一种滑动窗口下数据流 Disjoint 查询的增量处理算法

王少鹏[1)] 闻英友[1)] 赵 宏[1)] 孟颖辉[2)]

[1)](东北大学计算机科学与工程学院 沈阳 110819)

[2)](郑州轻工业学院计算机与通信工程学院 郑州 450002)

摘 要 对于滑动窗口下不具有全局约束机制的数据流 Disjoint 查询精确处理问题进行了研究,在现有 FSM 算法基础上提出了一种具有增量计算特征的查询处理算法 DQPIC. 该算法使用 FSM 算法处理第一个窗口中的数据流成员,同时保留了该窗口上的查询结果和窗口所对应 ***STWM*** 的最后一个列向量,除此之外还需要保留窗口 ***STWM*** 中所有列向量第 *curbound*.*highest* 个成员 DTW 路径的起始位置、距离值以及该成员在 ***STWM*** 中对应列向量的 *dmin* 值和候选查询结果这些信息. 从第二个窗口开始,继续使用 FSM 算法处理窗口成员,同时也保留和第一个窗口一样的信息. 在这个过程中,当处理相邻窗口中相同数据流成员时,通过比较该成员在前后两个窗口中分别对应的保留信息是否相同,可以确定算法有无继续处理剩余相同数据流成员的必要,能够在前一个窗口查询结果基础上增量地获得当前窗口查询结果. 基于公用数据样本 SST 与 Maskedchirp 的仿真实验验证了该算法的有效性. 提出的算法与现有其他算法执行结果相同,在空间开销增加 1.12～3.27 倍情况下,可以实现时间效率 2.5～25 倍的提高,对于与大窗口下的 Disjoint 查询相关应用场景,具有更好的时间效果.

关键词 数据流;动态时间扭曲;Disjoint 查询;滑动窗口;增量计算

中图法分类号 TP311 **DOI 号** 10.11897/SP.J.1016.2017.02381

An Incremental Processing Algorithm about Disjoint Query Based on Sliding Window over Data Stream

WANG Shao-Peng[1)] WEN Ying-You[1)] ZHAO Hong[1)] MENG Ying-Hui[2)]

[1)](*School of Computer Science and Engineering*, *Northeastern University*, *Shenyang* 110819)

[2)](*School of Computer and Communication Engineering*, *Zhengzhou University of Light Industry*, *Zhengzhou* 450002)

Abstract With the fast development of internet of things technology, more and more applications are producing data stream continuously. In order to obtain the knowledge in this kind of data, data stream mining technology is studied widely, and is great significant. All those obtained knowledge can be taken as our guideline for action. The similarity query over data stream is a very important one in all researches related to the data stream mining technology, and has been essential in many applications, like precision agriculture, and environmental monitoring. In all researches which is about the similarity query over data stream, Disjoint query has played a very important role, and can get all subsequences which are similar to the given query sequence in data stream. All these obtained subsequences are local minimal. However, the current researches did not focus on the Disjoint query based on the sliding window over data stream. In order to resolve the problem of precise Disjoint query processing based on the sliding window without global warping constraints over data stream, an effective query processing algorithm DQPIC which has

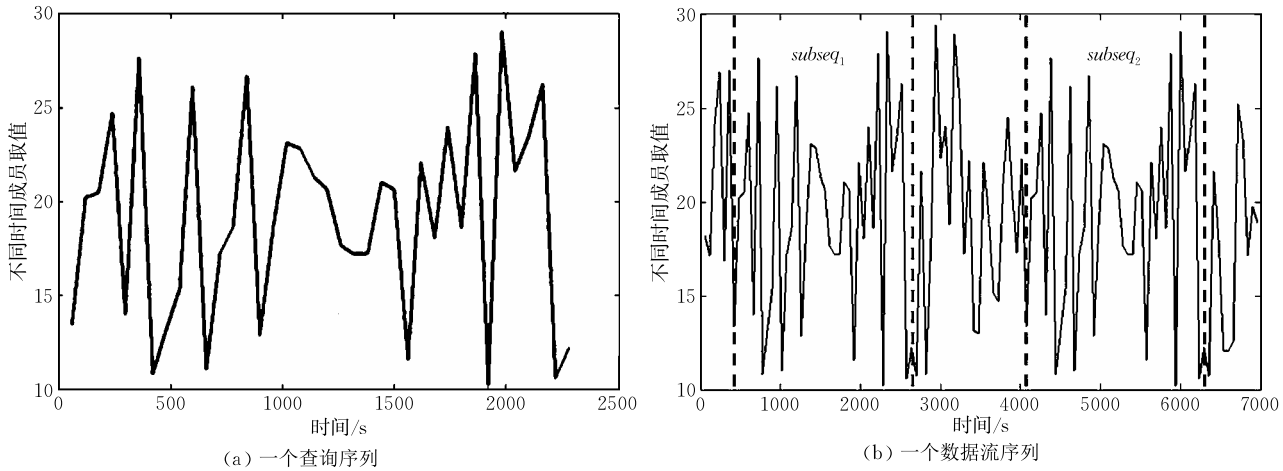
收稿日期:2016-05-18;在线出版日期:2016-11-25. 本课题得到国家自然科学基金(60903159,61173153,61402096,61501405)、中央高校基本科研业务费资助项目(N110818001,N100218001,N130504007,N120104001)、沈阳市科技计划项目(1091176-1-00)、国家"八六三"高技术研究发展计划基金(2015AA016005)资助. **王少鹏**,男,1984 年生,博士研究生,主要研究方向为数据流、事件流、数据挖掘. E-mail: wangshaopeng1984@163.com. **闻英友**,男,1974 年生,博士,副教授,主要研究方向为下一代网络、无线传感器网络. **赵 宏**,男,1954 年生,博士,教授,博士生导师,主要研究领域为网络管理、多媒体. **孟颖辉**,男,1984 年生,博士,讲师,主要研究方向为无线传感网络,传感器定位.

characteristic of incremental calculation is proposed based on the current FSM algorithm. This algorithm handles the data stream elements in the first sliding window with the FSM algorithm, and preserves query results and the last column vector of the ***STWM*** matrix about the window over data stream. In addition, some other kinds of information about the every column vector of ***STWM*** matrix are also needed to be reserved, such as the start position and distance value of the DTW road about the element whose index is *curbound.highest* in these column vectors of ***STWM*** matrix, the *dmin* value, and the candidate query result. Beginning from the second sliding window, the FSM algorithm is still be taken for the window elements processing by the DQPIC algorithm, and the same information as those retained in the first window are also still needed to be reserved. However, in this procedure, when all data stream elements which located in two both adjacent windows over data stream are processed, the DQPIC algorithm can determine whether the remaining elements of them still need to be processed by the FSM algorithm based on comparison of those information which are preserved respectively about the column vector corresponding to the same processing data stream element of two matrixes about these two sliding windows. By doing this, the algorithm can get query results of current window based on those ones of the adjacent last window over data stream, and has the characteristic of incremental calculation. The experimental results based on common data sample SST and Maskedchirp demonstrate the proposed algorithm is effective. It has the same results as the current other algorithms, and can improve the time efficiency by 2.5—25 times at the cost of increment of space cost by 1.12—3.27 times. In the application scenarios which are related to the Disjoint query based on the larger sliding window over data stream, the DQPIC algorithm has better performance in the time.

Keywords data stream; dynamic time warping; Disjoint query; sliding window; incremental calculation

1 引 言

Disjoint 查询作为基于 DTW 的数据流子序列相似性查询中非常重要的一类，可以帮助用户从数据流中获取有用信息，在很多应用领域都有着非常重要的作用[1-10]. 图 1 给出了 Disjoint 查询的过程描述，其中(a)是一个查询序列，(b)中实线条描述了一个数据流序列. Disjoint 查询可以求得数据流序列中与查询序列相似的所有不相交子列，而且每个子列都是与该子列相交且满足查询条件的子序列中的局部最优序列，如图中的两个子序列 $subseq_1$ 与 $subseq_2$ 就是当前数据流序列中 Disjoint 查询结果.



(a) 一个查询序列　　(b) 一个数据流序列

图 1 Disjoint 查询

在当前有关 Disjoint 查询的研究中，较少关注滑动窗口下 Disjoint 查询的处理机制. 而这类查询却有着较多的应用需求，比如在精细农业中，相关领域专家通过对大量样本的分析，可以得到适宜害虫孵化的温度取值走势，将其定义为查询序列. 为了了解一定时间范围内农作物可能发生虫害的情况，可以将查询序列与该时间范围的窗口序做 Disjoint 查询；又如在工业生产领域，通过对封装到生产过程的传感器所捕获的一定时间范围内流式数据与相应的异常数据模式做相似性查询，可以判断该时间范围内工业生产过程中的异常发生情况，进而采取相应措施减少因异常发生而导致的人力物力损失；再比如在环境监测领域，通过对藻类污染数据的分析可以得到适宜于该藻类生长的温度或湿度序列，以它们为查询序列与一定时间范内负责监控的相应传感设备所检测到的流式数据做 Disjoint 查询，该查询结果对于我们了解水域中藻类污染发生的可能性有重要的参考价值. 图 2 中给出了滑动窗口下 Disjoint 查询的具体说明.

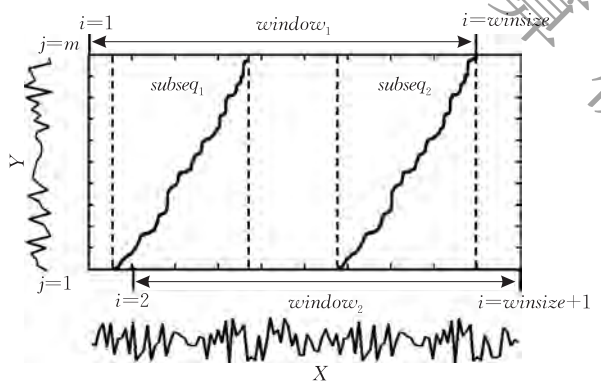


图 2　滑动窗口下 Disjoint 查询

其中 X 是一个数据流序列，而 Y 是一个长度为 m 的查询序列，$window_1$ 与 $window_2$ 是两个相邻的滑动窗口，$subseq_1$ 与 $subseq_2$ 是 $window_1$ 上的两个 Disjoint 查询结果，滑动窗口下的 Disjoint 查询就是要求得 $window_1$ 之后类似于 $window_2$ 一样的每个窗口上的 Disjoint 查询结果.

当前专门针对滑动窗口下数据流 Disjoint 查询的研究很少，文献[9]对该问题进行了研究，提出了一种基于边界路径技术的查询处理算法，该算法在产生和不产生查询结果的窗口上分别给出了边界路径、伪边界路径以及与之相关的创建元素的概念，通过比较前一个窗口上的创建元素在当前窗口中所对应的扭曲路径与前一个窗口上的边界路径或伪边界路径是否相同，来确定基本算法对于相邻 2 个窗口中相同数据流成员的处理量，能够增量地得到当前窗口上的 Disjoint 查询结果. 但该算法忽略了在使用基本算法处理相邻两个窗口中位于创建元素之前相同数据流成员时存在的冗余运算.

本文同样对滑动窗口下的 Disjoint 查询处理问题进行了研究，文中将这种查询命名为 DQ-SW 查询(Disjoint Query based on Sliding Window). 对于 naive 算法处理 DQ-SW 查询的过程进行了分析和研究，提出了具有增量计算特征的 DQ-SW 查询处理算法(DQ-SW query Processing algorithm with Incremental Calculation，简称 DQPIC). 该算法能够在前一个窗口查询结果的基础上增量获得当前窗口的查询结果，与现有算法相比，很好地减少了查询处理的时间开销. 特别与文献[9]中的 FSDQ 算法相比，在处理相邻两个窗口上相同数据流成员的时候，DQPIC 算法无需每次都对创建元素前的所有数据流成员进行处理，即可得到窗口上的查询结果，具有更好的时间效率. 文章实验部分证明了该算法是有效的. 本文的第 2 节回顾相关工作；第 3 节介绍相关预备知识，包括 Disjoint 查询，Spring 算法，FSM 算法以及文中要解决问题的形式化定义；第 4 节针对 naive 算法在处理 DQ-SW 查询时不具有增量计算的缺点，提出 DQPIC 算法，并着力介绍该算法的理论依据；第 5 节给出实验结果与分析；第 6 节总结全文，并说明下一步的研究方向.

2　相关工作

越来越多研究证明 DTW 是最好的距离测度函数[11]. 当前与基于 DTW 的数据流子序列相似性搜索相关的研究可分为如下几类：(1) 将数据流上的子序列长度固定，然后得到这些子列中与查询序列最相似的序列[12-13]，或者是判断这些子序列是否与查询序列相似[14-18]；(2) Disjoint 查询，即用于得到数据流中与查询序列相似的所有不相交子列，而且每个子列都是与该子列相交且满足查询条件的子序列中的局部最小序列[1-10]. 和第 1 类研究相比，Disjoint 查询并不需要分割原始数据流序列，且得到的查询结果不会相交且是局部最优. 自从 Sakurai 在 2007 年的 ICDE 会议上给出了 Disjoint 查询的定义，并提出了相应处理算法 Spring 之后，很多研究者都在此基础上做了大量的研究工作.

当前有关这类查询的研究按照查询结果的状态，可分为精确 Disjoint 查询和近似 Disjoint 查询 2

类.在精确 Disjoint 查询方面,按照是否考虑全局约束机制又为了 2 类,一类不考虑全局约束机制,如 Sakurai 等人在文献[1]中提出了 Spring 算法,该算法是第一个对该查询进行处理的算法;Zou 等人[2]改进了 Spring 算法,提出了 FSM 算法,该算法能有效地减少 Spring 算法的运算量;Kashyap 等人[3]对 Spring 算法进行了改进,提出了 eHaarSubseq 算法.该算法中使用了基于 haar 小波的 eHaar 技术,可以提前将数据流中与查询序列不相似的子列从中删除,然后对数据流中剩余部分成员与查询序列执行 Spring 算法,与原来 Spring 算法相比,时间效率更好;Gong 等人[10]同样对于 Spring 算法进行了改进,提出了 NSPRING 算法,该算法使得 Spring 算法支持所处理数据的正交化操作. 另一类是考虑全局约束机制,如 Niennattrakul 等人在文献[4-5]中对 Spring 算法中的全局约束机制与归一化处理问题分别作了研究,提出了 ASM 与 MSM 算法,可以更合理地处理数据流的子序列相似性搜索问题;Wang 等人在文献[6]中结合 FPGA 技术提出了一种 Disjoint 查询处理方法,该方法分为算法软设计与算法的 FPGA 实现 2 个部分.在算法软设计中,作者提出了混合下界函数 LB_pDTW 来估计 Disjoint 查询中 DTW 距离的下边界.该算法会先使用下界函数去掉数据流中不符合查询条件的子序列,然后在此基础上再使用 Spring 算法进行处理,不过该算法的效果极大地依赖于 FPGA 技术的实现;近似 Disjoint 查询通常会以一定的精度为代价,换得查询执行在时空方面的效率,比如文献[7-8],其中文献[7]中的 EBSM 算法是 Athitsos 等人在 Spring 算法基础上提出的,与 Spring 算法相比,DTW 距离的计算次数要少很多,但也付出了查询结果不够精确的代价;Zhou 等人在文献[8]中,提出了一种数据流维数约减技术 NPLA,然后在此基础上给出了 Disjoint 查询的处理算法.相比于使用了 PAA 维数约减技术的 Spring 算法,该算法具有时间优势.

还有些研究虽然涉及了 Disjoint 概念,但其研究的本质与我们研究的内容并不相同,如文献[19-20]关注的是数据流外包查询认证.其中涉及的 Disjoint 查询与我们研究的数据流 Disjoint 查询并不是一类查询方式.前者主要为了减少 client 多个查询的认证代价,将它们中重合的部分进行解构,使其成为不相交的子查询(Disjoint 查询),通过对这些子查询结果的 union 操作可以得到最初 client 查询的认证结果,这样可以减少操作代价.总体来看它们中所谓的 Disjoint 涉及两个方面,一个是不同数据流间的 Disjoint(即不相交),另一个是多个查询所针对的对象之间尽量解构为不相交的情况.而我们研究的是数据挖掘中的相似性查询,其中 Disjoint 查询指的是数据流中的查询结果间是不相交的,而且每一个结果都是与之相交的子列中的局部最小值,所以这两类研究是不同的.文献[21]关注的是数据流管理系统(DSMS)中的查询语言方面存在的问题,定义了一些新的数据流操作符和相应的查询语言使用结构.其中有些概念与我们的研究在命名上一样,但实质并不相同.如它们对于模式的定义是基于数据流成员使用正则表达式来描述的.主要关注的是数据流中不同成员间的关系,也就是说它们的模式匹配指的是到来的数据流成员满足正则表达式.这与我们的相似性查询定义不同;另外文献中的 Disjoint union 操作符是对于多个数据流的操作,其实质与关系数据库中 union 操作符相类似,这与我们 Disjoint 查询的定义不同.

值得注意的是在前面与我们所关注的 Disjoint 查询相关的研究中,它们所得到的查询结果都是从第一个处理的数据流成员开始,直到要处理的数据流成员被处理完为止,整个数据流上的 Disjoint 查询结果.这个处理过程中随着数据流成员的不断到来,数据流上 Disjoint 查询结果也在持续产生,其间并不涉及数据流成员的删除操作,相关算法不断地处理新到来的数据流成员,以得到新的 Disjoint 查询结果.但正如第一部分所述,有时候我们可能会对滑动窗口下 Disjoint 查询的结果产生兴趣,而这样的 Disjoint 查询对数据流上每一个滑动窗口都需要重新执行,在窗口滑动的过程中会涉及新数据的进入与旧数据的删除操作.王少鹏等人在文献[9]中对滑动窗口下数据流的 Disjoint 查询处理问题进行了研究,提出了一种 FSDQ 算法,但该算法忽略了在使用基本算法处理相邻两个窗口中位于创建元素之前相同数据流成员时存在的冗余运算.

总体来看,当前有关滑动窗口下数据流 Disjoint 查询处理问题的研究并不多,仅有的方法也存在冗余运算.得到一种更有效的处理方法是本文研究目的.需要说明的是在我们研究中,暂不考虑查询全局约束机制和对数据的正交化处理,主要对滑动窗口下数据流精确 Disjoint 查询处理问题进行研究.

3 预备知识

3.1 Spring 算法

Spring 算法是 Sakurai 提出的第 1 个用来处理

数据流 Disjoint 查询问题的算法. 该算法有 2 个特点：(1)提出了 star padding 思想；(2)使用了子序列时间扭曲矩阵 ***STWM***. 其中前者为查询序列加“*”前缀，在求得查询序列与数据流子序列的 DTW 距离取值时，使用新查询序列代替旧查询序列. 计算过程中规定“*”成员与任何数据流成员距离为 0，这样原查询序列中第 1 个成员在 ***STWM*** 中对应行向量成员取值的计算只会涉及该查询序列成员与对应数据流成员间的计算，所以可将数据流中每个成员作为子序列起点情况考虑进去，最终实现只用 1 个时间扭曲矩阵获得数据流中 Disjoint 查询结果的目的. ***STWM*** 是为了配合 star padding 思想而设计的数据结构，主要目的在于获得结果子序列起点位置. ***STWM*** 中(t,i)成员含有 2 个成员：$d(t,i)$和 $sp(t,i)$，它们的意义是 X 上 $sp(t,i)$与 t 之间的数据流序列与 Y 中长为 i 的查询序列前缀之间的距离值为 $d(t,i)$，该值也是 X 上长为 t 的数据流前缀与 Y 中长为 i 的查询序列前缀之间执行相似性匹配时的最小 DTW 距离值. 这两个成员取值的计算会按如下公式进行.

$$D(X(t_s:t_e),Y)=d(t_e,m)=\min(d(t,m)) \quad (1)$$

$$d(t,i)=dist(x_t-y_i)+d_{\text{best}} \quad (2)$$

$$d_{\text{best}}=\min\begin{cases}d(t,i-1)\\ d(t-1,i)\\ d(t-1,i-1\end{cases} \quad (3)$$

$$d(t,0)=0,d(0,i)=\infty,\ t=1,\cdots,n,i=1,\cdots,m \quad (4)$$

$$sp(t,i)=\begin{cases}sp(t-1,i), & d(t-1,i)=d_{\text{best}}\\ sp(t-1,i-1), & d(t-1,i-1)=d_{\text{best}}\\ sp(t,i-1), & d(t,i-1)=d_{\text{best}}\end{cases} \quad (5)$$

Spring 算法会在 ***STWM*** 每个列向量构建过程中得到 Disjoint 查询结果，有关该算法详尽描述可见文献[1]. 另外本文中所有 DTW 距离都使用 L1 形式[2]距离函数，当然亦可扩展为其他距离函数形式.

3.2　Disjoint 查询

定义 1[1]. Disjoint 查询(Disjoint query). 假定数据流 X 的长度为 n，Y 是一个长度为 m 的查询序列，$epsilon$ 为相似阈值，$X(t_{\text{start}}:t_{\text{end}})$是数据流 X 中起止时间分别为 t_{start} 到 t_{end} 的子序列，Disjoint 查询是要得到 X 的子序列集 S_{range}，序列集中的每个成员要满足如下的条件：

(1) 如果 $X(t_{\text{start}}:t_{\text{end}})\in S_{\text{range}}$，则有 $D(X(t_{\text{start}}:t_{\text{end}}),Y)\leq epsilon$；

(2) 对于 X 上任意与 $X(t_{\text{start}}:t_{\text{end}})$相交的子序列 $X(jt_{\text{start}}:jt_{\text{end}})$，如果 $D(X(jt_{\text{start}}:jt_{\text{end}}),Y)\leq epsilon$，一定有 $D(X(t_{\text{start}}:t_{\text{end}}),Y)\leq D(X(jt_{\text{start}}:jt_{\text{end}}),Y)$；

(3) 如果 $X(t_{\text{start}}:t_{\text{end}})$，$X(t_{\text{start1}}:t_{\text{end1}})\in S_{\text{range}}$，则这两个序列要满足 $X(t_{\text{start}}:t_{\text{end}})\cap X(t_{\text{start1}}:t_{\text{end1}})=\varnothing$.

其中 D 函数用于计算两个序列的 DTW 测度函数值，总得来说 Disjoint 查询就是要得到 X 上与 Y 相似但不相交的所有子列. 该查询所使用的数据流模型为界标窗口模型.

3.3　FSM 算法

FSM 算法是一个由邹鹏等基于 Spring 算法提出的同样用于处理数据流环境下 Disjoint 查询的算法. 同 Spring 算法不同的是，该算法采用了一种边界技术，最大程度地减少 ***STWM*** 构建过程中不必要的成员计算，以及相关的比较操作. 具体来说，该算法分别用 *curbound.lowest* 与 *curbound.highest* 保留 ***STWM*** 中正在被处理列向量的第一个和最后一个 d 值小于等于 *epsilon* 的成员位置；同时也分别用 *pribound.lowest* 与 *pribound.highest* 保留相邻前一个列向量中第一个和最后一个 d 值小于等于 *epsilon* 的成员位置. 在求得 ***STWM*** 中正在被处理列向量所有成员 d 值的过程中，如果正在处理的成员 d 值大于 *epsilon*，假设此时前一列向量的 *pribound.highest* 为 0，则无需计算当前列向量所有剩余成员取值；否则当这个成员的位置小于 *pribound.lowest* 时，则当前向量上该成员位置与 *pribound.lowest* 之间的所有成员 d 值都无需计算，直接处理索引为 *pribound.lowest* 的成员即可；当该成员的位置大于 *pribound.highest*+1 时，则无需计算当前列向量所有剩余成员的 d 值，处理当前向量到此结束；当该成员位置大于等于 *pribound.lowest*，且小于等于 *pribound.highest*+1 时，需要按 Spring 算法计算成员的 d 值. 这个过程也要记录下当前列向量的 *curbound.highest* 与 *curbound.lowest*，当处理完该向量后，需要将 *curbound* 取值赋予 *pribound*，接着开始处理 ***STWM*** 中下一个数据流成员所对应的列向量. 有关 FSM 算法的详细情况可见文献[2].

3.4　问题定义

定义 2.　数据流. 数据流 X 被看作连续到达的离散、无限的元组序列，表示为 $x_1,x_2,\cdots,x_{t-1},x_t,\cdots$，其中 t 为时间，本文中设数据流速度恒定.

定义 3.　基本窗口 EW 与滑动窗口 SW. EW 是定长时间内连续到达的数据流序列，即有 $EW=\{x_i,x_{i+1},\cdots,x_{i+m}\}$，其中 m 是基本窗口大小，表示

为$|EW|=m$. SW 对应 SW_{size} 个连续相邻基本窗口序列，即有 $SW=\{EW_j, EW_{j+1}, \cdots, EW_{j+SW_{\text{size}}}\}$，其中 SW_{size} 为滑动窗口大小，表示为 $|SW|=SW_{\text{size}}$，且有 $\forall k, l \in [j, j+SW_{\text{size}}]$，如果 $k \neq l$，有 $EW_k \cap EW_l = \varnothing$. 当窗口已满，$SW$ 会随数据流成员到来，以 EW 为单位不断滑动.

定义 4. DQ-SW 查询(Disjoint Query based on Sliding Window). $\forall Y, epsilon, SW_{\text{size}}, |EW|$，令 $seq_{SW}=\{SW_1, SW_2, \cdots, SW_n\}$ 是我们关注的数据流上 n 个连续的滑动窗口序列，其中 $n \in N$，SW_n 是 seq_{SW} 中第 n 个窗口. 对于每个 SW_n，DQ-SW 查询就是求得其上数据流成员与 Y 的 Disjoint 查询结果，这也是本文着力要解决的问题.

4 具有增量计算特征的 DQ-SW 查询处理算法

表 1 说明了文中所用到的变量和表达式所代表的含义.

表 1 变量和表达式含义说明

变量和表达式	含义
$candidate$	FSM 算法中的参数，是 ***STWM*** 中每个列向量构造完成后的候选查询结果；
$dmin$	FSM 算法的参数，是 $candidate$ 的距离；
$Para_{\text{cur}}$	DQPIC 算法中对应于当前窗口的参数；
$Preserve\_FSM\_P$	基于相邻前一窗口最后一个成员对应列向量及其参数对当前窗口最后一个基本窗口 EW 中的成员执行 FSM 算法，并保留 DQPIC 算法中所需要的相关参数；
$tempara$	用于临时存放滑动窗口 SW 中一个数据流成员对应的参数，这些参数都是 DQPIC 算法中用到的；
$Para_{\text{cur}.v}$	$Para_{\text{cur}}$ 中与当前窗口成员 v 对应的参数；
$Para_{\text{cur}}(index)$	$Para_{\text{cur}}$ 中与当前窗口索引为 $index+1$ 的基本窗口 EW 中成员对应的参数；
$hbound$	DQPIC 算法中与当前窗口 SW 每个成员对应的参数，用于存放 ***STWM*** 中与该成员对应列向量相关的 $curbound.highest$；
V_{hbound}	与滑动窗口 SW 每个成员相关的参数，用于存放 ***STWM*** 每个列向量中索引为 $hbound$ 的成员 d 值；
$winelenum$	从 DQPIC 算法执行开始进入到滑动窗口 SW 中的 EW 总数；
$Para_{\text{cur}}.lastvec$	$Para_{\text{cur}}$ 中的参数，包括与当前窗口相关的 ***STWM*** 中最后一个列向量以及与之相关的参数；
$result_{\text{cur}}$	当前窗口的查询结果；
$getresult$	DQPIC 算法在处理相邻窗口相同数据流成员的过程中，会寻找一个在两个窗口中相关参数都相同的成员，在找到该成员后，$getresult$ 会从 $result_{\text{cur}}$ 中得到其他相同数据流成员在被 DQPIC 算法处理过程中产生的查询结果；

4.1 naive 算法

由 Spring 算法可知，解决 DQ-SW 查询最直接方法即在每个数据流窗口上执行 Spring 算法. 但由文献[2]可知，这种算法处理 Disjoint 查询时存在冗余运算. 邹鹏等在该文献中所提出的 FSM 算法，不仅与 Spring 算法执行效果相同，还大大减少了这种冗余运算，所以我们也能通过在每个数据流窗口上执行 FSM 算法解决这类查询处理问题，而且这种方法比第 1 种方法冗余计算量少，本文把这种方法称为 naive 算法. 由 FSM 算法可知，每个窗口上 naive 算法查询结果是在该窗口上 $m \times (|EW| \times SW_{\text{size}})$ 阶矩阵 ***STWM*** 构建过程中产生的，而该矩阵的构成则是循环使用了两个大小为 m 的列向量.

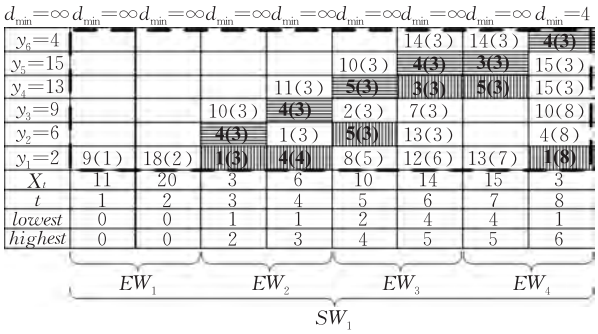
4.2 具有增量计算特征的 DQ-SW 查询处理算法

naive 算法需要在每个窗口上重新执行 FSM 算法以得到该窗口上 DQ-SW 查询结果，不同窗口上查询结果的获得过程彼此独立，所以算法不具有增量计算的特点，而这个问题的存在使得该算法执行效率并不是很高. 为了解决该问题，我们在这部分提出了具有增量计算特征的 DQ-SW 查询处理算法 DQPIC. 具体来说，首先给出该算法理论依据的直观描述；其次，对于算法理论依据的正确性进行了严格证明；接着基于该理论依据给出了 DQPIC 算法的详细执行步骤；最后分析了算法的时间、空间复杂度.

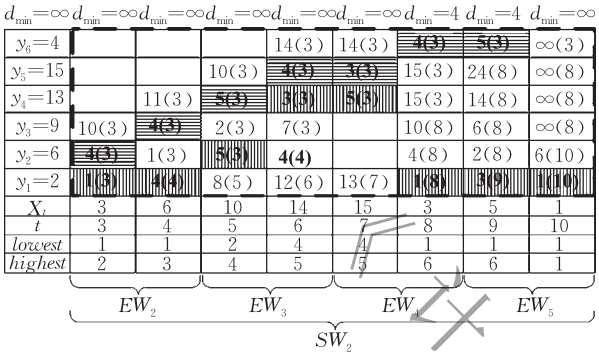
4.2.1 DQPIC 算法理论依据的直观描述

为便于对比分析，我们使用了文献[1-2]中的查询实例. 假设数据流 $X=\{11,20,3,6,10,14,15,3\}$，查询序列 $Y=\{2,6,9,13,15,4\}$，$epsilon=15$，$SW_{\text{size}}=4$，$|EW|=2$，图 3 中(a)和(b)虚线区域分别给出相邻 2 个窗口在执行完 FSM 算法后的 ***STWM***，我们分别将其命名为 $\boldsymbol{STWM}_1$ 与 $\boldsymbol{STWM}_2$；2 个 ***STWM*** 中列向量内横线背景和竖线背景成员的数字分别标出了该向量中 $highest$ 与 $lowest$ 成员，而它们在向量中索引分别是该向量中 $highest$ 与 $lowest$ 变量取值.

在图 3 的 ***STWM*** 中，每个成员有两部分取值，括号内外分别为起始位置 sp 与成员 d 值，每个列向量上面是按照 FSM 算法处理完该向量后得到的 $dmin$ 值. 通过对比 $\boldsymbol{STWM}_1$ 与 $\boldsymbol{STWM}_2$ 中成员取值，可以发现：EW_2，EW_3，EW_4 中数据流成员在两个矩阵中对应列向量成员取值完全相同. 而由 FSM 算法可知，***STWM*** 中每个列向量成员取值只与相邻前



(a) ***STWM***$_1$



(b) ***STWM***$_2$

图 3　***STWM***$_1$ 和 ***STWM***$_2$ 的示例

一个列向量及与该向量相关参数取值有关，所以根据 x_3 在两个矩阵中对应列向量成员取值以及相关参数取值相同，就可推得 EW_2 中其他数据流成员以及 EW_3，EW_4 中所有数据流成员对应列向量取值，以及在这些列向量成员取值计算过程中产生的查询结果是相同的. 显然基于该思想处理 DQ-SW 查询比 naive 算法的时间效率好. 但这个思想应用前提是记录当前窗口 ***STWM*** 中每个列向量及其相关参数，还有与该向量相关的查询结果取值情况，这样在求相邻下个窗口上查询结果时，处理两个窗口相同数据流成员可用“边处理，边比较”方式进行.

具体来说在得到每个相同数据流成员在下一个窗口中对应的列向量后，将该向量与同一数据流成员在当前窗口中对应的列向量取值进行比较，如果两个列向量取值及对应参数取值完全相同，则不必处理其他相同数据流成员，因为这些成员在两个 ***STWM*** 中对应列向量取值，以及在这个过程中产生的查询结果完全相同，我们只需将这些成员在当前窗口处理过程中产生的结果直接拿来即可. 当然这个时候若要得到相邻下一个窗口上的完整查询结果，还需要在当前窗口最后一个列向量及相关参数取值的基础上对下一个窗口新加入的 EW 中成员执行 FSM 算法.

但这种方法缺点明显，在保留信息中会涉及到每个列向量成员取值，空间开销大. 若能以较少保留信息表征要比较的这部分内容，则可以极大改善该算法空间开销. 我们在进一步观察后有个猜想，即能否以 ***STWM*** 每个列向量对应的 $highest$ 以及与该值对应的成员取值来描述整个列向量成员取值信息，也即能否以相邻两个窗口任意相同数据流成员在这两个窗口中相关参数以及该数据流成员在这两个 ***STWM*** 中对应列向量的 $highest$ 和与 $highest$ 对应的列向量成员取值相同，推出这两个窗口其他相同数据流成员在使用 FSM 算法处理过程中构建的列向量成员取值以及产生的查询结果也相同. 结合图 3 就是能否以 x_3 在 ***STWM***$_1$ 与 ***STWM***$_2$ 中处理完成后的相关参数相同以及 x_3 在这两个矩阵中对应列向量 $highest$ 值相同和列向量中第 $highest$ 个成员取值相同，推出 $x_4 \sim x_8$ 在两个矩阵中对应列向量成员取值以及在这些列向量构建过程中产生的查询结果也相同. 若可以，则设想成立. 该猜想是 DQPIC 算法的理论基础. 下面分析了它的正确性.

4.2.2　关于 DQPIC 算法理论依据的分析证明

为便于证明猜想，这部分先给出相关知识前提，接着基于分析证明了 DQPIC 算法的理论依据.

4.2.2.1　知识前提

本部分内容主要包括 DTW 路径定义，以及以引理形式给出的相关性质；同时也包括基于此分析出的 Spring 算法中 ***STWM*** 成员取值与 Disjoint 查询结果间的关系，这个关系我们以定理形式呈现.

令 ***SM***$_i$ 是滑动窗口 SW_i 上的成员在执行完 Spring 算法后得到的 ***STWM***，***SV***$_{i,j}$ 是 SM_i 中第 j 个列向量. 由 Spring 算法可知，该算法在确定矩阵中任一成员 sp 与 d 值时，需要得到该成员的 d_{best} 值，同时找到该 d_{best} 所在的矩阵成员. 进一步由 Spring 算法可知，在确定矩阵成员 sp 与 d 值的过程中，每个成员的 d_{best} 所在成员都是唯一的. 且每个成员的 sp 值与该成员 d_{best} 所在矩阵成员的 sp 值是相同的（可参考文献[1]与 3.1 节）.

定义 5.　DTW 路径. 设 $I(i,j)$ 是 ***SV***$_{i,j}$ 对应的数据流成员索引，则 ***SV***$_{i,j}$ 与该 SW_i 中的数据流成员 $x_{I(i,j)}$ 相对应. 对于 ***SV***$_{i,j}$ 中任意成员 ***SV***$_{i,j}(t)$，（由下而上的顺序），如果该成员对应的 sp 值 ***SV***$_{i,j}(t)$. $sp=p$，则 x_p 对应的向量 ***SV***$_{i,u}$（其中 $u=j-(I(i,j)-p)$）中的第 1 个成员 ***SV***$_{i,u}(1)$ 与 ***SV***$_{i,j}(t)$ 之间一定存

在一条路径(即扭曲路径),路径上任意成员是该路径上相邻下个成员的 d_{best} 所在矩阵成员,我们把该路径称为 $\boldsymbol{SV}_{i,j}(t)$ 在滑动窗口 SW_i 中的 DTW 路径,记为 $R_{i,j}(t)$. 而 p 被称为该路径的起始位置,记为 $R_{i,j}(t).sp$.

同样为了便于对比分析,我们使用了文献[1-2]中的查询实例. $X=\{11,20,3,6,10,14,15,3\}$,查询序列 $Y=\{2,6,9,13,15,4\}$,$epsilon=15$,$SW_{size}=3$,$|EW|=2$,我们以 X 在第 1 个滑动窗口上的实例来说明 DTW 路径的概念,详情见图 4.

$y_6=4$	29(1)	38(1)	24(1)	22(3)	16(3)	14(3)
$y_5=15$	22(1)	23(1)	32(3)	20(3)	10(3)	4(3)
$y_4=13$	18(1)	23(1)	20(3)	11(3)	5(3)	3(3)
$y_3=9$	16(1)	25(1)	10(3)	4(3)	2(3)	7(3)
$y_2=6$	14(1)	23(1)	4(3)	1(3)	5(3)	13(3)
$y_1=2$	9(1)	18(2)	1(3)	4(4)	8(5)	12(6)
X_i	11	20	3	6	10	14
i	1	2	3	4	5	6

EW_1 EW_2 EW_3

SW_1

图 4 DTW 路径示例

图 4 中虚线区域是 SW_1 上执行完 Spring 算法后的 ***STWM***,矩阵中每个成员包括两部分取值,括号内为起始位置 sp,括号外为成员 d 值. $\boldsymbol{SV}_{1,3}(1)\sim\boldsymbol{SV}_{1,6}(6)$ 之间阴影部分成员构成了 $\boldsymbol{SV}_{1,6}(6)$ 在滑动窗口 SW_1 中的 DTW 路径 $R_{1,6}(6)$,该路径起始位置 $R_{1,6}(6).sp$ 为 3.

引理 1. 对于 $\boldsymbol{SV}_{i,j}$ 中的任一成员 $\boldsymbol{SV}_{i,j}(t)$,$1\leq t\leq m$,该成员的起始位置 sp 为 p,(1) 在向量 $\boldsymbol{SV}_{i,u}$(其中 $u=j-(I(i,j)-p)$)中的第 1 个成员 $\boldsymbol{SV}_{i,u}(1)$ 与 $\boldsymbol{SV}_{i,j}(t)$ 之间一定存在唯一的 DTW 路径,路径上每个成员起始位置都为 p;(2) 对于 $\boldsymbol{SV}_{i,j}$ 中索引小于 t 的成员来说,其 sp 值都大于等于 p;而对于 $\boldsymbol{SV}_{i,j}$ 中索引大于 t 的成员来说,其 sp 值小于等于 p;(3) $\boldsymbol{SM}_i$ 中位于路径 $R_{i,j}(t)$ 下成员的 sp 值都大于等于 p.

证明. (1) 由定义 5 及 Spring 算法,结论(1)显然成立;

(2) 可以用反证法证明. $\forall\boldsymbol{SV}_{i,j}$ 中索引小于等于 t(这里 $\boldsymbol{SV}_{i,j}$ 中成员的索引是由下而上的顺序)的成员 $\boldsymbol{SV}_{i,j}(t_1)$,假设 $\boldsymbol{SV}_{i,j}(t_1).sp=p_1<p$,则由结论(1)可知,在 $\boldsymbol{SV}_{i.l}(1)$ 与 $\boldsymbol{SV}_{i,j}(t_1)$ 间一定存在一条 DTW 路径 $R_{i,j}(t_1)$,路径上所有成员起始位置都为 p_1,其中 $l=j-(I(i,j)-p_1)$. 而由(1)可知,在 $\boldsymbol{SV}_{i.u}(1)$ 与 $\boldsymbol{SV}_{i,j}(t)$ 之间存在一条 DTW 路径 $R_{i,j}(t)$,路径上每个成员起始位置都为 p,其中 $u=j-(I(i,j)-p)$. 因为 $p_1<p$,所以有 $l<u$,也即对于 $R_{i,j}(t_1)$ 与 $R_{i,j}(t)$,有 $t_1\leq t$,且 $R_{i,j}(t_1).sp<R_{i,j}(t).sp$ 成立,所以 $R_{i,j}(t_1)$ 与 $R_{i,j}(t)$ 一定相交,由结论 1 可推得,位于交点位置的成员会有 2 个起始位置,p_1 与 p,再由 Spring 算法可知这是不可能的,矛盾,所以假设不成立,即有 $\boldsymbol{SV}_{i,j}(t_1).sp\geq\boldsymbol{SV}_{i,j}(t).sp=p$ 成立. $\boldsymbol{SV}_{i,j}$ 中索引大于 t 成员的证明与此类似,不再赘述.

(3) 由定义 5 及 Spring 算法可知,$R_{i,j}(t)$ 在 $\boldsymbol{SV}_{i.u}$ 与 $\boldsymbol{SV}_{i,j}$ 之间(包括 $\boldsymbol{SV}_{i.u}(1)$ 与 $\boldsymbol{SV}_{i,j}(t)$)的每个列向量上都有成员,而 ***STWM*** 中位于这些成员位置下的矩阵成员构成了 ***STWM*** 中位于 $R_{i,j}(t)$ 路径下的成员. 针对每个这类成员使用(2)中的反证法,容易推得结论成立. 证毕.

我们仍以前面说明 DTW 路径时用到的数据流事例来说明引理 1 的内容,具体如图 5 所示.

$y_6=4$	29(1)	38(1)	24(1)	22(3)	16(3)	14(3)
$y_5=15$	22(1)	23(1)	32(3)	20(3)	10(3)	4(3)
$y_4=13$	18(1)	23(1)	20(3)	11(3)	5(3)	3(3)
$y_3=9$	16(1)	25(1)	10(3)	4(3)	2(3)	7(3)
$y_2=6$	14(1)	23(1)	4(3)	1(3)	5(3)	13(3)
$y_1=2$	9(1)	18(2)	1(3)	4(4)	8(5)	12(6)
X_i	11	20	3	6	10	14
i	1	2	3	4	5	6

EW_1 EW_2 EW_3

SW_1

图 5 引理 1 的说明

引理 1 的结论 1 告诉我们,***STWM*** 列向量中任一成员(也即 ***STWM*** 中任一成员)与数据流中索引为该成员起始位置的成员间一定存在唯一的 DTW 路径,如图 5 中 $\boldsymbol{SV}_{1,2}(4)\sim\boldsymbol{SV}_{1,1}(1)$ 之间具有横线背景的成员所构成的路径即为 $\boldsymbol{SV}_{1,2}(4)$ 在滑动窗口 SW_1 中的 DTW 路径 $R_{1,2}(4)$,而 $\boldsymbol{SV}_{1,3}(1)\sim\boldsymbol{SV}_{1,6}(6)$ 之间具有阴影背景的成员所构成的路径即为 $\boldsymbol{SV}_{1,6}(6)$ 在滑动窗口 SW_1 中的 DTW 路径 $R_{1,6}(6)$;

由引理 1 的结论 2 可知,对于矩阵中任一成员,考虑该成员所在列向量中索引值小于该成员索引值的成员,它们 sp 值都大于等于该成员的 sp 值;而对于该成员所在列向量中索引值大于等于该成员索引值的成员,它们的 sp 值都小于等于该成员的 sp 值,比如对于图 5 的 $\boldsymbol{SV}_{1,2}$ 中索引大于 $\boldsymbol{SV}_{1,2}(4)$ 在 $\boldsymbol{SV}_{1,2}$ 中索引值的成员(图中具有竖线背景的成员),每个

成员的 sp 值都小于等于 $\boldsymbol{SV}_{1,2}(4)$ 的 sp 值 1；而对于 $\boldsymbol{SV}_{1,2}$ 中索引小于 $\boldsymbol{SV}_{1,2}(4)$ 在 $\boldsymbol{SV}_{1,2}$ 中索引值的成员（$\boldsymbol{SV}_{1,2}$ 中点虚线区域），每个成员的 sp 值都大于等于 $\boldsymbol{SV}_{1,2}(4)$ 的 sp 值 1；又比如图 5 的 $\boldsymbol{SV}_{1,6}$ 中索引小于 $\boldsymbol{SV}_{1,6}(5)$ 在 $\boldsymbol{SV}_{1,6}$ 中索引值的成员（$\boldsymbol{SV}_{1,6}$ 中点虚线区域），每个成员 sp 值都大于等于 $\boldsymbol{SV}_{1,6}(5)$ 的 sp 值 3.

引理 1 的结论 3 告诉我们，位于矩阵任一成员 DTW 路径下的所有矩阵成员，它们的 sp 值都大于等于该成员的 sp 值. 比如位于图中 $R_{1,2}(4)$ 下的所有矩阵成员，它们的 sp 值都大于等于 $\boldsymbol{SV}_{1,2}(4).sp$ 值，也即 $R_{1,2}(4).sp$ 值 1；而位于图中 $R_{1,6}(6)$ 下的所有矩阵成员 sp 值都大于等于 $\boldsymbol{SV}_{1,6}(6).sp$ 值 3.

定理 1.　Spring 算法在构建 $\boldsymbol{SM}_i$ 的过程中获得该窗口上的 Disjoint 查询结果，但这个查询结果与 $\boldsymbol{SM}_i$ 中 d 值大于 $epsilon$ 的成员取值并没有关系，这个没有关系是指省略掉 $\boldsymbol{SM}_i$ 中 d 值大于 $epsilon$ 的成员取值计算，并不会对查询结果产生影响.

证明.　对于 $\boldsymbol{SM}_i$ 中的列向量 $\boldsymbol{SV}_{i,j}$，如果该向量最后一个成员的 sp 值为 p（即 $\boldsymbol{SV}_{i,j}(m).sp=p$），由引理 1 可知在 x_p 对应列向量中的第一个成员 $\boldsymbol{SV}_{i,p-i+1}(1)$ 与 $\boldsymbol{SV}_{i,j}(m)$ 之间一定存在一条 DTW 路径 $R_{i,j}(m)$，该路径在数据流中对应 $x_p\sim x_{I(i,j)}$，同时路径上所有成员的 sp 值都为 p. 如果 $\boldsymbol{SV}_{i,j}(m).d\leq epsilon$，则 $x_p\sim x_{I(i,j)}$ 会被作为一个候选查询结果. 之后每个列向量产生后，都会对该候选查询结果进行判定，以确定该查询结果是否是真正查询结果，而当新产生列向量的最后一个成员 d 值小于 $\boldsymbol{SV}_{i,j}(m).d$ 时，该候选查询结果会被更新为新列向量最后一个成员所在 DTW 路径对应的数据流序列. 所以我们要证明的包括两点：(1) 省略掉 $\boldsymbol{SM}_i$ 中 d 值大于 $epsilon$ 的成员取值计算后，$\boldsymbol{SM}_i$ 中所有候选查询结果序列不会发生变化；(2) 省略掉 $\boldsymbol{SM}_i$ 中 d 值大于 $epsilon$ 的成员取值计算，不会影响对这些候选查询结果的判断结果.

(1) 假设 $x_p\sim x_{I(i,j)}$ 是任意一个候选查询结果，由 Spring 算法可知，$\boldsymbol{SV}_{i,j}(m).sp$ 小于等于 $epsilon$，另有该查询结果对应的 DTW 路径（即在 $\boldsymbol{SV}_{i,p-i+1}(1)$ 与 $\boldsymbol{SV}_{i,j}(m)$ 之间存在的 DTW 路径 $R_{i,j}(m)$）上所有成员的 d 值都会小于等于 $epsilon$. 因为任何 d 值大于 $epsilon$ 的成员都不可能存在于候选查询结果的 DTW 路径上，所以对于 $\boldsymbol{STWM}$ 中 d 值大于 $epsilon$ 的成员取值计算的省略都不会影响最终的候选查询结果，于是所有的候选查询结果都不会受 $\boldsymbol{SM}_i$ 中 d 值大于 $epsilon$ 的成员取值计算省略后的影响.

(2) 由 Spring 算法可知，当 $x_p\sim x_{I(i,j)}$ 成为候选查询结果后，之后每个列向量产生后，都需要判别该候选查询结果是否是真正的查询结果. 具体方法是对于新列向量中起始位置 sp 属于区间 $[p,I(i,j)]$ 中的成员，如果它们的 d 值都大于 $\boldsymbol{SV}_{i,j}(m)$ 的 d 值 $\boldsymbol{SV}_{i,j}(m).d$，则该候选查询结果为真实查询结果. 假设在向量 $\boldsymbol{SV}_{i,l}$ 产生后，我们得到结论 $x_p\sim x_{I(i,j)}$ 是真实查询结果，则有该向量中起始位置在区间 $[p,I(i,j)]$ 中的成员，其 d 值都大于 $\boldsymbol{SV}_{i,j}(m).d$. 如果这些成员中没有 d 值大于 $epsilon$ 的成员，显然，省略掉 $\boldsymbol{SM}_i$ 中 d 值大于 $epsilon$ 的成员取值计算并不会影响最终的判断结果；如果这些成员中有 d 值大于 $epsilon$ 的成员，则有这些成员的 d 值一定大于 $\boldsymbol{SV}_{i,j}(m).d$，所以最终结果取决于这些成员中 d 值小于等于 $epsilon$ 的成员，因为这个省略操作并不会省略掉矩阵列向量中 d 值小于等于 $epsilon$ 的成员取值，所以不会影响最终判断结果.

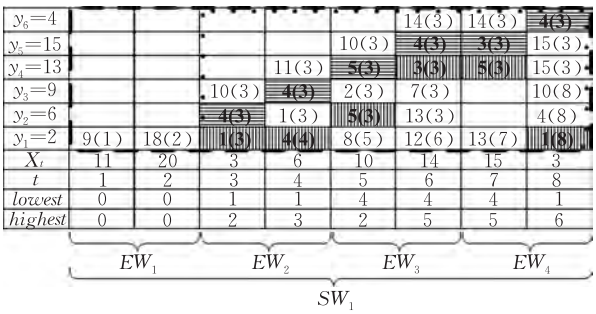
综上，定理 1 成立.　　证毕.

设 FSM 算法中的 $\boldsymbol{STWM}$ 为 $\boldsymbol{FM}$，Spring 算法中的 $\boldsymbol{STWM}$ 为 $\boldsymbol{SM}$. 由 Spring 与 FSM 算法可知，前者与后者相比，省略了很多不必要成员取值的计算，而这些省略不会影响矩阵中剩余成员的取值计算，也不会影响最终查询结果. 所以在本部分 $\boldsymbol{SM}$ 中成立的性质，在 FSM 算法下的 $\boldsymbol{FM}$ 中同样成立.

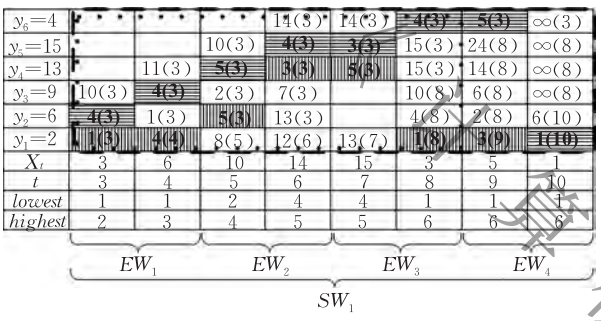
4.2.2.2　关于猜想的分析证明

设 SW_i，SW_{i+1} 是任意 2 个相邻滑动窗口，$\boldsymbol{FM}_i$ 与 $\boldsymbol{FM}_{i+1}$ 是这 2 个窗口成员在 FSM 算法下对应的 $\boldsymbol{STWM}$. 由滑动窗口特点可知，SW_i 上后 $|SW|-|EW|$ 个数据流成员和 SW_{i+1} 上前 $|SW|-|EW|$ 个数据流成员完全相同. 另设 $\boldsymbol{FM}_i$ 中后 $|SW|-|EW|$ 个列向量组成的子矩阵为 $\boldsymbol{FLM}_i$，前 $|SW|-|EW|$ 个列向量组成的子矩阵为 $\boldsymbol{FFM}_i$. 显然 $\boldsymbol{FLM}_i$ 与 $\boldsymbol{FFM}_{i+1}$ 分别为由 SW_i，SW_{i+1} 上相同数据流成员在 2 个矩阵中对应的列向量组成的矩阵. 我们可以通过对 SW_i 与 SW_{i+1} 上相同数据流成员重新执行 FSM 算法来得到 $\boldsymbol{FFM}_{i+1}$. 设 $\boldsymbol{Fv}_{i,j}$ 是 $\boldsymbol{FLM}_i$ 中第 j 个列向量，$\boldsymbol{UFv}_{i,j}$ 为 $\boldsymbol{Fv}_{i,j}$ 所对应的数据流成员在 $\boldsymbol{FFM}_{i+1}$ 中对应的列向量，显然 $\boldsymbol{UFv}_{i,j}$ 是 $\boldsymbol{FFM}_{i+1}$ 中的第 j 个列量，且 $\boldsymbol{FFM}_{i+1}$ 由所有 $\boldsymbol{UFv}_{i,j}$ 组成. 同样

使用文献[1-2]的查询实例，假设数据流 $X=\{11,20,3,6,10,14,15,3,5,1\}$，查询序列 $Y=\{2,6,9,13,15,4\}$，$epsilon=15$，$SW_{size}=4$，$|EW|=2$，我们在图 6 中实例说明了 $\boldsymbol{FLM}_i$ 与 $\boldsymbol{FFM}_{i+1}$ 以及 $\boldsymbol{Fv}_{i,j}$ 与 $\boldsymbol{UFv}_{i,j}$ 之间的关系.



(a) $\boldsymbol{FM}_1$



(b) $\boldsymbol{FM}_2$

图 6 $\boldsymbol{FLM}_i$ 与 $\boldsymbol{FFM}_{i+1}$ 示例

在图 6 的 $\boldsymbol{FM}$ 中，每个成员包括 2 部分取值，括号内外分别为起始位置 sp 与成员 d 值. 图中线虚线区域分别是 $\boldsymbol{FM}_1$ 与 $\boldsymbol{FM}_2$. 点虚线区域分别是 $\boldsymbol{FLM}_1$ 与 $\boldsymbol{FFM}_2$；其中 $\boldsymbol{FLM}_1$ 是由 $\boldsymbol{Fv}_{1,1}$，$\boldsymbol{Fv}_{1,2}$，$\boldsymbol{Fv}_{1,3}$，$\boldsymbol{Fv}_{1,4}$，$\boldsymbol{Fv}_{1,5}$，$\boldsymbol{Fv}_{1,6}$ 构成；$\boldsymbol{FFM}_2$ 是由 $\boldsymbol{UFv}_{1,1}$，$\boldsymbol{UFv}_{1,2}$，$\boldsymbol{UFv}_{1,3}$，$\boldsymbol{UFv}_{1,4}$，$\boldsymbol{UFv}_{1,5}$，$\boldsymbol{UFv}_{1,6}$ 构成. 我们通过对 $\boldsymbol{Fv}_{1,1}$，$\boldsymbol{Fv}_{1,2}$，$\boldsymbol{Fv}_{1,3}$，$\boldsymbol{Fv}_{1,4}$，$\boldsymbol{Fv}_{1,5}$，$\boldsymbol{Fv}_{1,6}$ 所对应的数据流成员执行 FSM 算法来得到 $\boldsymbol{UFv}_{1,1}$，$\boldsymbol{UFv}_{1,2}$，$\boldsymbol{UFv}_{1,3}$，$\boldsymbol{UFv}_{1,4}$，$\boldsymbol{UFv}_{1,5}$，$\boldsymbol{UFv}_{1,6}$.

考虑 $\boldsymbol{FFM}_{i+1}$ 的构造过程，对于相邻两个窗口中的相同数据流成员，按照当前使用 FSM 算法重新处理完的成员中是否有查询结果产生，可分为有查询结果和没有查询结果产生 2 种情况.

(1) 在当前已被处理完成的数据流成员中没有查询结果产生

为了便于对当前情况下理论依据进行证明，本部分我们会先对 Spring 算法下相邻窗口的相同数据流成员在这两个窗口 $\boldsymbol{STWM}$ 中对应列向量成员取值关系进行分析，并以引理形式给出分析结果. 接着基于此在定理 2 中完成对理论依据的证明.

令 $\boldsymbol{SM}_i$ 与 $\boldsymbol{SM}_{i+1}$ 分别是 2 个相邻滑动窗口 SW_i，SW_{i+1} 中成员在 Spring 算法下对应的 $\boldsymbol{STWM}$. 另设 $\boldsymbol{SM}_i$ 中后 $|SW|-|EW|$ 个列向量组成的子矩阵为 $\boldsymbol{SLM}_i$，前 $|SW|-|EW|$ 个列向量组成的子矩阵为 $\boldsymbol{SFM}_i$. 显然 $\boldsymbol{SLM}_i$ 与 $\boldsymbol{SFM}_{i+1}$ 分别为由 SW_i，SW_{i+1} 上相同数据流成员在 $\boldsymbol{SM}_i$ 与 $\boldsymbol{SM}_{i+1}$ 中对应列向量组成的矩阵. 与 FSM 算法下情况类似，我们可以通过对 SW_i 与 SW_{i+1} 上相同数据流成员重新执行 Spring 算法的来得到 $\boldsymbol{SFM}_{i+1}$. 设 $\boldsymbol{Sv}_{i,j}$ 是 $\boldsymbol{SLM}_i$ 中的第 j 个列向量，而 $\boldsymbol{USv}_{i,j}$ 为该向量所对应的数据流成员在重新执行 Spring 算法后得到的新向量，显然 $\boldsymbol{USv}_{i,j}$ 是 $\boldsymbol{SFM}_{i+1}$ 中的第 j 个列量，且 $\boldsymbol{SFM}_{i+1}$ 由所有 $\boldsymbol{USv}_{i,j}$ 组成.

引理 2. 在 $\boldsymbol{SFM}_{i+1}$ 的构造过程中，如果存在 $\boldsymbol{SLM}_i$ 中成员 $\boldsymbol{Sv}_{i,j}(t)$ 满足条件 $\boldsymbol{Sv}_{i,j}(t).sp=\boldsymbol{USv}_{i,j}(t).sp$，且 $\boldsymbol{Sv}_{i,j}(t).d=\boldsymbol{USv}_{i,j}(t).d(1\leq t\leq m)$，则对于 $\boldsymbol{Sv}_{i,j}$ 与 $\boldsymbol{USv}_{i,j}$ 中索引小于 t 的所有成员有如下结论成立：$\boldsymbol{Sv}_{i,j}(t_1).sp=\boldsymbol{USv}_{i,j}(t_1).sp$，$\boldsymbol{Sv}_{i,j}(t_1).d=\boldsymbol{USv}_{i,j}(t_1).d$ 成立，其中 $t_1<t$.

证明. 设 $\boldsymbol{Sv}_{i,j}(t).sp=p$，由引理 1 的结论 1 可知，$\boldsymbol{Sv}_{i,j}(t)$ 在 $\boldsymbol{SLM}_i$ 中与 x_p 间存在一条唯一 DTW 路径，我们把该路径命名为 R_1，因为当满足条件 $\boldsymbol{Sv}_{i,j}(t).sp=\boldsymbol{USv}_{i,j}(t).sp$，且 $\boldsymbol{Sv}_{i,j}(t).d=\boldsymbol{USv}_{i,j}(t).d$ 的 $\boldsymbol{USv}_{i,j}(t)$ 出现时，当前已被处理完成的所有数据流成员中并没有查询结果产生，所以路径 R_1 上的所有成员在当前已得到的 $\boldsymbol{SFM}_{i+1}$ 中依然存在，我们把此时的这条路径轨迹称为 R_2，我们只要能证明 R_2 上所有成员的取值与 R_1 都相同，则由 Spring 算法可知，$\boldsymbol{SFM}_{i+1}$ 中位于 R_2 下的成员与 $\boldsymbol{SLM}_i$ 中位于 R_1 下矩阵成员取值都相同，所以有 $\boldsymbol{Sv}_{i,j}(t_1).sp=\boldsymbol{USv}_{i,j}(t_1).sp$，$\boldsymbol{Sv}_{i,j}(t_1).d=\boldsymbol{USv}_{i,j}(t_1).d$，$(t_1<t)$ 成立，也即我们要证明的结论成立.

考虑 R_1 上的成员，由 Spring 算法可知，每个成员都是其相邻下一个成员的 d_{best} 所在成员. 又因为 R_2 上所有成员的位置与其在 R_1 上的对应成员置位置都相同，所以在 R_2 中，如果每个成员仍是其相邻下一个成员的 d_{best} 所在成员，结合 Spring 算法可知，R_2 与 R_1 上的成员取值完全相同，由前面分析可知，结论成立；如果 R_2 上有成员的 d_{best} 成员发生变化，由 Spring 算法可知，新 d_{best} 成员 d 值一定小于原来的 d_{best} 成员 d 值，R_2 最后一个成员 d 值与其在

R_1中的 d 值相比也会变小，也就是说此时 $\boldsymbol{USv}_{i,j}(t)$ 与 x_p间存在一条路径，该路径的 DTW 值小于 R_1的 DTW 值 $\boldsymbol{Sv}_{i,j}(t).d$，所以此时 $\boldsymbol{USv}_{i,j}(t)$的 d 值不可能与 $\boldsymbol{Sv}_{i,j}(t).d$ 相等，这与条件矛盾，所以这种情况不可能.

总之结论成立.　　证毕.

定理 2.　设$h_{i,j}$与$uh_{i,j}$分别是与向量$\boldsymbol{Fv}_{i,j}$,$\boldsymbol{UFv}_{i,j}$对应的 $highest$；而 $dmin_{i,j}$与 $udmin_{i,j}$分别是与向量 $\boldsymbol{Fv}_{i,j}$和 $\boldsymbol{UFv}_{i,j}$对应的 $dmin$ 值，$FR_{i,j}$和 $UFR_{i,j}$分别是与 $dmin_{i,j}$与 $udmin_{i,j}$对应的候选查询结果. 在使用 FSM 算法重新处理相邻窗口中相同数据流成员获得 $\boldsymbol{FFM}_{i+1}$的过程中，对于 $\boldsymbol{FFM}_{i+1}$中的任意列向量 $\boldsymbol{UFv}_{i,j}$，如果满足如下条件：① $h_{i,j}=uh_{i,j}$；② $\boldsymbol{Fv}_{i,j}(h_{i,j}).sp=\boldsymbol{UFv}_{i,j}(uh_{i,j}).sp$，$\boldsymbol{Fv}_{i,j}(h_{i,j}).d=\boldsymbol{UFv}_{i,j}(uh_{i,j}).d$；③ $dmin_{i,j}=udmin_{i,j}$；④ $FR_{i,j}=UFR_{i,j}$；则有 $\boldsymbol{FFM}_{i+1}$中剩余列向量成员与 $\boldsymbol{FLM}_i$中位于 $\boldsymbol{Fv}_{i,j}$之后的对应列向量成员取值都相同，同时在它们构造过程中产生的查询结果相同.

证明.　假设在求得 $\boldsymbol{FFM}_{i+1}$成员取值的过程中，$\boldsymbol{UFv}_{i,j-1}$的 $highest$ 位置并没有变化. 由 FSM 与 Spring 算法间的关系以及引理 2 可知，条件①、②的成立可以保证 $\boldsymbol{Fv}_{i,j}$与 $\boldsymbol{UFv}_{i,j}$中索引值小于 $h_{i,j}$的成员，其 d 值与 sp 值都对应相同. 而由 FSM 算法可知，该算法 ***STWM*** 的每个列向量成员取值只与前一个列向量的 $highest$ 值，$dmin$ 值，与 $dmin$ 值对应的候选查询结果以及前一个列向量中索引小于 $highest$ 的所有成员取值相关，所以再结合条件③、④，可推得 $\boldsymbol{FFM}_{i+1}$中剩余其他列向量成员与 $\boldsymbol{FLM}_i$中剩余列向量中对应成员取值相同，产生查询结果也相同.

假设在求得 $\boldsymbol{FFM}_{i+1}$成员取值过程中 $\boldsymbol{UFv}_{i,j-1}$的 $highest$ 位置发生变化，由 FSM 算法可知，此时只需对上面情况下 $\boldsymbol{FFM}_{i+1}$中位于 $\boldsymbol{UFv}_{i,j}$后(包括 $\boldsymbol{UFv}_{i,j}$)的矩阵成员作相应调整即可，而这个调整也仅与其中 d 值大于 $epsilon$ 的成员相关，也即调整后，其中 d 值小于等于 $epsilon$ 的成员位置与取值并没有发生变化，由定理 1 可推得这种情况下 $\boldsymbol{FFM}_{i+1}$中剩余列向量成员与 $\boldsymbol{FLM}_i$中位于 $\boldsymbol{Fv}_{i,j}$之后对应列向量成员在构造过程中产生相同查询结果.　　证毕.

仍然使用文献[1-2]中的查询实例. 假设数据流 $X=\{11,20,3,6,10,14,15,3,5,1\}$，查询序列 $Y=\{2,6,9,13,15,4\}$，$epsilon=15$，$SW_{size}=4$，$|EW|=2$，图 7 中(a)、(b)的线虚线区域分别给出了相邻 2 个窗口的 ***FM***；2 个 ***FM*** 中列向量内具有横线和竖线背景成员的数字分提标出了该向量中 $highest$ 与 $lowest$ 成员，而它们在向量中的索引分别是该向量中 $highest$ 与 $lowest$ 变量的取值. 图 7 的 ***FM*** 中，每个列向量上面是按照 FSM 算法处理完该向量后得到的 $dmin$ 值.

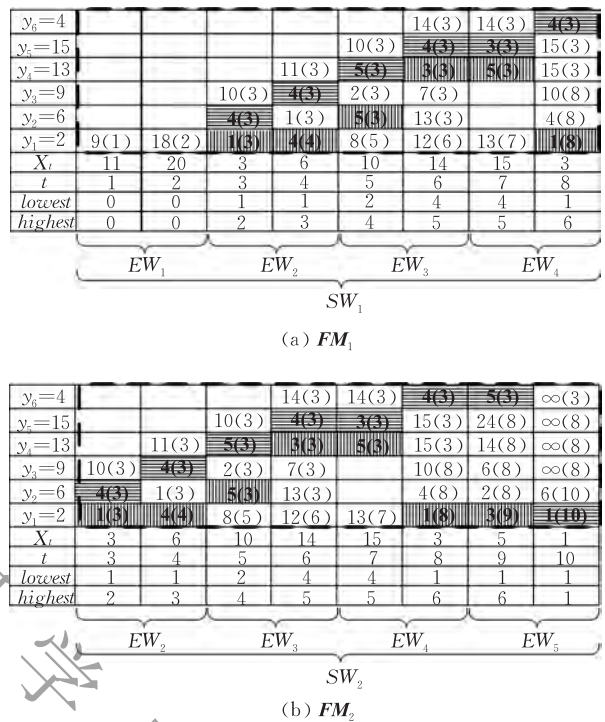


图 7　定理 2 的说明

定理 2 告诉我们，在使用 FSM 算法处理相邻两个窗口中的相同数据流成员的时候，如果存在一个数据流成员在相邻两个窗口的 ***FM*** 中对应的相关信息都分别相同，则有其他相同数据流成员在这 2 个窗口 ***FM*** 中对应列向量的成员取值，特别是其中 d 值小于 $epsilon$ 的成员取值都完全相同. 比如图 7 中的 x_3在 $\boldsymbol{FM}_1$与 $\boldsymbol{FM}_2$的列向量中，其 $highest$(这里该值为 2)，第 $highest$ 个成员取值(包括 d 值与 sp 值)，$dmin$ 取值(这里都为∞)以及候选查询结果(这里为 null)都相同，所以 2 个窗口中其他相同数据流成员在这两个窗口的 ***FM*** 中对应列向量成员取值完全相同，其中 d 值小于 $epsilon$ 的成员取值也完全相同.

(2) 在当前已被处理完成的数据流成员中有查询结果产生

当前情况下分析证明过程与情况 1 相类似，我们会先对 Spring 算法下相邻窗口相同数据流成员在这两个窗口 ***STWM*** 中对应列向量成员取值的关

系进行分析,并以引理形式给出分析结果.接着在此基础之上在定理 3 中完成了对理论依据的证明.

引理 3. 假设 $\boldsymbol{USv}_{i,j}$ 构造完后,在当前已被处理完成的数据流成员中有查询结果产生.这时如果有 $\boldsymbol{Sv}_{i,j}(t).sp=\boldsymbol{USv}_{i,j}(t).sp$,$\boldsymbol{Sv}_{i,j}(t).d=\boldsymbol{USv}_{i,j}(t).d$ $(1\leq t\leq m)$,则对于这 2 个向量中索引小于 t 所有成员有如下结论成立:$\boldsymbol{Sv}_{i,j}(t_1).sp=\boldsymbol{USv}_{i,j}(t_1).sp$,$\boldsymbol{Sv}_{i,j}(t_1).d=\boldsymbol{USv}_{i,j}(t_1).d$ 成立,其中 $t_1<t$.

证明. 与引理 2 证明相类似,设 $\boldsymbol{Sv}_{i,j}(t).sp=p$,$\boldsymbol{Sv}_{i,j}(t)$ 在 $\boldsymbol{SLM}_i$ 中与 x_p 间的 DTW 路径为 R_1.因为当满足条件 $\boldsymbol{Sv}_{i,j}(t).sp=\boldsymbol{USv}_{i,j}(t).sp$,且 $\boldsymbol{Sv}_{i,j}(t).d=\boldsymbol{USv}_{i,j}(t).d$ 的 $\boldsymbol{USv}_{i,j}(t)$ 出现时,当前已被处理完成的所有数据流成员中有查询结果产生,由 Spring 算法可知路径 R_1 上的成员在当前已得到的 $\boldsymbol{SFM}_{i+1}$ 中有 2 种情况:① R_1 上的成员在 $\boldsymbol{SFM}_{i+1}$ 中依然存在;② R_1 上有的成员在 $\boldsymbol{SFM}_{i+1}$ 中的 d 值变为 ∞.

情况①下的引理证明过程与引理 2 的证明相似,此处不再赘述,易得结论成立.

情况②的出现完全是因为 Spring 算法中当构建完 $\boldsymbol{STWM}$ 中的列向量产生查询结果 $x_s\sim x_e$ 时,该列向量中所有 sp 值小于等于 e 的成员,其 d 值都被设为 ∞,当 $\boldsymbol{SFM}_{i+1}$ 中 R_1 上存在符合这样情况的成员时,相关成员会因为 d 值被设为 ∞ 而使它们自身不存在于任何路径中.这种情况下,因为有条件 $\boldsymbol{Sv}_{i,j}(t).sp=\boldsymbol{USv}_{i,j}(t).sp$,且 $\boldsymbol{Sv}_{i,j}(t).d=\boldsymbol{USv}_{i,j}(t).d$ 成立,所以基于引理 1 的结论 1 可知 $\boldsymbol{USv}_{i,j}(t)$ 在 $\boldsymbol{SFM}_{i+1}$ 中与 x_p 间存在一条 DTW 路径,我们将其命名为 R_2,R_2 上不存在 d 值为 ∞ 的成员.接下来考虑 R_2 在原来 $\boldsymbol{SLM}_i$ 中的状态,我们把 R_2 在 $\boldsymbol{SLM}_i$ 中的路径轨迹称为 R_3.与前面分析类似,R_3 中成员也有两种情况:① R_3 中不存在 ∞ 成员;② R_3 中存在 ∞ 成员.考虑第 1 种情况,如果 R_2 与 R_3 上每个成员的 d_{best} 成员都一样,则 R_2 与 R_3 上每个成员的 d 值与 sp 值都相同,结合引理 2 证明过程可知,结论成立;如果 R_3 上存在某个成员的 d_{best} 成员发生变化,由 Spring 算法可知,新 d_{best} 成员 d 值一定会变小,最终 R_3 的 d 值也会变小,即有 R_3 的 d 值小于 $\boldsymbol{Sv}_{i,j}(t).d$,也就是说在 $\boldsymbol{SLM}_i$ 中的 $\boldsymbol{Sv}_{i,j}(t)$ 与 x_p 间存在一条 d 值小于 $\boldsymbol{Sv}_{i,j}(t).d$ 的路径,所以此时 $\boldsymbol{USv}_{i,j}(t)$ 的 d 值不可能与 $\boldsymbol{Sv}_{i,j}(t).d$ 相等,这与条件矛盾,所以这种情况不可能;考虑第 2 种情况,由前面可知此时 R_1 在 $\boldsymbol{SFM}_{i+1}$ 中有 d 值为 ∞ 的成员存在,又因为 R_2 上不存在 d 值为 ∞ 的成员,所以 R_2 一定位于 R_1 的下面,对于 $\boldsymbol{SLM}_i$ 中的 R_1 与 R_3,因为 R_3 与 R_2 具有相同的路径轨迹,所以 R_3 也一定位于 R_1 的下面.如果此时 R_3 中存在 ∞ 成员,由 Spring 算法可知,R_1 中一定也存在 ∞ 成员,但由条件可知,$\boldsymbol{SLM}_i$ 中的 R_1 并不存在 ∞ 成员,矛盾,即这种情况不存在.总之在所有可能出现的情况中,都有结论成立. 证毕.

定理 3. 设 $h_{i,j}$ 与 $uh_{i,j}$ 分别是与向量 $\boldsymbol{Fv}_{i,j}$,$\boldsymbol{UFv}_{i,j}$ 对应的 $highest$;而 $dmin_{i,j}$ 与 $udmin_{i,j}$ 分别是与向量 $\boldsymbol{Fv}_{i,j}$ 和 $\boldsymbol{UFv}_{i,j}$ 对应的 $dmin$ 值,$FR_{i,j}$ 和 $UFR_{i,j}$ 分别是与 $dmin_{i,j}$,$udmin_{i,j}$ 对应的候选查询结果.在使用 FSM 算法重新处理相邻窗口中相同数据流成员获得 $\boldsymbol{FFM}_{i+1}$ 的过程中,对于 $\boldsymbol{FFM}_{i+1}$ 中任意列向量 $\boldsymbol{UFv}_{i,j}$,如果在其构造完成后已有查询结果产生,同时也有如下条件成立:① $h_{i,j}=uh_{i,j}$;② $\boldsymbol{Fv}_{i,j}(h_{i,j}).sp=\boldsymbol{UFv}_{i,j}(uh_{i,j}).sp$,$\boldsymbol{Fv}_{i,j}(h_{i,j}).d=\boldsymbol{UFv}_{i,j}(uh_{i,j}).d$;③ $dmin_{i,j}=udmin_{i,j}$;④ $FR_{i,j}=UFR_{i,j}$,则可推得 $\boldsymbol{FFM}_{i+1}$ 中剩余列向量成员与 $\boldsymbol{FLM}_i$ 中位于 $\boldsymbol{Fv}_{i,j}$ 之后的对应列向量成员取值都相同,同时在它们构造过程中所产生查询结果也相同.

证明. 定理 3 证明过程与定理 2 的证明过程类似,只需将其中的证明依据引理 2 变为引理 3 即可.此处不再赘述. 证毕.

定理 3 的内容同样可以结合图 7 来说明.由定理 2 可知在我们处理相邻窗口上的相同数据流成员时,如果有如下两个条件成立:① 存在一个成员在两个窗口 $\boldsymbol{FM}$ 中对应列向量的相关信息都相同;② 对于两个窗口中所有已被处理完成的相同数据流成员,如果在处理它们的过程中没有查询结果产生,那么其他相同数据流成员在两个窗口 $\boldsymbol{FM}$ 中对应的列向量成员取值都相同,且产生的查询结果也相同.比如图 7 中的 x_3 就是这种情况,由定理 2 可知,其他相同数据流成员在两个窗口 $\boldsymbol{FM}$ 中对应列向量的相关信息都相同,而这些列向量在两个矩阵的构建过程中产生的查询结果也一样;定理 3 的内容是说,如果条件①满足,但是条件②不满足,即对于两个窗口中所有已被处理完成的相同数据流成员,如果在处理它们的过程中有查询结果产生,这种情况下,结论也是成立的.比如 x_3 之前如果已经处理了很多相邻 2 个窗口中的相同数据流成员,而且在处理的过程中已经有查询结果产生,这时如果出现了满足条件①的 x_3,则仍有结论成立.

4.2.3 DQPIC 算法

(1) DQPIC 算法执行步骤

定理 2、3 从两个角度对于 DQPIC 算法的理论基础进行了完整地分析与证明.基于这 2 个定理以

及 4.2.1 节的分析，可得到具有增量计算特点的 DQ-SW 查询处理算法 DQPIC. 具体执行步骤如下：

首先，在第 1 个窗口上执行 naive 算法得到查询结果. 在这个过程中保留每个数据流成员在当前窗口 ***FM*** 中对应列向量的相关参数，包括每个列向量的 *highest*，列向量中索引为 *highest* 的成员取值（*d* 值，*sp* 值），naive 算法在处理完该数据流成员后的 *dmin* 值，以及与该 *dmin* 值对应的候选查询结果，除此之外，该窗口上产生的查询结果以及窗口中最后一个数据流成员对应的列向量也需要保留.

接着处理下一个窗口上数据流成员. 此时分为 2 个阶段：对于相邻窗口上相同数据流成员进行处理和对新添加基本窗口中的数据流成员进行处理.

在第 1 个阶段，首先继续用 naive 算法处理相邻窗口上相同数据流成员. 与之前一样需要保留相关参数，同时还需要比较相同数据流成员在两个窗口上参数取值是否相同（定理 2、3）. 如果有成员满足这个条件，则将剩余相同数据流成员在前一个窗口上相关参数作为它们在当前窗口上相关参数，将剩余相同数据流成员在前一窗口处理过程中产生的查询结果，作为当前窗口上这些成员被处理时产生的查询结果（定理 2、3），它们和当前窗口上已被保留的查询结果构成了所有相同数据流成员在当前窗口上的查询结果；若不存在这样成员，则一直执行 naive 算法，当处理完所有相同数据流成员后，即得到这部分成员在当前窗口上查询结果；接着处理当前窗口中新加的数据流成员，即进入第 2 阶段.

在第 2 阶段，如果第 1 阶段是以第 1 种情况结束，由 FSM 算法可知，该算法 ***STWM*** 每个列向量成员取值只与前一个列向量的 *highest* 值，*dmin* 值，与 *dmin* 值对应的候选查询结果以及前一个列向量中索引小于 *highest* 的所有成员取值相关，所以只要知道最后一个相同数据流成员在当前窗口上相关参数和对应列向量取值即可，而因为此时在前一窗口上保留的最后 1 个数据流成员对应的列向量和与之相关的参数符合要求，所以只需基于它们对新基本窗口数据流成员执行 naive 算法即可，当然这个过程也需要保留数据流成员相关参数和最后 1 个数据流成员对应的列向量；如果第 1 阶段是以第 2 种情况结束时，则继续对新基本窗口中成员执行 naive 算法即可，这个过程也需要保留数据流成员相关参数和最后 1 个数据流成员对应的列向量.

最后处理完 2 个阶段后，由它们上的查询结果共同构成了当前窗口上的完整查询结果.

（2）DQPIC 算法的具体描述

根据上面的执行步骤，我们在算法 1 中给出了 DQPIC 算法的详细描述. 为减少因滑动窗口中内容更新而造成的资源损耗，我们采用可重写循环滑动窗口思想[22]存放新来的基本窗口成员.

算法 1.　DQPIC.

输入：数据流 X，查询序列 Y，滑动窗口大小 SW_{size}，基本窗口宽度 $|EW|$，相似阈值 *epsilon*

输出：满足相似性条件的 DQ-SW 查询的结果

```
1.  数据流 X 不断进行
2.  {
3.   新基本窗口 EW_{i+SW_size-1} 到达时，最近 SW_size 个基本窗口组成流上滑动窗口 SW_cur；
4.   [Para_cur, result_cur]=Preserve_FSM(SW_cur);
       //对当前窗口成员执行 FSM 算法，保留相关信息
5.   Report(result_cur); //输出当前窗口结果
6.   winelenum=SW_size;
7.   WHILE EW_{i+SW_size} arrives
8.   {winelenum++, result1_tem=null;
9.   SW[(winelenum-1)%|SW_size|]←EW_{i+SW_size};
       //使用循环滑动窗口存放新到来的 EW_{i+SW_size}
10.  T=(i+1);
11.  BEGIN from EW_T, FOR(each EW in SW_cur)
12.  {IF(EW!=EW_{i+SW_size})
13.   {FOR(each element v in the EW)
14.    {[tempara, result_tem]=Preserve_FSM(v);
15.    result1_tem=result1_tem ∪ result_tem;
16.    IF ((tempara(dmin, sp, candidate, hbound, V_hbound)=Para_cur,v(dmin, sp, candidate, hbound, V_hbound))
17.     {[Para_cur((winelenum-1)%|SW_size|), result_tem]←Preserve_FSM_P(EW_{i+SW_size}, Para_cur.lastvec);
18.      result_cur = result1_tem ∪ result_tem ∪ getresult(result_cur);
19.     BREAK;
20.     }
21.    ELSE
22.     Update Para_cur,v with tempara;
23.    }//FOR
24.   }//IF
25.   ELSE
26.   {[Para_cur((winelenum-1)%|SW_size|), result_tem]←Preserve_FSM_P(EW_{i+SW_size}, Para_cur.lastvec);
27.    result_cur=result1_tem ∪ result_tem;
28.   }
29.  }//BEGIN
```

30. $Report(result_{cur})$;
31. i++,并转到第 7 行;
32. }//WHILE
33. }

设 $\boldsymbol{FM}_{cur}$ 与 $\boldsymbol{FM}_{next}$ 分别是 FSM 算法在当前窗口与相邻下一个窗口上的 ***STWM***,而 $\boldsymbol{FLM}_{cur}$ 与 $\boldsymbol{FFM}_{next}$ 分别是它们中对应的 ***LSM*** 与 ***FSM***.

算法执行时首先对第 1 个窗口上数据流成员执行 FSM 算法,并保留它们在 $\boldsymbol{FM}_{cur}$ 中对应列向量的参数,这些参数有每个列向量的 $curbound$,$highest$(用 $hbound$ 表示),该列向量中索引为 $hbound$ 的成员取值 V_{hbound},sp 值以及与该向量相关的 $dmin$ 值,与 $dmin$ 对应的候选查询结果 $candidate$. 除此之外,$\boldsymbol{FLM}_{cur}$ 中的最后 1 个列向量,还有窗口上 ***STWM*** 构造过程中产生的查询结果,都需要保留. 当窗口中成员处理完成后,输出该窗口上结果(行 4～5);

当新数据到来后,更新窗口并处理新窗口成员(行 7～11);对相邻窗口中相同数据流成员执行 FSM 算法,保留相关参数信息,并将结果放入 $result1_{tem}$ 集合(行 14～15),这个过程也是 $\boldsymbol{FFM}_{next}$ 的构造过程,需要同时比较相同数据流成员在 $\boldsymbol{FLM}_{cur}$ 与 $\boldsymbol{FFM}_{next}$ 对应列向量中索引为 $hbound$ 的成员参数取值是否相同(行 16). 若相同,则无需处理其他相同数据流成员,直接将这些成员在前一窗口上相关信息作为它们在新窗口中相关信息保留. 在 $\boldsymbol{FLM}_{cur}$ 中最后 1 个列向量及其相关参数基础上,对新基本窗口中成员执行 FSM 算法,并保留结果到集合 $result_{tem}$ 中(行 17). 此时 $result_{cur}$ 中存放的是前一窗口上查询结果,得到前一个窗口上在处理当前成员之后所有成员时所产生的查询结果,则该结果与 $result1_{tem}$ 和 $result_{tem}$ 并集即为当前窗口上查询结果(行 18). 如果不相同,使用相同方法继续处理剩余相同数据流成员(行 21～22). 如果在处理相邻窗口相同数据流成员过程中没有任何成员在 $\boldsymbol{FLM}_{cur}$ 与 $\boldsymbol{FFM}_{next}$ 中对应参数完全相同,那么此时执行过程与 naive 算法相同,考虑到窗口成员处理的完整性,假设处理完最后 1 个基本窗口得到的查询结果集合为 $result_{tem}$,则此时 $result1_{tem} \cup result_{tem}$ 即为当前窗口上查询结果(行 25～30),新窗口上查询结果求解过程结束.

4.2.4 关于 DQPIC 算法时间和空间复杂度的分析

(1)时间复杂度

由前面分析可知,DQPIC 算法使用 naive 算法求得数据流上第一个窗口上的 DQ-SW 查询结果;对于其他窗口上的查询结果,该算法可基于前一窗口上的查询结果增量获得. 设数据流成员个数为 N,则所有窗口数为 $(N/|EW|)-SW_{size}+1$,算法在第 1 个窗口上时间复杂度为 $O(SW_{size}\times|EW|\times|Y|)$;其他每个窗口上的时间复杂度由两部分组成:处理相邻窗口上重叠成员部分和处理当前窗口相对于前一窗口新增基本窗口中成员所花时间开销部分. 假设对第 1 部分成员的平均处理个数为 T,在前一个窗口查询结果集合中搜索当前窗口上查询结果需要遍历的平均成员个数为 P,则其他每个窗口时间开销为 $O(|Y|\times T+P+|EW|\times|Y|)$. 设 $M=(N/|EW|)-SW_{size}+1$,则平均每个窗口时间复杂度为 $[O(SW_{size}\times|EW|\times|Y|)+O((M-1)\times(|Y|\times T+P+|EW|\times|Y|))]/M$,即 $O(SW_{size}\times|EW|\times|Y|)/M+O(|Y|\times T+P+|EW|\times|Y|)$. 当相邻 2 个窗口重叠部分成员在 2 个窗口的 ***STWM*** 中对应信息都不相同时,有 $T=(SW_{size}-1)\times|EW|$,同时此刻也无需使用前一个窗口的查询结果,即 $P=$null,DQPIC 算法退化为 naive 算法,由 FSM 算法可知,此时每个窗口上的最坏情况下的时间复杂度为 $O(SW_{size}\times|EW|\times|Y|)$.

(2)空间复杂度

DQPIC 算法需要保留每个窗口上数据流成员,同时也需要 2 个大小为 $|Y|$ 的列向量来完成当前窗口每个成员相关信息的获得;除此之外还需要额外空间存放与每个窗口成员相关的信息,因为窗口成员所占有的空间开销是 $O(SW_{size}\times|EW|)$,所以这部分额外空间所占有的空间开销为 $O(M_1\times SW_{size}\times|EW|)$,其中 $M_1>1$. 另外 DQPIC 算法需要保留当前窗口上最后一个数据流成员在 ***FM*** 中的对应列向量,再加上当前窗口查询结果的空间开销 Q,所以当前窗口上的空间总开销为 $O(3|Y|+M_1\times SW_{size}\times|EW|+Q)$;比较起来 naive 算法主要空间开销在于每个窗口上的数据流成员和 2 个大小为 $|Y|$ 的列向量,除此之外并不需要保留其他多余信息,所以空间开销为 $O(2|Y|+SW_{size}\times|EW|)$.

5 实验及结果分析

本部分的实验有 2 个目的:(1)DQPIC 算法有效性的验证,即实例证明 DQPIC 算法与 naive 算法的执行结果是相同的;(2)与现有的精确 Disjoint 查询处理算法进行对比,评价 DQPIC 算法的性能方面的优劣.

5.1 实验环境及数据样本

本文实验中的算法都用 C 语言实现，运行在虚拟机的 Linux 环境中. 电脑配置是 Intel(R) core (TM) i5 CPU 650@3.20 GHz CPU，虚拟机的版本是 vmware5.5.3，虚拟机上 Linux 使用的是 redhat2.4.20. 实验分别使用真实和人工合成数据验证算法性能. 具体来说真实数据使用了 SST，人工合成数据使用了文献[1]中的 Maskedchirp. SST 是太平洋海洋环境实验室测定的海面温度变化序列，共 61 437 个观测数据，每个数据元组有 4 个属性，本部分实验中只用温度属性；Maskedchirp 是由含有白噪声的不连续正弦波组成，其中任何 2 个不连续正弦波的周期都不相同，整个形式与真实的声音数据相类似，共有 20 000 个数据点. 实验中查询序列通过抽取数据样本中任意长度序列得到，而与 Maskedchirp 相关的查询序列在文献[1]中可获得. 为了简化实验，我们使用了文献[1-2]以及文献[9,16,22]中数据流模似方式，即将样本读入内存，然后顺序访问每个数据，从而模拟数据流场景. 所有数据都是 1 维，实验执行 10 次，结果取平均值.

5.2 DQPIC 算法的有效性

本部分实验中，因为查询结果数目较大，为了便于分析，这里定义了 DQPIC 算法相对于 naive 算法的查全率与查准率，我们分别用 R_{recall} 与 $R_{\text{precision}}$ 表示. 设 S_{Dresult} 与 S_{Nresult} 分别为相同参数设置下，DQPIC 和 naive 算法的执行结果集合，$|S_{\text{Dresult}}|$ 与 $|S_{\text{Nresult}}|$ 分别是这 2 个集合中成员数目，令 V_{equal} 表示两个集合中相等查询结果的个数. 这里定义 S_{Dresult} 中的任一查询结果 A 与 S_{Nresult} 中的查询结果 B，如果它们对应的窗口具有相同起始位置，且 A 与 B 也相同，则认为 A 与 B 相等；如果两者中有 1 项不同，则 A 与 B 不相等. 这样 R_{recall} 与 $R_{\text{precision}}$ 可表示为

$$R_{\text{recall}}=V_{\text{equal}}/|S_{\text{Nresult}}| \tag{6}$$

$$R_{\text{precision}}=V_{\text{equal}}/|S_{\text{Dresult}}| \tag{7}$$

显然如果 DQPIC 算法 R_{recall} 与 $R_{\text{precision}}$ 都为 1，则说明 DQPIC 与 naive 算法在每个窗口上执行结果相同. 实验参数设置与执行结果如表 2 所示(其中标出的 2 行也是性能评价实验中的参数取值).

表 2　实验 1 中的相关参数设置及 *DQPIC* 算法的 R_{recall} 与 $R_{\text{precision}}$

数据样本	样本大小	SW_{size}	$\|EW\|$	*dimension*	*epsilon*	$\|Y\|$	R_{recall}	$R_{\text{precision}}$
SST	**61437**	**25**	**200**	**1**	**1000**	**3000**	1	1
SST	61437	30	200	1	1500	3000	1	1
SST	61437	35	200	1	2000	3000	1	1
SST	61437	40	200	1	2500	3000	1	1
Maskedchirp	**20000**	**20**	**200**	**1**	**100**	**2048**	1	1
Maskedchirp	20000	25	200	1	200	2048	1	1
Maskedchirp	20000	30	200	1	300	2048	1	1
Maskedchirp	20000	35	200	1	400	2048	1	1

由表 2 可见在这些数据样本及相关参数设置下，DQPIC 算法查准率与查全率值都是 1，即 DQPIC 与 naive 算法结果相同，也即在 1 维数据中，DQPIC 算法能准确得到数据流上 DQ-SW 查询结果.

5.3 DQPIC 算法的性能评价

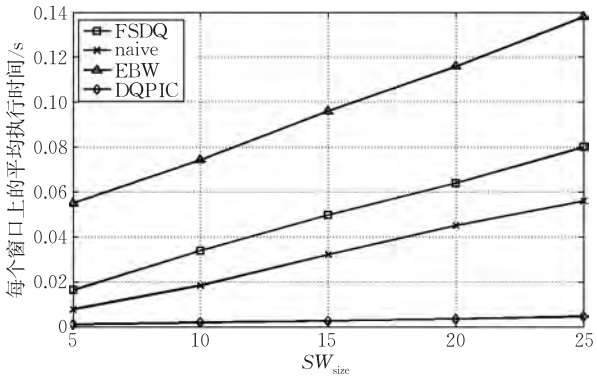
就现有研究来看能用于解决 DQ-SW 查询的研究并不多. 通过分析发现将文献[3]中算法用于每个滑动窗口可得到解决本文问题的算法，称为 EBW (eHaarSubseq Based on Window)算法；文献[9]中 FSDQ 算法与本文要解决问题相同. 它们与 naive 算法一起将作为 DQPIC 算法性能评价的对比算法.

5.3.1 DQPIC 算法运行时间分析

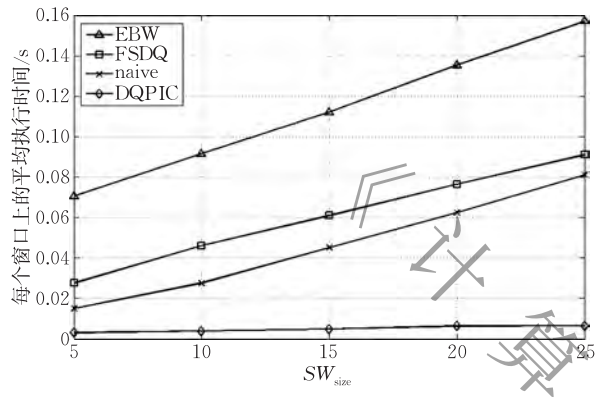
本部分实验使用了 SST 与 Maksedchirp 样本，具体实验分析如下：

首先分析 SW_{size} 对算法执行时间的影响. 实验中参数取值见表 2，实验结果如图 8 所示.

由图 8 中不同样本下几种算法的执行时间比较可以看出，几种算法的执行时间随 SW_{size} 的增加单调递增. 这主要因为窗口越大，需要处理的数据流成员就越多，对于 DQPIC 算法，由 4.2.4 节的分析可知，在该算法的时间开销中，$O(SW_{\text{size}}\times|EW|\times|Y|)/M$ 与 $O(|Y|\times T+P+|EW|\times|Y|)$ 部分都会因为 SW_{size} 大小的增加而变大，所以算法的时间开销会增大，而图中 DQPIC 算法时间所呈现出的变化状态，也印证了这一点. 另外由图中还可以看到，EBW 算法执行时间最多，这是因为在当前参数下，查询序列成员取值基本涵盖了窗口中所有数据流成员取值的最大与最小值，这使得 EBW 算法对于数据流中与查询序列不相似的序列的剪枝能力减弱，这样基本需要对大部分数据流成员执行 Spring 算法，所以执行时间较大；naive 算法执行时间小于 FSDQ 算法，这是因为虽然 FSDQ 算法使用了边界



(a) Maskedchirp



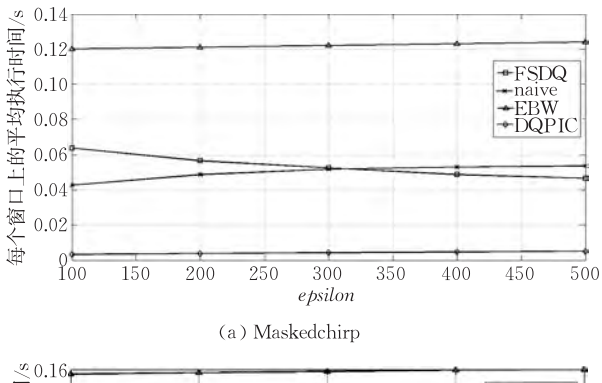
(b) SST

图 8 算法在 SW_{size} 发生变化时的执行时间对比

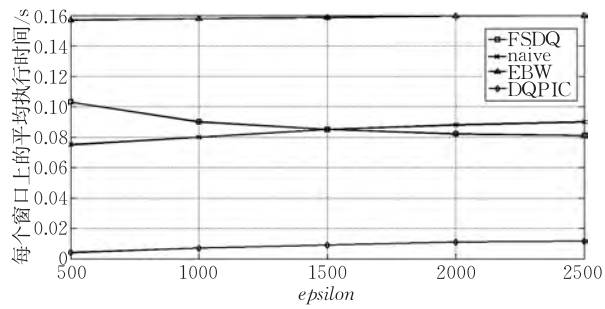
路径技术,但该算法对于创建元素之前的成员都使用 Spring 算法来处理,另外该算法也忽略了在使用 Spring 算法处理相邻两个窗口中位于创建元素之前相同数据流成员时存在的冗余运算,所以时间开销较大;另外由图中可以看出 DQPIC 算法时间开销最小,这也证明了 DQPIC 算法相对于其他算法的时间优势,说明了该算法的时间有效性.

分析相似阈值 *epsilon* 对于算法执行效果的影响. 实验中参数取值见表 2,实验结果如图 9 所示.

由图 9 可以看出 EBW,naive,DQPIC 这 3 种算法的执行时间随 *epsilon* 取值的增加而单调增加,而 FSDQ 算法的执行时间随 *epsilon* 取值的增加单调减少. 对于 EBW 算法,因为该算法具体仍然是用 Spring 算法来执行相似性查询的具体处理工作,当 *epsilon* 较小时,在构建窗口上 ***STWM*** 的过程中,多数列向量最后一个成员 d 值不小于 *epsilon*,所以不需要进一步判断该成员所在 DTW 路径对应的数据流序列是否是查询结果;随着 *epsilon* 的增加,满足该条件的成员越来越多,相应地也增加了很多判别操作,执行时间增加;对于 naive 算法,因为 *epsilon* 的增加,会使得在确定窗口上 ***STWM*** 中所有向量



(a) Maskedchirp



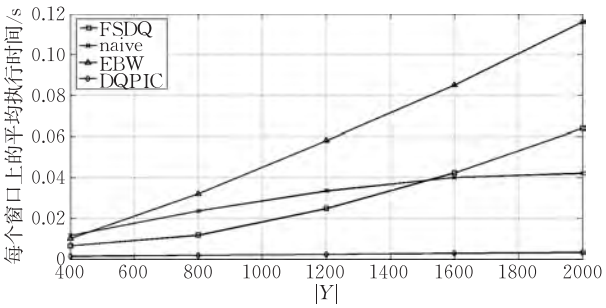
(b) SST

图 9 算法在 *epsilon* 发生变化时的执行时间对比

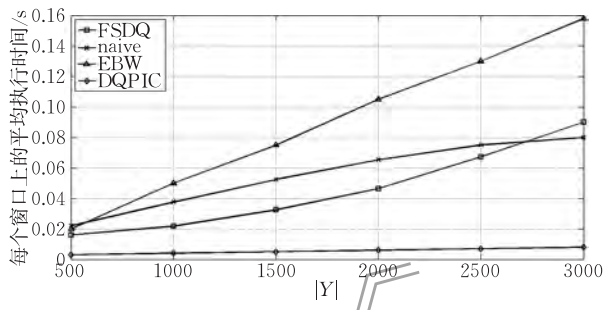
lowestbound 和 *highestbound* 的时候,计算更多成员值,所以执行时间增加;FSDQ 算法中,*epsilon* 的增加产生更多 DQ-SW 查询结果不为空的窗口,所以更多窗口上的 FSDQ 算法是利用边界路径来减少窗口上 naive 算法中存在的冗余运算,而由 FSDQ 算法可知,对于一个确定窗口,基于边界路径所导致的后一个窗口上 naive 算法运行时间的减小量大于等于基于伪边界路径所导致的后一个窗口上 naive 算法运行时间的减少量,所以 FSDQ 算法执行时间会减少. 对于 DQPIC 算法,考虑对于相邻 2 个窗口相同数据流成员的处理,如果 *epsilon* 取值的增加并没有改变该值变化前数据流中满足定理 2、3 的成员位置,显然在此时 4.2.4 节关于时间开销的分析中,T 部分的开销不会变化,如果 *epsilon* 取值的增加使该值变化前数据流中满足定理 2、3 中判别条件的成员位置发生了变化,则需要在之后的相同数据流成员中继续寻找这样的成员,显然此时,T 的开销会增加,也就是说随着 *epsilon* 取值的增加,T 部分的开销会单调增加,另外在使用 naive 算法处理相同数据流成员的过程中,时间开销也会增加,所以总体来看,时间开销会增加.

分析查询序列长度$|Y|$对于算法执行效果的影响,实验中参数取值见表 2,实验结果如图 10 所示.

由图 10 可以看到,几种算法的执行时间都会随着查询序列长度的增加而单调增加. 这是因为查询



(a) Maskedchirp

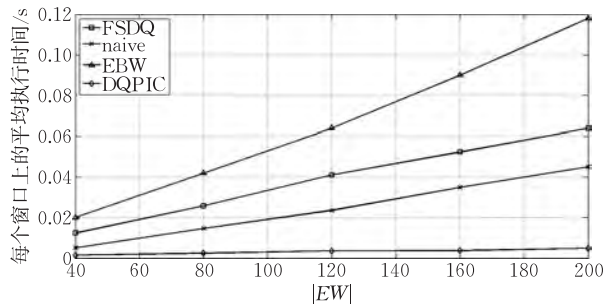


(b) SST

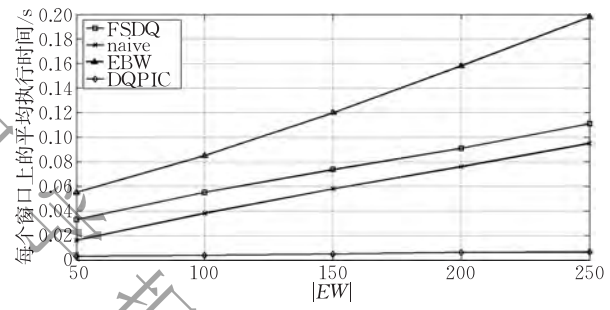
图 10　算法在查询序列长度发生变化时的执行时间对比

序列长度的增加，会使每个窗口上构建的 ***STWM*** 中行数增加，所以多出了很多相关的计算量；另外由图中我们也可以看到 EBW 算法的时间增长较快，这是因为该算法中所使用的 Spring 算法需要对 ***STWM*** 中每个新增加的成员进行处理；naive 算法的执行时间增加减慢，因为对于 Maskedchirp 样本的实验，当查询序列长度在 400～1600 间变化时，***STWM*** 中的多数列向量的 *highestbound* 都在变化(SST 实验中 Y 的长度在 200～2700 间变化，情况相同)，而由 FSM 算法可知，随着 $|Y|$ 的增加，***STWM*** 中列向量 *highestbound* 的位置只会上升，不会下降，所以基于 FSM 算法可推得，此时 naive 的算法时间增加较明显；而当 $|Y|$ 取值继续增加时，***STWM*** 中列向量的 *highestbound* 多数没有发生变化，所以此时 naive 算法的执行时间增加较为平缓；对于 FSDQ 算法，在查询序列相对较小时，所有窗口中能产生查询结果的较多，该算法在这些窗口上会基于边界路径来减少窗口上 Spring 算法中存在的冗余运算，而随着查询序列的增加，产生查询结果的窗口变少，该算法会在不产生查询结果的窗口上基于伪边界路径技术来减少窗口上 Spring 算法中存在的冗余运算. 因为 FSDQ 算法中使用边界路径技术比使用伪边界路径技术能更好地消除 naive 算法中的冗余运算，所以随着查询序列长度增加，FSDQ 算法对冗余运算的消除效果变差，算法时间增加变快；由图中还可以看到 DQPIC 算法执行时间最小，且随查询序列长度的增加而增加，我们可以结合 4.2.4 节时间开销来分析，因为 4.2.4 节中 $O(SW_{size}\times|EW|\times|Y|)/M$ 部分会增加，而 $O(|Y|\times T+P+|EW|\times|Y|)$ 部分也会因查询序列 $|Y|$ 的增加而增加，所以总体时间开销会增加，另外由图中也可以看到 DQPIC 算法的时间明显小于其他算法，这也说明了这种算法相对于其他算法的时间优势.

分析单元窗口大小 $|EW|$ 对于算法执行时间的影响. 实验中参数取值见表 2，结果如图 11 所示.



(a) Maskedchirp



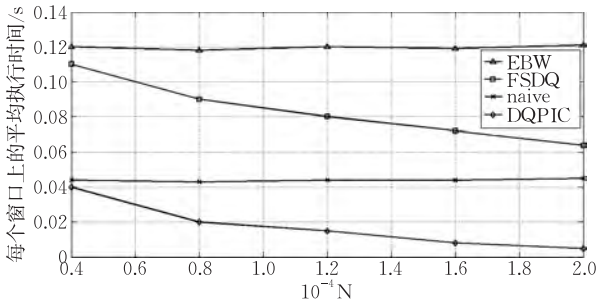
(b) SST

图 11　算法在 $|EW|$ 发生变化时的执行时间对比

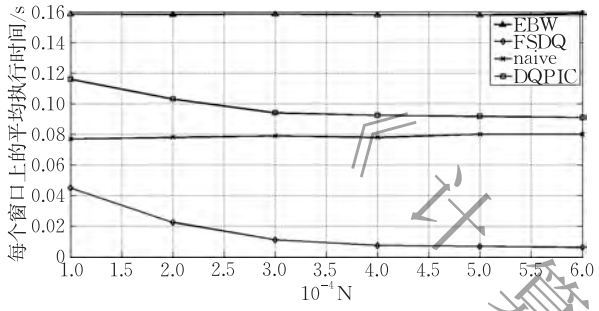
由图 11 可知，随着基本窗口大小 $|EW|$ 的增加，几种算法时间都呈单调增加趋势. 这是因为在窗口大小不变的情况下，基本窗口越大，窗口中数据流成员就越多，算法在处理这些数据流成员的时候所需的时间也就越多. EBW 算法中的 Spring 算法因为需要处理窗口 ***STWM*** 中每个成员，所以算法时间增加较快；FSDQ 与 naive 算法分别因为边界路径技术以及边界技术的使用，导致算法时间增加较慢；结合 4.2.4 节关于 DQPIC 算法时间开销的分析可知，当 $|EW|$ 增加时，$O(SW_{size}\times|EW|\times|Y|)/M$ 部分会增加，而 $O(|Y|\times T+P+|EW|\times|Y|)$ 中的 P 和 T 都会随 $|EW|$ 大小的增加而增加，所以总体来看 DQPIC 算法时间开销会增加，这一点由图中 DQPIC 算法时间呈现出的变化态势得以印证，另外也可以看到 DQPIC 算法时间增加最为平缓，同时在几种算法中时期开销也最小，这也证明了 DQPIC

算法相对于其他算法的时间有效性.

分析样本大小 N 对于算法执行时间的影响. 实验中参数取值见表 2,结果如图 12 所示.



(a) Maskedchirp



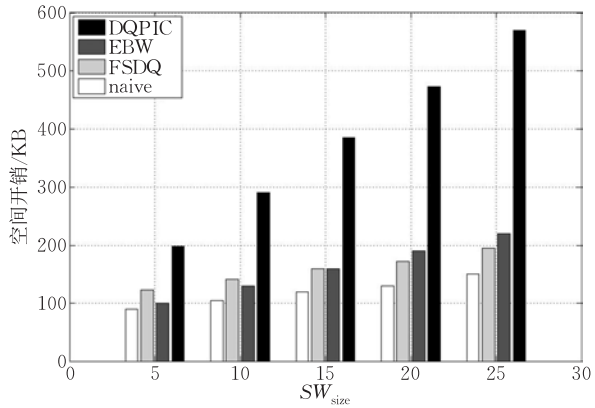
(b) SST

图 12 算法在样本大小 N 发生变化时的执行时间对比

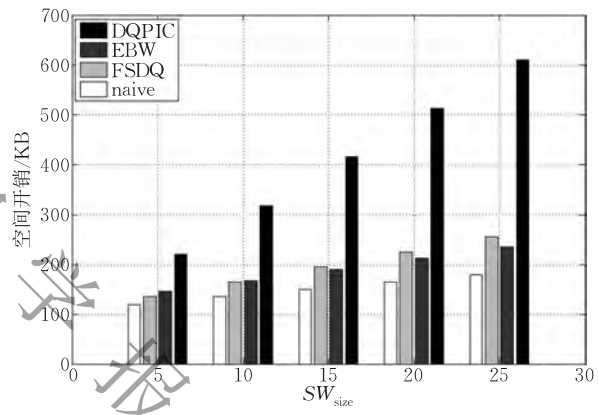
由图 12 可知,随着样本大小 N 的增加,EBW 以及 naive 算法的执行时间并不会发生变化;而 FSDQ 与 DQPIC 算法的执行时间都会变小. 对于 EBW 和 naive 算法来说,它们在每个窗口上的执行时间在 *epsilon* 取值不变的时候,只与窗口大小与查询序列长度相关,而这两个方面并不会随着样本大小的增加而发生变化,所以时间开销保持不变;对于 FSDQ 算法,因为在 N 较小时(Maskedchirp 中 N 为 4000 时,或 SST 中 N 为 10000 时),此时窗口数较少,而且由 FSDQ 可知,该窗口会使用 Spring 算法来处理第一个窗口,该窗口的时间效应没有被其他窗口削减太多,所以与 EWB 算法较为接近,而随着数据样本的增大,窗口数会越来越多,再加上边界界标窗口技术的作用,平均每个窗口上的执行时间会渐渐减少;同样对于 DQPIC 算法,因为在 N 较小时(Maskedchirp 中 N 为 4000 时,或 SST 中 N 为 10000 时),此时窗口数较少,第一个窗口上执行情况与 naive 算法相同,都是使用 FSM 算法来处理,与 FSDQ 算法一样,DQPIC 算法与 naive 算法时间较为接近. 随着样本大小增加,窗口个数也会增加,$O(SW_{size}\times|EW|\times|Y|)/M$ 部分时间开销会减少,而因为 $O(|Y|\times T+P+|EW|\times|Y|)$ 部分随样本大小增加,并无太大变化,所以总体时间开销会减少.

5.3.2 DQPIC 算法空间开销分析

首先分析 SW_{size} 对算法空间开销的影响. 实验参数取值见表 2,实验结果如图 13 所示.



(a) Maskedchirp



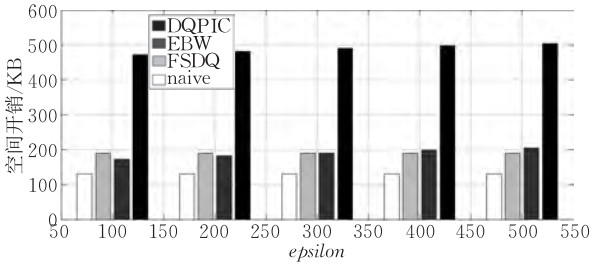
(b) SST

图 13 算法分别在 SW_{size} 发生变化时的空间开销对比

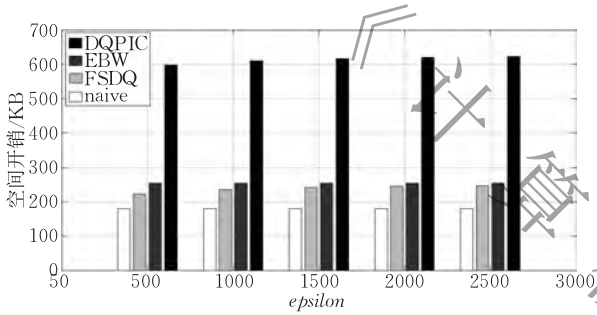
由图 13 可见不同样本下几种算法空间开销都随窗口大小的增加而增加. 这是因为窗口越大,需要开辟出存放窗口成员的空间就越大;FSDQ 算法因为需要保护现场,保留当前窗口上的查询结果和边界路径(伪边界路径),所以比 naive 算法需要更多的空间开销;EBW 算法除了保留当前窗口上的数据流成员和查询序列外,还需要保留当前窗口上可能含有查询结果的成员序列,而因为在当前参数下,整个窗口中大部分序列都含有查询结果,所以窗口中大部分成员都被保留下来,这也是 EBW 算法空间开销多于 naive 算法的原因;由图中还可以看到随着 SW_{size} 的增加,EBW 算法空间开销渐渐多于 FSDQ 算法,这是因为 EBW 算法中被保留下来的窗口序列会随窗口大小的增加而增加,而相对来说,FSDQ 算法空间开销部分在这个过程中只有当前窗

口上的查询结果会发生变化，但这个变化幅度并不大；DQPIC 算法与 FSDQ 算法相比，虽然不用保留边界路径(伪边界路径)，但却需要更多空间来存放每个窗口 ***STWM*** 中所有列向量最后一个成员的相关参数值，所以空间开销更大.

接着分析 *epsilon* 对算法空间开销的影响. 实验参数取值见表 2，实验结果如图 14 所示.



(a) Maskedchirp

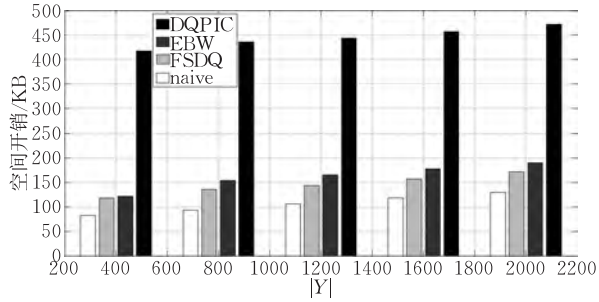


(b) SST

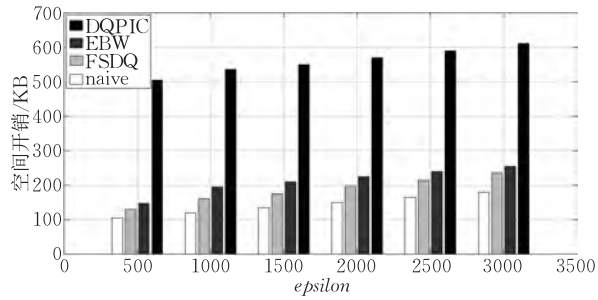
图 14　算法在 *epsilon* 发生变化时的空间开销对比

由图 14 可见，naive 算法空间开销在 *epsilon* 取值增加的过程中保持不变，这是因为 naive 算法只保留每个窗口上数据流成员、查询序列以及 ***STWM*** 中当前列向量与前一个列向量的 *lowestbound* 和 *highestbound*，而这些空间开销与 *epsilon* 取值都不相关；与 naive 算法相比，EBW 算法需要额外空间存放当前窗口上可能含有查询结果的成员序列，所以空间开销多于 naive 算法；另外由图中可见 EBW 算法空间开销在 *epsilon* 取值增加的过程中基本保持不变，这是因为在当前参数下，窗口上几乎全部成员都含有候选查询结果，所以在 *epsilon* 取值增加的过程中，并不会有更多含有候选查询结果的序列被保留；由图 14 还可以看到，DQPIC 算法与 FSDQ 算法的空间开销随着 *epsilon* 取值的增加而增加，这是因为这两种算法都需要保留每个窗口上 DQ-SW 查询的结果. 而随着 *epsilon* 取值的增加，每个窗口上的查询结果逐渐增加，所以这两种算法的空间开销也相应增加. 图中 FSDQ 算法相对于 EBW 算法空间开销的减少幅度随 *epsilon* 取值的增加渐渐变小，直至反超，这也是因为随着 *epsilon* 取值的变大，每个窗口上的查询结果都会逐渐增加.

分析 $|Y|$ 对于算法空间开销的影响. 实验中参数取值见表 2，实验结果如图 15 所示.



(a) Maskedchirp

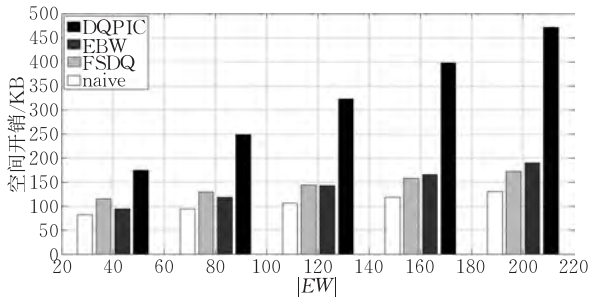


(b) SST

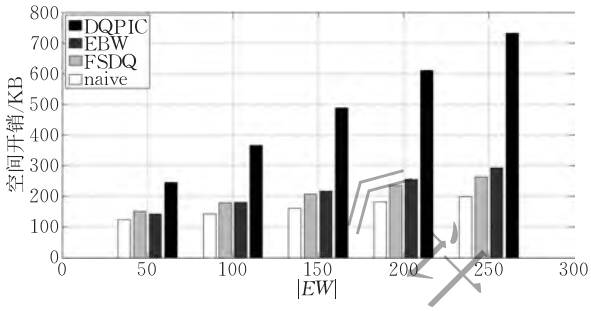
图 15　算法在 $|Y|$ 发生变化时的空间开销对比

由图 15 可见几种算法的空间开销随 $|Y|$ 的增加而增加. 这是因为 $|Y|$ 越大，总会使系统中存放它们的空间变大，所以会有这样的效果. 对于 FSDQ 与 DQPIC 算法，因为它们相对于 naive 算法需要保存更多信息，所以空间开销多于 naive 算法. 另外对于 FSDQ 与 DQPIC 算法，因为 DQPIC 相对于 FSDQ 算法还需要保留每个窗口 ***STWM*** 所有列向量索引为与之相关 *hbound* 的成员的相关参数，以及这些向量被处理后可能产生的查询结果，所以 DQPIC 算法的空间开销大于 FSDQ 算法. EBW 相对于 FSDQ 算法空间开销的增加幅度会先增加后减少，这是因为在开始的时候，随着 $|Y|$ 的增加，EBW 算法需要保留窗口中的候选查询结果序列会越来越多，当 $|Y|$ 的增加使得整个窗口序列中成员都含有候选查询结果序列时，这时 $|Y|$ 继续增加对算法空间开销的影响仅限于查询序列的存放；而随着查询序列长度的增加，FSDQ 算法的空间开销不仅有查询序列的存放，还有“保护现场”需要保留的当前窗口的最后一个成员在当前 ***STWM*** 中对应的列向量，所以应该是查询序列的 2 倍，所以随着 $|Y|$ 的增加，FSDQ 算法的空间开销增加幅度会越来越大，相对于 EBW 算法空间开销的差距也会越来越小.

分析|*EW*|对于算法空间开销的影响.实验中参数取值见表2,实验结果如图16所示.



(a) Maskedchirp



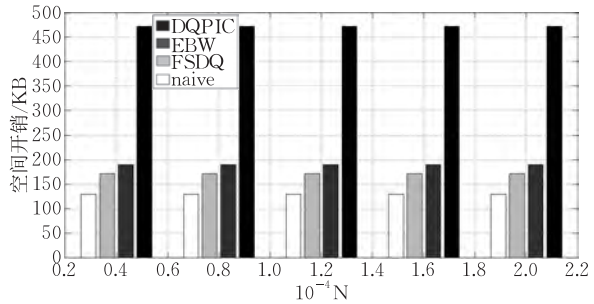
(b) SST

图16 算法在|*EW*|发生变化时的空间开销对比

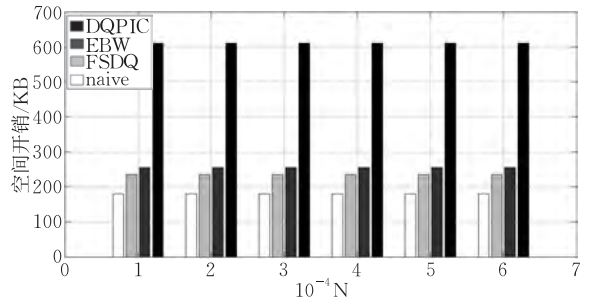
由图16可见,几种算法空间开销在|*EW*|取值增加的过程中严格单调增加.这是因为|*EW*|值越大,则窗口中存放的数据流成员就越多,系统为了存放它们也就需要开辟更多的空间.因为FSDQ算法相对于naive算法多出了保留当前窗口上查询结果,边界路径(伪边界路径)以及"保护现场"的空间开销,所以FSDQ算法的空间开销多于naive算法;由图中还可以看到随着|*EW*|的增加,EBW算法的空间开销渐渐多于FSDQ算法,这是因为EBW算法中被保留下来的窗口序列会随窗口成员的增加而增加,而相对来说,FSDQ算法的空间开销部分在这个过程中只有当前窗口上的查询结果会发生变化,但这个变化幅度并不太大;DQPIC算法相对于FSDQ算法不需要保留边界路径(伪边界路径),但却需要额外空间保留当前窗口上 ***STWM*** 中索引为与之相关*hbound* 的成员的相关参数,以及这些向量被处理后可能产生的查询结果,所以DQPIC算法的空间开销大于FSDQ算法.

分析样本大小N对于算法空间开销的影响,实验中参数取值见表2,实验结果如图17所示.

由图17可见,几种算法空间开销随 N 的增加,变化幅度并不大.对于naive算法,N 的增加不会影响到每个窗口上的查询序列长度与窗口中的成员个数;对于EBW算法,因为当前参数下窗口所有成员



(a) Maskedchirp



(b) SST

图17 算法在 N 发生变化时的空间开销对比

中都含有候选查询结果序列,所以数据样本大小的增加并不会影响到算法的空间开销;对于FSDQ与DQPIC算法,因为查询结果所占空间相对于其他空间开销部分较小,所以它们的空间开销主要决定于每个窗口上数据流成员个数,查询序列长度,以及相关参数,而这几部分在 N 增加的过程中并没有太大变化,所以空间开销也有这样的表现.

5.3.3 DQPIC算法的时间与空间开销对比分析

设 $At_{para,i}$ 与 $Bt_{para,i}$ 分别是参数 *para* 取第 i 个样本点时,算法DQPIC与其他算法执行时间,于是我们可以把DQPIC算法相对于其他算法在 *para* 取值变化时的时间平均减少率(Average Rate about the Decrement of Time,ARDT)定义为

$$ARDT=\left(\sum_{i=1}^{n}\left(|At_{para,i}-Bt_{para,i}|/Bt_{para,i}\right)\right)/n \quad (8)$$

同理,设 $As_{para,i}$ 与 $Bs_{para,i}$ 分别是参数 *para* 取第 i 个样本点的时候,DQPIC算法与其他算法的空间消耗,则DQPIC算法相对于其他算法在 *para* 取值变化时的空间平均增加率(Average Rate about the Increment of Space,ARIS)可定义为

$$ARIS=\left(\sum_{i=1}^{n}\left(|As_{para,i}-Bs_{para,i}|/Bs_{para,i}\right)\right)/n \quad (9)$$

使用式(8)和式(9)对实验数据进行处理,可以得到DQPIC算法相对于其他算法在不同参数下的 $ARDT$ 与 $ARIS$,这里我们用 $ARIS_{\text{algorithma}}$ 和 $ARDT_{\text{algorithma}}$ 分别表示,DQPIC算法相对于算法algorithma的 $ARIS$ 与 $ARDT$,具体结果如表3所示.

表 3　DQPIC 算法相对于其他算法在不同参数下的 *ARDT* 与 *ARIS*

数据样本	参数	$ARIS_{naive}$	$ARDT_{naive}$	$ARIS_{FSDQ}$	$ARDT_{FSDQ}$	$ARIS_{EBW}$	$ARDT_{EBW}$
Maskedchirp	SW_{size}	2.12	0.90	1.35	0.93	1.33	0.96
	epsilon	2.76	0.92	1.59	0.92	1.57	0.95
	$\|Y\|$	3.27	0.92	2.09	0.88	1.79	0.94
	$\|EW\|$	1.96	0.84	1.19	0.91	1.22	0.94
	N	2.63	0.60	1.74	0.81	1.48	0.84
SST	SW_{size}	1.69	0.87	1.12	0.91	1.07	0.94
	epsilon	2.41	0.89	1.57	0.90	1.41	0.94
	$\|Y\|$	3.01	0.90	2.10	0.85	1.69	0.94
	$\|EW\|$	1.92	0.91	1.27	0.93	1.17	0.94
	N	2.39	0.78	1.59	0.83	1.39	0.89

由表 3 不同参数下基于 Maskedchirp 与 SST 样本的实验结果可以看到，DQPIC 相对于 naive 算法能以增加 1.69～3.27 倍的空间开销，换来高达 60%～92%的时间减少幅度，换句话说能够实现 2.5～12.5 倍时间效率的提高；DQPIC 相对于 FSDQ 算法能以增加 1.12～2.1 倍的空间开销，实现 5.26～14.2 倍时间效率的提高；DQPIC 算法相对于 EBW 算法，能以 1.17～1.79 倍的空间开销的增加，实现 6.26～25 倍时间效率的提高. 通过对 DQPIC 算法在不同参数下的时间与空间性能变化范围取最大，最小值可得 DQPIC 算法增加了空间开销 1.12～3.27 倍，但却因此而换得了算法在执行时间效率上 2.5～25 倍的提高. 具体来说，DQPIC 算法时间在窗口大小增加过程中平均时间效率提高幅度可达 25 倍之多，这是因为窗口越大，DQPIC 算法能省略的冗余运算就越多，这点也可由 4.2.4 节时间复杂度的分析与 5.3.1 节实验 1 的分析可知；而 DQPIC 算法时间在样本大小增加过程中平均时间效率提高为 2.5 倍，这是因为 DQPIC 算法能减少的冗余运算量会随着样本大小的增加而变得越来越多，因为在我们实验中样本大小的变化幅度不太大，所以时间效率的提升幅度也并不太大.

总之，由上面仿真实验可以得出：(1) DQPIC 算法与 naive 算法执行结果相同，能够完整而准确地得到数据流 DQ-SW 查询的结果；(2) 相比于现有其他算法，DQPIC 算法增加了空间开销(1.12～3.27 倍)，但却因此换得了算法在执行时间效率上大幅度提高(2.5～25 倍). 总体来看，DQPIC 算法是一种处理数据流 DQ-SW 查询的有效方法，在对时间效率更重视的应用场景，相对于其他算法有时间优势，特别是在窗口较大的无限数据流 DQ-SW 查询应用场景中，算法时间优势最好.

6　总结及展望

本文对滑动窗口下不具有全局约束机制的数据流精确 Disjoint 查询处理问题进行了深入地研究. 首先提出了 DQ-SW 查询的概念；接着针对该查询提出了一种增量处理算法 DQPIC，该算法可以减少 naive 算法执行过程中因为不具有增量计算的特点而产生的冗余运算；最后通过对公用数据样本的实验证明了该算法的有效性. 接下来的研究会从如下几个方向入手：(1) 继续改进 DQPIC 算法，主要目的是减少该算法的空间开销；(2) 对于全局约束机制下 DQ-SW 查询的处理问题展开研究，分析基于 DQPIC 算法思想处理这类查询问题的可行性和相关条件；(3) 将该算法扩展到多维数据流环境中，验证算法在多维数据流环境下的有效性.

参　考　文　献

[1] Sakurai Y, Faloutsos C, Yamamuro M. Stream monitoring under the time warping distance//Proceedings of the 23rd International Conference on Data Engineering. Istanbul, Turkey, 2007: 1046-1055

[2] Zou P, Su L, Jia Y. Fast similarity matching on data stream with noise//Proceedings of the 24th International Conference on Data Engineering. Cancun, Mexico, 2008: 194-199

[3] Kashyap S, Lee M-L, Hsu W. Similarity subsequence search in time series databases//Proceedings of the 2011 International Conference on Database and Expert Systems Apllications. Toulouse, French, 2011: 232-246

[4] Niennattrakul V, Wanichsan D, Ratanamahatana C-A. Accurate subsequence matching on data stream under time warping distance//Proceedings of the Advances in Knowledge Discovery and Data Mining. Bangkok, Thailand, 2009: 752-755

[5] Rodpongpun S, Niennattrakul V, Ratanamahatana C-A. Efficient subsequence search on streaming data based on time warping distance. ECTI Transactions on Computer and Information Technology, 2011, 5(1): 2-8

[6] Wang Z-L, Huang H-T, Wang L-J. Accelerating subsequence similarity search based on dynamic time warping distance with FPGA//Proceedings of the 2013 ACM/SIGDA International Symposium on Field Programmable Gate Arrays. Monterey, USA, 2013: 53-62

[7] Athitsos V, Papapetrou P, Potamias M. Approximate embedding-based subsequence matching of time series// Proceedings of the ACM SIGMOD International Conference on Management of Data. New York, USA, 2008: 365-378

[8] Zhou X-Z, Li S-P, Yan Z, et al. A narrowest parallel line based approach for data stream similarity detection. International Journal of Advancements in Computing Technology, 2012, 4(8): 145-154

[9] Wang Shao-Peng, Wen Ying-You, Li Zhi, et al. A fast processing algorithm on section Disjoint query of data stream. Journal of Computer Research and Development, 2014, 51(5): 1136-1148(in Chinese)
(王少鹏，闻英友，李志等. 一种关于数据流区间 Disjoint 查询的快速处理算法. 计算机研究与发展，2014，51(5)：1136-1148)

[10] Gong X-Y, Si Y-W, Fong S-M, et al. NSPRING: Normalization-supported SPRING for subsequence matching on time series streams//Proceedings of the 15th IEEE International Symposium on Computational Intelligence and Informatics. Budapest, Hungary, 2014: 373-378

[11] Ding H, Trajcevski G, Scheuermann P, et al. Querying and mining of time series data: Experimental comparison of representations and distance measures. The VLDB Endowment, 2008, 1(2): 1542-1552

[12] Rakthanmanon T, Campana B, Mueen A, et al. Searching and mining trillions of time series subsequences under dynamic time warping//Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Beijing, China, 2012: 262-270

[13] Rakthanmanon T, Campana B, Mueen A, et al. Addressing big data time series: Mining trillions of time series subseuqneces under dynamic time warping. ACM Transactions on Knowledge Discovery from Data, 2013, 7(3): 10-40

[14] Huang S-T, Dai G-H, Sun Y-L, et al. DTW-based subsequence similarity search on AMD heterogeneous computing plateform//Proceedings of the 2013 IEEE International Confernece on High Performance Computing and Communications. Zhangjiajie, China, 2013: 1054-1063

[15] Hundt C, Schmidt S-E. CUDA-Accelerated alignment of subsequences in streamed time series data//Proceedings of the 2014 International Conference on Parallel Processing. Minneapolis, USA, 2014: 10-19

[16] Wu Feng, Zhong Yan, Wu Qing-Yuan, et al. Similarity search in data stream with adaptive segmental approximations. Journal of Software, 2009, 20(10): 2867-2884(in Chinese)
(吴枫，仲妍，吴清源等. 基于适应性分段估计的数据流相似性搜索. 软件学报，2009，20(10)：2867-2884)

[17] Zhou M, Wong M-H. Efficient online subsequence searching in data streams under dynamic time warping distance// Proceedings of the 24th International Conference on Data Engineering. Cancun, Mexico, 2008: 686-694

[18] Bui C-G, Duong T-A. Similarity search in multiple high speed time series streams under dynamic time warping// Proceedings of the 2nd National Foundation for Science and Technology Development Conference on Information and Computer Science. Ho Chi Minh City, Vietnam, 2015: 82-87

[19] Papadopoulos S, Cormode G, Deligiannakis A, et al. Lightweight authentication of linear algebraic queries on data streams//Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. New York, USA, 2013: 881-892

[20] Papadopoulos S, Cormode G, Deligiannakis A, et al. Lightweight query authentication on streams. ACM Transactions on Database Systems, 2014, 39(4): 30-75

[21] Herbst S, Pollner N, Tenschert J, et al. An alggebra for pattern matching, time-aware aggregates and partitions on relational data streams//Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems. Oslo, Norway, 2015: 140-149

[22] Li Jun-Kui, Wang Yuan-Zhen. Rewritable circular sliding window: Towards effective on-line data stream. Computer Science, 2007, 34(12): 51-55(in Chinese)
(李俊奎，王元珍. 可重写循环滑动窗口：面向高效的在线数据流处理. 计算机科学，2007，34(12)：51-55)

WANG Shao-Peng, born in 1984, Ph. D. candidate. His research interests include data stream, event stream and data mining.

WEN Ying-You, born in 1974, Ph. D., associate professor. His research interests include the next generation network, and wireless sensor networks.

ZHAO Hong, born in 1954, Ph. D., professor, Ph. D. supervisor. His research interests include network management and multimedia.

MENG Ying-Hui, born in 1984, Ph. D., lecturer. His research interests include wireless sensor network, sensor localization.

Background

Disjoint query over data stream plays an important role in many practical applications, such as firancial analysis, network monitoring, mobile services, and sensor network management. The aim of Disjoint query is to find out all subsequences which are similar to the given query sequence and not overlap with each other in the data stream, each of these subsequences is the local best.

However, few of the current relevant works focus on the Disjoint query based on sliding window over data stream. Although the methods based on existed studies can be taken to handle this problem, they are low efficiency, because the execution process over each window is independent to each other, and does not have the characteristic of incremental computations. In this paper, we addressed Disjoint query precise processing based on the sliding window without global warping constraints over data stream, which is known as DQ-SW query. After the relationship between query results of adjacent two windows is inferred by analyzing FSM algorithm execution processes over them, The DQPIC (DQ-SW query processing algorithm with incremental calculation) algorithm which is used to handle the DQ-SW query is proposed based on it, and has the characteristic of incremental calculations. The extensive experiments on common data sets SST and maskedchirp have evaluated the proposed algorithm and verified the validity of it. Compared with the current other algorithms, it can not only guarantee the accuracy of the query results, but also can improve the time efficiency by 2.5—25 times at the cost of increment of space cost by 1.12—3.27 times. In the application scenario which is related to the Disjoint query based on the larger sliding window over data stream, the proposed algorithm has better performance.

This work is supported by the National Natural Science Foundation of China (60903159, 61173153, 61402096, 61501405), the Special Fund from the Central Collegiate Basic Scientific Research Bursary (N110818001, N100218001, N130504007, N120104001), the Science and Technology Plan of Shenyang (1091176-1-00), and the National High Technology Research and Development Program of China (2015AA016005).

The authors of this paper have been engaged in the researches of the data processing, event processing and data mining over data stream. These studies are the key parts in all researches related to programs above and have laid the important theoretical basis and background knowledge, and the prototype system about data processing and event processing over data stream has also been constructed.