ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.17, No.10, October 2006, pp.2048−2056
DOI: 10.1360/jos172048

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

基于形状特征 k-d 树的多维时间序列相似搜索*

黄 河[1,2+], 史忠植[1], 郑 征[1,2]

[1](中国科学院 计算技术研究所,北京 100080)

[2](中国科学院 研究生院,北京 100049)

Similarity Search Based on Shape k-d Tree for Multidimensional Time Sequences

HUANG He[1,2+], SHI Zhong-Zhi[1], ZHENG Zheng[1,2]

[1](Institute of Computing Technology, The Chinese Academy of Sciences, Beijing 100080, China)

[2](Graduate School, The Chinese Academy of Sciences, Beijing 100049, China)

+ Corresponding author: Phn: +86-10-62565533 ext 5689, E-mail: huangh@ics.ict.ac.cn, http://www.intsci.ac.cn

Huang H, Shi ZZ, Zheng Z. Similarity search based on shape k-d tree for multidimensional time sequences. *Journal of Software*, 2006,17(10):2048−2056. http://www.jos.org.cn/1000-9825/17/2048.htm

Abstract: Multidimensional time sequences are an important kind of data stored in the information system. Similarity search is the core of their applications. Usually, these sequences are viewed as curves in multi-space, and the Euclidean Distance is computed to measure similarity between these curves. Although Euclidean Distance can reflect the whole deviation between two sequences or subsequences, it ignores their inherent changing features. To remedy it, this paper presents a new algorithm. In this algorithm, the shape features of sequences or subsequences are subtly combined with spatial index structure (k-d tree), which makes it possible to match shape of sequences or subsequences without any extra cost whiling searching the tree. The experimental result demonstrates that the algorithm is effective and efficient.

Key words: time sequence; similarity search; Euclidean distance; index structure; k-d tree

摘 要: 多维时间序列是信息系统中一类重要的数据对象,相似搜索是其应用的一个核心.两个序列(子序列)相似度加以比较的常用方法是:将序列(子序列)转换成空间中的曲线,然后计算曲线间的欧几里德距离.这种方法的主要缺陷是它仅考虑了序列(子序列)间的整体距离关系,而不能体现它们自身的局部变化.针对此问题,提出了一种新的可应用于多维时间序列的快速相似搜索方法.该方法将序列(子序列)的局部变化特性与检索结构(k-d 树)结合起来,使得在搜索 k-d 树的同时实现了序列(子序列)的局部变化匹配,从而极大地提高了查询效率和正确率.实验结果表明了算法的有效性.

关键词: 时间序列;相似搜索;欧氏距离;检索结构;k-d 树

中图法分类号: TP301 文献标识码: A

* Supported by the National Natural Science Foundation of China under Grant No.90604017 (国家自然科学基金); the National Grand Fundamental Research 973 Program of China under Grant No.2003CB317004 (国家重点基础研究发展规划(973)); the Beijing Natural Science Foundation of China under Grant No.4052025 (北京市自然科学基金)

Received 2004-03-03; Accepted 2006-03-31

时间序列是信息系统中一类重要的数据对象,相关的应用主要有时间序列的建模和查询等.相似搜索是这些应用的核心.例如:给定多个公司在 10 年期间股票收盘价组成的时间序列,找出具有相似收盘价变化的公司;给定多个商品的每日销售量序列,找出具有相似销售模式的商品等等.根据序列的长度,相似搜索可以分为两类[1]:(1) 整体匹配(whole matching):所有时间序列的长度相等,搜索的目的是从数据库中找到与查询序列相似的时间序列;(2) 子序列匹配(subsequence matching):当查询序列的长度小于数据库中时间序列的长度时,找出序列中与查询序列最相似的子序列.根据查询的标准,相似搜索还可以分为两类:(1) *k*-临近查询(*k*-nearest neighbor):找出 *k* 个与查询序列最相似的序列或子序列;(2) 范围查询(range query):找出与查询序列的距离不大于允许误差ε的序列或子序列.本文主要关注于子序列匹配和范围查询.

1993 年,Agrawal 等人首次提出了时间序列相似搜索的整体匹配算法[1],接着,Faloutsos 等人提出了子序列匹配算法[2],推广了相似搜索的应用领域.针对欧氏距离(Euclidean distance)的缺陷,文献[3]中提出一种基于时间弯曲距离(time warping distance)的相似搜索方法.上述算法均为一维时间序列的相似搜索方法,且在其各自的应用领域内取得了很大的成功.但随着音、视频设备和 Internet 的流行和普及,许多关于一维时间序列的相似搜索方法已经不适用于多媒体数据,从而提出了多维时间序列的相似搜索.

多维时间序列主要包括图形、图像、音频、视频等信息,它是由一组随时间改变的数据向量组成.对于多维时间序列,许多方法(例如文献[4])将多维时间序列看成多维空间中的一条曲线,然后比较曲线(或子曲线)与查询曲线间的欧氏距离.但欧氏距离仅能体现序列间的整体关系(序列中对应结点的空间距离之和或对应节点的平均空间距离是否小于某个值),而不能体现序列本身的局部变化(例如起伏变化),这可能导致查询结果的不正确.例如,给定 4 个一维的时间序列,如图 1 所示[4].序列 *S*1 和 *S*2 的凹凸方向恰好相反,序列 *S*3 和 *S*4 的变化曲线相似,但 *S*1 与 *S*2 间的欧几里德距离小于 *S*3 与 *S*4 间的欧几里德距离.尽管可以通过归一化来缓解此问题,但解决此问题的关键在于如何将局部变化的信息融入相似度的计算中.由于多维空间中曲线的局部变化在不同维上投影呈现的变化趋势可能不同,因此需要综合考虑曲线在每个维上起伏,并将它们作为相似性度量的准则之一.

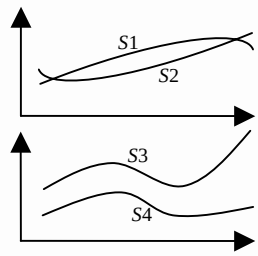

Fig.1 Similarity of two pairs of time sequences

图 1 两组序列的相似度比较

针对此问题,本文提出构造基于形状特征的 k-d 树[5]来实现多维时间序列的子序列匹配.给定 *k* 维的时间序列数据库,我们采用滑动窗口(长度 *w*)技术,将多维时间序列分成若干个可重叠的子序列.将子序列在每维上的形状特征编码为数值,并将代表 *k* 维的 *k* 个形状特征数值作为子序列形状特征向量 *SF* (由 *k* 个关键字组成),并以 *SF* 为关键值构造 k-d 树,其中每个树结点引导一个链表,每个表结点中存放着一个子序列的各种信息,且同一链表中的子序列具有相同的 *SF*.给定查询序列 *Q*,计算 *Q* 的 *SF*,并根据 *SF* 在 k-d 树中进行查找,得到的树结点所链接的子序列与 *Q* 具有相似的局部变化趋势.这些子序列作为候选集,并比较它们与 *Q* 的欧氏距离,返回小于允许误差ε的子序列作为查询的结果.该算法有以下优点:(1) 迅速性——能快速地返回与查询序列相似的子序列;(2) 完整性——能返回所有符合要求的子序列;(3) 小空间性——算法可以在小空间中运行,特别是当数据库足够大而不能一次装入内存时,这能够保证查询的效率;(4) 动态性——算法能适用于动态的数据库,即序列的长度随时间而增长,且能处理不同长度的查询序列.

本文第 1 节介绍相关工作.第 2 节具体描述相似搜索算法.第 3 节对算法的性能进行分析比较.第 4 节给出实验结果.第 5 节是我们的结论和展望.

1 相关工作

对于时序数据库,特别是一次无法装入内存的大型数据库,如何进行快速而准确的相似搜索,其重点在于:如何表示时间序列、如何度量序列间的相似性以及如何对数据库进行检索.

序列表示的主要作用是降维即解决维度的困扰(dimensionality curse)、提取序列的特征等.常用的方法有:

离散傅里叶变换(discrete Fourier transform,简称 DFT)[1,2]、离散小波变换(discrete wavelet transform,简称 DWT)[6]、奇异值分解(singular value decomposition,简称 SVD)[7]、分段法[3,8]等.DFT 能将时间序列转换到频率域,取前几个较强的傅里叶系数作为序列的表示以达到降维的目的.DFT 适合那些自然发生的正弦信号,但不适合表示不连续间断的信号.在 DWT 方法中,Haar 小波变换是最常用的.但由于基函数不光滑,Haar 小波采用阶梯状的结构近似地模拟信号.因此,仅用少数的 Haar 小波变换系数是不能很好地近似连续函数的,所以需要保留的小波系数的数目必须比较多.SVD 是一种依赖于数据内容的降维方法.通过计算给定数据集的特征值和特征向量,SVD 将数据进行转换使得大多数信息集中在某些维上,取数据在这些维上的坐标作为原数据集的压缩.它的主要缺陷是当数据改变时,特征向量需要重新计算.因此,SVD 不适合动态变化的数据库.分段法主要通过将序列分段,取每段的特征(例如,极值点、变化趋势等)组成序列的特征表示.

比较两个序列的相似性一般采用距离公式求得序列间的距离,当距离越小时,说明两个序列越相似(除最长公共子序列以外),反之亦然.常用的距离公式有:欧氏距离[1,2]、时间弯曲距离[3,9]和最长公共子序列(longest common subsequence)[10]等.欧式距离的主要优势在于计算复杂度低,并在正交变换下保持距离不变.与其他公式相比,它的主要缺陷是易受噪声和时间轴上偏移的影响.

常用的检索结构有:k-d 树[5]、R 树[11]及其改进等.R 树是一种存储多维空间对象的平衡的动态索引结构,主要以最小边界方形(minimum bounding rectangle,简称 MBR)为检索对象.R 树是一种平衡多叉树,它通过逐级缩小搜索的空间范围来定位要找的对象.R 树的改进主要体现在对结点的 MBR 覆盖区域、MBR 重叠区域和 MBR 的边缘长度等方面进行优化,从而提高查询效率.k-d 树是一种多维的二叉查找树,支持基于多关键字的二叉查找树,同时也可以用来存储 k 维空间中的数据点.

Lee 等人[4]将多维时间序列分割成若干个子序列,每个子序列用最小边界方形 MBR 表示,并用空间结构存储这些 MBR.该算法的主要特点在于提出了一种新的距离函数:规范化 MBR 距离 D_{norm}.在检索空间结构时,可以利用 D_{norm} 距离来得到查询的候选集.D_{norm} 表示两个序列在空间中欧式距离的下限.与欧式距离相似,D_{norm} 能够体现序列间的空间距离,但不能体现序列本身的局部变化.这可能导致查询结果的不正确.

在一维时间序列的相似搜索中,许多方法将长度为 n 的一维序列视为 n 维空间中的一个点,然后应用 DFT 或 DWT 等降维方法以提高计算效率.对于多维时间序列,序列中每一个点就是一个 k 维向量.若采用类似一维时间序列的方法,长度为 n 的多维序列则成为 $k\cdot n$ 维空间中的点.对于一个如此高维的空间,DFT 或 DWT 等方法则不适合提取多维时间序列的特征.同样,计算复杂度高的距离函数也不适合多维时间序列的比较.例如:对于两个长度为 n 的多维时间序列,计算它们之间欧式距离的时间复杂度为 $O(k\cdot n)$;计算时间弯曲距离或最大公共子序列的时间复杂度则均为 $O(k\cdot n^2)$.基于上述原因,我们采用分段法作为序列表示的方法,采用欧氏距离作为相似度函数.同时,为了描述局部变换特征,子序列在每维上的形状特征编码为数值.k-d 树作为多维的二叉查找树,它比 R 树家族的成员更适合存储每段子序列的 k 个关键字.

2 相似搜索

多维时间序列的相似搜索问题具体描述如下[4]:一个 k 维时间序列由多维空间中的点组成,记为 $S=(S[0],S[1],\ldots,S[n-1])$,其中,$S[i]=(S[i,0],\ldots,S[i,k-1])$是一个 k 元向量;n 表示序列的长度,k 表示序列的维数.给定一个时间序列数据库,它包括 N 个长度可能不相等的多维时间序列 $S_0,S_1,\ldots,S_{N-1}$,给定查询序列 Q 以及允许误差 ε,找出数据库中所有的多维时间序列 $S_i(0\le i\le N-1)$及其每个子序列 $S_i[j:j+Len(Q)-1](0\le j\le Len(S_i)-Len(Q))$,使得子序列与 Q 起伏形状相似且它们之间的距离小于 ε.

2.1 多维时间序列的性质

2.1.1 多维时间序列的形状特征向量

给定多维时间序列 $S=(S[0],S[1],\ldots,S[n-1])$,它在每一维上的投影 $S^j=(S[0,j],\ldots,S[n-1,j])(0\le j\le k-1)$是一个一维时间序列.通过描绘每个 S^j 的形状特征可以表示 S 的形状特征.对于 S^j,我们考虑一种简单且直观的方法:在每个点处用垂直线将序列分隔,对于分隔的每一条线段,若上升,则用“1”表示;若下降,则用“0”表示;若水平,则可以

统一地视为上升或下降.因此,S^j 的形状可以用“0”和“1”组成的长度为 $n-1$ 个比特的二进制数表示.如图 2 中的一维序列,形状特征为“10010100”.这种方法的主要缺陷是:曲线的起伏描述过于繁琐,且容易受噪声干扰.例如图 3 中两个序列 $L1$ 和 $L2$,它们的形状极为相似.但由于 $L2$ 在 i,j,k 三点处受噪声干扰,使得它们的“0”,“1”序列特征值不同,分别为“11111111000000111110000001110”和“11111011001000111110001001110”.为缓解噪声的干扰,可以将 $L1$ 和 $L2$ 分割成 7 个段.对于每个段,倘若它的总体趋势是上升的,则用“1”表示该段的特征;若总体趋势下降,则用“0”表示;其他情况可以统一地视为“1”或“0”.经过这一变换后,$L1$ 和 $L2$ 的形状特征值则均为“1101101”.描述每段的总体趋势,不仅能在有噪声情况下缓解噪声的干扰,同时还能在无噪声的情况下不致影响序列的特征提取.

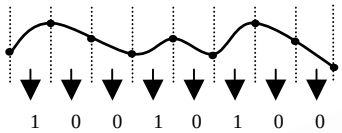

Fig.2 Waving feature of one dimensional time sequence

图 2 一维序列的起伏特征

Fig.3 Two sequences $L1$ and $L2$

图 3 两序列 $L1$ 和 $L2$

下面定义一维时间序列的形状特征:给定 $L=(l_0,\ldots,l_{n-1})$和分段参数 h(不妨设 $n-1$ 能被 h 整除),将 L 分割成 h 段,即 $L_0,\ldots,L_{h-1}$;令 $j=(n-1)/h$,每段(包含两个端点)由 $j+1$ 点组成,表示为 $L_i=(l_{i\cdot j},\ldots,l_{(i+1)\cdot j})(0\le i\le h-1)$;令

$$\delta(i)=\sum_{m=i\cdot j+1}^{(i+1)\cdot j}(l_m-l_{m-1}).$$

当 $\delta(i)>0$ 时,分段 L_i 的总体趋势是上升的;当 $\delta(i)<0$ 时,分段 L_i 的总体趋势是下降的.然后定义 L 的形状特征函数:

$$shape(i)=\begin{cases}1, & \delta(i)>0\\ 0, & \text{others}\end{cases}(0\le i\le h-1).$$

函数 $shape(i)$能够产生长度为 h 的“0”,“1”串,称为序列 L 的形状特征值.

采用上述方法的优点是:(1) 节省空间;(2) 既能描绘序列的形状,又能有效缓解噪声对数据带来的影响;(3) 在无噪声的情况下不会影响序列的特征提取;(4) 计算简单,仅需一次遍历就能计算出序列的形状特征值.

对于一个 k 维的时间序列 S,它的形状特征向量 SF 定义如下:令 SF^j 等于 S 在每一维上的投影 $S^j=(S[0,j],\ldots,S[n-1,j])(0\le j\le k-1)$的形状特征值,是由 k 个 SF^j 组成的向量 $SF=[SF^0,\ldots,SF^{k-1}]$成为 S 的形状特征向量.两个长度相等的 k 维时间序列,若它们的形状特征向量中每个分量依次相等,则表示这两个 k 维序列的波形近似.

2.1.2 多维时间序列的欧几里德距离

给定两个长度为 n 的 k 维时间序列 S_1 和 S_2.S_1 和 S_2 中任意两点间的距离定义为两点间的欧几里德距离[1]

$$d(S_1[i],S_2[j])=\left(\sum_{t=0}^{k-1}(S_1[i,t]-S_2[j,t])^2\right)^{1/2}.$$

S_1 和 S_2 间的距离定义为对应点间的平均距离[4]:

$$D(S_1,S_2)=\sum_{i=0}^{n-1}d(S_1[i]-S_2[i])\Big/n.$$

2.2 子序列的信息提取

在子序列查询过程中,面对一个长的时间序列,我们不可避免地要将序列进行分段处理.此时的关键在于要保证没有信息的丢失,即保证信息的完整性.因此,我们采用滑动窗口技术.

给定 k 维时间序列 S,将一个长度为 w 的滑动窗口放置在其上,窗口的左端与 S 的第一个数据点对齐,然后以步长为 1 的方式滑动窗口,直到窗口的右端与 S 的最后一个数据点对齐为止.对于每一个落入窗口的子序列 $(S[i],\ldots,S[i+w-1])(i=0,1,\ldots n-w)$,提取如下信息来构造信息结点 Inf_i:

1) S 在数据库中的 ID,用 Inf_i.ID 表示;
2) 子序列相对于 S 的偏移量($offset=i$),用 $Inf_i.Offset$ 表示;

3) 子序列的形状特征向量 SF,用 $Inf_i.SF$ 表示;

4) 指向下一个信息结点的指针 $Next$,用 $Inf_i.Next$ 表示.

2.3 检索结构——k-d树

2.3.1 k-d 树的性质

k-d 树是一种多维二叉搜索树.在本文中,k-d 树结点 P 由以下几部分组成:

1) 子序列的形状特征向量 $SF=[SF^0,\ldots,SF^{k-1}]$,用 $SF(P)$表示,其中每一个二进制特征值都作为一个键值,用 $K_i(P)(i=0,\ldots,k-1)$表示;

2) 类型为整数的鉴别器 $Disc(0\le Disc\le k-1)$,表示在该节点处应比较的哪个 SF 分量,用 $Disc(P)$表示;

3) 指向子序列信息节点的指针 $FNPnt$,作为具有相同形状特征向量的子序列信息节点组成的链表的头节点,用 $FNPnt(P)$表示;

4) 指向左孩子的指针 $LeftSon$ 和右孩子的指针 $RightSon$,分别用 $LeftSon(P)$和 $RightSon(P)$表示.

对于 k-d 树中的任意节点 P,令 $j=Disc(P)$,都有:

1) 对于 P 的左子树中任意节点 L,满足 $K_j(L)<K_j(P)$;

2) 对于 P 的右子树中任意节点 R,满足 $K_j(P)\le K_j(R)$;

3) 对于 $FNPnt(P)$所指链表中每一个节点的 SF 都相等.

在一棵 k-d 树中,同层节点的鉴别器是相等的.根节点(第 0 层)的鉴别器为 0,第 1 层的鉴别器为 1,…,第 $k-1$ 层的鉴别器为 $k-1$,第 k 层的为 0,第 $k+1$ 层的为 1,如此循环下去.由此可得

$$Disc(LeftSon(P))=Disc(RightSon(P))=(Disc(P)+1) \bmod k.$$

给定一个树节点 P 和子序列信息节点 q,判断 q 应插入 $FNPnt(P)$所指的链表中,或是在 P 的左子树,或是在 P 的右子树的函数 $Successor(P,q)$描述为:如果 $SF(P)$与 $q.SF$ 的每个分量依次相等,返回 P;否则,令 $j=Disc(P)$,如果 $q.SF[j]<K_j(P)$,返回 $LeftSon(P)$;否则,返回 $RightSon(P)$.

2.3.2 构造 k-d 树

给定一个时间序列数据库,将滑动窗口放置在每个时间序列上滑动,为每个落入窗口的子序列生成信息节点,并将节点依次插入到 k-d 树中,从而得到时间数据库的检索结构.给定一个子序列信息节点 q,将其插入到以 Root 为根的 k-d 树中的函数 $Insert(Root,q)$描述为:

```
Insert(Root,q){
    if (Root==NULL) then {
        生成一个新的树节点 P;
        SF(P)=q.SF;
        Disc(P)=0;
        FNPnt(P).Next=q;
        LeftSon(P)=RightSon(P)=NULL;
        return;
    }
    T=Root;
    do {
        R=T;
        T=Successor(R,q);
    } until ((T==NULL) or (T==R))
    if(T==NULL) then {    //树中不存在键值与 q.SF 相等的结点
        生成一个新的树节点 P;
        SF(P)=q.SF;
```

Disc(*P*)=(*Disc*(*R*)+1) mod *k*;
FNPnt(*P*).*Next*=*q*;
LeftSon(*P*)=*RightSon*(*P*)=*NULL*;
j=*Disc*(*R*);
if (*q*.*SF*[*j*]<K_j(*R*) then *LeftSon*(*R*)=*P*;
else *RightSon*(*R*)=*P*;
}
if(*T*==*R*) then 将 *q* 插入到 *FNPnt*(*R*)所指链表的尾部;
}

2.4 查询处理

给定查询序列 *Q*,滑动窗口 *w*,分段参数 *h*,以 Root 为根的 k-d 树以及用户设定的允许误差ε,查询处理的算法描述如下:

SimilarityQuery(*Root*,*Q*,ε,*w*,*h*){
将 *Q* 划分为长度为 *w* 的 *l* 个不相交子序列,并提取信息特征节点 $q_0,\ldots,q_{l-1}$;
对于每个结点 $q_i(0\le i\le l-1)$){
P=*SearchTreeNode*(*Root*,q_i); //找出树中键值与 q_i.*SF* 相等的节点 *P*
InfoNodeSet_i=*get*(*P*); //将 *P* 所引导链表中所有结点 *r* 放入集合中
}
InfoNodeSet=*Match*(*InfoNodeSet*_0,…,*InfoNodeSet_l*–1,*w*); //匹配所有 *InfoNodeSet_i*
ResultSet=*RealDis*(*InfoNodeSet*,*Q*,ε); //根据 *InfoNodeSet* 中信息节点从数据库中取得真正的子序列,并比较每个子序列与 *Q* 的欧几里德距离从而得到解集
}

其中,函数 *SearchTreeNode*(*Root*,*q*)与 *Insert*(*Root*,*q*)的搜索方法类似,此处不再详细描述.

函数 *Match*(*InfoNodeSet*_0,…,*InfoNodeSet_l*–1,*w*)描述为:设集合 *InfoNodeSet* 初始为空,对于 *InfoNodeSet*_0 中所有节点为 *r*,若在每个 *InfoNodeSet_i*(*i*=1,2,…,*l*–1)中都存在节点 *t*,使得 *t*.*ID*=*r*.*ID* 且 *t*.*offset*=*r*.*offset*+*i*·*w*,则将 *r* 加入 *InfoNodeSet*,最后返回 *InfoNodeSet*.

函数 *RealDis*(*InfoNodeSet*,*Q*,ε)描述为:设子序列集合 *SubSequenceSet* 初始为空,根据 *InfoNodeSet* 中每个节点的 ID 和 offset,从数据库中取得长度为 *Len*(*Q*)的相应的子序列并加入到 *SubSequenceSet* 中;然后,设结果集合 *ResultSet* 初始为空,对于 *SubSequenceSet* 中每一个子序列 *s*,若 *D*(*s*,*Q*)≤ε,则将 *s* 加入 *ResultSet*,最后返回 *ResultSet*.

下面,我们根据不同 *Q* 的长度来讨论查询时的不同点:

1) 当 *Len*(*Q*)=*w* 时,*Q* 只有一个信息特征节点 *q*,最多只有一个信息特征结点集合,即 *InfoNodeSet*_0,因此不需要执行 Match 函数;
2) 当 *Len*(*Q*)>*w* 时,*Q* 有 *l*=*int*(*Len*(*Q*)/*w*)个信息特征结点,因此最多有 *l* 个集合,即 *InfoNodeSet*_0,…,*InfoNodeSet_l*–1;
3) 当 *Len*(*Q*)<*w* 时,*Q* 只有一个信息特征节点 *q*,因此不需要执行 Match 函数.但 *q*.*SF* 的每一个分量的比特长度较小,使得 *SearchTreeNode*(*Root*,*q*)函数与前两种情况不同,可能会返回多个树结点.

SearchTreeNode(*Root*,*q*){
TreeNodeSet=∅; //将树结点集合置空
f_{min}=*q*.*SF* 的每一个分量高位部分加若干个“0”,使得每个分量的比特长度为 *h*;
f_{max}=*q*.*SF* 的每一个分量高位部分加若干个“1”,使得每个分量的比特长度为 *h*;
设置队列 *Queue* 为空;
Enqueue(*Queue*,*Root*); //将根结点入队
While (*NotEmpty*(*Queue*)) { //队列不为空
P=*Dequeue*(*Queue*); //取出队头元素

```
            对于形状特征向量 f_min 的每一个分量 f_min[i] {
                if ((f_min[i] &K_i(P))==f_min[i]) then {    //q.SF[i]与 K_i(P)的前缀相等
                    TreeNodeSet=TreeNodeSet∪P;
                }
                j=Disc(P);
                if((K_j(P)>f_min[j]) & & (K_j(P)<f_max[j])) then {
                    Enqueue(Queue,LeftSon(P));
                    Enqueue(Queue,RightSon(P));
                }
                else if (K_j(P)<f_min[j]) then Enqueue(Queue,RightSon(P));
                else Enqueue(Queue,LeftSon(P));
            }
            return TreeNodeSet;
        }
    }
```

在得到 *SearchTreeNode*(*Root*,*q*)函数的返回结果 *TreeNodeSet* 后,将所得结果合并,得到 *InfoNodeSet*.

3 算法分析

给定一个时间序列数据库 $S_0,\ldots,S_{N-1}$,假设每个序列的平均长度为 M,则子序列信息节点的个数为 $N\cdot(M-w+1)$.根据这些信息结点构造一棵 k-d 树,设树结点数为 m,则每个树结点所指的信息链表长度约为 $L=N\cdot(M-w+1)/m$.文献[5]中已经证明,在 k-d 树中插入和搜索节点的时间复杂度为 $O(\log_2 m)$.在最坏情况下,即所有信息节点的形状特征向量 *SF* 都不相同时,构造 k-d 树时间为 $O(MN\cdot\log_2(MN))$.给定查询序列 Q,当 $Len(Q)\geq w$ 时,令 $l=int(Len(Q)/w)$,则 SearchTreeNode 所花的时间为 $O(l\cdot\log_2 m)$,Match 所需的时间为 $O(l\cdot L)$,RealDis 所需的时间为 $O(L)$;当 $Len(Q)<w$ 时,在最糟情况下,SearchTreeNode 要遍历整棵数,查询部分的时间复杂度为 $O(MN\cdot Len(Q))$.因此,当查询序列的长度很小时,可采用顺序检索方法来简化相似搜索.

在文献[4]所提出的多维时间序列相似搜索的方法中,计算 D_{norm} 所需的时间为 $O(Len(Q)^2)$,查询 R 树所需的时间为 $O(p\cdot\log_u m)$,其中,m 为树结点数;u 为树的阶数;p 表示搜索过程中访问的多个子树,这主要是由 MBR 之间的重叠所引起的.由此可见,我们的方法与文献[4]中的方法在时间复杂度方面没有量级的区别.

4 实验结果

我们取随机数组成的时间序列和查询序列应用上述算法.首先查看分段参数 h 和滑动窗口长度 w 与建立 k-d 树所需时间(*C*Time)、树结点个数(*T*Node)、树的高度(Height)、在树中进行 10 000 次查找所需的时间(*Q*Time)和剪枝率(prune)之间的关系.其中:

Prune=(序列的长度–候选集中子序列长度之和)/(序列的长度–结果集中子序列长度之和).

表 1 显示了 h,w 与 *C*Time,*T*Node,Height 和 *Q*Time 之间的关系.从表 1 中可以看出:

1) 在 w 近乎不变的情况下,h 越小,建树时间越短、树结点个数越少、每个树结点所指链表越长.尽管搜索树所花费的时间较少,但在链表中搜索的时间有所增加,因而导致查找时间增加,剪枝率下降;
2) 随着 h 的增大,尽管链表中搜索的时间和总体查找时间减少,剪枝率加大,但建树时间却增加,并且 k-d 树的规模急剧增加.

从表 1 中总结出 h 的较好取值为 4 或 5.

令 $k=3$,$Len(Q)=36$,$w=36$,$h=5$,我们查看时间序列的长度 $Len(S)$与建 k-d 树所需时间(*C*Time)和在树中进行 10 000 次查找所需的时间(*Q*Time)之间的关系.如图 4 所示,建树时间与序列长度呈线性关系,查询时间与序列长度也近乎呈线性关系(与不同输入相关).

接下来与文献[4]中多维时间序列方法进行比较.给定一个多维时间序列(其中,k=3,$Len(S)$=500000)和查询序列 Q,取 w=36,h=5.令 QTime 表示我们的方法进行 10 000 次查找所需的时间,令 ScanTime 表示文献[4]中方法进行 10 000 次查找所需的时间.两种方法查询时间的比较如图 5 所示.随着 $Len(Q)$逐渐接近 w,QTime 急剧下降;当 $Len(Q)$大于等于 w 后,QTime 近乎保持不变.通过 QTime 与 ScanTime 的对比可以得出:我们的算法在查询速度方面优于文献[4]中的算法.

Table 1 Relationship between (h,w) and (CTime, TNode, Height, QTime, prune)

表 **1** h,w 与 CTime,TNode,Height,QTime 和 prune 之间的关系

h	w	CTime (s)	TNode	Height	QTime (s)	Prune
3	37	1.203	422	18	0.531	0.892
4	37	1.265	689	19	0.047	0.921
5	36	1.375	3 733	26	0.046	0.936
6	37	1.453	9 240	30	0.062	0.949
7	36	1.546	20 107	31	0.079	0.934
8	33	1.609	48 849	37	0.062	0.965
9	37	1.687	49 095	38	0.063	0.978
10	41	1.765	50 823	39	0.063	0.994

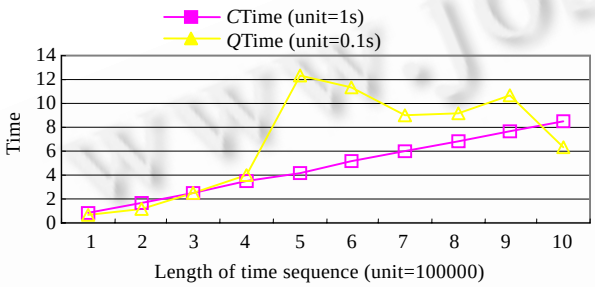

Fig.4 Relationship between $Len(Q)$ and (CTime, QTime)

图 4 $Len(Q)$与 CTime,QTime 的关系

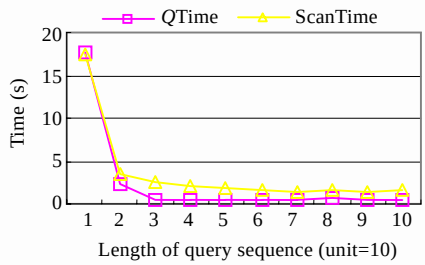

Fig.5 Comparison between QTime and ScanTime

图 5 QTime 与 ScanTime 的对比

最后,我们采用某一实际的数据库(某县的气象虫害数据库,长度为 100 000),取其中相关的 5 维作为测试数据集,比较两种方法在该数据集上的查询准确率(precision)和查全率(recall),其中

$Precision$=|{$Relevant$}∩{$Retrieved$}|/|{$Retrieved$}|, $Recall$=|{$Relevant$}∩{$Retrieved$}|/|{$Relevant$}|.

图 6 显示出随着允许误差ε的增加,两种算法的准确率发生变化.在我们的算法中,候选集的产生与ε的大小无关,因而随着ε的增加,准确率也有所增加;而文献[4]中的算法不同,候选集的产生与ε相关,所以准确率随着ε的增加而下降.图 7 显示出随着允许误差ε的增加,两种算法的查全率的变化情况.

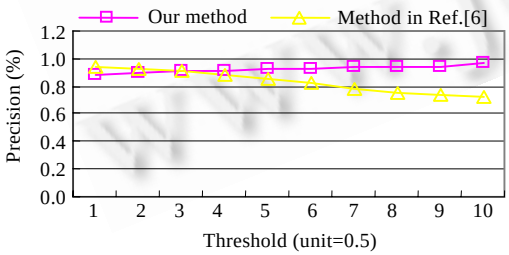

Fig.6 Precision comparison of two methods

图 6 两种算法的 Precision 比较

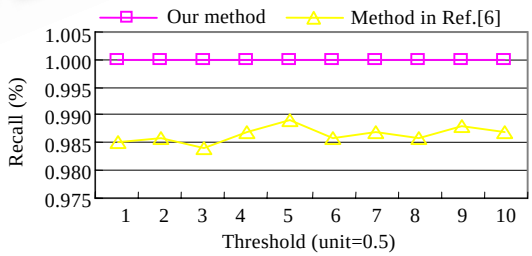

Fig.7 Recall comparison of two methods

图 7 两种算法的 Recall 比较

对于更高维的数据集会产生更多的形状特征值,从而导致 k-d 树规模的增加,树查询时间的增加,链表查询时间的减少和查询准确率的提高.对于更高维的数据集,R 树的规模增加要略低于 k-d 树,但会带来 MBR 重叠区域的增加,进而导致搜索过程中访问子树的增加,并降低文献[4]中方法的查询准确度.

我们的方法采用步长为 1 的方式滑动窗口,并提取落入窗口的子序列特征,因而保证了查询结果的完整性.

而文献[4]中采用序列的无重叠分段方法,必然存在各分段间信息的丢失,因而无法保证算法在查询结果方面的完整性.

5 结论和展望

相似搜索是多维时间序列应用的一个重要方面.现有的一些方法大多都采用欧几里德距离作为计算相似度的依据.但欧几里德距离不能顾及序列本身局部的起伏变化,极大地降低了它在比较相似度方面的准确性,从而会导致返回错误的结果.针对这一问题,本文提出一种抽取序列(或子序列)的形状特征的方法,并将形状特征向量与检索结构 k-d 树有机地结合,使得在搜索 k-d 树的同时实现了序列(子序列)的局部变化匹配,以弥补欧几里德距离的不足,从而极大地提高了查询效率和正确率.我们下一步的工作是:针对高维数据,考虑使用数据降维方法;在相似度比较时,考虑不同数据维所占的权重.

References:

[1] Agrawal R, Faloutsos C, Swami A. Efficient similarity search in sequences database. In: Lomet D, ed. Proc. of the 4th Int'l Conf. of Foundations of Data Organization and Algorithms. Chicago: Springer-Verlag, 1993. 69–84.

[2] Faloutsos C, Ranganathan M, Mandoponlos Y. Fast subsequence matching in time-series databases. In: Snodgrass R, Winslett M, ed. Proc. of the '94 ACM SIGMOD Conf. Mineapolis: ACM Press, 1994. 419–429.

[3] Huang H, Xiong FL, Hang XS, Huang K. Research on a fast retrieval of similarity patterns in a time series database. Journal of Pattern Recognition and Artificial Intelligence, 2003,16(2):169–173 (in Chinese with English abstract).

[4] Lee S, Chun S, Kim D. Similarity search for multidimensional data sequences. In: Proc. of the 16th Int'l Conf. on Data Engineering. Washington: IEEE Computer Society, 2000. 599–608. http://citeseer.ist.psu.edu/lee00similarity.html

[5] Bentley J. Multidimensional binary search trees used for associative searching. Communications of the ACM, 1975,18(9):509–517.

[6] Kahveei T, Singh A. Variable length queries for time series data. In: Proc. of 17th Int'l Conf. on Data Engineering. Heidelberg, 2001. 273–282. http://citeseer.ist.psu.edu/323328.html

[7] Korn F, Jagadish H, Faloutsos C. Efficiently supporting ad hoc queries in large datasets of time sequences. In: Peckham J, ed. Proc. of the ACM-SIGMOD Int'l Conf. on Management of Data. Tucson: ACM Press, 1997. 289–300.

[8] Toshniwal D, Joshi RC. Similarity search in time series data using time weighted slopes. Informatica, 2005,29(1):79–88.

[9] Fu AW, Keogh E, Lau LYH, Ratanamahatana CA. Scaling and time warping in time series querying. In: Proc. of the 31st Int'l Conf. on Very Large Data Bases. Trondheim, 2005. 649–660. http://www.cs.ucr.edu/~eamonn/vldb05.pdf

[10] Vlachos M, Hadjieleftheriou M, Gunopulos D, Keogh E. Indexing multi-dimensional time-series with support for multiple distance measures. In: Proc. of the 9th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. 2003. 216–225. http://portal.acm.org/citation.cfm?coll=GUIDE&dl=GUIDE&id=956777

[11] Guttman A. R-Trees: A dynamic index structure for spatial searching. In: Yormark B, ed. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. Boston: ACM Press, 1984. 47–57.

附中文参考文献:

[3] 黄河,熊范纶,杭小树,黄轲.时序数据库中快速相似搜索的算法研究.模式识别与人工智能,2003,17(2):169–173.

黄河(1977－),女,江西分宜人,博士生,主要研究领域为数据挖掘,语义 Web.

郑征(1980－),男,博士生,主要研究领域为数据挖掘,粗糙集,粒度计算.

史忠植(1961－),男,研究员,博士生导师,CCF 高级会员,主要研究领域为人工智能,机器学习,分布式人工智能.