基于 LSH 的时间子序列查询算法

汤春蕾 董家麒
(复旦大学计算机科学技术学院 上海 200433)

摘 要 子序列的相似性查询是时间序列数据集中的一种重要操作,包括范围查询和 k 近邻查询.现有的大多算法是基于欧几里德距离或者 DTW 距离的,缺点在于查询效率低下.文中提出了一种新的基于 LSH 的距离度量方法,可以在保证查询结果质量的前提下,极大提高相似性查询的效率;在此基础上,给出一种 DS-Index 索引结构,利用距离下界进行剪枝,进而还提出了两种优化的 OLSH-Range 和 OLSH-kNN 算法.实验是在真实的股票序列集上进行的,数据结果表明算法能快速精确地找出相似性查询结果.

关键词 相似性查询;时间序列数据库;子序列;LSH;索引
中图法分类号 TP311 **DOI 号**: 10.3724/SP.J.1016.2012.02228

Similarity Query of Time Series Sub-sequences Based on LSH

TANG Chun-Lei DONG Jia-Qi
(*School of Computer Science, Fudan University, Shanghai* 200433)

Abstract Subsequence Similarity Query is an important operation in time series, including range query and k nearest neighbor query. Most of these algorithms are based on Euclidean distance or DTW distance, weak point of which is the time inefficiencies. We propose a new distance measure, based on Locality Sensitive Hash (LSH), which improve the efficiency greatly while ensuring the quality of the query results. We also propose an index structure named DS-Index. Using DS-Index, we prune the candidates of query and thus propose two optimal algorithms: OLSH-Range and OLSH-kNN. Our experiments conducted on real stock exchange transaction sequence datasets show that algorithms can quickly and accurately find similarity query results.

Keywords similarity query; time-series databases; subsequence; Locality Sensitive Hash (LSH); index

1 引 言

时间序列数据是一种重要的数据类型[1-2],在计量经济学的研究中与横截面数据和纵面数据并列为 3 大数据形态.随着 Web 技术的迅速发展和金融信息学的兴起,在时间序列中的各种挖掘分析是当前工商业界和学术界共同关注的热点问题.

时间序列相似性查询分为全序列匹配和子序列匹配两种.查找与查询序列长度相同且相似的内容,称为全序列匹配;而查找与查询序列长度不同但相似的,称为子序列匹配.时间序列子序列匹配,根据不同查询标准,可分为范围查询和 k 近邻查询两类.k 近邻查询能找出 k 个与查询序列最相似的子序列;而范围查询则能找出与查询序列的距离不大于允许误差 Δ 的子序列.

近年来在时间序列数据集上的许多子序列匹配算法被提出来,这些算法主要是采用传统欧几里德距离或者动态时间规整(DTW)作为相似性度量的.由于时间序列存在海量性和超高维特性,这两种距

收稿日期:2012-06-05;最终修改稿收到日期:2012-08-14.本课题得到上海市重点学科建设基金(B114)资助.**汤春蕾**,女,1976 年生,博士研究生,工程师,主要研究方向为数据库、数据挖掘.E-mail: towne@fudan.edu.cn. **董家麒**,男,1987 年生,硕士研究生,主要研究方向为数据库、数据挖掘.

离函数都会造成系统运行效率低下，即若时间序列时间长度为 l，欧几里德距离的时间复杂度是 $O(l)$，而 DTW 时间复杂度是 $O(l^2)$.

为提高查询效率，目前最常用的技术是“降维-剪枝-验证”的方法. 然而这种方法在验证阶段，仍需要使用现有耗时的距离度量，所以算法时间的减少主要取决于剪枝效果，一旦剪枝效果不理想算法开销要比暴力算法更大. 因而如何设计一种有效的且不依赖于时间序列长度的相似性度量成为一个迫切需要解决的问题. 由此还需要设计符合新相似性度量的基于索引的剪枝技术.

基于上述讨论，本文主要贡献如下：

(1) 给出了一种基于位置敏感 Hash(LSH) 的时间序列距离度量方法，并且证明了该距离度量的正确性.

(2) 设计了基于 LSH 距离的 DS-Index 索引结构，并给出了 DS-Index 的建立、插入、删除算法.

(3) 提出了两种优化的基于 DS-Index 索引结构的时间序列子序列相似性查询算法 OLSH-Range 和 OLSH-kNN.

(4) 将基于 LSH 的时间序列固定长度子序列查询算法扩展成任意长度子序列查询.

本文第 2 节介绍时间序列子序列查询的相关工作；第 3 节描述基于 LSH 的时间序列距离度量方法；第 4 节给出基于 LSH 距离的 DS-Index 索引及其建立、插入和删除算法；第 5 节给出基于索引的 OLSH-Range 和 OLSH-kNN 两种优化算法；第 6 节给出基于 LSH 的时间序列任意长度子序列查询算法；第 7 节是在真实股票数据集上的实验结果及其分析；最后一节是本文总结.

2　相关工作

本文相关的工作主要涉及到以下 3 个方面：

(1) 时间序列的距离度量

距离度量主要有 3 种：传统的欧几里得距离度量、基于动态时间规整(DTW)[3-4] 和基于编辑距离的距离度量.

欧几里德距离度量把时间序列的第 i 个点和另一个时间序列的第 i 个点比较，相对简单和直观，计算距离的时间复杂度是线性的；DTW 允许时间序列的延伸和压缩，查找结果优于欧几里得距离度量，如果两条时间序列的长度分别是 m 和 n，则计算距离的时间复杂度是 $O(mn)$，时间复杂度较高，但可以使用下界函数加速查询速度. 而基于编辑距离的距离度量需要先将时间序列离散化成字符序列，常用的有 LCSS[5]，其利用最长公共子序列模型，优点是对噪声点具有鲁棒性，若阈值参数 ε 已知且它们的距离小于 ε，则两条时间序列的两个点是匹配的；EDR[6] 是基于编辑距离的另一种相似性度量，其利用了空白长度数据率计算两条时间序列间的差距. ERP[7] 结合了 DTW 和 EDR 的优点，通过连续变化的参考点来计算两条时间序列的距离. 本文所设计的基于 LSH 的距离度量是基于 L_p-$norm$ ($p=1, 2, \cdots, \infty$) 准则的，和欧几里德距离较为相似，其特点是计算速度快、便于索引.

(2) 时间序列相似性查询

查询主要分 4 种：全序列范围查询、全序列 k 近邻查询、子序列范围查询和子序列 k 近邻查询.

时间序列相似性查询最早是由 IBM 的 Agrawal 等人[8] 于 1993 年提出的，该问题被描述为“给定某个时间序列，要求从一个大型时间序列数据库中找出与之最相似的序列”. 文献[8]同时解决了全序列范围查询问题. 算法共有两个步骤：第 1 步是进行降维，并使用 R* 树[9] 对转换后的点进行保存；第 2 步是查询，首先将查询点也进行维度归约，然后使用转换的点进行查询，由于映射函数保证了距离下界，因此可以保证召回率. Keogh 等人[10] 提出了全序列的 k 近邻查询算法，并提出了一种 APCA 降维算法. 该算法首先找到 k 近邻上界，然后使用范围查询找到所有的 k 近邻. Faloutsos 等人[11] 提出了一种 FRM 算法，能解决子序列范围查询问题. 该算法首先使用滑动窗口对序列进行切分，然后使用降维技术对窗口降维并保存于 R* 树. 查询阶段首先将查询序列切分成互不相交几段窗口，将窗口进行降维，并使用转换后的点查找到所有候选集，最后对候选集进行验证找到所有满足条件的子序列. 窗口切分算法还有 DualMatch[12] 和 GeneralMatch[13]. DualMatch 在切分窗口时引入了二元性概念，GeneralMatch 则引入了 J 滑动窗口和 J 不相交窗口的概念. Han 等人[14] 开发了排名子序列匹配算法，解决时间序列子序列的 k 近邻问题. 该算法使用了最小距离匹配窗口对(MDMWP)，有效地减少了需要匹配的子序列的数量，极大地减少了 I/O 开销. 本文和现有这些算法的区别是，本文提出了一种新的时间序列度量方法，同时分别设计了范围查询和 k 近邻查询算法，

并设计了维度划分索引用于剪枝.

(3)时间序列降维算法

由于时间序列的超高维特性,目前已经有很多降维的方法,主要包括离散傅里叶变换(DFT)、离散小波变化(DWT)、主成分分析(PCA),或奇异值分解(SVD).本文使用的是位置敏感 Hash(LSH),主要用于设计新的距离度量.

3 基于 LSH 的时间序列距离度量

我们先简单介绍位置敏感 Hash(Locality Sensitive Hash, LSH). LSH 是一种高维的空间最近邻搜索算法,基本思想是将距离上较近的点大概率地映射到同一个 Hash 桶内(Hash 桶的个数远远地小于输入点数的总和),形式化定义[15]如下.

定义 1(位置敏感 Hash 族). 对于 Hash 族 H,如果任意两点满足以下条件,则认为 H 是(R,c,P_1,P_2)敏感:

(1)如果$\|p-q\|\leq R$,则$Pr_H(h(p)=h(q))\geq P_1$.

(2)如果$\|p-q\|\leq cR$,则$Pr_H(h(p)=h(q))\leq P_2$.

条件(1)保证了两个相近的点以高概率被映射到同一个 Hash 桶,条件(2)则保证了两个相异的点以低概率被映射到同一个 Hash 桶.值得注意的是,只有当 $P_1>P_2$时,此 Hash 族才有实际意义.

这里我们采用文献[16]中的 Hash 函数定义 $h_i(v)=\left\lfloor\frac{\boldsymbol{a}_i\cdot v+b_i}{\omega}\right\rfloor$,其中$\omega$是窗口长度参数(文献[17]推荐使用 $\omega=4$),$\boldsymbol{a}_i$是一个 d 维向量,其中每一维的值都满足标准正态分布.b_i是随机偏离参数,满足[0,ω]均匀分布.

定义 2(d-Hash 函数). 将长度为 l 的序列 v 进行 d 次 Hash 并连接,构成 d-Hash 签名:$H(v)=\langle h_1(v),h_2(v),\cdots,h_d(v)\rangle$.

定义 3(LSH 距离). 对于两个长度为 l 的序列 x,y,其 LSH 距离为 $D_{\mathrm{LSH}}(x,y)=Pr_H(h(x)\neq h(y))$.

注意:这里的距离表示为概率值,因此距离值域是[0,1],0 代表距离最近,1 代表距离最远.

定理 1. 距离保序性.

令查询序列为 q,如果时间序列 q_A和 q_B满足:

$\|q_A-q\|>\|q_B-q\|\geq 0$,则 $D_{\mathrm{LSH}}(q_A-q)>D_{\mathrm{LSH}}(q_B-q)\geq 0$.

证明. 根据定义 1,对于时间序列 x,y,若设$\sigma=\|x-y\|$,就有

$$p_q(\sigma)=Pr_H(h(x)=h(y))=\int_0^{\omega}\frac{1}{\sigma}f_p\left(\frac{t}{\sigma}\right)\left(1-\frac{t}{\omega}\right)\mathrm{d}t,$$

其中 $f_p(t)=\int_0^t\sqrt{2/\pi}\exp(-x^2/2)\mathrm{d}x$.

可见 $f_p(t)$是一个严格递增函数,可知$Pr_H(h(x)=h(y))$是一个随着 σ 递减的函数,又 $D_{\mathrm{LSH}}(x,y)=Pr_H(h(x)\neq h(y))=1-Pr_H(h(x)=h(y))$.

因此,$D_{\mathrm{LSH}}(x,y)$随$\|x-y\|$递增. 证毕.

在此体系结构中,定义 3 的 LSH 距离等价为 Hash 签名的海明距离(Hamming Distance)[18],即

$$D_{\mathrm{Hamming}}(H(x),H(y))=\sum_{i=1}^{d}f_H(h_i(x),h_i(y))\Big/d,$$

其中,$f_H(h_i(x),h_i(y))=\begin{cases}1, & h_i(x)\neq h_i(y)\\ 0, & h_i(x)=h_i(y)\end{cases}$,$d$ 为 Hash 签名的长度.

由于海明距离无法区分微小的距离差距,因此在这里我们使用曼哈顿距离(Manhattan Distance)[19],即

$$\begin{aligned}D_{\mathrm{LSH}}&=D_{\mathrm{Manhattan}}(H(x),H(y))\\&=\sum_{i=1}^{d}f_M(h_i(x),h_i(y))\Big/d,\end{aligned}$$

其中

$$f_M(h_i(x),h_i(y))=\begin{cases}|h_i(x)-h_i(y)|/\varphi, & |h_i(x)-h_i(y)|<\varphi\\ 1, & |h_i(x)=h_i(y)|\geq\varphi\end{cases},$$

在这里 φ 是一个规约参数,实验中设为 $\varphi=10$.

对于两个长度为 l 的序列 x,y,其计算 LSH 距离的时间分为两个步骤:第 1 步,转换为 d-Hash 签名;第 2 步计算 $D_{\mathrm{Manhattan}}(H(x),H(y))$.由于转换为 d-Hash 签名可以预处理,因而计算 LSH 距离的复杂度是仅仅和 d-Hash 签名长度有关,为 $O(d)$.

距离保序性证明了 LSH 距离的正确性,即两条序列的欧几里德距离越小,其 LSH 距离也越近.序列在 HASH 签名空间下和原空间的距离是等价的.所以使用 LSH 距离进行子序列的相似性度量可以得到正确的相似结果集.

4 DS-Index 索引

本节介绍一种全新的基于 LSH 的序列索引结构,称作 DS-Index 索引(Dimension Split Index).DS-Index 是二叉树结构,下文中我们使用 NL 代表 DS-Index 非叶结点,用 L 代表 DS-Index 叶节点.

叶节点 L 的结构为简单的数据块,保存不超过 θ 的 d-Hash 签名数据点.由此下文中的 L 除了代表

叶节点，同时也代表该叶节点的 Hash 签名点集.

4.1　相关定义与定理

定义 4(维度边界). 设有 d-Hash 签名集合 $A=\{x\mid x=\langle x^1,x^2,\cdots,x^d\rangle\}$，那么 A 在第 i 维的维度边界被定义为 $b^i(A)=\langle\min\{x^i\},\max\{x^i\}\rangle$，$x\in A$，这里上下界分别是该 d-Hash 签名集合的最小和最大值. 使用 $b^i_{\text{lower}}(A)$ 代表下界，使用 $b^i_{\text{upper}}(A)$ 代表上界.

定义 5(索引边界). 设 L 为索引叶节点，Hash 签名维数是 d，叶节点 L 的索引边界由 d 个维度边界组成，定义为 $B(L)=\langle b^1(L),b^2(L),\cdots,b^d(L)\rangle$.

定义 6(索引距离). Hash 签名 x 到某索引叶节点 L 的距离定义为 $D(x,L)=\sum_{i=1}^{d}D_{\text{Manhattan}}(x^i,b^i(L))$. 其中

$$D_{\text{Manhattan}}(x^i-b^i(L))=\begin{cases}f_M(x^i,b^i_{\text{lower}}(L)), & x^i<b^i_{\text{lower}}(L)\\ 0, & b^i_{\text{lower}}(L)\le x^i\le b^i_{\text{upper}}(L)\\ f_M(x^i,b^i_{\text{upper}}(L)), & x^i>b^i_{\text{upper}}(L)\end{cases}.$$

定理 2. 索引距离下界定理.

Hash 签名 x 到叶节点 L 的距离满足 $D(x,L)\le\min\{D_{\text{Manhattan}}(x,y)\mid y\in L\}$.

证明. 由于曼哈顿距离是所有维度距离分量的累加，因此我们只需要证明索引距离的每个维度距离分量都是最小的即可. 共有 3 种情况：

(1) 当 $x^i<b^i_{\text{lower}}(L)$ 时，由于 $b^i_{\text{lower}}(L)\le\{y^i\mid\forall y\in L\}$，所以 $f_M(x^i,b^i_{\text{lower}}(i))\le f_M(y^i,x^i)$，$\forall y\in L$.

(2) 当 $b^i_{\text{lower}}(L)\le x^i\le b^i_{\text{upper}}(L)$ 时，由于索引距离该维分量为 0，必然是最小值.

(3) 当 $x^i\ge b^i_{\text{upper}}(L)$ 时，由于 $b^i_{\text{upper}}(L)\ge\{y^i\mid\forall y\in L\}$，所以 $f_M(x^i,b^i_{\text{upper}}(i))\le f_M(y^i,x^i)$，$\forall y\in L$.

以上 3 种情况中，索引距离的每一维距离分量都最小. 证毕.

索引距离下界定理保证了数据点到该索引叶子节点的距离是距离的下界. 通过这个性质，我们可以在保证正确性的前提下进行剪枝.

4.2　索引的建立、插入和删除算法

DS-Index 索引结构主要利用了维度划分的二分 k-means 聚类算法. L 代表索引叶节点，θ 代表叶节点所能容纳的最大数据点数；NL 代表索引非叶节点，非叶节点有两个属性，第 1 个属性表示其子孙在哪个维度进行分裂，用 ∂ 表示；另一个属性表示其子孙在 ∂ 维的哪个值进行分裂，用 $\hbar$ 表示.

算法 1. 索引建立算法(DS-IndexBuiling).

输入：Hash 签名点集 D，叶节点所能包含最大数据点数 θ

输出：非叶结点 NL

1. if $\|D\|<\theta$ then 将 D 作为叶节点 L 返回；
2. 选择维度 ∂；
3. 在 ∂ 维上，执行一次二分 k-means 聚类，得到 $D\to\{D_1,D_2\}$；
4. 计算 $\hbar=\dfrac{\max\{x^\partial\mid x\in D_1\}+\min\{y^\partial\mid y\in D_2\}}{2}$；
5. 生成非叶结点 $NL=\langle\partial,\hbar\rangle$；
6. $Node1=$DS-IndexBuilding(D_1,θ)；
7. $Node2=$DS-IndexBuilding(D_2,θ)；
8. 设 NL 左子孙为 $Node1$、右子孙为 $Node2$；
9. 返回 NL.

索引建立算法是一个递归调用的算法，每次将数据点集 D 一分为二，即 $D\to\{D_1,D_2\}$，其中 $D_1\cap D_2=\varnothing$，$D_1\cup D_2=D$ 且 $\forall x\in D_1$，$\forall y\in D_2$，$x^\partial\le y^\partial$. 由于使用了二分 k-means，因此该分裂算法与参数无关，由算法自行选择分裂点.

当有新数据点 x 来到时，使用索引插入算法进行更新. 索引插入算法主要使用非叶节点的两个属性 $\langle\partial,\hbar\rangle$ 来定位应该插入的叶节点. 首先从根节点开始，新数据点如果 $x^\partial\le\hbar$，选择根节点的左子树，否则选择根节点右子树，同样的过程反复调用直到找到叶节点. 当插入叶节点后，所在叶节点中数据点个数超过了 θ，则叶节点需要进行分裂. 分裂过程调用 DS-IndexBuilding 算法，生成两个新的叶节点和非叶结点，然后将原叶节点删除替换成新的非叶结点.

算法 2. 索引插入算法(DS-IndexInserting).

输入：索引 $index$，新数据点 x

输出：x 所属叶节点

1. $node\leftarrow index.root$；
2. while ($node\in NL$) //$node$ 不是叶节点
3. 　if $x^{node.\partial}\le node.\hbar$
4. 　then $node=node.left$
5. 　else $node=node.right$；
6. 将数据点 x 插入 $node$；
7. if $node.size>\theta$
8. then $newnode=$DS-IndexBuilding(L,θ)；
9. 替换 $node$ 为 $newnode$.

当有数据点需要删除时，使用索引删除算法来实现. 首先使用和插入算法同样的方法找到数据点所在的叶节点，然后将该数据点从叶节点中删除，若发现叶节点为空时，删除其父节点，将其父节点的父节点的子孙指针指向其兄弟节点，并删除空叶节点.

算法 3. 索引删除算法(DS-IndexDeleting).

输入:索引 $index$,需要删除的数据点 x

输出:根节点 $root$

1. $node \leftarrow index, root$;
2. while ($node \in NL$) //$node$ 不是叶节点
3. if $x^{node.\partial} \leq node.\hbar$
4. then $node = node.left$
5. else $node = node.right$;
6. 将数据点 x 从 $node$ 删除;
7. if $node.size = \theta$
8. then $node.father.father.child = node.brother$;
9. 删除 $node$, $node.father$.

该索引并非一颗平衡树,理想的树高是 $\log\|D\|$,其中 $\|D\|$ 代表数据集 D 的大小,而 DS-Index 最差情况是 $\|D\|/\theta$. 值得注意的是,我们使用索引是为了快速查询 k 近邻而并非搜索某一个特定的点,因此叶子节点内数据点的相似性成为索引好否的标准.

由于高维数据的"维灾"问题,往往使得聚类效果不尽如人意,而基于 LSH 距离度量能够有效规避"维灾"[20].

5 基于 DS-Index 索引的两种优化查询算法

由于时间序列的超高维特性使得匹配结果候选集非常大,即序列中任何一个时间点开始的子序列都可能是查询结果,从而使得子序列匹配中消耗过多的时间[5]. 所以,在本节中我们给出基于索引的 OLSH-Range 和 OLSH-kNN 两种优化算法.

定理 3. k 近邻搜索原则.

令 S 为目前访问的所有数据点中距离查询点 q 最近的 k 个数据点集合,S_{farthest} 代表这 k 个数据点中距离 q 最远的数据点. 对于数据点 p,如果 $D_{\text{Manhattan}}(p,q) > D_{\text{Manhattan}}(S_{\text{farthest}},q)$,则数据点 p 必定不是 q 的 k 最近邻.

需要注意的是,这里的 p 和 q 都是经过 LSH 签名转换后的新数据点,而非原始时间序列.

定理 4. 近邻分区剪枝原则.

令 S 为目前访问的所有数据点中距离查询点 q 最近的 k 个数据点集合,S_{farthest} 代表这 k 个数据点中距离 q 最远的数据点. 对于叶节点 L,如果 $D(q,L) > D_{\text{Manhattan}}(q,S_{\text{farthest}})$,则 $\forall x \in L$,x 必定不是 q 的 k 最近邻.

证明. 由于 $D(q,L)$ 是 q 到数据点集合 L 的距离下界,故必有 $\forall x \in L \geq D(q,L) > D_{\text{Manhattan}}(S_{\text{farthest}},q)$,根据定理 3 得证. 证毕.

由此,我们给出基于索引的固定子序列长度 k 近邻查询优化算法 OLSH-kNN.

算法 4. k 近邻优化查询算法(OLSH-kNN).

输入:索引 $index$,查询序列 q 的 Hash 签名 x,k

输出:查询序列 q 的 k 近邻结果集 S

1. $S \leftarrow \varnothing$; //S 为最大堆,最大容量为 k
2. 遍历 $index$ 中的叶节点 L
3. 计算 $D(x,L)$;
4. 根据 $D(x,L)$ 升序排序所有叶节点;
5. for each L
6. if $D(x,L) > D_{\text{Manhattan}}(x,S_{\text{farthest}})$ then break; //剩余的 L 被剪枝
7. 计算 $D(x,y)$, $\forall y \in L$,如果 $D(x,y) \leq D_{\text{Manhattan}}(x,S_{\text{farthest}})$,将 y 添加至 S;
8. return S.

k 近邻算法采用了剪枝的思想体系,将分区从近到远排序,依次检查叶子节点中的每一个数据点的距离,然后将已经计算过的数据点插入优先队列中. 当分区的索引距离大于优先队列中数据点最远距离,则表示剩下的点已经不可能成为数据点的实际 k 近邻,算法可以终止. 优先队列中的数据点即实际 k 近邻.

我们使用最大堆 S 保存 k 近邻查询结果. 该最大堆的容量为 k,当堆中已经有 k 个签名的时候,如果新加进来的签名比 S_{farthest} 小,则从最大堆 S 中删除 S_{farthest} 并将新的签名插入最大堆. 注意到要遍历 $index$ 中的叶节点 L,叶节点的个数 $n \approx \|D\|/\theta$,因此这部分的时间和叶节点的容量 θ 成反比. 排序使用快排或者归并排序,时间为 $n\log n$. 第 6 行的剪枝和叶节点内 Hash 签名的紧凑程度有关,而 θ 越大,叶节点数据越分散,剪枝效果越差,因此这部分时间和 θ 成正比.

基于索引的固定子序列长度范围查询优化算法 OLSH-Range 和 OLSH-kNN 算法相同,但不需要使用 S_{farthest} 来动态描述更新剪枝的阈值,因此比 OLSH-kNN 算法简单. 由于篇幅关系这里不给出伪代码.

6 基于 LSH 的时间序列任意长度子序列查询算法

由于位置敏感 Hash 族 H 的参数 P_1 和 P_2 都是固定长度的,LSH 距离理论上只能对这种固定长度为 d 的时间序列进行比较. 本节我们针对长度不等的时间序列,设计了一种 Hash 函数构造方法并解

决了这个问题.

定义 7(同位连接向量). 设有同一起始点但不同维数($i\neq j$)的两个向量分别为 $\boldsymbol{A}=[a_{1\cdots i}]$ 和 $\boldsymbol{B}=[b_{1\cdots j}]$,则存在($i+j$)维向量 $\boldsymbol{V}$ 是这两个向量的连接,则记作 $\boldsymbol{V}=[\boldsymbol{A};\boldsymbol{B}]$.

定义 8(错位连接向量). 设有不同起始点且不同维数($i\neq j$)的两个向量分别为 $\boldsymbol{A}=[a_{1\cdots i}]$ 和 $\boldsymbol{B}=[b_{1\cdots j}]$,若 $j>i$ 且起始点差值为 t,则存在($j+t$)维向量 $\boldsymbol{V}'$ 是这两个向量的连接,记作 $\boldsymbol{V}'=[\boldsymbol{A}_{1\cdots t};(\boldsymbol{A}_{(t+1)\cdots i}+\boldsymbol{B}_{1\cdots[(j-i)+2t]});\boldsymbol{B}_{[(j-i)+2t+1]\cdots j}]$,简记为 $\boldsymbol{V}=[(\boldsymbol{A};\boldsymbol{B})_t]$.

根据以上定义,任意长度子序列查询可分成以下两种情况:

(1) 对于长度为 $k\cdot d$ 长度的时间子序列,可以直接使用同位连接向量进行构造. 参数 a_{new} 构造为 k 个 a 的同位连接 $a_{\text{new}}=[\overbrace{a;a;\cdots,a}^{k}]$,$b_{\text{new}}$ 同理.

(2) 对于长度为 $k\cdot d+t$ 长度的时间序列,需要先使用同位连接向量,再使用错位连接向量进行构造. 参数 a_{new} 构造为 k 个 a 的同位连接加上一个错位连接 $a_{\text{new}}=[([\overbrace{a;a;\cdots,a}^{k}];a)_t]$,$b_{\text{new}}$ 同理.

通过这样的构造,我们能够实现长度大于 d 的任意长度的时间子序列相似性查询,而不需要重新构造不同长度的 Hash 函数. 令时间序列为 T,记 T 的一段从第 i 个时间点开始的长度为 d 的子序列为 $T_{i,d}$. 由于任何一段长度为 d 的子序列的 Hash 值都计算过,因此对于从第 i 时间点开始长度为 $k\cdot d$ 的时间子序列 $H(T_{i,k\cdot d})=\left\lfloor\frac{a_{\text{new}}\cdot T_{i,k\cdot d}+b_{\text{new}}}{\omega}\right\rfloor=\sum_{y=0}^{k}H(T_{i+y\cdot d,d})$;对于从第 i 时间点开始长度为 $k\cdot d+t$ 的时间子序列,同理可得,$H(T_{i,k\cdot d+t})=\left\lfloor\frac{a_{\text{new}}\cdot T_{i,k\cdot d+t}+b_{\text{new}}}{\omega}\right\rfloor=\sum_{y=0}^{k}H(T_{i+y\cdot d,d})+H(T_{i+k\cdot d-t,d})$.

即任何一段长度大于 d 的时间子序列 Hash 值都可以通过已经计算得到的长度为 d 的时间子序列 Hash 值得到.

7　实验结果及其分析

在这一节,我们实现了前文表述的多个算法,采用上海与深圳交易所时间跨度自 2007 年 1 月 1 日到 2012 年 4 月 1 日,共有 2347 只股票、2 110 544 个数据点、1 882 791 条长度为 100 的时间子序列. 所有实验在配置为 2.2 GHz 的 CPU 和 2 GB RAM 的 PC 上实现,语言为 JAVA.

我们选择基于索引的 OLSH-kNN 优化算法作为主要实验对象. 相应对比算法有 2 个,一个是使用欧几里得距离的原始序列暴力查询算法(Brute Force Euclidean-kNN);另一个是 d-Hash 签名暴力查询算法(Brute Force LSH-kNN).

7.1　DS-Index 索引开销

由于基于索引的 OLSH-kNN 优化算法使用了 DS-Index 索引,所以需要对索引的额外开销进行专门实验,以证明索引是否在时间或空间的开销上都较小.

(1) 时间开销

DS-Index 索引的时间开销主要是计算 Hash 签名和建立索引两部分:计算 Hash 签名,即使用 LSH 函数将 1 882 791 条子序列映射为 d-Hash 签名,这里取 $\omega=4$;建立索引,即在 d-Hash 签名的基础上,构造相应的 DS-Index,需要说明的是 LSH-k 近邻查询是基于 d-Hash 签名而非原始时间子序列.

图 1 是不同 Hash 签名长度的 DS-Index 索引开销比较. 从图 1 可以看到,计算 Hash 签名和建立索引的时间都与 Hash 签名长度参数 d 成正比. 其次,将计算 Hash 签名和建立索引两部分累加,不同 Hash 签名长度预处理开销都在 200 s～300 s 之间;而据多次实验结果显示,Brute Force Euclidean-kNN 算法进行 1 次传统的欧几里得距离的 k 近邻查询时间超过 13 s,在需要执行上万次 OLSH-kNN 算法的情况下,这部分时间开销可忽略.

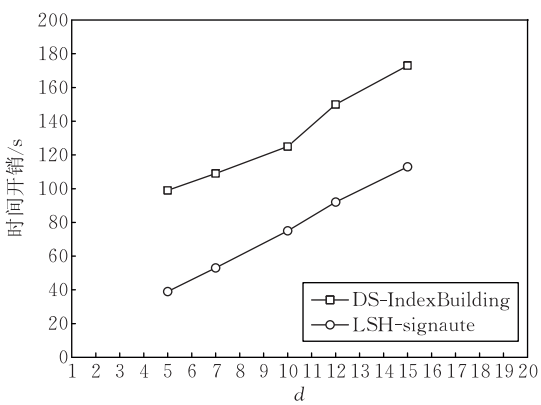


图 1　不同 Hash 签名长度的 DS-Index 索引时间开销比较

另外,由于 DS-Index 索引的数据结构是二叉树,DS-Index 索引的时间开销还与树的深度有关,这是因为树的深度决定了 DS-Index 索引插入或删

减操作的时间开销. 理想的树深是 $\log_2 \frac{\|D\|}{\theta}$，最差情况是 $\frac{\|D\|}{\theta}$. 表 1 是不同叶节点容量 θ 的 DS-Index 索引实际与理想树深比较.

表 1 不同叶节点容量 θ 的 DS-Index 索引树深比较

叶节点容量 θ	实际树深	理想树深
30	24	16
50	22	15
100	17	14
150	18	14
200	21	13

从表 1 可以看到，实际树深接近于理想树深，虽然并没有保证 DS-Index 是一颗平衡树，但是维度选择机制和基于聚类的分裂点选择使得实际树深接近于理想树深.

(2) 空间开销

由于叶节点内存储的是所有的数据点，叶节点的存储开销等于数据集的大小，因而 DS-Index 索引的额外空间开销来自于索引非叶节点个数，由于 DS-Index 索引是二叉树，其非叶结点个数是也节点个数减 1.

表 2 是不同叶节点容量 θ 的所需额外存储开销比较. 可以看出，除了叶节点对所有序列保存所必须的开销以外，索引所需要的额外开销非常的小.

表 2 不同叶节点容量 θ 的预处理空间开销比较

叶节点容量 θ	所需额外存储/KB
30	969
50	580
100	289
150	192
200	146

7.2 算法时间效能比较

我们知道，Hash 签名的长度 d 是影响时间效能的一个重要因素. d 越长，OLSH-kNN 优化算法的查询时间也越长，同时 Hash 签名的曼哈顿距离就越接近于实际概率值 $Pr_H(h(x)\neq h(y))$，距离计算也越准确.

OLSH-kNN 算法的时间分布如表 3 所示，其运行时间大致可分为两个部分：阶段 1 是计算查询点到所有叶节点的索引距离，并对索引距离进行排序；阶段 2 是依次搜索索引叶节点，直到剪枝条件达成. 阶段 2 中被搜索的点集称为候选集，候选集中点的数量除以所有点的总数为候选集占比(Candidate Rate).

表 3 OLSH-kNN 算法运行时间分布

Hash 签名长度 d	阶段 1 时间/ms	阶段 2 时间/ms	候选集占比/%
5	38	1	0.019
7	40	5	0.165
10	43	15	0.320
12	53	24	1.009
15	54	46	1.437

从表 3 可以看到，阶段 1 的时间随着 Hash 签名长度参数 d 增长而线性增长，那是因为阶段 1 的时间主要来自于索引距离的计算，索引叶节点个数不变的情况下，维度的增加导致了计算时间的增加；而阶段 2 的时间变化较大，这是因为随着维度的增加，“维灾”效应慢慢体现，索引的剪枝能力渐渐降低，因此运行时间不仅随维度增加而增加，更会随着剪枝能力的降低而增加. 总的来说，表 3 显示 OLSH-kNN 算法的剪枝能力较好.

不同算法运行时间比较如图 2 所示. 从图 2 中可以看到，随着 Hash 签名长度参数 d 的增长，Brute Force LSH-kNN 的运算时间呈线性增长，这是由于计算 Hash 签名曼哈顿距离的时间是呈线性增长的；而 OLSH-kNN 算法虽然也是增长的，但运行时间仍远小于相同情况下的 Brute Force LSH-kNN 算法，这是由于 OLSH-kNN 采用候选集剪枝策略，需要查询的数据点总比 Brute Force LSH-kNN 来得少.

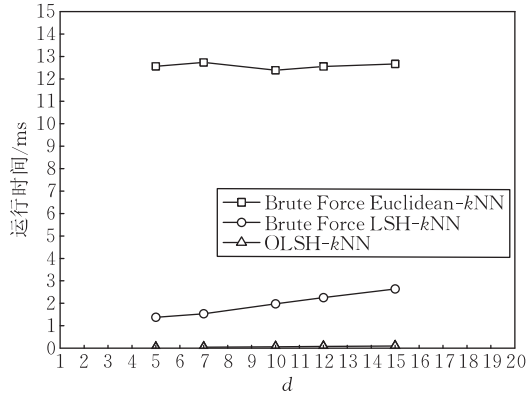


图 2 不同算法运行时间比较

OLSH-kNN 优化算法的不同叶节点容量 θ 的时间效能如图 3 所示.

可以看到，图 3 说明了叶节点最大容量 θ 和查询时间的关系：叶节点最大容量越小，叶节点的签名之间距离就越紧凑，所以剪枝效果就越好，但是由于叶节点的增多使得计算索引距离的时间增加，因此总时间反而增加. 在实验中，我们还发现使用 $\theta=100$ 能达到最优查询效果.

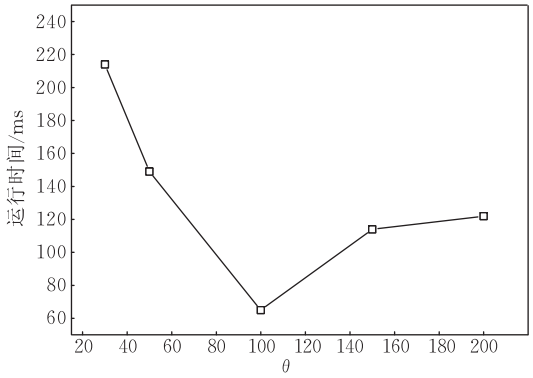


图 3　OLSH-kNN 算法的不同叶节点容量 θ 的时间效能

7.3　候选集占比关系比较

从前面实验可以知道，算法时间效能与候选集的占比(Candidate Rate)情况密不可分. 如表 3 所示，随着候选集占比不断增加，阶段 1 的时间变化随 Hash 签名长度参数 d 的变化不大，而阶段 2 则随之快速上升. 特别地，当 d 较大时，OLSH-kNN 优化算法时间将主要由阶段 2 决定，即候选集占比关系决定.

图 4 说明了候选集占比和 Hash 签名长度参数 d 的关系，横坐标是 d 以 10 为底的对数. 虽然从表 3 中发现，随着 d 的增长，候选集占比出现快速增长. 然而从图 4 的实验中，我们使用最小二乘法线性拟合，发现候选集占比和 lg(d)现线性相关，其相关系数是 0.9168. 这说明了算法在 d 较大时有良好的可扩展性.

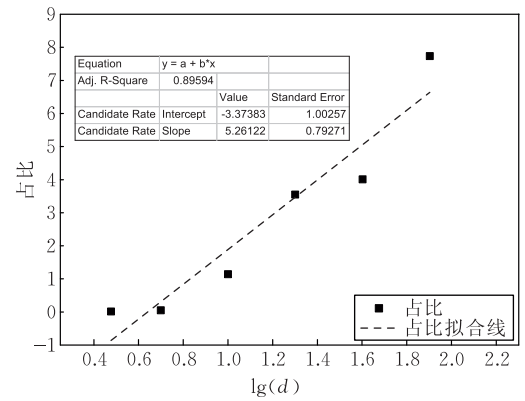


图 4　候选集占比和 Hash 签名长度参数 d 的关系

图 5 说明了候选集占比和叶节点容量 θ 的关系. 从图 5 可以看出，随着叶节点最大容量 θ 的线性增长，候选集占比以超过线性速度增长. 这是因为叶节点容量越小，每个叶节点的索引距离所代表的距离下界越紧、剪枝能力更强，而叶节点容量变大时，剪枝能力迅速降低. 另外从图 3 可以发现，OLSH-kNN 优化算法总运行时间并非是 θ 越小越好的，因为 θ 越小则代表叶节点越多，导致计算索引距离次数越多，反而增加了运行时间.

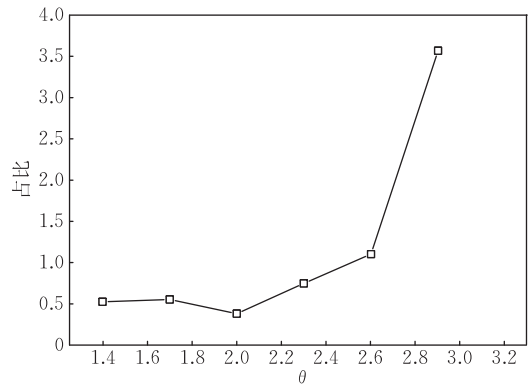


图 5　候选集占比和叶节点容量 θ 的关系

图 6 说明了候选集占比和近邻参数 k 的关系，横坐标是 k 以 10 为底的对数. 从图 6 可以看出，lg(k)线性增长时，即 k 线性增长时，而候选集占比却没有出现快速增长. 这是因为候选集占比在一定比例下，对近邻参数 k 是有范围覆盖作用的，因为候选集实际个数要比 k 大许多. 图 6 使用最小二乘法进行线性拟合，发现候选集占比和 lg(k)呈现线性相关，相关系数 0.898. OLSH-kNN 优化算法在参数 k 上的可扩展性得到了证明.

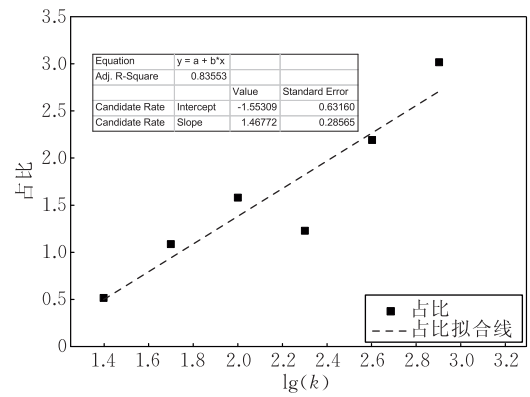


图 6　候选集占比和近邻参数 k 的关系

8　结束语

在本文中，我们较好地解决了时间子序列海量匹配的查询问题. 设计了一种全新快速的时间序列距离度量，同时设计了基于该距离度量上的索引结构 DS-Index，使用剪枝策略进一步加快了范围查询和 k 近邻查询的搜索过程. 一系列的实验表明我们的算法相比传统的基于欧几里德距离的算法快了数百倍，而额外内存开销却很小.

参 考 文 献

[1] Keogh E. Exact indexing of dynamic time warping//Proceedings of the VLDB. Hong Kong, China, 2002: 406-417

[2] Rafiei D, Mendelzon A O. Querying time series data based on similarity. IEEE Transactions on Knowledge and Data Engineering, 2000, 12(5): 675-693

[3] Berndt D, Clifford J. Finding patterns in time series: A dynamic programming approach//Advances in Knowledge Discovery and Data Mining. American Association for Artificial Intelligence. Menlo Park, CA, USA, 1996: 229-248

[4] Sakoe H, Chiba S. Dynamic programming algorithm optimization for spoken word recognition. IEEE Transactions on ASSP, 1978, 26(1): 43-49

[5] Vlachos M, Gunopulos D, Kollios G. Discovering similar multi-dimensional trajectories//Proceedings of the ICDE. San Jose, CA, USA, 2002: 673-684

[6] Chen L, Ozsu M T, Oria V. Robust and fast similarity search for moving object trajectories//Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data. New York, USA, 2005: 491-502

[7] Chen L, Ng R T. On the marriage of Lp-norms and edit distance//Proceedings of the 30th International Conference on Very Large Data Bases. 2004: 792-803

[8] Agrawal R, Faloutsos C, Swami A. Efficient similarity search in sequence databases//Proceedings of the FODO. Chicago, Illinois, USA, 1993: 69-84

[9] Beckmann N et al. The R*-tree: An efficient and robust access method for points and rectangles//Proceedings of the SIGMOD. Atlantic City, NJ, USA, 1990: 322-331

[10] Keogh E et al. Locally adaptive dimensionality reduction for indexing large time series databases//Proceedings of the SIGMOD. Santa Barbara, CA, USA, 2001: 151-162

[11] Faloutsos C, Ranganathan M, Manolopoulos Y. Fast subsequence matching in time-series databases//Proceedings of the SIGMOD. Minneapolis, Minnesota, USA, 1994: 419-429

[12] Moon Y S, Whang K Y, Loh W K. Duality-based subsequence matching in time-series databases//Proceedings of the ICDE. Heidelberg, Germany, 2001: 263-272

[13] Moon Y S, Whang K Y, Han W S. General match: A subsequence matching method in time-series databases based on generalized windows//Proceedings of the SIGMOD. Madison, Wisconsin, USA, 2002: 382-393

[14] Han W S, Lee J, Moon Y S. Jiang H. Ranked subsequence matching in time-series databases//Proceedings of the VLDB. Vienna, Austria, 2007: 423-434

[15] Andoni A, Indyk P. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. Communications of ACM, 2008, 51(1): 117-122

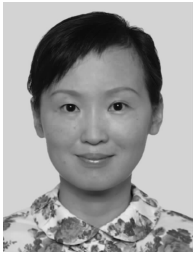
[16] Datar M, Immorlica N, Indyk P, Mirrokni V. Locality-sensitive hashing scheme based on *p*-stable distributions//Proceedings of the ASCG. New York, USA, 2004: 253-262

[17] Andoni A, Indyk P. E2LSH 0.1 user manual. MIT, Cambridge, MA, USA: Technical Report, 2005

[18] Hamming R W. Error-detecting and error-correcting codes, Bell System. Technical Journal, 1950, 29(2): 147-160

[19] Eugene F K. Taxicab Geometry: An Adventure in Non-Euclidean Geometry. New York: Dover Publications, 1987

[20] Berchtold S, Bohm C, Kriegel H-P. ThePyramid-Technique: Towards breaking the curse of dimensionality//Proceedings of the SIGMOD. Seattle, Washington, USA, 1998: 142-153



TANG Chun-Lei, born in 1976, Ph. D. candidate, engineer. Her research interests include database, data mining.

DONG Jia-Qi, born in 1987, M. S. candidate. His research interests include database, data mining.

Background

The work of this paper is supported by the Shanghai Leading Academic Discipline Project under Grant No. B114.

Time series data is a common type of data that are found in a wide range of applications such as music data, stock prices and network traffic data. Similarity search is an important problem in time series data, which can help discovering interesting patterns under complexity sequences. However, the scale of time series data is significant and grows dramatically fast. Current algorithms using DTW or Euclidean distance is time consuming, especially for subsequence matching. Therefore how to design an algorithm which can quickly find meaningful subsequence similarity search results remains to be a challenge.

The current solutions for accelerating the similarity search can be concluded as "dimension reduction-pruning-verification". The authors review the existing methods with a main focus on similarity distance measure and index construction and then propose a theoretical model to design a fast subsequence similarity search algorithm in time series data.