doi:10.3969/j.issn.1005-3026.2016.07.005

滑动窗口下数据流完全加权最大频繁项集挖掘

王少鹏[1],闻英友[1,2],赵 宏[1,2]
(1. 东北大学 信息科学与工程学院,辽宁 沈阳 110819;2. 东北大学 医学影像计算教育部重点实验室,辽宁 沈阳 110819)

摘 要:针对当前关于数据流加权最大频繁项集WMFI(weighted maximal frequent itemsets)的研究无法有效地处理频繁阈值和加权频繁阈值不一致情况下WMFI的挖掘问题,提出了完全加权最大频繁项集FWMFI(full weighted maximal frequent itemsets)的概念.为了减少naive算法在处理滑动窗口下完全加权最大频繁项集挖掘时存在的冗余运算,提出了FWMFI-SW(FWMFI mining based on sliding window over data stream)算法.所提出的算法通过基于频繁约束条件的优化策略减少了naive算法中MaxW优化策略的无效调用次数,采用编辑距离比率作为WMFP-SW-tree的重构判别函数,可以有效减少该树的重构次数.实验结果表明FWMFI-SW算法是有效的,且比naive算法更有时间优势.

关 键 词:数据流;滑动窗口;编辑距离比率;加权最大频繁项集;重构判别函数
中图分类号:TP 311 **文献标志码**:A **文章编号**:1005-3026(2016)07-0931-06

Mining Full Weighted Maximal Frequent Itemsets Based on Sliding Window over Data Stream

WANG Shao-peng[1], *WEN Ying-you*[1,2], *ZHAO Hong*[1,2]
(1. School of Information Science & Engineering, Northeastern University, Shenyang 110819, China; 2. Key Laboratory of Medical Image Computing, Ministry of Education, Northeastern University, Shenyang 110819, China. Corresponding author: WANG Shao-peng, E-mail: wangshaopeng1984@163.com)

Abstract: Aiming at the problem that none of current researches on the WMFI (weighted maximal frequent itemsets) over data stream emphasizes the WMFI mining on the condition that the frequent threshold is not equal with the weighted frequent threshold, the concept of FWMFI (full weighted maximal frequent itemsets) was firstly promoted in this work. In order to reduce redundant operations existing in the naive algorithm which is used to handle the FWMFI mining based on sliding window over data stream, the FWMFI-SW (FWMFI mining based on sliding window over data stream) algorithm was proposed. The mining optimization strategy was adopted based on the frequent character to reduce the unnecessary call about the MaxW optimization strategy in the naive algorithm. In addition, the edit distance ratio was taken as reconstruction judge function to decide whether the updated WMFP-SW-tree should be reconstructed as the window slides. The extensive experiments showed that the FWMFI-SW algorithm is effective, and outperforms the naive algorithm in running time.

Key words: data stream; sliding window; edit distance ratio; weighted maximal frequent itemsets; reconstruction judge function

挖掘数据流环境下的频繁项集是数据流挖掘领域中的重要内容[1-4].在当前有关数据流频繁项集挖掘的研究中,挖掘最大频繁项集比挖掘基本频繁项集和频繁闭项集更有效率,因为最大频繁项集个数更少,且隐含了所有频繁项集[5].但这类研究没有区分最大频繁项集中成员的重要

收稿日期:2015-05-11
基金项目:国家自然科学基金资助项目(60903159,61173153,61402096);中央高校基本科研业务费专项资金资助项目(N110818001,N100218001,N130504007,N120104001);沈阳市科技计划项目(1091176-1-00);国家高技术研究发展计划项目(2015AA016005).
作者简介:王少鹏(1984-),男,内蒙古乌兰察布人,东北大学博士研究生;赵 宏(1954-),男,辽宁沈阳人,东北大学教授,博士生导师.

性. 针对该问题产生了加权最大频繁项集[6-9]. Yun 等[6]首次研究了数据流加权最大频繁项集挖掘问题,使用了界标窗口模型;Lee 等[7]首次讨论了滑动窗口下数据流加权最大频繁项集挖掘问题,提出了 WMFP-SW 算法.

但这些研究存在一个共同问题,即项集加权频繁性与基本频繁性都基于同一个阈值判别,而实际中它们的判别阈值并不经常相同. 为解决这个问题,本文将频繁条件从原有加权频繁项集条件中独立出来,给出完全加权最大频繁项集(FWMFI,full weighted maximal frequent itemsets)概念,接着研究了滑动窗口下该项集挖掘问题. 为减少 naive 算法的冗余运算,提出了 FWMFI-SW(full weighted maximal frequent itemsets mining based on sliding window over data stream)算法,最后通过实验证明了算法的有效性.

1 相关知识

WMFP-SW 算法是 Lee 等提出的首个用于处理滑动窗口下数据流 FWMFI 挖掘的算法[7]. 该算法借助 WMFP-SW-tree, WMFP-SW-array 和 WMFP-tree 执行挖掘操作. WMFP-SW-tree 用于挖掘所有可能加权最大频繁项集,结构与 FP-tree 类似. 数据流到来时,算法先构造窗口的 WMFP-SW-tree,接着按由下而上顺序分别挖掘 WMFP-SW-tree 头表成员. 当借助 WMFP-tree 消除所有加权频繁项集间超集关系后,树的每个分枝都是 WMFP. WMFP-SW-array 用于减少构建条件 WMFP-SW-tree 时扫描条件模式基的次数.

2 基本概念和问题说明

2.1 基本概念

设 $IS=\{I_1,I_2,\cdots,I_m\}$ 是 m 个 item 项的集合,$TS=\{T_1,T_2,\cdots,T_n\}$ 是事务集,其中 T_i 为 IS 的子集.

定义 1 数据流 DS. 数据流 DS 是一个由连续到来的事务组成的事务序列,表示为 $DS=\{T_1,T_2,T_3,\cdots,T_i,\cdots\}$,$T_i$ 为第 i 个到来的事务.

定义 2 频繁项集. 任给项集 S,在 TS 中支持度为 TS 中包含 S 的事务个数与 TS 中事务总数 $|TS|$ 的比值,记为 $SUP(S)$. 若 $SUP(S)$ 大于等于频繁阈值 $\varepsilon(0\leqslant\varepsilon\leqslant1)$,则 S 为频繁项集.

定义 3 基本窗口 EW 与滑动窗口 SW. 基本窗口 EW 是一段时间内连续到达的数据流成员,$EW=\{T_i,T_{i+1},\cdots,T_j\}$ $(0<i\leqslant j)$. 滑动窗口 SW 对应一个连续相邻的 EW 序列,$SW=\{EW_1,EW_2,\cdots,EW_i,EW_j,\cdots\}$,其中 EW_i 和 EW_j 分别是 SW 中第 i 和第 j 个基本窗口,且 $\forall i,j$,如果 $i\neq j$,$EW_i\cap EW_j=\varnothing$. SW 的滑动步长为一个基本窗口,SW 大小为其中 EW 的个数,表示为 $|SW|$;$|EW|$ 是基本窗口中数据流成员个数.

定义 4 加权支持度[6-7]. 给定项集 $S=\{I_1,I_2,\cdots,I_l\}$,权值集合 $W=\{w_1,w_2,\cdots,w_l\}$,S 的加权支持度定义为 $\left(\sum_{i=1}^{l}w_i/l\right)\times SUP(S)$,记为 $WSUP(S)$. 本文中 w_i 满足 $0\leqslant w_i\leqslant1$. 实际中可以对所有 w_i 执行归一化处理,使其满足这一条件.

定义 5 完全加权最大频繁项集. 设 S' 是项集 S 的一个超集,S_{super} 是所有 S' 的集合. 给定频繁阈值 $\varepsilon(0\leqslant\varepsilon\leqslant1)$,加权阈值 $\Delta(0\leqslant\Delta\leqslant1)$,如果 S 满足如下条件,即为完全加权最大频繁项集:

① $SUP(S)\geqslant\varepsilon$ 且 $WSUP(S)\geqslant\Delta$;

② $\forall S^*\in S_{super}$,$WSUP(S^*)<\Delta$ 或 $SUP(S^*)<\varepsilon$.

2.2 问题说明

给定数据流 DS,频繁阈值 ε,加权阈值 Δ,基本窗口及滑动窗口大小 $|EW|$ 和 $|SW|$. 本文关注的是挖掘每个 SW 上数据流完全加权最大频繁项集.

3 FWMFI-SW 算法

3.1 naive 算法

由定义 5 及文献[6-7]可知,使用频繁条件过滤加权最大频繁项集能得到 FWMFI 集合,所以可基于 WMFP-SW 算法解决本文问题. 另外由 FP-growth 算法与频繁项集向下闭包性可知,该过滤操作可在 WMFP-SW 算法处理每个 WMFP-SW-tree 及条件 WMFP-SW-tree 头表成员时执行. 这样添加过滤条件的 WMFP-SW 算法被称为 naive 算法.

3.2 FWMFI-SW 算法

naive 算法头表中成员是否加入 prefix,决定于该成员在执行完基于频繁阈值和基于 $\max W$ 优化策略 2 个判别操作后结果是否为真. 但很多头表成员只需执行频繁判别操作就能确定,所以 naive 算法在这方面有冗余运算;另外,如果 naive 算法中窗口更新使原来 WMFP-SW-tree 头表中成员位置发生变化,则该树重构. 但由文献[10]可知,这种情况下树的编辑距离比率若小于重构阈值 $\gamma(0\leqslant\gamma\leqslant1)$,则无需重构. 所以 naive 算法在这方面也有冗余运算. 针对这 2 个问题做了如下工作:①提出基于频繁约束条件的优化策

略；②给出 WMFP – SW – tree 重构判别函数；③将①，②引入 naive，得到能减少冗余运算的 FWMFI – SW 算法.

3.2.1　基于频繁约束条件的优化策略

设 N 为窗口中数据流成员数，WS 是窗口 WMFP – SW – tree，树中项的权值范围为 $[w_h, w_l]$，item 项出现次数和权值分别为 item. count，item. weight，$S_l = \Delta / w_h$，$S_h = \Delta / w_l$.

性质 1　当 $\varepsilon < S_l$，①如果 item. count $< N \times S_l$，则不能加入 prefix；②如果 item. count $\geqslant N \times S_h$，则能加入 prefix；③如果 $N \times S_l \leqslant$ item. count $< N \times S_h$，只要 max$W$ 优化策略关于 item 的判别为真，则能加入 prefix.

①当 item. count $< N \times S_l$，因为 $S_l = \Delta / w_h$，故 $w_h \times$ SUP(item) $< \Delta$. 又因为 item. weight $\leqslant w_h$，故 item. weight $\times$ SUP(item) $< \Delta$，item 不能放入 prefix.

②当 item. count $\geqslant N \times S_h$，因为 $S_h = \Delta / w_l$，所以 $w_l \times$ SUP(item) $\geqslant \Delta$. 因为 item. weight $\geqslant w_l$，所以 item. weight $\times$ SUP(item) $\geqslant \Delta$. 又因为 item. count $\geqslant N \times S_h > \varepsilon$，所以 item 可以被放入 prefix.

③当 $N \times S_l \leqslant$ item. count $< N \times S_h$，此时 item. count $\geqslant N \times S_l > \varepsilon$ 成立，所以当 max$W$ 优化策略判别结果为真，该成员可放入 prefix. 证毕.

性质 2　当 $S_l \leqslant \varepsilon < S_h$，①如果 item. count $< N \times \varepsilon$，则不能加入 prefix；②如果 item. count $\geqslant N \times S_h$，则能加入 prefix；③如果 $N \times \varepsilon \leqslant$ item. count $< N \times S_h$，只要 max$W$ 优化策略判别结果为真，就能加入 prefix.

性质 3　当 $\varepsilon \geqslant S_h$，①如果 item. count $< N \times \varepsilon$，则不能加入 prefix；②如果 item. count $\geqslant N \times \varepsilon$，则能加入 prefix.

性质 2，3 证明与性质 1 相似. 不再赘述.

把基于性质 1 ~ 性质 3 优化 naive 算法的方法称为基于频繁约束条件的优化策略(OSF2C，optimization strategy based on frequent constraint condition).

3.2.2　重构判别函数 RJF(reconstruction judge function)

为减少 naive 算法中第 2 类冗余运算，采用文献[10]中编辑距离比率(edit distance proportion，EDP)作为 WMFP – SW – tree 重构判别函数. 每个窗口上 WMFP – SW – tree 是否执行重构，取决于该树 RJF 值是否小于重构阈值 γ($0 \leqslant \gamma \leqslant 1$). 下面给出 RJF 计算过程，其中 M 为窗口中所有项个数.

$$\mathrm{RJF} = \mathrm{EDP} = \sum_{k=1}^{M} \mathrm{abs}(d_k) / \sum_{k=1}^{M} \max(k-1, M-k)\ , \tag{1}$$

$$d_k = \mathrm{pos}_{t_n}(k) - \mathrm{pos}_{t_{n-1}}(k)\ , \tag{2}$$

$$\sum_{k=1}^{M} \max(k-1, M-k) = [M(M-1)/2] + \mathrm{floor}(M^2/4)\ . \tag{3}$$

3.2.3　FWMFI – SW 算法

将基于频繁约束条件的优化策略与关于 WMFP – SW – tree 的重构判别机制引入 naive 算法，得到解决本文所关注问题的 FWMFI – SW 算法.

算法 1　FWMFI – SW 算法

输入：DS，|SW|，|EW|，ε，Δ，γ

输出：完全加权最大频繁项集集合 P

1) T = Order = $H = \varnothing$，N = |EW| × |SW|；// T 是待建的 WMFP – SW – tree

2) While there are transactions to be processed in DS

3)　If $T = \varnothing$

4)　　naive. Create_Window(T, Order)；

5)　　naive. Restructure_Tree(T, Order)；

6)　　H = Order；

7)　Else

8)　　[Order, T] ←naive. Update_Window(T, Order)；

9)　　If (RJF(Order, H) > γ)

10)　　naive. Restructure _ Tree (T, Order)；

11)　　H = Order；

12)　　Else

13)　　Order = H；

14)　prefix = $P = \varnothing$；

15)　get the upper and low bound of the weight about T, w_h, w_l；

16)　$S_l \leftarrow \Delta / w_h$, $S_h \leftarrow \Delta / w_l$；

17)　IFMine_WMFP(T, prefix, P, N, ε, Δ, S_l, S_h).

首次执行的算法先构建窗口 WMFP – SW – tree，记下头表中项的排列顺序(行 3 ~ 6)；当窗口滑动时，先更新 WMFP – SW – tree 中成员(行 8)，然后基于 RJF 判断新 WMFP – SW – tree 是否重构(行 9 ~ 13)；在 WMFP – SW – tree 确定后，先求得 prefix，P，S_l，S_h，然后用 IFMine_WMFP 挖掘 FWMFI(行 14 ~ 17). 这里，IFMine_WMFP 算法与 naive 算法的 Mine _ WMFP 子算法基本相同，不同点在于：①对于 WMFP – SW – tree 头表成员，先用 OSF2C 优化策略进行处理，根据处理结果决定是否执行 max$W$ 优化策略；②在递归调用 IFMine_WMFP 算法前，先确定条件树所有项

权值的最大、最小以及 S_1 和 S_h 的取值.

3.2.4 FWMFI – SW 算法正确性分析

FWMFI – SW 与 naive 相比有 2 处不同:① FWMFI – SW 算法引入 WMFP – SW – tree 的重构判别函数;② IFMine_WMFP 子算法中引入基于频繁约束条件的优化策略. 前者不改变 naive 算法结果,因为由文献[10]可知是否重构只影响树的紧凑程度;由性质 1 ~ 3 可知,后者也不会改变 naive 算法结果.

图 1 是 FWMFI – SW 算法执行过程. 这里仍用文献[7]中例子,其中 $\Delta = 0.3$, $\varepsilon = 0.25$, $\gamma = 0.2$.

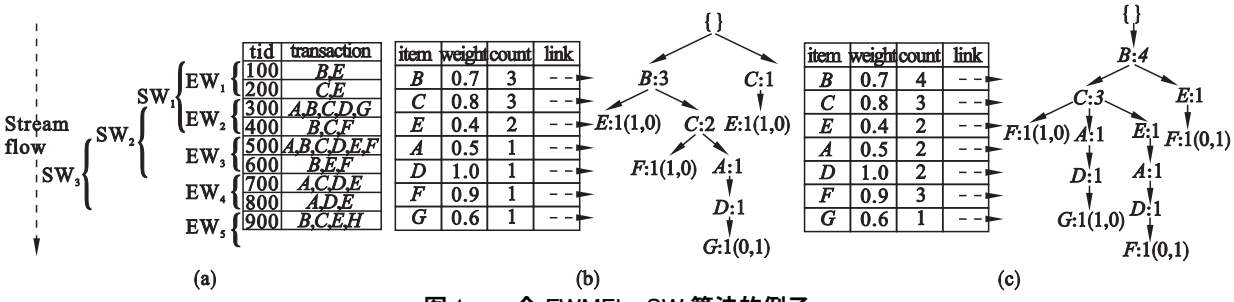

图 1 一个 FWMFI – SW 算法的例子

Fig. 1 A FWMFI – SW algorithm example

(a)—数据流及其上的滑动窗口;(b)—SW_1 上的 WMFP – SW – tree;(c)—SW_2 上没有重构的 WMFP – SW – tree.

图 1a 是数据流及窗口划分情况;图 1b 是 FWMFI – SW 算法处理 SW_1 中成员后得到的 WMFP – SW – tree. $S_1 = 0.3$, $S_h = 0.75$,由 OSF2C 可知 G, F, D, A 无需执行 max$W$ 优化策略. E 的 max$W$ 优化策略结果为 true,放入 prefix. 构建 E 的条件 WMFP – SW – tree,该树 $S_{11} = 0.375$, $S_{h1} = 0.42$. 由 OSF2C 可知 B, C 不在结果中. 又 WSUP(E) = 0.2 < 0.3,故 E 不在结果中. 用相同方法处理 B, C 可得 BC 模式为所求. 图 1c 是更新 SW_1 后的 WMFP – SW – tree,其 RJF 值 0.182 < 0.2,所以不重构.

4 实验分析

所有实验在一台具有 Intel(R) core(TM) i5 处理器,4 GB 内存的 PC 机上完成. 算法由 java 实现. 实验以 mushroom 和 retail 为样本. mushroom 是稠密数据集,8 124 条记录,大小 0.54 MB,120 个 item,平均记录长度 23;retail 是稀疏数据集,88 162 条记录,大小 3.97 MB,16 470 个 item,平均记录大小 13. 使用这 2 个样本时 |EW| 分别为 500,1 000;item 权值取 0.5 ~ 0.8 之间随机数;实验执行次数为 10.

4.1 FWMFI – SW 算法的有效性

设 SN_{result} 与 SF_{result} 分别为相同参数下 naive 与 FWMFI – SW 算法结果集合,$|SF_{result}|$ 与 $|SN_{result}|$ 分别是它们中成员数目,V_{equal} 是 2 个集合中相等成员个数. 这里规定 SF_{result} 中任意成员 A 与 SN_{result} 中成员 B,如果它们所在窗口起始位置和成员内容都相同,则 A 与 B 相等;否则不相等. 基于此定义了 FWMFI – SW 算法相对于 naive 算法的查全率(recall)与查准率(precision),分别用 $R_{FWMFI-SW}$ 与 $P_{FWMFI-SW}$ 表示.

$$R_{FWMFI-SW} = V_{equal} / |SN_{result}| \ ; \quad (4)$$

$$P_{FWMFI-SW} = V_{equal} / |SF_{result}| \ . \quad (5)$$

类似可定义 WMFP – SW 算法相对于 naive 算法的查全率($R_{WMFP-SW}$)与查准率($P_{WMFP-SW}$). 图 2 给出了 mushroom 下算法的查全率与查准率.

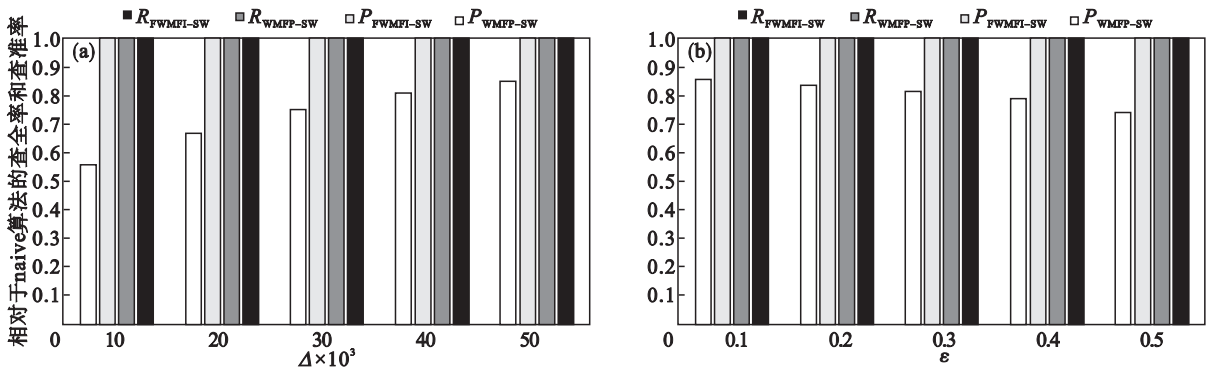

图 2 不同算法相对于 naive 算法的查全率和查准率对比

Fig. 2 Comparison of recall and precision of different algorithms over the naive

(a)—查全率和查准率与 Δ($\varepsilon = 0.1$, |SW| = 8, $\gamma = 0.2$);(b)—查全率和查准率与 ε ($\Delta = 0.05$, |SW| = 8, $\gamma = 0.2$).

由图 2 可见，$R_{\mathrm{FWMFI-SW}} = P_{\mathrm{FWMFI-SW}} = 1$. 这说明 FWMFI – SW 与 naive 算法结果相同. 另外可以看到 $R_{\mathrm{WMFP-SW}} = 1$. 这是因为 FWMFI 集合是通过使用另外一个频繁阈值 ε 过滤 WMFI 集合后得到的子集；同样理由 WMFI 集合中才会有不满足频繁阈值条件的成员，所以 $P_{\mathrm{WMFP-SW}} < 1$，该现象也说明 WMFP – SW 算法不适合挖掘 FWMFI.

4.2　FWMFI – SW 算法的性能评价

分析 |SW| 对算法时间的影响效果见图 3.

由图 3 可见 2 种样本下 WMFP – SW 算法时间随 |SW| 增加分别减少和增加，这与文献[7]描述一致. 另外还可以看到 WMFP – SW 时间大于 naive，这是因为 naive 算法不用处理 WMFP – SW – tree 与条件 WMFP – SW – tree 头表中很多非频繁成员；另外 naive 与 FWMFI – SW 算法时间都随 |SW| 增加而增加. 这是因为窗口越大，其中频繁项越多，WMFP – SW – tree 头表会有更多成员需要处理.

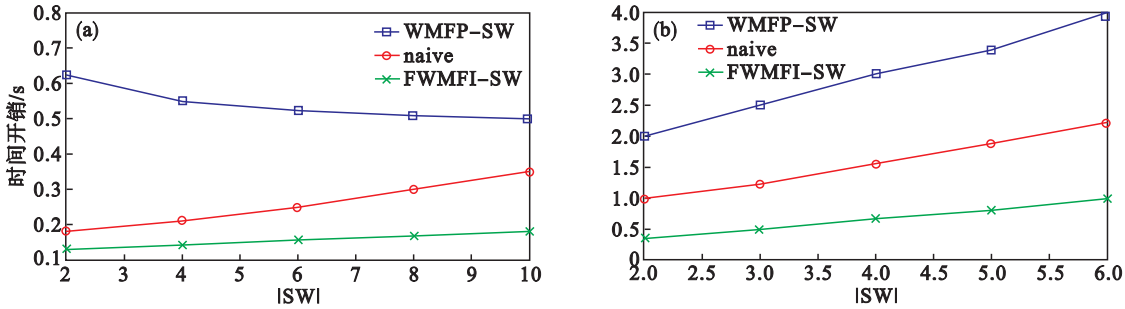

图 3　算法在 |SW| 变化时的执行时间比较

Fig. 3　Comparison of the execution time about different algorithms when |SW| changes

(a)—mushroom 样本($\Delta = 0.05$, $\varepsilon = 0.1$, $\gamma = 0.2$)；(b)—retail 样本($\Delta = 0.001$, $\varepsilon = 0.1$, $\gamma = 0.2$).

Δ 对算法执行时间的影响，见图 4. 由图 4 可见，WMFP – SW 算法时间随 Δ 的增加而减少. 这是因为 Δ 越大，$\max W$ 优化策略效果越好，另外还可以看到 naive 与 FWMFI – SW 算法执行时间都与 Δ 无关. 这是因为当前参数下所有频繁项也满足 $\max W$ 优化策略判别条件. FWMFI – SW 算法因为不需要执行 naive 算法中所有的 $\max W$ 优化策略，同时也引入了重构判别机制，所以执行时间小于 naive 算法.

ε 对算法执行时间的影响见图 5. 由图 5 可以看到 WMFP – SW 算法时间不随 ε 变化发生改变. 这是因为该算法与 ε 无关，另外还可以看到

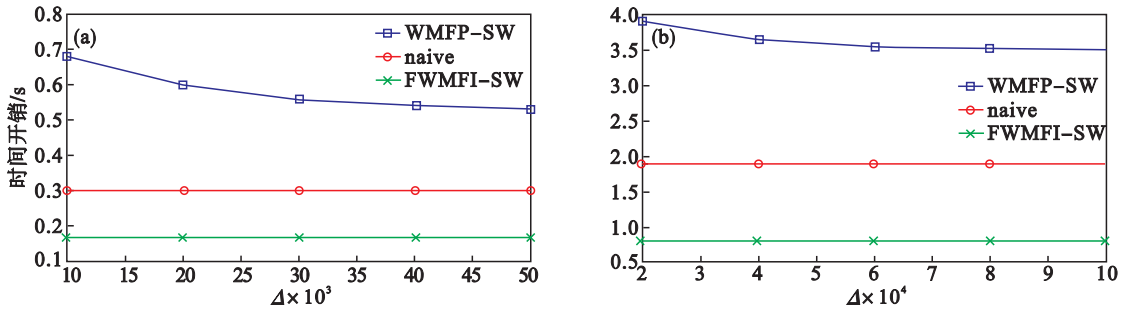

图 4　算法在 Δ 变化时的执行时间比较

Fig. 4　Comparison of the execution time about different algorithms when Δ changes

(a)—mushroom 样本(|SW| = 8, $\varepsilon = 0.1$, $\gamma = 0.2$)；(b)—retail 样本(|SW| = 5, $\varepsilon = 0.1$, $\gamma = 0.2$).

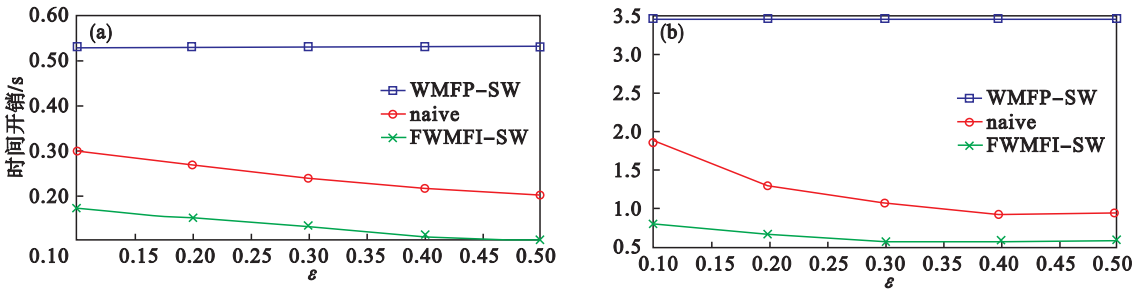

图 5　算法在 ε 变化时的执行时间比较

Fig. 5　Comparison of the execution time about different algorithms when ε changes

(a)—mushroom 样本(|SW| = 8, $\Delta = 0.05$, $\gamma = 0.2$)；(b)—retail 样本(|SW| = 5, $\Delta = 0.001$, $\gamma = 0.2$).

WMFP – SW 算法时间大于 naive 算法,这是因为 naive 算法不用处理 WMFP – SW – tree 与条件 WMFP – SW – tree 头表中的非频繁成员;其他 2 类算法时间随 ε 取值增加而递减. 这是因为 ε 越大,WMFP – SW – tree 头表中要处理的成员越少. 另外可以看到 FWMFI – SW 执行时间小于 naive 算法,说明 FWMFI – SW 使用的策略有效.

γ 对算法执行时间的影响见图 6. 由图 6 可见 WMFP – SW 算法时间大于 naive 算法,这是因为 naive 算法忽略 WMFP – SW – tree 与条件 WMFP – SW – tree 头表中很多非频繁成员. 另外可以看到,当 γ 在 0 ~ 0.2 间递增取值时,FWMFI – SW 算法的时间小于 naive 算法的,而且会随 γ 增加而减少. 这是因为 RJF 值小于 γ 的 WMFP – SW – tree 在重构后结构变化不大,重构操作会增加时间开销. 当 γ 在 0.2 ~ 1 间递增取值时,FWMFI – SW 算法时间增加. 这是因为随着 γ 增加,很多 RJF 值较大的 WMFP – SW – tree 没有重构.

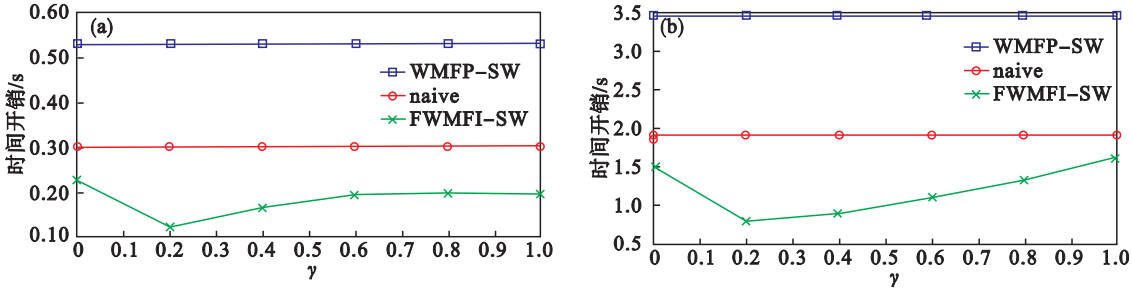

图 6 算法在 γ 变化时的执行时间比较

Fig. 6 Comparison of the execution time about different algorithms when γ changes

(a)—mushroom 样本(|SW| = 8, Δ = 0.05, ε = 0.1);(b)—retail 样本(|SW| = 5, Δ = 0.001, ε = 0.1).

另外由图中可见,当 γ 在[0,0.2]与[0.2,1]范围内递增取值时,FWMFI – SW 算法时间分别会单调递减和单调递增. 所以 γ = 0.2 时 FWMFI – SW 算法中的重构判别效果最好.

5 结 论

本文提出了数据流完全加权最大频繁项集概念,并给出了滑动窗口下该项集挖掘算法——FWMFI – SW. 该算法在 naive 算法上增加了基于频繁约束条件的优化策略;同时利用编辑距离比率作为窗口更新后是否重构 WMFP – SW – tree 的判别函数. 这些策略有效减少了 naive 算法的冗余运算. 实验结果表明 FWMFI – SW 时间开销小于 naive 算法,是处理滑动窗口下数据流 FWMFI 挖掘的有效方法.

参考文献:

[1] 李国徽,陈辉. 挖掘数据流任意滑动窗口内频繁模式[J]. 软件学报,2008,19(10):2585 – 2596.
(Li Guo-hui, Chen Hui. Mining the frequent patterns in an arbitrary sliding window over online data stream [J]. *Journal of Software*, 2008, 19(10): 2585 – 2596.)

[2] Gao C C, Wang J Y, Yang Q Y. Efficient mining of closed sequential patterns on stream sliding window [C]// Proceedings of the 11th IEEE International Conference on Data Mining. Los Alamitos: IEEE Computer Society, 2011: 1044 – 1049.

[3] Yang S Y, Chao C M, Chen P Z, et al. Incremental mining of closed sequential patterns in multiple data streams [J]. *Journal of Networks*, 2011, 6(5): 728 – 735.

[4] Li H F, Zhang N. A simple but effective stream maximal frequent itemset mining algorithm [C]// Proceedings of the 7th International Conference on Computational Intelligence and Security. Piscataway: IEEE Computer Society, 2011: 1268 – 1272.

[5] 敖富江,颜跃进,刘宝宏,等. 在线挖掘数据流滑动窗口中最大频繁项集[J]. 系统仿真学报,2009,21(4):1134 – 1139.
(Ao Fu-jiang, Yan Yue-jin, Liu Bao-hong, et al. Online mining maximal frequent item sets in sliding window over data stream [J]. *Journal of System Simulation*, 2009, 21(4): 1134 – 1139.)

[6] Yun U, Lee G, Ryu K H. Mining maximal frequent patterns by considering weight conditions over data streams [J]. *Knowledge-Based Systems*, 2014, 55: 49 – 65.

[7] Lee G, Yun U, Ryu K H. Sliding window based weighted maximal frequent pattern mining over data streams [J]. *Expert Systems with Applications*, 2014, 41(2): 694 – 708.

[8] Wang J, Yu Z. DSWFP: efficient mining of weighted frequent pattern over data streams [C]//Proceedings of the 8th International Conference on Fuzzy Systems and Knowledge Discovery. Piscataway: IEEE Computer Society, 2011: 942 – 946.

[9] Yun U, Shin H, Ryu K H, et al. An efficient mining algorithm for maximal weighted frequent patterns in transactional databases [J]. *Knowledge-Based Systems*, 2012, 33: 53 – 64.

[10] Yun S K, Gillian D. Efficient single pass ordered incremental pattern mining [C]//Proceedings of the 13th International Conference on Data Warehousing and Knowledge Discovery. Berlin: Springer, 2011: 265 – 276.