Burkhard 和 Keller 在1973年提出的 BK-tree(Burkhard-Keller Tree)，其后人们在它的基础上又做出改进，但大都在结构上保持了相似的特点。

(1)BK-tree[6]

BK-tree 是最早提出的度量空间树状索引结构，适用于离散的距离函数。其具体构造方法为：从数据集中任意选取元素 $p\in W$ 作为索引树的根节点。对于任意离散距离 $i>0$，定义 $W_i=\{w\in W, d(w,p)=i\}$，即 W_i 为距离根节点为 i 的所有点的集合。对于每一个非空的 W_i，循环为其建立 BK-tree，直到剩余元素数目小于一个预定义的阈值 b 时停止，其中每一个被选择为子树根节点的元素称为支点(pivot)。图1就是一个 BK-tree 的简单例子，它反映了 BK-tree 的特点。

当给定一个查询 q 和范围 r 时，BK-tree 的查询算法是从根节点出发，找出所有满足不等式 $d(p,q)-r\leqslant i\leqslant d(p,q)+r$ 的子节点 i，递归直至到叶子节点。对路径上所有的支点和叶子节点中的每一个元素都要与 q 进行比较，看是否满足不等式 $d(q,u)\leqslant r$，记录下所有满足不等式的点作为结果返回。在查询中，三角不等式性质保证了所有满足查询条件的点均会被检索到。



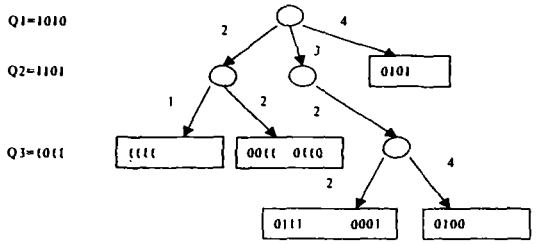
左边是原始数据，右边是 BK-tree 的第一层

图1 简单的 BK-tree

(2)BK-tree 类的其它几种索引结构

在 BK-tree 提出之后，人们对它的结构进行了各种改进，以期得到更好的性能，下面列出主要的几种：

·FQ-tree(Fixed-Queries Tree)[2]。FQ-tree 对 BK-tree 做出的改进主要体现在两点：第一，数据库中的所有数据都是存放在叶子节点上，在非叶子节点中存放的可能是数据库中的数据，也可能是其它的关键字；第二，位于同一高度上的所有中间节点存放的都是同一个关键字，这是 FQ-tree 最新颖的地方。由于同一层的中间节点所对应的是同一个关键字，因此在检索时可以减少距离比较的次数，但是同时也会增加遍历的节点数，即树的高度会增加。图2是一个简单 FQ-tree 的例子。



叶子节点所包含数据的最大数目为2

图2 使用 Hamming 距离定义的 FQ-tree

·FHQ-tree(Fixed-Height FQT)[1]。FHQ-tree 在 FQ-tree 的基础上又做出了改进，即强行规定所有的叶子节点都必须在同一高度上。这样，在进行查询时，对于同一层的节点可以

统一考虑，而不必再分为叶子节点和中间节点两种情况考虑。其代价为某些叶子节点不得不增加高度以满足这一条件。图3是FHQ-tree的一个例子。

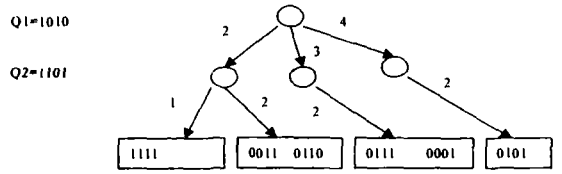


图3 FHQ-tree

·FQ-array(Fixed-Queries Array)[7]。FQ-array是一种数组查询结构，其实质是对FHQ-tree的一种压缩存储格式。它的构成是将一棵已有的FHQ-tree的叶子节点从左到右依次存放入数组的每一个元素，并且记录下从根节点至叶子节点所历经的所有边的权值。利用FQ-array进行查找时实际是进行二分查找，比较FHQ-tree而言需要多花O(log n)的时间，但是并没有增加距离函数的计算次数。图4是FQ-array的一个例子。

1111	0011	0110	0111	0001	0100	0101
2	2	2	3	3	3	4
1	2	2	2	2	2	2

图4 FQ-array

4.2 BS-tree类

BS-tree类的索引结构不仅可用于离散的距离函数，也适用于连续的距离函数，比BK-tree类的索引结构有更广泛的适用范围。

(1)BS-tree[13]。BS-tree(Bisector Trees)是BS-tree类中最早被提出的索引结构，它是一棵二叉查找树。其构造方法为：从根节点出发，为每一个节点选择两个中心数据点 c_1 和 c_2，依照距离中心点 c_1 和 c_2 谁更近的原则，依次将剩余元素分别放入节点的左右子树中；为每一个中心点记录下它的覆盖半径，即在它的子树中离它最远元素的距离。BS-tree的查找过程很简单，在每一层上对于中心点 c_i，如果 $d(q,c_i)-r$ 小于 c_i 的覆盖半径，则进入 c_i 对应的子树进行查找，否则删除这一分支。图5给出了BS-tree的一个例子。

(2)其他几种BS-tree类的索引结构。

·VT(Voronoi Tree)[11]。具有和BS-tree类似的结构，改进之处在于当必须生成一个新的节点以容纳新插入的一个元素时，不仅把新插入的元素放入该新节点中，而且把它父节点中所包含距离最近的元素也放入该节点。实验证明，它比BS-tree具有更高的查询效率和更好的平衡性。

·GHT(Generalized-Hyperplane Tree)[18]。在构造上与BS-tree完全类似，不同之处在于查询时不是使用中心点的覆盖半径而是两个中心点之间的超平面来选择分支：如果 $d(q,c_1)-r<d(q,c_2)+r$，则进入 c_1 的子树查找；如果 $d(q,c_2)-r\leqslant d(q,c_1)+r$，则进入 c_2 的子树查找。需要注意的是有可能会查找所有的分支。

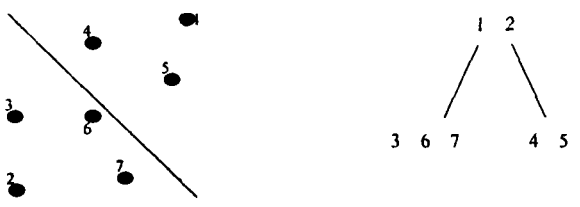


图5 一棵只有一层的BS-tree

·Monotonous BS-tree[17]。对BS-tree做出的改进在于当为每个节点选取两个中心点时，其中一个中心点是父节点中的中心点。这样做的好处是查询时随着树的深度的增加，中心点的覆盖半径会递减。

·GNA-tree(Geometric Near-neighbor Access Tree)[5]。本质思想与BS-tree完全一致，只是对BS-tree做出了扩展，即为每个节点所选取的中心点不是两个而是 m 个，相应地我们得到的索引树也不再是二叉树而是 m 叉树。

4.3 VP-tree类

(1)VP-tree[22](Vantage-Point Trees) 是一种基于连续距离函数的二叉平衡树，它的思想在于如何将二分查找用至只有距离信息的多维度量空间中。类似于KD-tree[3]，VP-tree的每一个节点都相当于对空间的一个划分，所不同的是VP-tee使用的是距离信息而不是坐标信息。具体的构造方法为：选取一个元素 p(vantage-point)作为根节点，将剩余的所有元素根据至 p 的距离远近分别均匀地放至 p 的左右子树中，离 p 较近的一半元素放入 p 的左子树中，离 p 较远的一半元素放入 p 的右子树中；设 M 为左右子树的临界距离值，将它作为节点的附加信息保存。分别对左右子树重复以上过程，直到节点中的元素数目小于阈值 k。

在VP-tree上的查询同样是利用三角不等式性质减少查询分支。如果 $d(q,p)-r\leqslant M$，则进入 p 的左子树；如果 $d(q,p)+r>M$，则进入 p 的右子树；类似于BS-tree，我们同样可能会查找它的所有分支。

构造VP-tree的空间复杂度为O(log n)，时间复杂度为O(n log n)，在查询半径很小的条件下查询间复杂度为O(log n)。图6就是一个VP-tree的例子。

(2)其它几种VP-tree类的索引结构

· MVP-tree(Multi-Vantage-Point Tree)[4]：MVP-tree对VP-tree的结构做了少许改变。它将中间节点所包含元素(Vantage Point)的数目由VP-tree的一个增加到两个，其中每个元素的划分份数为 m。这样提高了每个节点的输出能力，降低了树的高度，从而减少了距离函数的计算次数。文[15]中指出，构造一棵MVP-tree所需要的距离函数的计算次数为 $O(n\log_m n)$，比VP-tree少了 $\log_2 n$，并且查询性能上也有了很大的提高。

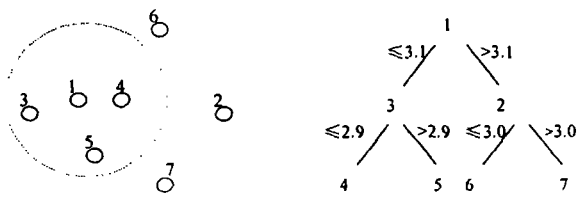


图6 选择元素1为根节点的VP-tree

·VPF(Vantage Point Forest)[23]。VPF是VP-tree的又一个变种，它基于VP-tree的一个缺点做出改进：在VP-tree中，当一个查询点距离支点很近的时候，将不得不分别进入支点的左右子树进行查询。因此，VPF的主要思想是在树的每一层上，将所有距离支点适中的元素独立地选取出来，并用这些元素构造第二棵树，以此类推得到有多棵树构成的索引结构。这里"适中"的具体定义为：设 r_0 和 r_* 分别代表距离支点最近和最远的元素到支点的距离，如果距离 M 满足 $d(p,r_0)+\delta\leqslant M\leqslant d(p,r_*)-\delta$，则 M 被认为是适中的距离。当在进行范围查询时，如果查询半径 $r'\leqslant(r_*-r_0-2\delta)/2$，则避免了回溯，否则就不得不搜索所有的树。文[23]中指出，在VPF上进

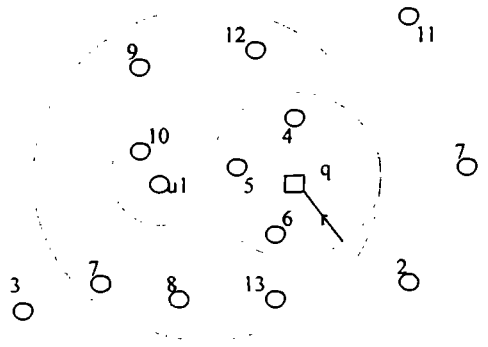
行查询时所需要的距离函数计算次数为 $O(n^{1-p}\log n)$，其中 p 依赖于查询半径 r^*，并满足 $0<p<1$。只有在很小的条件下，VPF 才有比较好的性能。

4.4 AESA 类

严格说来，AESA[19](Approximating Eliminating Search Algorithm)并不是索引结构，而只是一种查询算法。它所用到的数据结构是一个 $n*n$(n 为数据库中元素的个数)的矩阵 E，E 中的元素表示数据库中两两元素之间的距离。在查询时，首先任意选取元素 $p\in U$，计算距离 $r_p=d(p,q)$，去除数据库中所有不满足条件 $r_p-r\leqslant d(u,p)\leqslant r_p+r$ 的元素 u，由于元素与元素之间的距离都是在构造矩阵时得到，所以这里我们只需要进行一次距离的计算。重复以上过程，直到剩余的元素足够少。最后将剩余的所有元素直接与 q 做比较，得到结果。

AESA 的查询时间复杂度为 O(1)，它的主要缺点在于它的构造所需要的时间复杂度和空间复杂度都是 $O(n^2)$，对于海量数据库来说这是不可接受的。

图7是一个 AESA 的例子。



第一次随机选取的元素为 u1，大圈外的的元素和元素 10为第一次舍弃的元素。

图7 一个 AESA 的例子

LAESA(Linear AESA)[14]是 AESA 的一个变种，它使用了 k 个固定的支点，这样在构造时空间和时间的复杂度就降为了 $O(kn)$，但是在查询时效率有所降低。

4.5 M-tree 类

(1)M-tree[9] 是度量空间中第一个真正动态的索引结构，它不仅考虑了减少距离函数的计算次数，而且考虑了 I/O 性能。M-tree 的结构特点非常类似于 R-tree[16]，具体体现在以下几个方面：

·它是平衡树，所有指向实际数据的指针都存放在叶子节点上。

·每个节点所存放的项的数目最大不得超过 N。

·在动态插入一个数据项时，选择最"相近"的分支往下插入；当引起节点溢出时，要对节点进行分裂。

它区别于 R-tree 的方面有：

·在中间节点中存放的是有代表性的数据项和该数据项覆盖其所有子数据项需要的最小半径。

·只用到了距离信息，没有坐标信息。

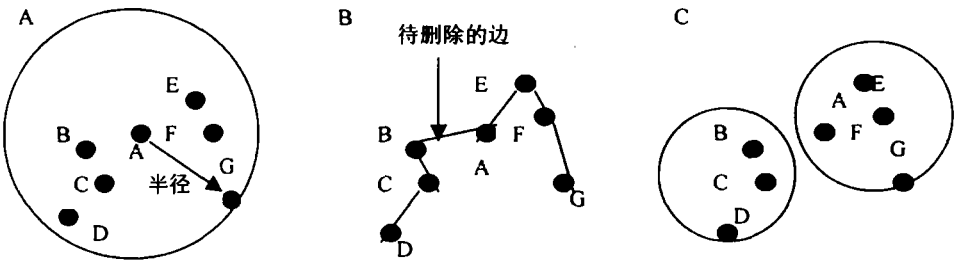
通过分析可以看出，M-tree 是将向量空间索引结构的动态性与平衡性和度量空间索引结构的距离定义结合起来，以期达到较好的检索性能。如果给 M-tree 加上坐标信息，实际上就成为了向量空间中的另一种索引结构—SS-tree[20]。

(2)Silm-tree[21] 是在 M-tree 的基础上对它进行了优化，具体体现在：

·提出了一种基于 MST(Minimal Spanning Tree)新的节点分裂算法，如图7所示。首先为待分裂节点中的所有元素构造一棵最小生成树(边的权值以距离表示)，选择权值最大的边(AB)进行切分；将剩余的两个连通图之中的元素分别作为两个新节点之中的元素；最后为每个新节点选取有代表性的元素和最小覆盖半径。

·提出了一种衡量度量空间中空间重叠程度的办法，并且利用这种衡量标准给出了一种在节点之间移动数据项的算法，以期达到降低节点相互重叠程度的目的。具体算法请参见文[21]。

(3)M^2-tree 在 M-tree 的基础上，有人提出了 M^2-tree[10]。传统的 M-tree 只能根据数据的某一个固定的特征进行检索，而 M^2-tree 可以同时根据数据的多个特征对数据进行检索，即使用了多种距离函数，因此在节点的数据项中包含了多个特征以及对应于每个特征的距离值。M^2-tree 可以应用在多媒体数据库中进行复杂查询。



A 表示分裂前的节点，B 表示构造的 MST，C 表示分裂后的节点

图8 基于 MST 的节点分裂示意图

4.6 SA-tree

与传统索引结构划分数据集的思想不同，SA-tree(Spatial Approximation Tree)[15]的基本思想是随着查询的进行，可以不断地在空间位置上越来越靠近查询点，以此得到查询结果。其构造方法为：选择一数据点 a 作为根节点，并求得它的邻居节点集合 N，N 中的每一个数据点 b 都满足条件 $\forall r\in N, d(a,b)<d(b,r)$；将 N 中的每一个数据点都作为 a 的一棵子树插入；把剩余的数据节点分别插入距离最近的子树中，并以原 N 中的数据点为根节点重复以上过程。显然，它是一种完全静态的索引结构。

在进行范围查询时，我们首先求出 $\{a\}\cup N$ 中距离查询点 q 最近的点 c 和 $d(q,c)$，对于 $\forall b\in N$，如果 $d(b,q)\leqslant d(q,c)+2r$，则进入 b 的子树查询，否则删除该路径。图8就是一个构造好的 SA-tree 和查询的例子。

总结 在很多领域，度量空间已经越来越普遍地被作为相似性检索的模型。从上面的介绍可以看出，在近30多年的时间里，人们对度量空间下的索引结构进行了大量的研究，并提出了众多的索引结构和算法，它们各有优缺点。本文只是选取

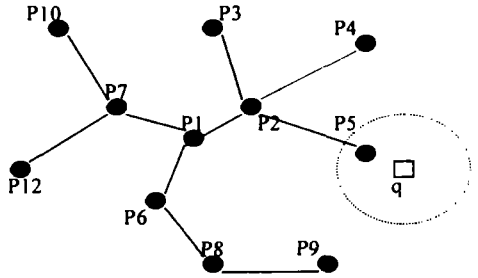


图9 以 P1为根节点的 SA-tree 和查询 q

了其中一些具有代表性的索引结构进行了概括性的介绍，而不去评价它们孰优孰劣。表1给出了一些索引结构的复杂度。事实上，到目前为止，度量空间高维索引结构仍没有得到广泛的应用，这是因为目前索引结构的性能都受到以下两个因素的制约：①随着范围查询半径的增大，查询的时间复杂度迅速上升；②随着维数的增加，索引结构的性能也迅速下降。

在今后的研究工作中，有以下几个方面是需要注意的：①索引结构可以完全动态地支持数据的存取；②将查询算法的研究重点从范围查询转向近邻查询；③衡量索引结构性能的标准从单一的距离函数计算次数转向消耗的 cup 代价；④提出具有实际应用价值的索引结构。

表1 度量空间中的索引结构复杂度表[8]

Data Structure	Space Complexity	Construction Complexity	Claimed Query Complexity	Extra CPU Query time
BK-tree	n pointers	$O(n \log n)$	$O(n^a)$	——
FQ-tree	$n..\log n$ pointers	$O(n \log n)$	$O(n^a)$	——
FHQ-tree	$n..nh$ bits	$O(nh)$	$O(\log n)(*)$	$O(n^a)$
FQ-array	nhb bits	$O(nh)$	$O(\log n)(*)$	$O(n^a \log n)$
BS-tree	n pointers	$O(n \log n)$	not analyzed	——
VT-tree	n pointers	$O(n \log n)$	not analyzed	——
GH-tree	n pointers	$O(n \log n)$	not analyzed	——
GNA-—tree	nm^2 pointers	$O(nm \log_m n)$	not analyzed	——
VP-tree	n pointers	$O(n \log n)$	$O(n \log n)(**)$	——
MVP-tree	n pointers	$O(n \log n)$	$O(n \log n)(**)$	——
VPF	n pointers	$O(n^{2-a})$	$O(n^{1-a}\log n)(**)$	——
AESA	n^2 pointers	$O(n^2)$	$O(1)(***)$	$O(n)..O(n^2)$
LAESA	kn^2 pointers	$O(n^2)$	$k+O(1)(***)$	$O(\log n)..O(kn)$
M-tree	n pointers	$O(n(m..m^2 \log_m n)$	not analyzed	——
SA-tree	n pointers	$O(n \log n/\log \log n)$	not analyzed	——

(*)如果 $h=\log n$ (**)仅在查询半径非常小时成立 (***)经验结论

参 考 文 献

1 Baeza-Yates R. Searching: an algorithmic tour. In: A. Kent, J. Williams, eds. Encyclopedia of Computer Science and Technology, Marcel Dekker Inc., 1997, 37: 331～359

2 Baeza-Yates R, Cunto W, Manber U, Wu S. Proximity matching using fixed-queries trees. In: Proc. 5th Combinatorial Pattern Matching (CPM'94), LNCS 807, 1994. 198～212

3 Bentley J L. Multidimensional Binary Search Trees Used for Associative Searching. Communications of the ACM, 1975, 18(9): 509～517

4 Bozkaya T, Ozsoyoglu M. Distance-based indexing for high-dimensional metric spaces. In: ACM SIGMOD Intl. Conf. on Management of Data, Sigmod Record, 1997, 26(2): 357～368

5 Brin S. Near neighbor search in large metric spaces. In: Proc. 21st Conf. on Very Large Database(VLDB'95), 1995. 574～584

6 Burkhard W, Keller R. Some approaches to best-match file searching. Comm. of the ACM, 1973, 16(4): 230～236

7 Chavez E, Marroquin J, Navarro G. Overcoming the curse of dimensionality. In: European Worshop on Content-Based Multimedia Indexing (CBMI'99), 1999. 57～64

8 Chavez E, Navarro G, Baeza-Yates R, Marroquin J. Searching in Metric Spaces. ACM Computing Surveys, 2001

9 Ciaccia P, Patella M, Zezula P. M-tree: an efficient access method for similarity search in metric spaces. In: Proc. Of the 23rd Conf. on Very Large Databases(VLDB'97), 1997. 426～435

10 Ciaccia P, Patella M. The M²-tree : Processing Complex Multi-Feature Queries with Just One Index. DELOS Workshop : Information Seeking, Searching and Querying in Digital Libraries 2000

11 Dehne F, Nolteimer H. Voronoi trees and clustering problems. Information Systems, 1987, 12(2): 171～175

12 Faloutsos C, Lin K. Fastmap: a fast algorithm for indexing, data mining and visualization of traditional and multimedia datasets. ACM SIGMOD Record, 1995, 24(2): 163～174

13 Kalantari I, McDonald G. A data structure and an algorithm for the nearest point problem. IEEE Transactions on Software Engineering, 1983, 9(5)

14 Mico L, Oncina J, Vidal E. A new version of the nearest-neighbor approximating and eliminating search (AESA) with linear preprocessing-time and memory requirement. Pattern Recognition Letters, 1994, 15 : 9～17

15 Navarro G. Searching in metric spaces by spatial approximation. In: Proc. String Processing and Information Retrieval (SPIRE'99), IEEE CS Press, 1999. 141～148

16 Guttman. R-tree: A dynamic index structure for spatial searching. In: Proc. of the ACM SIGMOD Intl. Conf. on Management of Data, 1984. 47～54

17 Nolteimer H, Verbarg K, Zirkelbach C. Monotonous Besector * Trees - a tool for efficient partitioning of complex schenes of geometric objects. In Data Structures and Efficient Algorithms, LNCS 594, Springer-Verlag, 1992. 186～203

18 Uhlman J. Satisfying general proximity/similarity queries with metric trees. Information Processing Letters, 1991, 40: 175～179

19 Vidal E. An algorithm for finding nearest neighbors in (approximately) constant average time. Pattern Recognition Letters, 1986, 4: 145～157

20 White D A, Jain R. Similarity indexing with the SS-tree. In: Proc. 12th Int Conf on Data Engineering, New Orleans, LA, 1996

21 Train C Jr, Traina A J M, Seeger B, Faloutsos C. Slim-trees: High performance meric trees minimizing overlap between nodes. In: EDBT 2000, pp. 51-65, Konstanz, Germany, Mar. 2000

22 Yianilos P. Data structures and algorithms for nearest neighbor search in general metric spaces. In: Proc. 4th ACM-SIAM Symposium on Discrete altgorithms (SODA'93), 1993. 311～321

23 Yianlios Y. Locally lifting the curse of dimensionality for nearest neighbor search. In: DEMACS Implementation Challenge, ALENEX'99, Baltimore, MD, 1999

24 董道国，薛向阳，罗航哉. 多维数据索引结构回顾. 计算机科学, 2002, 29(3)