2019 年 10 月 第 10 期
Vol. 40 No. 10 2019

基于 LSH 的时间序列 DTW 相似性查询

李 敏,于长永,张 峰,马海涛,赵宇海

(东北大学 计算机科学与工程学院,沈阳 110819)
E-mail: 1045317573@qq.com

摘 要: 提出了一种新颖的基于 LSH 的时间序列 DTW 相似性近似查询算法,较好地解决了 DTW 相似性查询速度慢的问题. 首先,分析了 DTW 相似性度量的特点,即时间弯曲的重要特性; 其次,将该特性与 LSH 函数相结合,设计了高效的 DTW 相似时间序列过滤方法. 在很大概率程度上保证了相似的时间序列至少具有一个相同的 LSH 函数值; 最后,给出了一个基于过滤加验证框架的时间序列 DTW 相似性近似查询算法,该算法利用低维的 Hash 索引加快候选集合的筛选,从而加快查询速度. 实验结果表明,在保持较好的召回率的情况下,本文提出的方法较现有算法有效地提高了 DTW 相似性查询速度.

关 键 词: 时间序列; 相似性查询; LSH; DTW
中图分类号: TP301 **文献标识码**: A **文 章 编 号**: 1000-1220(2019)10-2155-05

LSH and DTW-based for Searching Similar Time Series

LI Min, YU Chang-yong, ZHANG Feng, MA Hai-tao, ZHAO Yu-hai
(College of Information Science and Engineering, Northeastern University, Shenyang 110819, China)

Abstract: In this paper, we propose a novel algorithm, time series similarity approximate query algorithm based on LSH and DTW, which solves the problem of low efficiency of DTW similarity query. Firstly, the characteristics of DTW similarity measure, namely the important characteristics of time warping, are analyzed. Secondly, this feature is combined with LSH function to design an efficient DTW similar time series filtering method. To a large extent, it is guaranteed that similar time series has at least one identical LSH function value. Finally, a time-series DTW similarity approximate query algorithm based on filtering and verification framework is presented, which accelerates the filtering of candidate sets by using low-dimensional Hash index. The experimental results show that the proposed method effectively improves the DTW similarity query speed compared with the existing algorithms while maintaining a good recall rate.

Key words: time series; similarity query; LSH; DTW

1 引 言

时间序列挖掘目的是发现时序数据中潜在的有价值的知识,主要包括时间序列预测、聚类、异常检测、分类、相似性查询等内容. 其中相似性查询是时间序列挖掘中最根本的一个问题,在时间序列数据的分析中,处理相似性问题是研究其他相关任务的基础. 时间序列相似性查询就是在时序数据库中找到与给定序列相似的序列. 鉴于时间序列的海量性和高维性,如何快速准确的找到相似的序列是我们亟需解决的问题.

时间序列的相似度通过计算两条序列间的距离来衡量,所以时间序列的距离度量方式在相似性查询中扮演了至关重要的作用. 最常用的两种距离度量分别是欧氏距离和动态时间弯曲. 鉴于欧氏距离只能处理等长的时间序列,不允许时间轴弯曲等缺点,动态时间弯曲[14]被认为是较准确的一种度量方式,成为人们的首选. 然而由于使用动态时间弯曲计算距离的时间复杂度为 $O(n^2)$,n 为时间序列的长度,对于处理海量的高维时间序列来说,计算成本太高. 所以在尽可能的保留序列的原始信息的前提下,将高维的时间序列降到低维空间中去处理,从而降低计算的成本,是时间序列相似性查询的关键.

为加快动态时间弯曲的计算速度,Sakoe 等人[9]提出通过限制路径的搜索范围来提高计算效率,范围的大小由参数控制. Sakoe-Chiba band[9]是对称型的弯曲窗口,而 Itakura 等人[10]提出的是平行四边形的弯曲窗口,这两种经典的方法都得到了广泛的应用,减少了计算时间的消耗. 本文我们采用的局部敏感哈希技术可以实现将高维的时间序列降到低维空间中去处理,进而加快计算的速度.

为了减少数据库中序列的数量,尽可能的避免不相似的序列之间的计算,加快查询的速度,研究者提出多种计算 DTW 下界的算法. 比如 LB_Kim[1], LB_Yi[2], LB_Keogh[3],

收稿日期: 2018-12-06 收修改稿日期: 2019-01-18 基金项目: 国家自然科学基金项目(61772124, 61332014, 61401080, 61402087) 资助; 河北省自然科学基金项目(F2015501049) 资助; 河北省教育厅项目(QN2014339) 资助; 中央高校基本科研业务费项目(N150402002) 资助. 作者简介: **李 敏**,女,1995 年生,硕士研究生,研究方向为时间序列挖掘; **于长永**,男,1981 年生,博士,副教授,研究方向为大数据挖掘与知识发现、生物信息学、机器学习; **张 峰**,男,1995 年生,硕士研究生,研究方向为时间序列分析与挖掘; **马海涛**,男,1977 年生,博士,讲师,研究方向为生物信息学、卷积神经网络优化、图像识别; **赵宇海**,男,1975 年生,博士,教授,CCF 会员,研究方向为大数据挖掘与知识发现、机器学习、社交网络数据分析、推荐系统.

其中 LB_Keogh 一经提出,该方法就得到了广泛应用. LB_Keogh 找到每一条查询序列的上包络线 U 和下包络线 L,计算候选序列超出上下包络线区域的部分之和作为下界,通过证明该方法较 LB_Kim,LB_Yi 更为紧凑,不会产生漏报. 鉴于时间序列的高维性,Keogh 等人又提出了低维空间下计算下界的方法 LB_PAA[3],在一定程度上降低了动态时间弯曲的计算时间. 下界的提出是为了提前筛选掉一些不相似的序列,但是,当处理的时间序列走势波动程度较大时,现有的下界算法往往效果欠佳. 下界作为一种过滤技术是为了提前筛选掉一些不相似的序列,避免过多冗余的计算,进而加快检索的速度. 本文我们使用的局部敏感哈希同样可以作为一种过滤技术,通过提前过滤掉一些不相似的序列,大大加快了查询的速度.

时间序列相似性查询分为全序列查询和子序列查询,本文目标旨在解决全序列查询问题. 因为时间序列的高维性特点,利用 dtw 直接在原始的序列上操作会存在时间复杂度高等问题. 所以我们采用一种新颖的技术,即局部敏感哈希. Lsh 的第一个优势是将高维时间序列降到低维空间,加快了计算的速度; 第二个优势就是提前过滤掉一些不相似的序列,减少了冗余的计算,加快了查找的速度.

主要工作如下:

1) 将实数形式的时间序列转化成整型的 ID 序列.

2) 将时间序列映射到局部敏感哈希空间中. 为了保证时间序列的相似性在动态时间弯曲空间下比较和在局部敏感哈希空间下比较是等价的. 将时间序列随机抻长为原始序列的 w 倍.

3) 实现了本文的算法. 将局部敏感哈希算法和动态时间弯曲相结合,通过实验证明大大加快了检索的速度.

2 问题定义

定义 1. (时间序列) : 时间序列 T 是按相等的时间采样的数据点构成的序列,$T=(t_1,t_2,t_3,\cdots,t_n)$,$t_i(i\in\cdots n)$ 是任意的实数,n 为时间序列的长度.

定义 2. (时间序列数据库) : 时间序列数据库 TS 是 N 条时间序列的集合,$TS=(t_1,t_2,t_3,\cdots,t_n)$,$t_i(i\in 1\cdots N)$ 是一条长度为 n 的时间序列.

定义 3. (时间序列相似性查询) : 给定一条查询序列 Q、时间序列数据库 TS、相似性度量函数 SIM 和阈值 τ,时间序列相似性查询就是要从时间序列数据库 TS 中找到和查询序列 Q 相似程度大于等于 τ 的所有时间序列. 即

$$R=\{C\in TS\mid SIM(Q,C)\geqslant\tau\} \tag{1}$$

动态时间弯曲首次被应用是在语音识别领域中,是一种准确性较高的时间序列距离度量方法. 该方法区别于传统的欧氏距离,可以通过弯曲时间轴,实现一点对应多点的匹配,从而可以度量两条不等长的时间序列.

给定两条时间序列 $X=(x_1,x_2,x_3,\cdots x_m)$ 和 $Y=(y_1,y_2,y_3,\cdots y_n)$,它们的动态时间弯曲距离定义为:

定义 4. (动态时间弯曲) :

$$D(<>,<>)=0 \tag{2}$$

$$D(X,<>)=D(<>,Y)=\infty \tag{3}$$

$$D(X,Y)=d(x_1,y_1)+\min\begin{cases}D(X,Rest(Y))\\D(Rest(X),Y)\\D(Rest(X),Rest(Y))\end{cases} \tag{4}$$

$$d(x,y)=\|x-y\|_p,x\in X,y\in Y \tag{5}$$

其中,$<>$ 表示空序列,$Rest(X)=x_2,x_3,x_4,\cdots x_m$,$Rest(Y)=y_2,y_3,y_4,\cdots y_n$,$d(x_i,y_j)$ 表示 x_i 与 y_j 之间的距离.

动态时间弯曲距离就是利用动态规划的思想寻找一条具有最小弯曲代价的弯曲路径,该算法的时间复杂度为 $O(mn)$.

局部敏感哈希是一种应用于高维空间中的近似最近邻查询方法. 它的基本思想是将原本相近的对象以很大的概率 hash 到同一个桶中,而原本较远的对象 hash 到同一个桶中的概率会很低. 形式化描述[14] 如下:

定义 5. (局部敏感哈希) :

令 U 为对象的集合,d 为距离函数. 对于哈希家族 H,如果任意两个对象 x、y 满足如下两个条件,则认为 H 是($d1$,$d2$,$p1$,$p2$) 敏感的.

如果 $d(x,y)\leqslant d1$,则 $h(x)=h(y)$ 的概率至少为 $p1$;

如果 $d(x,y)\geqslant d2$,则 $h(x)=h(y)$ 的概率至多为 $p2$;

其中 $d(x,y)$ 表示 x 和 y 之间的距离,$d1<d2$,$h(x)$ 和 $h(y)$ 分别表示对 x 和 y 进行 hash 变换. 只有当 $p1>p2$ 时 H 才有意义.

条件 1. 解释了相近的两个对象会以很高的概率哈希到同一个桶中;

条件 2. 解释了不相似的两个对象会以很低的概率哈希到同一个桶中.

本文我们采用的是汉明距离下的哈希函数即:

$H=\{v_i: v_i(x_1,x_2,x_3,\cdots,x_m)=x_i\mid i\lceil m\rceil\}$.

其是 $(d1,d2,1-d1/m,1-d2/m)$ 敏感的.

2.1 将时间序列映射为 ID 序列

为了能够使用汉明距离度量时间序列的相似度,第一步需要将实数形式的时间序列数据转化成字符串. 为了避免平移、缩放等外界因素对衡量时间序列相似度带来的影响,首先需要对时序数据进行标准化,最常用的标准化公式如下:

$$S=\frac{S-\mu}{\sigma} \tag{6}$$

经过标准化之后,平移、缩放的影响就可以消除. 然后对每一维度的数据分配 ID,实数形式的数据即被整型的 ID 所代替,减少了存储代价,同时也达到了降维的目的,并且为利用汉明距离度量提供了方便. 具体计算 ID 的公式如下所述:

$$row=\lfloor\frac{x-x_{min}}{length}\rfloor+1 \tag{7}$$

$$col=\lfloor\frac{t-1}{width}\rfloor+1 \tag{8}$$

$$ID=(row-1)*COLUMN\_NUM+col \tag{9}$$

给定参数 $COLUMN\_NUM$ 即桶的个数,遍历原始序列数据找到其最大值和最小值,根据给定的 $COLUMN\_NUM$,确定每个桶的宽度,然后就可以将原始的序列分配到每一个桶中,并为其分配 ID. 实数值的时间序列则映射为 ID 序列. 算法如下所示.

算法 1. TimeSeriesTranstoIDSequence

Input:

S: time series

COLUMN_NUM: parameter

ROW_NUM: parameter

Output:

S′: ID sequence presentation of S

1. tempS←TimeSeriesNormalization(S) ;
2. MaxMin←findMaxMin(tempS) ;
3. $S'\leftarrow\emptyset$;
4. length←(($x_{max}-x_{min}$) /*ROW_NUM*) * 1.01;
5. width←((*Length*(*S*) −1) /*COLUMN_NUM*) * 1.01;
6. for every point s_i of S do
7. $row=\lfloor(x_i-x_{min})/length\rfloor+1$;
8. $col=\lfloor(i-1)/width\rfloor+1$;
9. $ID_i=(row-1)*COLUMN\_NUM+col$;
10. $S'\leftarrow S'\cup ID_i$;
11. end
12. return S';

桶的宽度决定了所能容纳的序列的点的个数,原始维度为 n 的时间序列经过映射为 ID 序列之后,维度至多为 *COLUMN_NUM*(小于 n) ,达到了降维的目的. 所以在计算 ID 序列的 DTW 距离时,会大大加快计算的速度,ID 序列的弯曲路径如图 1 所示.

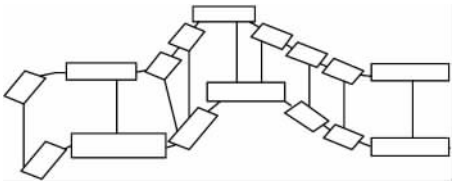

图 1　ID 序列弯曲路径匹配图

Fig. 1　ID sequence warping path matching graph

2.2　将 ID 序列投影到局部敏感哈希空间

我们的目的是为了应用汉明距离下的局部敏感哈希技术,所以第二步就是将 ID 序列映射到局部敏感哈希空间中. 映射的方法就是将不同长度的 ID 序列抻长为原始最大长度的 w 倍,每一个 ID 可能被随机抻长一次或者多次. 这样做的优势有如下几点. 第一点考虑到ID序列长短不一,大多数的

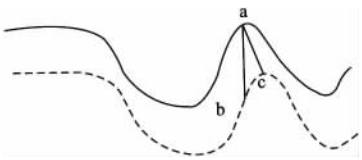

图 2　局部偏移匹配图

Fig. 2　Local offset matching graph

距离度量不能处理不等长的序列,利用抻长技术将 ID 序列抻长为长度一致的序列,从而可以更好地度量序列之间的相似程度. 第二点 DTW 在计算的过程中允许时间轴的弯曲,所以为了保证时间序列的相似性在动态时间弯曲空间下比较和在局部敏感哈希空间下比较是等价的,则将 ID 序列上的每一点随机抻长,从而满足了序列间一点对应多点的可能,保证了局部偏移不变性. 比如在图 2 中,两条序列整体上非常相似,但在 x 轴上发生了局部偏移,如果利用欧氏距离度量时,会将实线中的 a 点和虚线中的 b 点匹配,但实际应该和 c 点对应,这就会产生较大的误差,而我们类比 dtw 中允许时间轴弯曲的思想,通过抻长技术将每一个点重复一次或多次就可以消除这种误差.

为了尽可能的提高准确度,我们会对序列抻长多次,并且将每次抻长的参数保存,以便对查询序列采取相同的操作. 抻长的具体实现算法如下所示:

算法 2. StrecthIDSequence

Input:

S′: IDsequence

Output:

StretchS: stretched representation of S′

1. SeriesLength←*findLongestSeries*(*S*′) ;
2. rand←∅;
3. for i←0 to 3* SeriesLength
4. num_i←generaterandomnumber(0 or 1) ;
5. *rand*←*rand*∪{ num_i} ;
6. end
7. StretchS←∅;
8. for every point s_i of *S*′ do
9. *j*←0; *k*←0;
10. while j<3* SeriesLength
11. if(k < length(*S*′))
12. *StretchS*←*StretchS*∪{ s_k} ;
13. k←k + $rand_j$;
14. else
15. StretchS←StretchS∪{ 0} ;
16. j←j + 1;
17. end
18. end
19. return StretchS;

2.3　建立候选集并验证

在将 ID 序列映射到局部敏感哈希空间中以后,第三步就是利用局部敏感哈希技术进行过滤. 将数据库中抻长后的 ID 序列中随机选取某几个位置进行哈希,构成哈希家族,同样的我们会对抻长后的序列进行多次哈希,并将哈希过程中的参数保存,对每次的哈希家族构建前缀索引. 判断查询序列的哈希族和数据库序列的哈希族是否相同,数据库中的序列只要满足有一次的哈希家族和查询序列的相匹配,我们就将该序列作为候选序列. 然后计算该哈希族对应的原始序列和查询序列之间的 DTW 距离,如果二者的 DTW 距离小于或者等于阈值 τ,则认为二者是相似的. 每一条候选序列的验证步骤重复如此. 对一条序列的哈希算法如下所示:

算法 3. HashIDSequence

Input:

S′: IDsequence

m: hash location number

Output:

HashS: hashed representation of *S*′

1. hash←∅;
2. for *i*←0 to m
3. $num_i\leftarrow random\ number(0\sim Length(S'))$; 4. $hash\leftarrow hash\cup\{num_i\}$;
5. for *j*←0 to *i*
6. while $hash_j==hash_i$
7. $num_i\leftarrow random\ number(0\sim Length(S'))$;
8. end

9. $hash \leftarrow hash \cup \{num_i\}$;
10. end
11. end
12. Hash$S \leftarrow \emptyset$;
13. for i←0 to m
14. $HashS \leftarrow HashS \cup \{S'_{hash_i}\}$;
15. end
16. return HashS;

3 实验结果

3.1 实验环境与数据

本文实现了提出的算法 TQLD,采用基于下界的 DTW 算法 LB_DTW 和基于集合的相似性查询算法 STS3,进行了准确性和有效性的对比. 全部算法采用 Java 语言实现,运行在一台 CPU 为 Intel(R) Core(TM) i5-4200U 1.60gHz,主存大小为 4GB 的 PC 机上,操作系统为 win10.

表 1 数据集描述

Table 1 DataSet description

DataSet	Query	DataBase	Length	DTW 阈值	Jaccard 阈值
Plane	105	105	144	10.0	0.70
Symbols	90	995	398	23.0	0.65
CBF	90	900	128	41.0	0.40
TwoPatterns	90	4000	128	25.0	0.35

表 1 是对数据集的描述,分别为数据集的名称,查询序列的数目,数据库中序列的数目,时间序列的长度,以及设定的 DTW 的阈值和 Jaccard 阈值. Jaccard 阈值的设定是根据给定 DTW 阈值下相似性序列的查询数目所决定的,目的是为了使 STS3 方法查到的相似性序列的数目和 DTW 查到的数目尽可能的一致,在数目大体相同的情况下,和其他方法进行准确性和有效性的比较. 本实验所采用的数据集来源于 UCR[1] 公开数据集.

3.2 TQLD 算法分析

TQLD 算法第一步需要将原始的实数形式的时间序列转变为整型的 ID 序列,即将每条序列映射到不同的格子中,用每个格子的 ID 组成的序列来表示原始的序列,每个格子的大小等同. 因此 TQLD 算法的准确度会受到格子的长度和宽度这两个参数的影响,长和宽的取值越小,和原始序列的贴近程度会越强.

第二步是我们通过分析 DTW 距离度量的性质,对 Plane 数据集中相似的时间序列进行了 DTW 弯曲路径比对的计算,结果如图 3 所示. 我们充分利用 DTW 距离度量允许时间弯曲这一重要特性,将 ID 序列抻长,抻长为原始最长序列长度的 3 倍,在抻长的过程中,每个点可能重复一次或多次.

第三步是将汉明距离下的 LSH 函数应用到抻长的 ID 序列中,随机选取 ID 序列中的某几个位置进行哈希,若数据库中存在序列的哈希值和查询序列的哈希值相同,则将该序列纳入候选集合中. 在这个过程中我们通过将高维的时间序列降到低维中去处理,提前对数据库中的序列的哈希值建立前缀索引,进而加快了候选集合的筛选. 哈希位置的选取也会对查询结果的准确性产生影响.

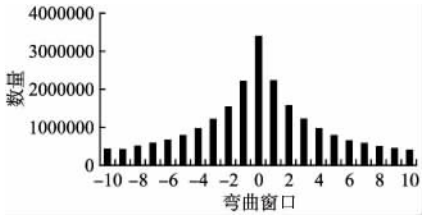

图 3 DTW 弯曲窗口分布

Fig. 3 Warping windows distribution

最后对候选集中的序列利用 DTW 验证,候选集合中无效的序列数量越少,TQLD 算法的查询效率越高. 因此当候选集中无效和有效的序列数目都较多时,尽管召回率比较理想,但相应的查询速度也会减慢. 通过在 Symbols 数据集上测试,当 DTW 阈值设为 23 时,暴力算法需要进行 89550 次 DTW 计算,而 TQLD 算法只需要进行 13195 次计算,并且召回率达到 0.8 左右.

3.3 算法性能比较

3.3.1 固定阈值的算法性能比较

我们针对表 1 中提供的数据集在三种不同的算法下进行了准确性和有效性的对比. 准确性采用找到给定 DTW 阈值下(DTW 阈值和 Jaccard 阈值在表 1 中给出) 相似的时间序列的召回率和准确率度量,有效性采用找到给定阈值下相似时间序列的平均查询时间代价度量.

表 2 平均时间代价

Table 2 Average time cost

DataSet	DTW	TQLD	STS3	Lb_DTW
Plane	0.003s	0.015s	0.002s	0.029s
Symbols	2.005s	0.331s	0.010s	0.721s
CBF	0.240s	0.120s	0.006s	0.247s
TwoPatterns	1.020s	0.333s	0.012s	1.061s

我们的实验分别在 4 种不同的数据集上进行测试,表 2 描述了在进行范围查询时不同方法的运行速度,图 4 和图 5 分别展示了在阈值不变的情况下,不同方法在不同数据集上

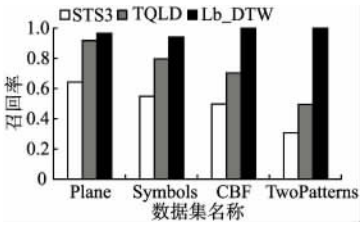

图 4 同一阈值的召回率对比

Fig. 4 Comparison of recall at the same threshold

的召回率和准确率. 通过分析可以得出 LB_DTW 虽然查全率和准确率都较高,但是在查询速度较 DTW 并没有明显的改善,因此可以判断对于某些数据集,下界的紧致性并不是很好,没有起到很好的剪枝作用,甚至执行时间有时会超过暴力

[1] www.cs.ucr.edu/~eamonn/time series data/.

算法. 而基于集合的算法即 STS3 虽然在很短的时间内就可以完成相似性序列的查询,但是通过实验结果得出,STS3 的查全率和查准率没有达到理想的指标,在某些情况下,不能够近似于DTW度量完成相似性序列的查询. 而我们的方法虽然召回率略低于 Lb_DTW 算法,但是查询速度明显快于 Lb_DTW,证明 TQLD 算法的过滤能力要优于 Lb_DTW. 因此我们的算法在保持较好的召回率的情况下,较现有算法有效地提高了 DTW 相似性查询速度.

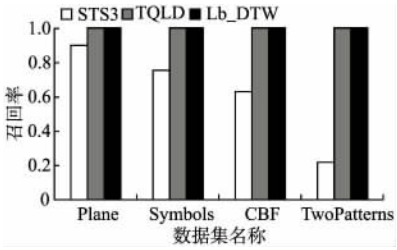

图 5　同一阈值的准确率对比

Fig. 5　Comparison of accuracy at the same threshold

3.3.2　不同阈值的算法性能比较

我们对 Symobols 数据集进行了不同 DTW 阈值下的范围查询,Jaccard 阈值会根据 DTW 阈值的设定相应的改变,实验运行时间比较如图 6 所示.

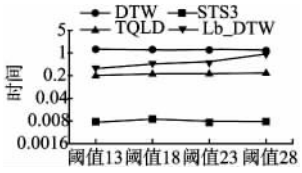

图 6　时间对比

Fig. 6　Time comparison

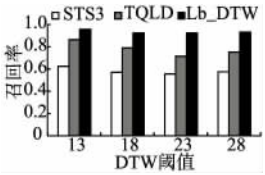

图 7　不同阈值的召回率对比

Fig. 7　Comparison of recall for different thresholds

通过实验结果可以得出,我们的方法的运行时间不会受到阈值大小的影响,由图7可以得出. 并且当给定DTW阈值较小的情况下,TQLD 的召回率会达到 0.88 左右,并且查询时间较 DTW 方法显著提升. 对比另外两种方法, 尽管基于下界的 Lb_DTW 方法召回率、准确率一直维持完美的指标,但是 Lb_DTW 的查询速度会受到阈值变化的影响. 由图 7 可以看出,随着阈值增大,Lb_DTW 的查询速度不断减慢. 而基于集合的 STS3 方法尽管运行时间不会受到阈值的影响,但是当 DTW 阈值较小时,STS3 方法的准确率会受到影响,并且不管阈值如何改变,STS3 方法的召回率和准确率都略逊色于 TQLD 方法.

4　结　论

时间序列的相似性查询是时间序列挖掘中的关键问题之一. 本文提出一种基于 LSH 的时间序列 DTW 相似性近似查询算法,较好地解决了 DTW 相似性查询速度慢的问题. 通过与现有的相似性查询算法的实验对比,实验结果显示了 TQLD 方法的准确性和有效性,表明该算法在保持较好的召回率的情况下,较现有算法有效地提高了 DTW 相似性查询速度. 如何进一步提高算法的准确性和降低算法的时间代价,以处理更大规模的时间序列数据的相似性查询是论文未来工作方向之一.

References:

[1] Kim S W, Park S, Chu W W. An index-based approach for similarity search supporting time warping in large sequence databases [C]//Proceedings 17th International Conference on Data Engineering(ICDE) , Berlin: IEEE, 2001: 607-614.

[2] Yi B K, Jagadish H V, Faloutsos C. Efficient retrieval of similar time sequences under time warping [C]//Proceedings of 14th International Conference on Data Engineering (ICDE) , Alando: IEEE, 1998: 201-208.

[3] Keogh E, Ratanamahatana C A. Exact indexing of dynamic time warping [J]. Knowledge and Information Systems(KAIS) , 2005, 7 (3) : 358-386.

[4] Berndt D J, Clifford J. Using dynamic time warping to find patterns in time series [C]//Knowledge Discovery and Data Mining Workshop, 1994, 94(3) : 359-370.

[5] Zhou M, Wong M H. Boundary-based lower-bound functions for dynamic time warping and their indexing [C]//IEEE International Conference on Data Engineering, IEEE ICDE, 2011, 181(19) : 4175-4196.

[6] Keogh E, Ratanamahatana C A. Exact indexing of dynamic time warping [J]. Knowledge and Information Systems(KAIS) , 2005, 7 (3) : 358-386.

[7] Rakthanmanon T, Campana B, Mueen A, et al. Addressing big data time series: mining trillions of time series subsequences under dynamic time warping [C]//ACM Transactions on Knowledge Discovery from Data(TKDD) , 2013, 7(3) : 1-31.

[8] Sakurai Y, Yoshikawa M, Faloutsos C. FTW: fast similarity search under the time warping distance [C]//Proceedings of the Twenty-fourth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, ACM, 2005, 24(5) : 326-337.

[9] Sakoe H, Chiba S. Dynamic programming algorithm optimization for spoken word recognition [J]. IEEE Transactions on Acoustics, Speech, and Signal Processing, 2003, 26(1) : 43-49.

[10] Itakura F. Minimum prediction residual principle applied to speech recognition [J]. IEEE Transactions on Acoustics Speech and Signal Processing, 1975, 23(1) : 67-72.

[11] Li Hai-lin, Guo Chong-hui. Review of feature representation and similarity measure in time series data mining [J]. Application Research of Computers, 2013, 30(5) : 1285-1291.

[12] Tang Chun-lei, Dong Jia-qi. Time-sequence query algorithm based on LSH [J]. Chinese Journal of Computers, 2012, 35(11) : 2228-2236.

[13] Andoni A, Indyk P. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions [C]//IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, 2006: 459-468.

[14] Li Hai-lin, Liang Ye, Wang Shao-chun. A survey of dynamic time bending in time series data mining [J]. Control and Decision, 2018, 33(8) : 4-12.

[15] Liu Gen-ping, Chen Ye-fang, Du Cheng-tou. An LSH-based time subsequence matching query algorithm [J]. Telecommunications Science, 2015, 31(8) : 63-71.

附中文参考文献:

[11] 李海林,郭崇慧. 时间序列数据挖掘中特征表示与相似性度量研究综述[J]. 计算机应用研究,2013,30(5) : 1285-1291.

[12] 汤春蕾,董家麒. 基于 LSH 的时间子序列查询算法[J]. 计算机学报,2012,35(11) : 2228-2236.

[14] 李海林,梁　叶,王少春. 时间序列数据挖掘中的动态时间弯曲研究综述[J]. 控制与决策,2018,33(8) : 4-12.

[15] 刘根平,陈叶芳,杜呈透. 一种基于 LSH 的时间子序列匹配查询算法[J]. 电信科学,2015,31(8) : 63-71.