一种基于滑动窗口的数据流相似性查询算法

王考杰[1,2] 郑雪峰[1] 宋一丁[2]

(北京科技大学信息工程学院 北京 100083)[1] (总后勤部后勤科学研究所 北京 100071)[2]

摘 要 相似性查询是一种非常重要的数据挖掘应用。由于数据流具有无限、高速等特性,传统的查询算法不能直接应用于数据流。提出了一种基于小波滑动窗口的多数据流相似性查询算法。算法首先将滑动窗口划分成若干等宽基本窗口,然后对每个基本窗口内的数据进行小波分解与系数约简,从而形成小波摘要窗口。执行相似性查询时,直接基于小波摘要进行计算,而无需数据重构。由于利用了小波分解的线性处理优点,算法具有较低的时间复杂度。最后,基于实际数据对算法进行了实验,实验结果证明了算法的有效性。

关键词 数据流,相似性查询,滑动窗口,小波分解

中图法分类号 TP311 **文献标识码** A

Algorithm Based on Sliding Window for Similarity Queries over Data Stream

WANG Kao-jie[1,2] ZHENG Xue-feng[1] Song Yi-ding[2]

(School of Information Engineering, University of Science and Technology Beijing, Beijing 100083, China)[1]

(Logistics Scientific Institute, General Logistics Department, Beijing 100071, China)[2]

Abstract Similarity queries are fundamental part of modern data mining application. But traditional query algorithms can not be applied on data stream, which is an unbounded sequence of data elements generated at a rapid rate. We proposed a novel approach for computing similarity over multi data streams based on wavelet sliding window model. The basic idea is to divide sliding window into equally-sized basic windows and represent the data elements of a basic window using wavelet coefficients, then form wavelet synopses window. As a result, queries toward data streams can be converted to queries toward such wavelet synopses. This algorithm takes advantage of the merit of wavelet decomposition for linear computing and achieves superior runtime performance. The extensive experiments verified the effectiveness of our algorithm.

Keywords Data stream, Similarity query, Sliding window, Wavelet decomposition

1 引言

数据流是近年出现的一种新兴的数据形式,基于数据流的查询处理、挖掘研究正成为研究热点。由于数据流的动态、无限、时变特性,不可能全部存储在传统数据库中,因此其查询处理一般是基于各种窗口技术进行的,滑动窗口[1]便是其中之一。

滑动窗口是一个有限的内存空间,存储数据流在某一个时间区间内的数据,并以数据的起始与截止时间戳为标识。窗口的截止时间戳为当前时刻,起始时间戳则随着新的数据流项的到来而不断移动,滑出时间区间的数据被丢弃,因此被称为滑动窗口。滑动窗口技术在数据流频繁模式挖掘[2-4]、数据抽样[5,6]、Top-k[7]查询等方面已得到较深入的研究。

为了节省内存空间,加快数据查询处理速度,可以只保留窗口内数据的摘要。采样、直方图、小波分解等是常用的数据摘要构造方法。其中,小波分解由于具有多尺度表达数据及线性的处理时间等优点,得到了广泛的研究[8-10]。

相似性查询作为一种数据挖掘典型应用,已有广泛的研究。相似性查询的关键是确定相似性度量标准,常见的是通过计算两个数据序列之间的欧式距离来确定其相似性。文献[11]提出一种从哈尔小波系数计算欧式距离的方法,文献[12]则将基于小波变换的相似性查询技术从哈尔小波扩展到其他的小波类型,并取得了良好的性能表现。文献[13]中,作者提出了一种X-Tree数据结构,进一步降低了基于小波域进行相似性查询的时间复杂度。文献[14]则针对轨迹数据流的局部连续性特征,提出了一种局部聚类的方法,用以计算数据流之间的相似性,进而检测出数据流中存在的异常。

本文研究了任意时间区间内多数据流相似性查询问题,并基于"分而治之"的思想,提出了一种基于滑动窗口的数据流相似性查询方法。

算法首先将数据流划分成等长的基本窗口,多个基本窗口构成滑动窗口。针对每一个基本窗口内的数据进行小波分解,并进行小波系数约简,形成小波摘要窗口。相似性查询则完全基于小波摘要滑动窗口进行。

到稿日期:2009-11-20 返修日期:2010-01-20 本文受国家科技支撑计划重点项目(2006BAG01A07)资助。

王考杰(1972—),男,博士生,主要研究方向为数据流分析、数据挖掘,E-mail:ewagkj@tom.com;郑雪峰(1951—),男,硕士,教授,主要研究方向为数据流分析、网络安全;宋一丁(1959—),男,硕士,高级工程师,主要研究方向为数据仓库、数据集成、软件质量。

2 背景及相关工作

为方便起见，我们把本文中用到的符号列在表 1 中。

表 1 本文中用到的符号

符号	描述
N	数据流长度
k	基本窗口大小
ω	滑动窗口大小
ε	相似性度量阈值
d_i	数据流项权重
c_i	小波系数
$\vec{x}$，$\vec{y}$	由基本窗口内数据构成的子序列
T_l，T_h	相似性查询时间起始点、结束点
$path_i$	小波错误树中数据流项 e_i 的祖先节点
δ_{ij}	小波系数因子，取值为±1

2.1 数据流

数据流可以看作是由无数数据项构成的无边界序列，而每一个数据项可看作是一个三元组：$e_i=\langle x_i,d_i,t_i\rangle$，此处 x_i 为数据流项的标识，d_i 为该数据流项的权重取值，t_i 为该数据项的时间戳。

例如，在一个由 IP 网络包构成的数据流中，x_i 为数据包的目的地址；d_i 为数据包字节长度；t_i 为数据包发送的时间戳。

2.2 滑动窗口

滑动窗口是数据流处理模型之一。在这种处理模型中，并不在内存中保留所有的数据项，而只保留给定时间区间内到达的最近的 N 个数据项。

例如，在图 1 中，给定窗口大小 ω，只有最近的 ω 个数据项被保留下来，用于数据分析与挖掘。当新的数据项到达时，滑动窗口中最旧的数据项因过期而自动被丢弃。

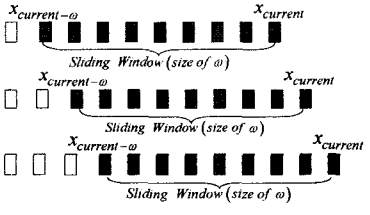

图 1 滑动窗口模型

2.3 小波分解

小波分解是一种数学工具，用于层次化分解函数，在信号及图像处理领域已有多年的成功应用。

通过实例可以很好地理解小波分解的过程。如给定一个大小为 $N=8$ 的数据序列 $\vec{x}=(2,2,0,2,3,5,4,4)$，以哈尔小波为例，其分解过程如下：

首先计算相邻成对数据的均值，从而得到 4 个数值[2,1,4,4]作为矢量 $\vec{x}$ 的“低分辨率”表达。由于这种表达缺少细节信息，因此为了保留完整的 $\vec{x}$ 的信息，计算相邻成对数据的差值，从而得到 4 个数值[0,−1,−1,0]，用以表达 $\vec{x}$ 的细节信息。从上述 8 个数值中可以将 $\vec{x}$ 完全重构出来。

基于 4 个均值，重复上述计算过程，最后得到一个总的均值[11/4]，以及每一次重复运算所得到的系数[−5/4,1/2,0,0,−1,−1,0]。

误差树是一种便于理解小波系数的树结构，图 2 表示的是 $\vec{x}$ 的小波错误树表达。树结构的每一个内部节点 $c_i(i=0,\cdots,7)$ 表示一个小波系数，每一个叶节点 $d_i(i=0,\cdots,7)$ 表示一个数据项，$l_i(i=0,\cdots,3)$ 表示小波系数所在的层次。

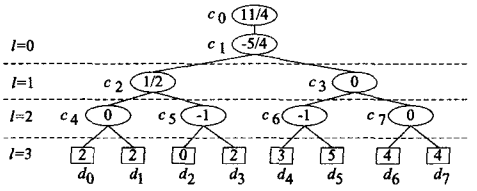

图 2 小波误差树

2.4 相似性

相似性搜索的关键问题是确定数据序列之间的距离度量方法。

为简单起见，本文采用欧几里德距离度量方法。

给定一个阈值 ε，两个长度均为 n 的时间序列 $\vec{x}$ 和 $\vec{y}$，如果满足如下条件，则认为 $\vec{x}$ 和 $\vec{y}$ 是相似的。

$$D(\vec{x},\vec{y})=\left(\sum_{i=0}^{n-1}(y_i-x_i)^2\right)^{\frac{1}{2}}\leqslant\varepsilon \tag{1}$$

假定 T_l，T_h 分别为起始时间点和截止时间点，则两个数据流在时间范围 $T_l:T_h$ 内的相似性定义为

$$D_{(T_l:T_h)}^{(\vec{x},\vec{y})}=\left(\sum_{i=T_l}^{T_h}(y_i-x_i)^2\right)^{\frac{1}{2}}\leqslant\varepsilon \tag{2}$$

这样，我们的问题就转换成计算两个数据流在任意指定的时间范围内的欧式距离。

3 基于滑动窗口的数据流近似行搜索

本文提出一种基于滑动窗口的欧式距离计算方法。算法的本质是基于“分而治之”的思想，将数据流划分成若干等宽基本窗口，每个基本窗口内数据进行小波分解与系数约简，从而形成小波摘要窗口。在线相似性查询应答则直接基于这些小波摘要窗口进行。

算法主要由两部分构成，即滑动小波摘要窗口构造算法，以及基于小波摘要窗口的相似性查询应答算法。

3.1 滑动小波摘要窗口构造算法

给定一个时间点 t 和一个时间区间 k，则所有在[t,$t+k-1$]时间区间内到达的数据项形成一个子序列，将其存储在一个缓存区，并称之为基本窗口。对于窗口内数据，进行哈尔小波分解和小波系数约简，从而构造出小波数据摘要。经过分解后的基本窗口又可称为小波摘要窗口。对于这些窗口，只存储它的起始、截至时间戳以及该窗口内数据的小波摘要。若干小波摘要窗口链接在一起，形成滑动窗口。当滑动窗口已满时，对应的起始时间戳最久的小波摘要窗口将被删除，新的小波摘要窗口加入。

算法详细描述如下：

算法 1 WaveletSlidingWindow(k,ω)

输入：k，基本窗口大小；ω，滑动窗口大小

输出：小波摘要窗口集合

1)初始化 Buffer 数据结构，用于缓存当前基本窗口未满时的数据流项

2)初始化 WaveletWindow 数据结构，用于存储滑动窗口内的所有小波摘要窗口

3)While(有新的数据流项)

4) 初始化该数据流项 $e_i=\langle x_i,w_i,t_i\rangle$

5) 计数器countOfItem 加 1
//记录数据流项数量
6) If CountOfWaveletWindow<ω
// 滑动窗口未满
7) If countOfItem<k
//基本窗口未满
8) 将 $e_i=\langle x_i,w_i,t_i\rangle$缓存至 Buffer
9) Else
//当前基本窗口 B_i 已满
10) Call WaveletWindow(B_i)
//对 B_i 内数据进行哈尔小波变换，形成小波摘要窗口 $W_{\vec{d}}(B_i)$
11) 小波摘要窗口 $W_{\vec{d}}(B_i)$缓存至 WaveletWindow
12) 计数器 CountOfWaveletWindow 加 1
13) 清空 Buffer
14) Else
//滑动窗口已满
15) 丢弃起始时间戳最久的小波摘要窗口
16) 计数器 CountOfWaveletWindow 减 1
17) 更新 WaveletWindow

上述算法中，基本窗口内小波摘要构造算法如下：

算法 2 WaveletWindow(B_i)

输入：$B_i=[x_i,\cdots,x_{i+(k-1)}]$，最近的 k 个数据流项

输出：小波摘要 $W_{\vec{d}}$

1)初始化数据结构 coefficientTree，用于存储小波系数

2)初始化数据结构 dataIndex，用于存储窗口内数据流项索引，即该数据项的时间戳

3)初始化变量 *startTimeStamp* 和 *endTimeStamp*，用于标识窗口 B_i 的起始时间戳和截止时间戳

4) 执行哈尔小波变化，并以链表存储小波系数
//小波系数以其层次为序存储在链表中

5) For $l=1,\cdots,$系数链表长度

6) 将系数 c_l插入 coefficientTree

7) 将 c_0 存储为 average

8) 执行小波系数约简

9) For $l=0,\cdots,k$

10) 将数据流项索引存储在 dataIndex 中

11) 存储 *startTimeStamp* 和 *endTimeStamp*

3.2 基于小波摘要窗口的相似性查询算法

针对一个时间范围，执行相似性查询(T_l，T_h)。T_l 表示起始时间点，T_h 表示结束时间点。首先要查询索引，确定 T_l，T_h 所对应的基本窗口位置，并由此得到所有相关基本窗口。

这些窗口可划分成 3 类：

第一类窗口，即第一个窗口 W_{first}，窗口的结束时间点包含在 T_l，T_h 所指定的范围内，起始时间戳等于 T_l。

第二类窗口，即中间窗口 W_{middle}，窗口的起止时间戳完全包含在由 T_l，T_h 所指定的范围内。

第三类窗口，即最后一个窗口 W_{last}，窗口的起始时间点包含在 T_l，T_h 所指定的范围内，结束时间戳则等于 T_h。

针对每一个基本窗口，计算两个数据流之间的欧式距离。最后将这些欧式距离累加，便可得到总的指定时间范围内的欧式距离，如式(3)所示。

$$D(W_{l:h})=D(W_{first})+D(W_m)+D(W_{last}) \tag{3}$$

此处，

$$D(W_{first})=D(T_l:startTimeStamp\ \text{of}\ B_{first})$$

$$D(W_m)=\sum_{i=0}^{k}D(W_{middle_i})$$

$$D(W_{last})=D(startTimeStamp\ \text{of}\ B_{last}:T_h)$$

算法详细描述如下：

算法 3 WaveSliWin(T_l，T_h，ε)

输入：时间范围的起始时间戳、截止时间戳，以及相似性度量阈值

输出：两个数据流在指定时间范围内是否相似

1)初始化欧式距离变量

2)查询所有位于 T_l，T_h 时间范围内的小波摘要窗口

3)For 每一个窗口 W

4) If $T_l\geqslant W_{start\text{-}timestamp}$并且 $T_l\leqslant W_{end\text{-}timestamp}$

5) If $T_h\leqslant W_{end\text{-}timestamp}$//只有一个窗口

6) Call calOnlyWin(T_l，T_h)
//计算唯一窗口 $D(W_{first})$

7) Else

8) Call calFirstWin(T_l，$W_{end\text{-}timestamp}$)
//计算第一个窗口 $D(W_{first})$

9) Start=TRUE;End=FALSE

10) Else if $T_l<W_{start\text{-}timestamp}$
并且 $T_h>W_{end\text{-}timestamp}$

11) Call calMiddleWin($W_{start\text{-}timestamp}$，$W_{end\text{-}timestamp}$)
//计算中间窗口 $D(W_{m_i})$

12) Else if $T_l>W_{start\text{-}timestamp}$
并且 $T_h\leqslant W_{end\text{-}timestamp}$

13) Call calLastWin($W_{start\text{-}timestamp}$，$T_h$)
//计算最后一个窗口 $D(W_{last})$

14)$D(W_{l:h})=D(W_{first})+D(W_{m_i})+D(W_{last})$
//根据式(3)计算

15)If $D(W_{l:h})>\varepsilon$ 返回 TRUE

16)Else 返回 TRUE

对于计算第一类、第三类窗口欧式距离的函数，即 calOnlyWin，calFirstWin，calLastWin，采用文献[9]中的办法。

假设 $d_i^{\vec{x}}$，$d_i^{\vec{y}}$ 分别是数据流$\vec{x}$，$\vec{y}$基本窗口内的数据项，则依据文献[9]，有

$$d_i^{\vec{x}}=\sum_{c_j^{\vec{x}}\in path_i}\delta_{ij}\cdot c_j^{\vec{x}};\ d_i^{\vec{y}}=\sum_{c_j^{\vec{y}}\in path_i}\delta_{ij}\cdot c_j^{\vec{y}}$$

则

$$\begin{aligned}d_i^{\vec{y}}-d_i^{\vec{x}}&=\sum_{c_j^{\vec{y}}\in path_i}\delta_{ij}\cdot c_j^{\vec{y}}-\sum_{c_j^{\vec{x}}\in path_i}\delta_{ij}\cdot c_j^{\vec{x}}\\&=\sum_{\substack{c_j^{\vec{y}}\in path_i\\ c_j^{\vec{x}}\in path_i}}\delta_{ij}\cdot(c_j^{\vec{y}}-c_j^{\vec{x}})\end{aligned}$$

因此，在基本窗口的一个任意指定的时间范围(t_1：t_2)内，假定对应时间 t 的数据流项的索引为 i，则

$$\begin{aligned}D_{(t_1:t_2)}^{(\vec{x},\vec{y})}&=\Big(\sum_{t=t_1}^{t_2}(d_i^{\vec{y}}-d_i^{\vec{x}})^2\Big)^{\frac{1}{2}}\\&=\Big(\sum_{t=t_1}^{t_2}\Big(\sum_{\substack{c_j^{\vec{y}}\in path_i\\ c_j^{\vec{x}}\in path_i}}\delta_{ij}\cdot(c_j^{\vec{y}}-c_j^{\vec{x}})\Big)^2\Big)^{\frac{1}{2}}\end{aligned} \tag{4}$$

对于计算第二类窗口距离的函数 calMiddleWin，采用文献[15]的方法：

$$D(\vec{x},\vec{y})=\Big[\sum_{\substack{n_{(l,p)}^{\vec{x}}\in C_B\\ \text{or}\\ n_{(l,p)}^{\vec{y}}\in C_B}}D_{(l,p)}^{(\vec{x},\vec{y})}*2^l\Big]^{1/2} \tag{5}$$

此处，$\vec{x},\vec{y}$表示一个基本窗口内的数据流子序列。C_B 表示小波摘要，l,p 则表示基本窗口内小波系数节点所在的层次以及在该层的位置。

$$D_{(l,p)}^{(\vec{x},\vec{y})}=\begin{cases}[c_{(l,p)}^{(\vec{x})}-c_{(l,p)}^{a(\vec{y})}]^2, & \text{if } c_{(l,p)}^{(\vec{x})}\in C_B \& c_{(l,p)}^{(\vec{y})}\in C_B\\ [c_{(l,p)}^{(\vec{x})}]^2, & \text{if } c_{(l,p)}^{(\vec{x})}\in C_B \& c_{(l,p)}^{(\vec{y})}\notin C_B\\ [c_{(l,p)}^{(\vec{y})}]^2, & \text{if } c_{(l,p)}^{(\vec{x})}\notin C_B \& c_{(l,p)}^{(\vec{y})}\in C_B\end{cases}$$

从图 2 可以看出，小波系数存储可以认为是一个完全二叉树。为了便于索引，我们采用数组存储小波系数，如图 3 所示。

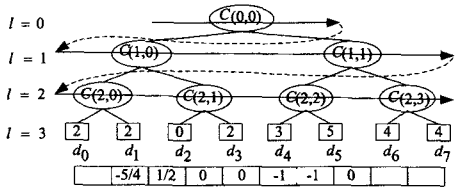

图 3　小波系数存储结构

这样，小波系数在数组中以 l,p 递增次序存储。只要依据数组索引递增次序针对数组元素逐个进行计算，就可以很方便地计算出 $D_{(l,p)}^{(\vec{x},\vec{y})}$。

4　实验分析

我们基于一组实际数据对算法的执行时间效率进行了测试。为了便于比较，我们实现了文献[13]中的算法，并以 X-Tree 标记，而本文提出的算法以 WaveSliWin 标识。算法均以 Java1.5 实现，测试环境是酷睿双核 1.6G，内存 1G。

测试数据选用两组海洋表面温度测量值，每组数据包含 65536 项数据(1980 年至 1998 年)，取自近赤道太平洋上布置的浮标，用于研究厄尔尼诺现象。经过处理，每项数据包含(序号，温度，时间戳)3 个数据元素。

图 4、图 5、图 6 显示了实验结果。

图 4 显示的是在固定基本窗口大小的前提下，查询算法执行时间随滑动窗口大小变化的情况。

图 5 显示的是在固定滑动窗口大小的前提下，查询算法执行时间随基本窗口大小变化的情况。

从图中可以看出，无论哪种情况，查询执行时间除了有个别的变动外，基本都能保持一个相对稳定的时间值，说明算法有比较好的适用性。同时，相较于 X-Tree 算法，WaveSliWin 表现更稳定。

图 6 显示的是在固定滑动窗口、基本窗口大小的前提下查询算法执行时间随时间区间变化的情况。从图中可以看出，算法的查询执行时间随查询时间范围均呈现次线性变化，表明算法具有良好的可扩展性，能够适应数据流，尤其是大规模数据流的应用情形。相较于 WaveSliWin 算法，X-Tree 曲线斜率要更大一些，表明 WaveSliWin 的可扩展性要好于 X-Tree。

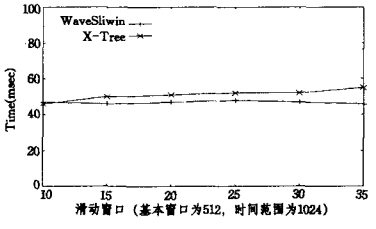

图 4　查询时间随滑动窗口变化情况

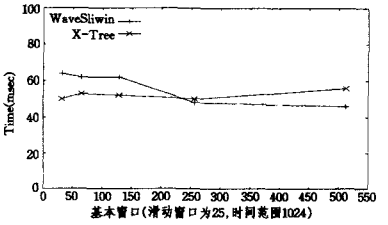

图 5　查询时间随基本窗口变化情况

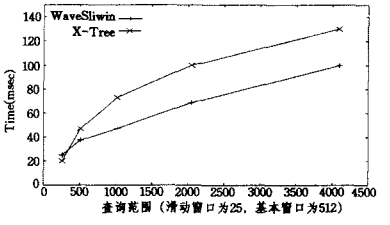

图 6　查询时间随查询范围变化情况

结束语　本文提出一种新的基于滑动窗口的数据流相似性查询算法。算法采用“分而治之”的策略，首先将指定时间范围内的数据流划分成若干等宽的基本窗口，并针对基本窗口内数据进行小波变换，构造小波摘要。然后基于小波摘要滑动窗口，进行任意时间范围内的数据流相似性查询。实验结果表明，算法具有良好的适应性与可扩展性。

参 考 文 献

[1] Mayur D, Aristides G, Piotr I, et al. Maintaining stream statistics over sliding windows[C]// Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms. San Francisco, California, Society for Industrial and Applied Mathematics, 2002: 635-644

[2] Lukasz G, David D, Erik D D, et al. Identifying frequent items in sliding windows over on-line packet streams[C]// Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement. Miami Beach, FL, USA, ACM, 2003: 173-178

[3] Hua-Fu L, Suh-Yin L. Mining frequent itemsets over data streams using efficient window sliding techniques[J]. Expert Syst. Appl., 2009, 36(2): 1466-1477

[4] 敖富江，颜跃进，黄健. 数据流频繁模式挖掘算法设计[J]. 计算机科学，2008，35(3)：1-5

[5] Rainer G, Wolfgang L. Sampling time-based sliding windows in bounded space[C]// Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data. Vancouver, Canada, ACM, 2008: 379-392

[6] Vladimir B, Rafail O, Carlo Z. Optimal sampling from sliding windows[C]// Proceedings of the twenty-eighth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. Providence, Rhode Island, USA, ACM, 2009: 147-156

[7] Cheqing J, Ke Y, Lei C, et al. Sliding-window top-k queries on uncertain streams[J]. Proc. VLDB Endow., 2008, 1(1): 301-312

[8] Kaushik C, Minos G, Rajeev R, et al. Approximate query processing using wavelets[J]. The VLDB Journal, 2001, 10(2/3): 199-223

[9] Panagiotis K, Nikos M. One-pass wavelet synopses for maximum-error metrics[C]// Proceedings of the 31st International Conference on Very Large Data Bases. Trondheim, Norway, VLDB Endowment, 2005: 421-432

(下转第 201 页)

结束语 PSO算法的搜索机制是随机的，通过迭代计算来找到近似最优值或是问题的近似最优解决方案，无论适应度函数是不是连续的或是可导的，都可以应用PSO来解决。因此，PSO的应用范围就更广泛，更有可行性。本文提出的IDPSO是改进的离散型PSO算法，在IDPSO算法中融入了遗传算法的部分操作算子，包括变异算子和交叉算子。普通的PSO算法一般以一定的概率按照式(1)、式(2)更新粒子的位置和速度，可以比较迅速地得到次优适应度函数值，但是有时候会因为忽略的粒子有较好的优化信息，从而陷入局部最优。IDPSO通过粒子群优化算法本身的更新规则代替遗传算法中的选择规则，在每次的迭代过程中都以一定的概率对每个粒子进行交叉、变异操作，尽可能地保留种群中各个粒子的优化信息，尽可能避免迅速陷入局部最优。

本文采用的IDPSO算法通过多组实例的测试得到了比较不错的布线方案；通过与遗传算法在最优值进化曲线和平均适应度函数值的比较，表明IDPSO具有更快的收敛速度、简单的操作，更具可行性和有效性。在下一步的工作中，将寻找一种能够表示更为广泛的矩形Steiner树的编码，用以表示非连通布线模型中的矩形Steiner树。

参考文献

[1] Hashimoto A, et al. Wire Routing by Optimizing Channal Assignment Within Large Appertures[C]// Proc. of 8th IEEE/ACM Design Automation Workshop. 1971

[2] 邓爱姣，李强，张嘉为. 改进的Prim启发式算法在VLSI布线中的应用[J]. 沈阳工业大学学报，2006，28(5)：557-559

[3] 杨昌玲，严晓浪. 基于树形编码的MRST混合遗传算法及其并行处理[J]. 微电子学，1999，29(2)：89-95

[4] Wey Chin-Long. Efficient Rectilinear Steiner Tree Construction with Rectangular Obstacles[C]// Proc. 5th WSEAS International Conference on Circuits System Selectronics Control & Signal Processign. Dallas, USA, 2006：204-208

[5] Rita M H, Bryant A J. A Spanning-Tree-based Genetic Algorithm for Some Instances of the Rectilinear Steiner Problem with Obstacles[C]// Proceedings of the 2003 ACM Symposium on Applied Computing. 2003：725-729

[6] Eberhart R C, Kennedy J. A New Optimizer Using Particles Swarm Theory[C]// Proc. 6th International Symposium on Micro Machine and Human Science. Nagoya, Japan, 1995：39-43

[7] Clerc M, Kennedy J. The Particle Swarm-explosion, Stability, and Convergence in a Multidimensional Complex Space[J]. IEEE Transactions on Evolutionary Computation, 2002, 6(1)：58-73

[8] Hanan M. On steiner's problem with rectilinear distance[J]. SIAM Journal of Applied Mathematics, 1996, 14(2)：255-265

[9] Kennedy J, Eberhart R C. A Discrete Binary Version of the Particle Swarm Optimization Algorithm[C]// Proc. of the IEEE International Conference on Systems Man and Cybernetics. Orlando, USA, 1997, V：4104-4109

[10] Clerc M. Discrete Particle Swarm Optimzation illustrated by the Traveling Salesman Problem[EB/OL]. http://www.maurice-clerc.net, 2000-02-29

[11] 郭文忠，陈国龙，夏添. 异构机群下数据流自适应分配策略[J]. 计算机辅助设计与图形学学报，2009，21(8)：1175-1181

[12] 郭文忠，陈国龙. 一种求解多目标最小生成树问题的有效离散粒子群优化算法[J]. 模式识别与人工智能，2009，22(4)：597-604

[13] 王晓东，吴英杰. Prufer编解码的最优算法[J]. 小型微型计算机系统，2008，29(4)：687-690

[14] 张会生，翁史烈，张小兵，等. 基于内弹道改进型零维模型的装药优化仿真[J]. 弹道学报，2000，12(3)：32-36

[15] 洪先龙，严晓浪，乔长阁. 超大规模集成电路布图理论与算法[M]. 北京：科学出版社，1998

[16] Shi Y H, Eberhart R C. A Modified Particle Swarm Optimizer[C]// IEEE International Conference of Evolutionary Computation. Piscataway, NJ：IEEE, 1998：69-73

(上接第164页)

[10] Kang Donghyun, Han Saeyoung, Yoo Seohee, et al. Prediction-based Dynamic Thread Pool Scheme for Efficient Resource Usage[C]// Proceedings of the 2008 IEEE 8th International Conference on Computer and Information Technology Workshops. 2008：159-164

[11] Goetz B. Thread Pools Help Achieve Optimum Resource Utilization[EB/OL]. http://www.900.ibm.com/developerWorks/en/java/j-jtp0730/index_eng.shtml, 2002-07

[12] 邵鸣年，张昕. 一种分布式系统中线程池的设计与实现[J]. 计算机工程与设计，2005，26(1)：7-11

[13] Mercury Interactive. LoadRunner controller user's guide windows version8.0 [EB/OL]. http://www.mercury.com/us/products/performance-center/loadrunner/

(上接第172页)

[10] Dimitris S, Antonios D, Timos S. Hierarchically compressed wavelet synopses[J]. The VLDB Journal, 2009, 18(1)：203-231

[11] Franky Kin-Pong C, Ada Wai-chee F, Clement Y. Haar Wavelets for Efficient Similarity Search of Time-series：With and Without Time Warping[J]. IEEE Trans. on Knowl. and Data Eng., 2003, 15(3)：686-705

[12] Popivanov I, Miller R J. Similarity Search Over Time-Series Data Using Wavelets[C]// Proceedings of the 18th International Conference on Data Engineering. IEEE Computer Society, 2002：212-216

[13] Liabotis I, Theodoulidis B, Saraaee M. Improving Similarity Search in Time Series Using Wavelets[J]. International Journal of Data Warehousing and Mining, 2006, 2(2)

[14] Yingyi B, Lei C, Ada Wai-Chee F, et al. Efficient anomaly monitoring over moving object trajectory streams[C]// Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Paris, France, ACM, 2009：159-168

[15] Hao-Ping H, Ming-Syan C. Efficient range-constrained similarity search on wavelet synopses over multiple streams[C]// Proceedings of the 15th ACM International Conference on Information and Knowledge Management. Arlington, Virginia, USA, ACM, 2006：327-336