



一种基于 LSH 的时间子序列匹配查询算法

刘根平, 陈叶芳, 杜呈透, 钱江波

(宁波大学信息科学与工程学院 宁波 315211)

摘要:提出了一种基于 LSH(locality sensitive hashing,局部敏感散列)算法处理时间子序列匹配问题的方法 LSHSM。不同于 FRM 和 DualMatch 方法,该方法不需要对时间序列做 DFT、DWT 等特征变换,而是直接把序列看成高维数据点,利用 LSH 能处理高维数据的特性来查找相似时间子序列。实验采用 3 种不同的时间序列数据集,通过与线性扫描算法比较,验证了算法的有效性,性能有很大的提高。

关键词:时间子序列;LSH;匹配查询

doi: 10.11959/j.issn.1000-0801.2015196

An LSH Based Time Subsequence Matching Algorithm

Liu Genping, Chen Yefang, Du Chengtou, Qian Jiangbo

(Faculty of Electrical and Computer Science, Ningbo University, Ningbo 315211, China)

Abstract: An algorithm called LSHSM, which uses locality sensitive hashing (LSH) to process time subsequence matching, was proposed. Different to the FRM and DualMatch algorithms, the LSHSM does not require feature transformation such as DFT and DWT. It just directly regards the sequence as a high-dimensional object to find similar subsequences. Comparing to a linear algorithm on three real datasets, the LSHSM algorithm demonstrates the effectiveness and efficiency.

Key words: time subsequence, locality sensitive hashing, match searching

1 引言

时间序列是按时间顺序排列、随时间变化且相互关联的数据序列^[1,2]。时间序列数据出现在各种各样的领域中,包括科学计量、财务数据、传感网络、音频、视频和人类活动等。在这些序列中,典型查询有找出股票价格趋势相似

的公司、找到与某公司有相似销售模式的公司、找到近几个月和某产品销售模式相似的例子等。

时间序列的相似性查询可分为全序列匹配查询和子序列匹配查询。全序列匹配查询是从时间序列数据集中找出所有与查询序列在整体上相似的时间序列,即给定 N 个数据序列 $S_1, S_2, \dots, S_N$ 、一个查询序列 Q 以及相似性阈

收稿日期:2014-12-25;修回日期:2015-07-30

基金项目:国家自然科学基金资助项目(No.61472194),浙江省自然科学基金资助项目(No.LY13F020040),宁波市自然科学基金资助项目(No.2014A610023, No.2015A610119),“信息与通信工程”浙江省重中之重学科开放基金资助项目

Foundation Items: The National Natural Science Foundation of China(No.61472194), The Natural Science Foundation of Zhejiang Province(No.LY13F020040), The Natural Science Foundation of Ningbo City (No.2014A610023, No.2015A610119), Zhejiang Key Discipline Fund of Information and Communication Engineering

论文引用格式:刘根平,陈叶芳,杜呈透等.一种基于 LSH 的时间子序列匹配查询算法.电信科学,2015196

Liu G P, Chen Y F, Du C T, *et al.* An LSH based time subsequence matching algorithm. Telecommunications Science, 2015196



值 ε , 找到所有与 Q 相似性小于 ε 的数据序列。这里, 数据序列和查询序列长度相等。例如, 给定一支股票, 已知它在 2012–2014 年每日的收盘价, 要找出这 3 年中与它相似的所有股票, 这个问题就是全序列匹配问题。子序列匹配查询则是从比查询序列长很多的时间序列中找出所有与查询序列相似的子序列的位置偏移和长度, 即给定 N 个不同长度的数据序列 $S_1, S_2, \dots, S_N$, 一个查询序列 Q 以及相似性阈值 ε , 找到所有与 Q 相似性小于 ε 的子序列。如给定一支股票, 已知它在 2003–2005 年每日的收盘价, 在过去 30 年中找出与该股票 3 年走势相似的股票。

子序列匹配是全序列匹配的一般化问题。在数据库序列远长于查询序列, 且最优子序列匹配可能出现在数据库的任何位置时, 实现高效的子序列匹配是一个重要的问题^[9]。对于现实世界中的应用, 如哼唱查询 (query by humming, QbH)、手写文档中的字识别以及大型视频数据库和动作捕捉数据库中基于内容的检索等, 改进子序列匹配算法意义重大。

子序列匹配的算法已有很多, Faloutsos 等人^[4]提出 FRM 算法, 把数据库序列划分成滑动窗口, 查询序列划分成翻转窗口, 窗口大小为 u , 最后把这些窗口用离散傅立叶变换 (discrete fourier transform, DFT) 转换成 f 维的点, 它能处理不同长度的子序列查询。与查询序列相似的子序列能在数据库的任意位置找到。然而, 通过把数据序列划分成滑动窗口, FRM 产生太多的点需要存储。DualMatch 方法^[5]解决了 FRM 算法的这一问题。它把数据库序列划分成翻转窗口, 查询序列划分成滑动窗口。数据库序列划分成翻转窗口后, DualMatch 方法把需要存储的点降低到 FRM 算法的 $1/u$ 。

虽然通过 DFT 后, 时间序列的主要特征被提取出来, 但不能保证提取的特征是完整的, 且转换后点的维度 f 的值不能太大, 否则建立的索引结构会遭遇维数灾难。为避免这一问题, 本文不通过 DFT 的方法对时间序列进行处理, 而是采用局部敏感散列 (locality sensitive hashing, LSH) 技术。

LSH 有坚实的理论依据, 在高维数据空间中表现优异。LSH 的基本思想是将距离近的对象以很高的概率散列到同一个桶中, 而距离远的对象以很低的概率散列到同一个桶中。通过这样的处理, 可以过滤很多不相似的对象, 避免不必要的比较, 从而大大提高检索速度。因此, 被广泛应用于许多场景中, 包括基于内容的图像检索^[6]、音频检索^[7]、视频复制检测^[8,9]等。

2 相关工作

2.1 序列匹配

全序列匹配第一次出现在 Agrawal^[10]的文章中, 它的主要思想如下: 首先, 把长度为 n 的数据库序列通过 DFT 转换到频域空间, 分别提取它们的前 f 个特征 ($f < n$)。把这 f 个特征看作一个 f 维的点, 再利用空间索引方法如 R*-tree 对它们进行索引。DFT 的过程实质是一个降维的过程, 把原来 n 维的点降到 f 维。查询序列到来时, 同样采用 DFT 进行转换, 提取 f 个特征。然后在索引中进行范围查询, 找到相似性小于阈值 ε 的候选集。方法保证了得到的结果没有假阴性序列, 但会有假阳性序列, 因此最后需要进行验证。

FRM 算法是 Faloutsos^[4]提出的, 用来解决子序列匹配问题。它包括索引建立和子序列匹配两个阶段。在索引建立阶段, 首先把数据库序列划分成大小为 u 的滑动窗口, 对每个窗口进行 DFT, 转换为 f 维的点; 然后把把这些点组织成最小边界矩形 (minimum bounding rectangle, MBR); 最后把 MBR 存储在 R*-tree 中。在子序列匹配阶段, 当查询序列 Q 到来时, 把它划分成大小为 u 的翻转窗口。对每个窗口进行 DFT, 转换为 f 维的点, 然后每个点在 R*-tree 中进行范围查询, 找到候选集, 最后进行验证, 得到相似子序列。

由于 FRM 算法把数据库序列划分成滑动窗口, 存储这些数据需要很大的空间消耗, DualMatch^[5]解决了这一问题。DualMatch 在构造窗口时与 FRM 算法相反, 它把数据库序列划分为翻转窗口, 查询序列划分为滑动窗口。通过把数据库序列划分为翻转窗口而不是滑动窗口, 使 R*-tree 需要的数据存储空间减小到原来的 $1/u$, 减少了所需的存储空间, 但它允许的窗口大小比 FRM 算法小。DualMatch 可以直接在索引中存储数据点, 而不是最小边界矩形。

以上方法不管如何划分时间序列, 在划分完之后, 都要通过特征提取函数对窗口进行特征提取, 形成 f 维的点, 这个过程实质是一个降维的过程。但提取的特征不能保证信息的完整性, 且 f 的值不能太大。因为 f 太大, 索引的性能会受到影响, 导致维度灾难。基于这个考虑, 本文采用 LSH 算法对时间序列进行处理。

2.2 局部敏感散列

局部敏感散列算法^[11,12]是一种适合高维数据的流行算法。它的主要思想是距离相近的点会以很高的概率散列到

同一个桶中,距离远的点散列到同一个桶中的概率会很低。它的定义如下^[13]。

定义 1 散列函数族 $H=\{h:S \rightarrow U\}$ 是 (r_1, r_2, p_1, p_2) -敏感的, 如果对于任意的 $v, q \in S$ 有 (其中 S 为 d 维点的数据空间, $D:S \times S \rightarrow R$ 为相似性度量):

- 如果 $D(v, q) \leq r_1$, 则 $\Pr[h(v)=h(q)] \geq p_1$;
- 如果 $D(v, q) \geq r_2$, 则 $\Pr[h(v)=h(q)] \leq p_2$ 。

为使 LSH 的散列函数族有效, 需要保证 $r_1 < r_2, p_1 > p_2$ 。

不同的距离度量有不同的 LSH 函数。欧式距离下的 LSH 函数为^[14]:

$$h(x) = \left\lfloor \frac{a \cdot x + b}{w} \right\rfloor \quad (1)$$

其中, x 是 d 维的向量, a 是另一个 d 维向量, 它的每一维都服从 p -稳定分布。当 $p=2$ 时, 这个分布为均值为 0、方差为 1 的高斯分布 $N(0, 1)$ 。 $a \cdot x$ 是向量 a 和 x 的点积。 w 是一个常量, 变量 b 服从 $[0, w]$ 的均匀分布。

通过使用 LSH, 可以保证距离近的点冲突的概率大于距离远的点。这样可以过滤很多不相似的对象, 避免不必要的比较, 从而大大提高检索速度。因此, 被广泛应用于许多场景中, 包括基于内容的图像检索、音频检索、视频复制检测等。

参考文献[6]使用 LSH 进行卫星图像检索, 参考文献[15]使用 LSH 对局部描述子进行索引, 参考文献[16]中使用 LSH 进行 3D 物体索引, 参考文献[7]中使用两级 LSH 索引机制对多变音轨进行检测, 许多视频复制检测系统也使用 LSH 创建索引^[8,9]。除此外, LSH 还被用于处理 Web 数据。参考文献[17]使用 LSH 对 Web 数据进行重复检测。Google news 使用 min-hash 对 Web 上的新闻进行协同过滤^[18]。在生物数据方面, Buhler^[19,20]使用 LSH 对大规模的 DNA 序列进行相似性比对。参考文献[21,22]利用 LSH 进行离群点的检测, 参考文献[23]中利用 LSH 对时间序列产生散列签名, 并把对象不冲突的概率作为距离, 然后利用这个距离进行剪枝操作。

与参考文献[23]不同, 本文把数据库序列划分为大小相等的翻转窗口, 查询序列划分成滑动窗口。但不采用 DFT 对窗口进行转换, 而是把划分好的每个窗口看成一个高维的点, 直接用 LSH 对窗口进行散列, 既保留了时间序列信息, 又会避免维度灾难。通过散列步骤把数据库序列的窗口建成索引, 待查询序列到来时, 用查询序列的窗口在索引中进行查询, 得到候选序列。

3 LSHSM 算法及实现

3.1 基础知识

本文所用的符号及含义见表 1。

表 1 符号及含义

符号	含义
MBR	最小边界矩形
Q	查询序列
S	数据库序列
$ Q $	Q 的长度
S_{ij}	起始于 i 、终止于 j 的 S 的子序列
s_i	序列 S 的第 i 个元素
$D(Q, S_{ij})$	Q 与 S_{ij} 之间的距离

定义 2 时间序列 S 是一个随时间变化的有序序列 $S = \langle (t_1, s_1), (t_2, s_2), \dots, (t_m, s_m) \rangle$ 。其中 $(t_i, s_i) (i \in [1, m])$ 表示 t_i 时刻的值为 s_i , 后文为方便把 S 简记为 $S = s_1, s_2, \dots, s_m$ 。

时间序列可以很长, 有时包含上亿个观察值。但在本文中, 只关注时间序列的子部分, 即时间子序列。

定义 3 时间序列 S 的子序列 S_{ij} 是起始于 s_i , 终止于 s_j 的短序列, $S_{ij} = s_i, s_{i+1}, \dots, s_j$ 。

定义 4^[24] $X = x_1, \dots, x_N, Y = y_1, \dots, y_N, X$ 与 Y 之间的欧几里德距离 D 为 $D(X, Y) = \sqrt{\sum_{i=1}^N (x_i - y_i)^2}$ 。

定义 5 序列 $X = x_1, \dots, x_N, Y = y_1, \dots, y_N$ 是 ε -匹配, 当且仅当 $D(X, Y) \leq \varepsilon$ 。

引理 1^[4] 如果长度相同的两个序列 S 和 Q 是 ε -匹配, 则任意的序列对 (S_{ij}, Q_{ij}) 也是 ε -匹配。即:

$$D(S, Q) \leq \varepsilon \Rightarrow D(S_{ij}, Q_{ij}) \leq \varepsilon \quad (2)$$

这个引理保证了用窗口进行处理时不会有假阴性的序列。

引理 2^[5] 当长度相同的两个序列 S 和 Q 分别划分为 p 个窗口 ws_i 和 $wq_i (1 \leq i \leq p)$, 如果 S 和 Q 是 ε -匹配, 则至少有一对对应的窗口 (wq_i, ws_i) 是 $\frac{\varepsilon}{\sqrt{p}}$ -匹配。即:

$$D(S, Q) \leq \varepsilon \Rightarrow \exists D(ws_i, wq_i) \leq \varepsilon / \sqrt{p} \quad (3)$$

利用查询序列 Q 的某个窗口进行 $\frac{\varepsilon}{\sqrt{p}}$ 范围查询, 若在数据序列 S 中找到某一窗口, 则把与 Q 窗口偏移位置相同、长度与 Q 相等的子序列当成候选集。所以当序列划被分成窗口后, 能够找到候选子序列。



3.2 LSHSM 算法的思想

要解决的问题是：给定一个查询序列，在给定的数据库中找出与查询序列最相似的几个子序列。

首先把数据库序列分成大小为 u 的翻转窗口，不采用 DFT，而直接把这个窗口看成 u 维的点，利用 LSH 的散列函数 h 对这些窗口进行散列。

LSH 算法是一个“过滤+验证”的框架。从 LSH 的原理可以看出，散列函数的过滤效果越好，需要进行的距离计算次数会越少，查询效率也会越高。即距离近的窗口冲突的概率越大，距离远的窗口冲突的概率越小，LSH 的查询质量和查询效率就会越高。如果只使用一个 LSH 函数会产生很多的假阳性点，过滤效果不理想。但如果使用的散列函数个数太多，又会导致假阴性点增多。所以从 H 散列函数族中随机均匀独立地选择 k 个 $h_i(\cdot)$ 组成函数 $g(\cdot) = \langle h_1(\cdot), h_2(\cdot), \dots, h_k(\cdot) \rangle$ 。通过散列函数 $g(\cdot)$ 将数据序列所有窗口映射到一个散列表中 $T(\cdot)$ ；但一个散列表可能会把真正相似的窗口遗漏，故采用 L 个这样的函数 $g_1(\cdot), g_2(\cdot), \dots, g_L(\cdot)$ ，每一次把数据库中所有的窗口都散列到散列表中，共有 L 个散列表。

待给定的查询序列 Q 到来后，先把它划分成长度为 u 的滑动窗口 wq_i ，每次滑动一格。对每个窗口分别计算 $g_1(wq_i), g_2(wq_i), \dots, g_L(wq_i)$ 。以落入散列表 $T_j(\cdot)$ 的桶 $g(wq_i)$ 中的所有窗口作为候选集。

经过上述处理，假设与查询窗口 wq 距离为 r 的窗口通过 $h(\cdot)$ 散列与 wq 冲突的概率为 p_0 ，那么该窗口与 wq 在 L 张散列表中至少冲突一次的概率为 $1 - (1 - p_0^k)^L$ ，于是与 wq 距离不大于 r 的点被作为查询结果而返回的概率就不小于 $1 - (1 - p_0^k)^L$ 。选择合适的 k, L 会得到比较好的结果。

这样对于候选集中的所有窗口，计算其起始偏移位置，获取与查询序列 Q 长度相同的子序列作为候选子序列，比较其与 Q 之间的距离，选出距离最近的 K 个子序列。

所以对数据序列的每个窗口根据式 (1) 进行散列计算，计算 k 次，得到一个 $g(\cdot)$ ，存储在散列表中，如此重复 L 次，得到 L 个散列表，这样就建立好了索引。待查询序列 Q 到来时，首先把它划分成滑动窗口，总共有 $|Q| - u + 1$ 个滑动窗口，对每个窗口同样运用 LSH 进行散列处理，取出 Q 的每个窗口所在的 L 个桶 $g(\cdot)$ 中的数据序列窗口，由之前分析可知，这些窗口将以 $1 - (1 - p_0^k)^L$ 的概率被返回。

图 1 说明了 LSH 处理序列的过程。数据序列 S 被划分成翻转窗口， ws 是它的一个窗口。每个窗口经过两个 $g(\cdot)$ 函数处理后，映射到散列表 T_1, T_2 中。查询序列 Q 到来时，

首先划分成滑动窗口， wq 是它的一个窗口。 wq 经过两个 $g(\cdot)$ 处理后，映射在 T_1 的 2 号桶中、 T_2 的 2 号桶中。取出这两个桶中的数据序列窗口作为候选，进行后处理。在后处理中，由于要找的与查询序列长度相同的子序列，所以要计算子序列的起始偏移。图 2 描述了计算子序列起始偏移的过程。

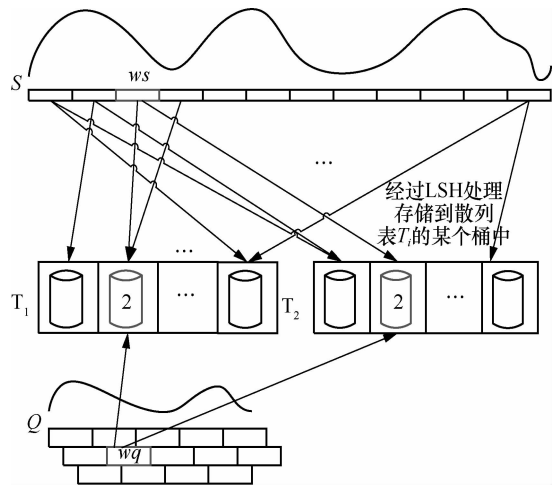


图 1 LSH 处理过程

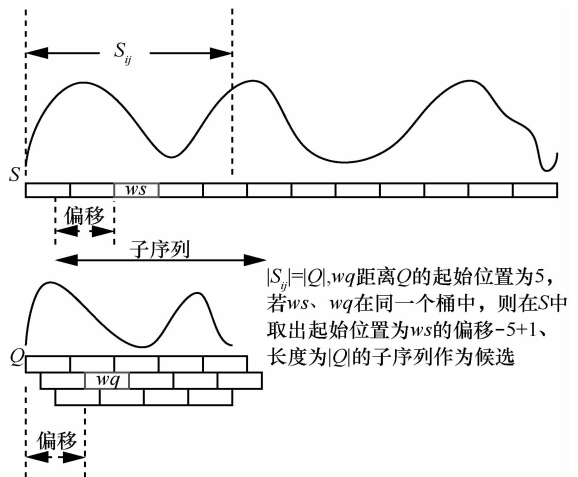


图 2 算法后处理

3.3 算法实现

3.3.1 构建索引算法实现

输入：数据库序列 S ，窗口大小 u ，LSH 参数 k, L 。

输出：LSH 索引。

构建索引算法的实现步骤如下：

- (1) 初始化索引；
- (2) 对于数据序列 S ，把 S 划分为 $\lfloor |S|/u \rfloor$ 个翻转窗口；
- (3) 对于每个窗口，对 L 个散列表中的每个表，用 LSH 散列函数 h_i 进行散列值的计算，重复 k 次；
- (4) 存储信息到散列表中。

3.3.2 查询算法实现

输入:索引,查询序列 Q 。

输出:与 Q 相似的子序列。

查询算法实现步骤如下。

(1)初始化。把 Q 划分为 $|Q|-u+1$ 个滑动窗口。

(2)查询索引。对于每个滑动窗口,对 L 个散列表中的每个表,用 LSH 散列函数 h_i 进行散列值的计算,重复 k 次,得到桶信息;取出窗口所在桶的所有数据当成候选。

(3)后处理。对候选集中的每个记录;在数据库找出对应的子序列;计算与 Q 的距离;若相似,则输出。

4 实验验证

本节将通过合成数据集和真实数据集来评估本文所提出算法的性能,并和线性扫描算法进行比较。算法用 Java 语言实现,实验在 Intel Core i3-320 3.3 GHz、4 GB 内存的机器上进行。

4.1 数据集

利用几个常用的用于评估算法的时间序列数据集,包括合成数据集 Walk1M、真实数据集 stock 和 stock2,它们的数据集信息见表 2。

表 2 数据集信息

数据集	长度/数据点个数
Walk1M	1 000 000
stock	1 916 833
stock2	3 240 000

(1)Walk1M

Walk1M[4, 5, 25]包含 1 000 000 个数据点。它的第一个点设置为 1.5,而其之后的点是通过在前一个点的基础上加上一个范围在 $[-0.001, 0.001]$ 的随机值形成的。

(2)stock

stock 数据集包含纽约证券交易所的 408 家公司 25 年中每日的收盘价。数据集是从 Yahoo Finance^[26]收集的。序列总长度为 1 916 833 个数据点。

(3)stock2

stock2^[26]数据集包含纽约证券交易所的 500 家公司 25 年中每日的开盘价。数据集是从 Yahoo Finance 收集的。每个序列长度为 6 480 个数据点,把所有序列级联成一个长序列,长度为 3 240 000 个数据点。

4.2 实验对比

LSHSM 算法是一个近似算法,需要验证它的有效性。

因此,采用与线性扫描算法进行 Top5 查询对比的方式验证 LSHSM 算法的有效性。

(1)评价标准

3 个评价标准:运行时间、准确率、访问率。

运行时间:在相同运行环境下,算法运行时间越少,说明算法越有效。

假设查询点为 Q ,用 $I(Q)$ 表示查询返回的结果集合, $A(Q)$ 表示数据集中所有满足查询条件的点的集合。召回率和精度的定义为:

$$recall = \frac{|A(Q) \cap I(Q)|}{|A(Q)|} \tag{4}$$

$$precision = \frac{|A(Q) \cap I(Q)|}{|I(Q)|} \tag{5}$$

召回率 $recall$ 表示查询结果集中满足查询条件的点的数量与数据集中所有满足查询条件点的数量比例,精度 $precision$ 表示查询结果集中满足查询条件的点的数量与查询结果集中所有点的数量的比例^[27]。在 $Topk$ 查询中,召回率等于精度。用召回率表示准确率。

访问率是指访问数据库的比例。用 $C(Q)$ 表示 LSH 查询得到的候选子序列集合, $M(Q)$ 表示数据库中所有长度为 $|Q|$ 的子序列集合。访问率可以表示为:

$$selectivity = \frac{|C(Q)|}{|M(Q)|} \tag{6}$$

(2)算法的有效性

图 3、图 4、图 5 验证了 LSHSM 算法的有效性。在实验中,由于各数据集的特性各不相同,所以 LSHSM 算法取的参数值不尽相同。在 Walk1M 数据集中,窗口大小 u 为 100,散列表个数 L 为 1,散列函数个数 k 为 5;stock 数据集中,窗口大小 u 为 50,散列表个数 L 为 2,散列函数个数 k 为 4;stock2 数据集中,窗口大小 u 为 50,散列表个数 L 为 3,散列函数个数 k 为 3。

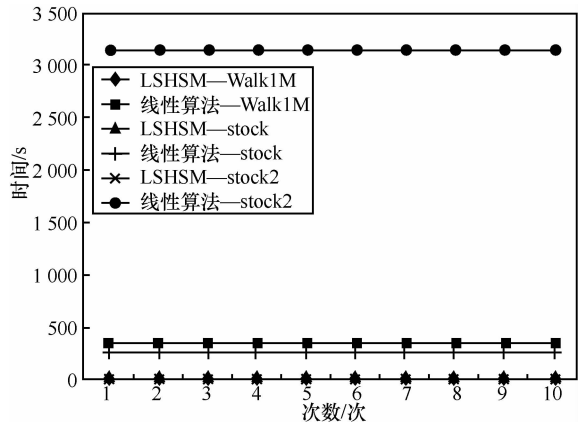


图 3 与线性扫描算法运行时间的对比

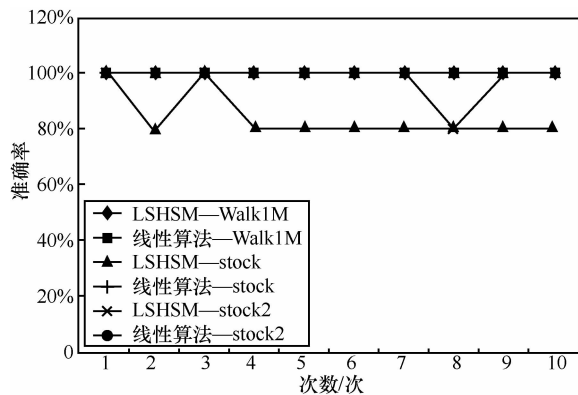


图4 与线性扫描算法准确率的对比

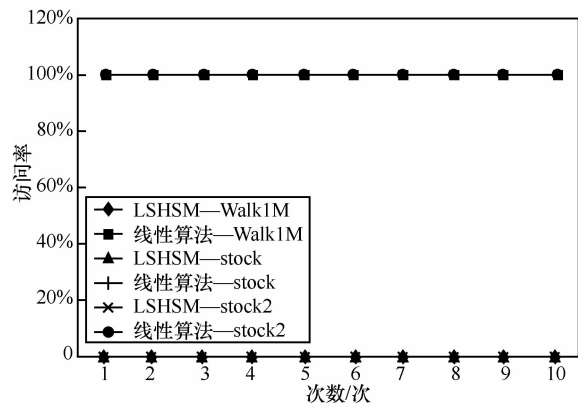


图5 与线性扫描算法访问率的对比

图3、图4、图5分别是LSHSM算法与线性扫描算法在3个数据集上进行Top5查询运行10次的运行时间、准确率、访问量的对比。可以看出,LSHSM算法的运行时间有很大的提高;LSHSM算法的准确率接近100%,几乎和线性扫描相同;相比于线性扫描,LSHSM算法访问的数据库比率小了3个数量级。

实验证明LSHSM算法是有效的,且性能比线性扫描算法有极大的提高。

4.3 参数的影响

LSHSM算法中参数 k 、 L 、 u 的选择对算法性能有很大的影响。 k 太小,散列到同一个桶的概率比较大,得到的候选对象会很多,其中包括许多假阳性对象,增加计算开销; k 太大,散列到同一个桶的概率变小,得到的候选对象变少,这样不仅把假阳性对象过滤掉,同时也把真正近似的对象过滤掉,准确率不高。 L 越小,散列到同一个桶中的概率越小,得到的候选对象越少,找到真正近似对象的准确性越低; L 越大,散列到同一个桶中的概率越大,得到的候选对象越多,找到真正近似对象的准确性越高,但同时访问数据库的比率会越大。所以,需要考虑这两个参数的影响,使算法的性能最好。

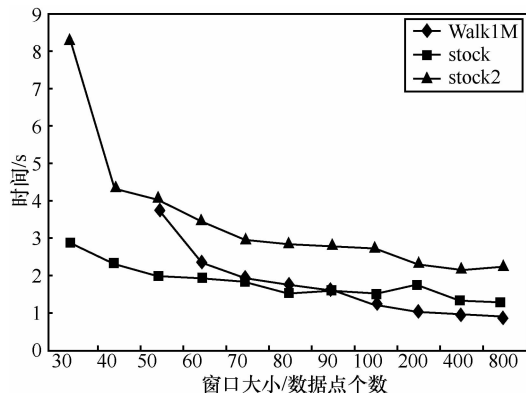
实验中考虑3个参数对算法性能的影响:散列表的个

数 L 、散列函数个数 k 以及划分窗口大小 u 。理论上来说, u 越大,LSHSM算法计算越快, u 太大可能导致找不到相似子序列。另一方面, u 越小,越能精确找到子序列,准确度越高,但是窗口太小,LSHSM算法所花的时间会越多。

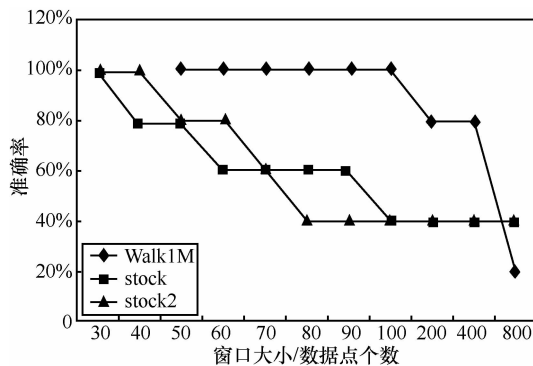
在Walk1M数据集中, $k=5$, $L=1$, $u=80$;stock数据集中, $k=4$, $L=2$, $u=50$;stock2数据集中, $k=4$, $L=3$, $u=50$ 。实验中,在改变其中一个参数时,其他两个参数固定不变。

(1) 窗口大小 u 对算法的影响

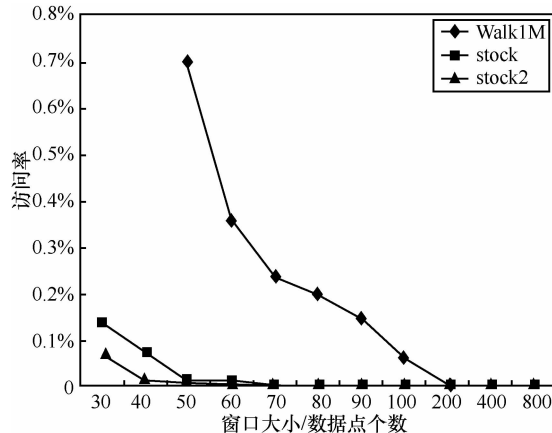
图6显示了在固定 k 和 L 时,窗口大小 u 对算法性能的影响。



(a) 运行时间



(b) 准确率



(c) 访问率

图6 窗口大小 u 对算法性能的影响

从图 6 中可以看出, u 变大时, 算法构建的索引会减小, 查询窗口个数也随着减小, 时间呈整体下降的趋势。但随着 u 的增大, LSH 得到的候选集会减小, 导致假阴性节点增加, 算法的准确性会有所下降。

(2) 散列表个数 L 对算法的影响

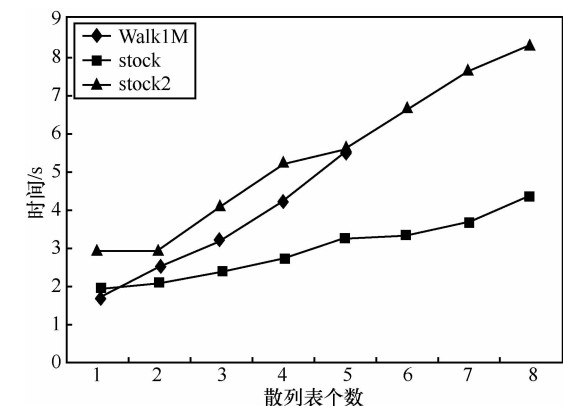
图 7 显示了在固定 k 和 u 时, 散列表个数 L 对算法性能的影响。 L 达到一定数值, 准确率已达到很高, 再继续增大 L , 得到的效果也已经达到饱和。特别是 Walk1M 数据集中, 在

$L=5$ 时的效果已经非常好, 没有必要再进行更大 L 的比较。

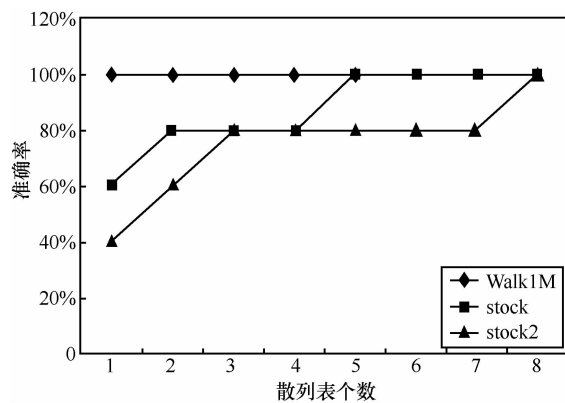
从图 7 中可以得出, 其他参数固定时, 随着散列表个数的增加, 算法构建的索引增大, 需要查询的散列表增多, 候选集增加, 因此数据访问率呈增长趋势, 查询所用的时间也呈上升趋势。散列表个数增大会增加两个对象冲突的概率, 使得算法准确性增加。

(3) 散列函数个数 k 对算法的影响

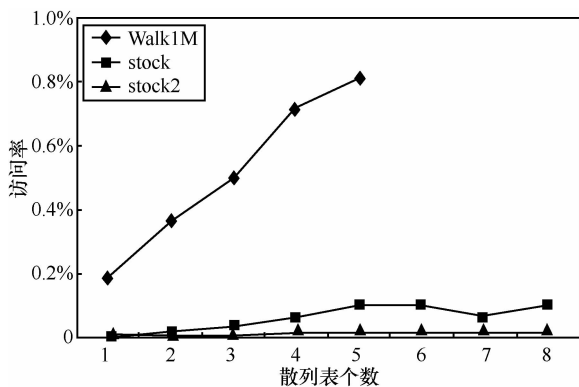
图 8 显示了在固定 u 和 L 时, 散列函数个数 k 对算法



(a) 运行时间

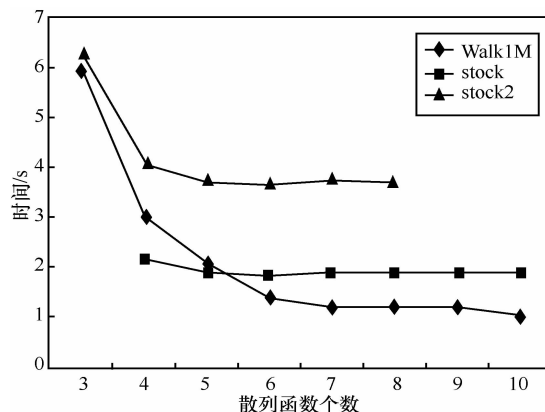


(b) 准确率

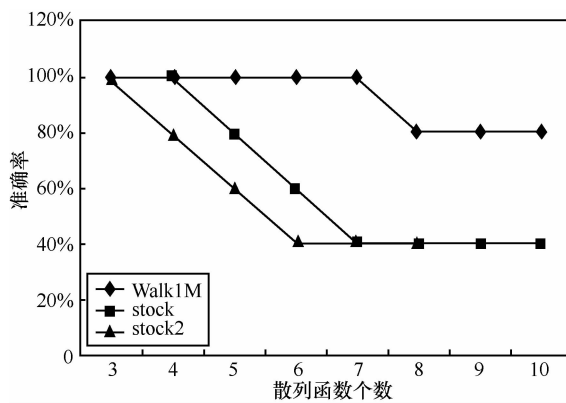


(c) 访问率

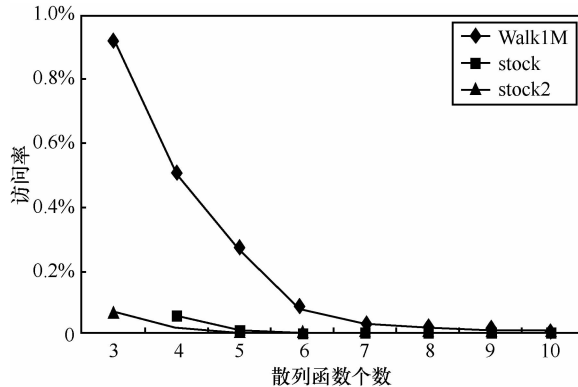
图 7 散列表个数 L 对算法性能的影响



(a) 运行时间



(b) 准确率



(c) 访问率

图 8 散列函数个数 k 对算法性能的影响



性能的影响。

从图 8 可知,其他参数固定时,随着散列函数个数 k 的增加,候选集会减少,数据访问率会降低,所需查询时间呈下降趋势。但是随着散列函数个数的增加,会把真正的近邻散列到其他桶中,准确率会有所下降。

5 结束语

时间序列的相似性查询已有很长的研究历史,在子序列匹配问题中,传统基于树的方法虽然能很好地解决低维空间的问题,但当数据维度增加时,它们的效果甚至不如线性扫描算法。本文提出了用 LSH 技术与时间序列结合的方法,既能避免高维数据导致的维度灾难,又能保留时间序列的主要特征。利用 LSH 的局部敏感性,把相似的子序列聚集到一起,过滤掉很多不相似的对象,减少所需的比较次数,提高效率。实验结果表明,该方法在性能上有很大的提升。

参考文献

- 肖辉. 时间序列的相似性查询与异常检测(博士学位论文). 复旦大学, 2005
Xiao H. Similarity search and outlier detection in time series (doctor dissertation). Fudan University, 2005
- 曾海泉. 时间序列挖掘与相似性查找技术研究(博士学位论文). 复旦大学, 2003
Zeng H Q. Research on mining and similarity search in time series database (doctor dissertation). Fudan University, 2003
- Fu T. A review on time series data mining. *Engineering Applications of Artificial Intelligence*, 2011, 24(1): 164~181
- Christos F, Ranganathan M, Manolopoulos Y. Fast subsequence matching in time-series databases. *Proceedings of ACM SIGMOD Conference*, Minneapolis, USA, 1994
- Moon Y S, Whang K Y, Loh W K. Duality-based subsequence matching in time-series databases. *Proceedings of the 17th International Conference on Data Engineering*, Heidelberg, Germany, 2001: 263~272
- Ruben B, Homaifar A, Gebril M, *et al.* Satellite image retrieval using low memory locality sensitive hashing in Euclidean space. *Earth Science Informatics*, 2011, 4(1): 17~28
- Yi Y, Michel C, Vincent O, *et al.* Local summarization and multi-level LSH for retrieving multi-variant audio tracks. *Proceedings of the 17th ACM International Conference on Multimedia*, Beijing, China, 2009: 341~350
- Paisitkriangkrai S, Mei T, Zhang J, *et al.* Scalable clip-based near-duplicate video detection with ordinal measure. *Proceedings of the ACM International Conference on Image and Video Retrieval*, Xi'an, China, 2010: 121~128
- Zhu L, Liu T, Gibbon D, *et al.* Effective and scalable video copy detection. *Proceedings of the 11th ACM International Conference on Multimedia Information Retrieval*, Philadelphia, USA, 2010: 119~128
- Agrawal R, Faloutsos C, Swami A. Efficient similarity search in sequence databases. *Proceedings of the 4th International Conference FODO*, Chicago, USA, 1993: 69~84
- Indyk P, Motwani R. Approximate nearest neighbors: towards removing the curse of dimensionality. *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, Dallas, USA, 1998: 604~613
- Gionis A, Indyk P, Motwani R. Similarity search in high dimensions via hashing. *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB)*, Edinburgh, UK, 1999: 518~529
- 甘骏豪. 基于动态碰撞检测的位置敏感哈希(硕士学位论文). 中山大学, 2013
Gan J H. Locality-sensitive hashing scheme based on dynamic collision counting (master dissertation). Sun Yat-sen University, 2013
- Datar M, Nicole I, Indyk P, *et al.* Locality-sensitive hashing scheme based on p-stable distributions. *Proceedings of the 20th Annual Symposium on Computational Geometry*, New York, USA, 2004: 253~262
- Yan K, Sukthankar R, Huston L. Efficient near-duplicate detection and sub-image retrieval. *ACM Multimedia*, 2004, 4(1): 869~876
- Bogdan M, Shan Y, Sawhney H S, *et al.* Rapid object indexing using locality sensitive hashing and joint 3D-signature space estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2006, 28(7): 1111~1126
- Manku G S, Jain A, Sarma A D. Detecting near-duplicates for web crawling. *Proceedings of the 16th International Conference on World Wide Web*, Banff, Alberta, Canada, 2007: 141~150
- Das A S, Datar M, Garg A, *et al.* Google news personalization: scalable online collaborative filtering. *Proceedings of the 16th International Conference on World Wide Web*, Banff, Alberta, Canada, 2007: 271~280
- Buhler J. Efficient large-scale sequence comparison by locality-sensitive hashing. *Bioinformatics*, 2001, 17(5): 419~428
- Buhler J. Provably sensitive indexing strategies for biosequence similarity search. *Journal of Computational Biology*, 2003, 10(3~4): 399~417
- Wang Y, Parthasarathy S, Tatikonda S. Locality sensitive outlier detection: a ranking driven approach. *Proceedings of the IEEE 27th International Conference on Data Engineering (ICDE)*, Hannover, Germany, 2011: 410~421

- 22 Pillutla M R, Raval N, Bansal P, *et al.* LSH based outlier detection and its application in distributed setting. Proceedings of the 20th ACM International Conference on Information and Knowledge Management, Glasgow, Scotland, UK, 2011: 2289~2292
- 23 汤春蕾, 董家麒. 基于 LSH 的时间子序列查询算法. 计算机学报, 2012, 35(11): 2228~2236
Tang C L, Dong J Q. Similarity query of time series subsequences based on LSH. Chinese Journal of Computers, 2012, 35(11): 2228~2236
- 24 李俊奎. 时间序列相似性问题研究 (博士学位论文). 华中科技大学, 2008
Li J K. Research on time series sequence similarity (doctor dissertation). Huazhong University of Science and Technology, 2008
- 25 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 ACM SIGMOD International Conference on Management of Data, Madison, Wisconsin, USA, 2002: 382~393
- 26 Cai Y, Ng R. Indexing spatio-temporal trajectories with Chebyshev polynomials. Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, 2004: 599~610
- 27 凌康. 基于位置敏感哈希的相似性搜索技术研究 (硕士学位论文). 南京大学, 2012
Ling K. Research on locality sensitive hashing-based similarity Search (master dissertation). Nanjing University, 2012

[作者简介]



刘根平,女,宁波大学信息科学与工程学院硕士生,主要研究方向为数据流处理和挖掘。



陈叶芳,女,宁波大学信息科学与工程学院讲师,主要研究方向为数据处理和挖掘。



杜呈透[通信作者],男,宁波大学信息科学与工程学院副教授,主要研究方向为数据处理和挖掘。



钱江波,男,博士,宁波大学信息科学与工程学院教授,主要研究方向为数据处理和挖掘、逻辑电路设计。