基于时间衰减模型的数据流频繁模式挖掘

吴 枫 [1] 仲 妍 [1] 吴泉源 [1]

摘 要 频繁模式挖掘是数据流挖掘中的重要研究课题. 针对数据流的时效性和流中心的偏移性特点, 提出了界标窗口模型与时间衰减模型相结合的数据流频繁模式挖掘算法. 该算法通过动态构建全局模式树, 利用时间指数衰减函数对模式树中各模式的支持数进行统计, 以此刻画界标窗口内模式的频繁程度; 进而, 为有效降低空间开销, 设计了剪枝阈值函数, 用于对预期难以成长为频繁的模式及时从全局树中剪除. 本文对出现在算法中的重要参数和阈值进行了深入分析. 一系列实验表明, 与现有同类算法 MSW 相比, 该算法挖掘精度高 (平均超过 90 %), 内存开销小, 速度上可以满足高速数据流的处理要求, 且可以适应不同事务数量、不同事务平均长度和不同最大潜在频繁模式平均长度的数据流频繁模式挖掘.

关键词 数据流, 频繁模式挖掘, 数据挖掘, 时间衰减模型
DOI 10.3724/SP.J.1004.2010.00674

Mining Frequent Patterns over Data Stream under the Time Decaying Model

WU Feng[1] ZHONG Yan[1] WU Quan-Yuan[1]

Abstract Frequent-pattern mining is an important task in mining data streams. Considering the timeliness of data stream and the shift of the stream center, we propose an algorithm of data stream frequent pattern mining, named DFPMiner, which combines both the landmark window and the time decaying model. By dynamically constructuring the global pattern tree, the method uses time exponential decay function to account the support of each pattern to describe the frequent degree within the landmark window. Moreover, to reduce the space cost effectively, the constructed pruning threshold function is used to cut in time the pattern that is not able to turn into the frequent pattern from the global tree. This paper deeply analyzes the important parameters and thresholds in the algorithm. The experimental results show that comparing with the algorithm MSW, DFPMiner has higher mining precision (over 90 % in average) and lower storage cost, and it can meet the request of processing high-speed data streams in the execution. Our method can be applied to the streams of different numbers of transactions, different average transaction sizes, and different maximal potentially frequent pattern sizes.

Key words Data stream, frequent pattern mining, data mining, time decaying model

随着通信技术的不断发展, 数据采集手段多样且方便快捷, 诸如各类监控系统, 包括传感器网络监控、安全态势的实时感知、天气或环境监测、网络服务商对流量的监控等, 每时每刻都在产生无穷、时序和快速到达的数据流[1]. 这些数据的高效实时处理以及相关的在线监控、持续提供近似查询结果等现实需求, 与计算/存储能力、网络带宽资源等方面的矛盾日益突出, 对传统的面向静态数据集的数据处理技术提出了诸多挑战[1−2]. 如何管理和分析这些数据流, 成为新一代计算理论和应用的研究难点之一.

目前, 数据流分析的研究主要包括查询和挖掘, 在数据流挖掘方面, 频繁模式挖掘是一个重要的研究课题. 由于数据流不同于静态数据集, 导致传统的挖掘方法无法适用于流环境中[2]: 首先, 无法获得数据流的先验模式信息; 其次, 有限的内存不足以保存数据流中完整的模式信息; 此外, 还有实时性、精度和自适应性等限制. 为此需要研究适合动态环境下的高效频繁模式挖掘算法.

为了能够及时、精确、全面地挖掘数据流上最近的频繁模式, 本文提出了界标窗口模型与时间衰减模型相结合的数据流频繁模式挖掘算法. 该算法通过动态构建全局模式树, 利用时间指数衰减函数对模式树中各模式的支持数进行统计, 以此刻画界标窗口内模式的频繁程度; 并设计了剪枝阈值函数, 用以对预期难以成长为频繁的模式及时进行删除, 从而有效降低了空间开销. 本文对全局模式树的性质、有关周期删除机制的重要参数和阈值进行了深入分析. 通过一系列真实、人工数据集上的实验验证了本文算法的高效性.

1 相关工作

数据流频繁模式挖掘算法大致分为如下两大类:

收稿日期 2008-12-15 录用日期 2009-11-06
Manuscript received December 15, 2008; accepted November 6, 2009
国家高技术研究发展计划 (863 计划) (2006AA01Z451, 2007AA01Z474) 资助
Supported by National High Technology Research and Development Program of China (2006AA01Z451, 2007AA01Z474)
1. 国防科学技术大学计算机学院 长沙 410073
1. School of Computer, National University of Defense Technology, Changsha 410073

1) 基于概率误差区间的近似算法[3]. 这类算法利用采样、哈希等随机方法进行模式频率计数, 所以导致不确定误差, 但能够以较高的置信度将误差控制在一定范围内, 从而可获得较好的近似结果. 文献 [4] 通过 “Sticky 采样” 完成模式频率的近似计数. 文献 [5] 在滑动窗口模型下结合 “Sticky 采样” 方法, 通过动态删除数据流上的模式信息, 能够有效消除过期数据的影响. 由于 “Sticky 采样” 计数不精确, 为此产生了基于哈希的模式计数方法[6−7], 该方法对数据流上的模式信息逐个进行哈希映射, 并以较小的内存空间加以维护.

2) 基于确定误差区间的近似算法[3]. 相对于前一类算法而言, 这类算法没有采用随机方法, 因此可以确定相对误差的上界. 文献 [4] 提出了一种称为 “Lossy 计数” 的方法, 能够确保无漏报, 但该方法所产生的候选项集数量非常巨大, 内存存储开销很高, 而且没有消除过期数据的影响, 不能动态反映数据流上的最新变化. 为降低存储开销, 文献 [8−9] 采用前缀子模式复用方式压缩存储数据流上的模式信息, 并利用子模式估计模式频率, 快速过滤掉不频繁模式信息, 降低了存储空间. 为实时反映数据流上最近的频繁模式信息, 提出了时间衰减模型[10−11] 下频繁模式挖掘方法. 文献 [11] 提出了 FP-stream 算法, 结合 FP-Growth 和倾斜时间窗口策略, 以不同时间粒度分别保存数据流上不同时期的频繁模式信息. 文献 [12] 在基于滑动窗口的频繁模式挖掘中结合时间衰减模型处理方法, 提出了 MSW 算法. 该算法能够挖掘任意大小滑动窗口内的频繁模式, 而且通过周期删除过期和不频繁的模式信息, 降低了算法的储存开销. 此外, 文献 [13] 提出了 SS-BE 和 SS-MB 两种算法, 分别通过不同的删除策略, 能够确保无漏报, 而且算法执行效率高.

2　预备知识

一般来讲, 数据流频繁模式挖掘可描述为: 给定数据项集合 $I=\{i_1,i_2,\cdots,i_M\}$, 其中元素按照全序 $\prec$(字典序) 进行排列; 数据流 $DS=\{TA_1,TA_2,\cdots,TA_j,\cdots\}$ 为无限可数的事务序列, 任意 $TA_j\subseteq I$; 定义 $sup(P)$ 为模式 P ($P\subseteq I$) 在数据流中的支持集, 当 $|sup(P)|/N\geq s$ 时称为频繁模式[3], 其中, N 表示当前数据流中事务的数量, s 为支持度阈值. 数据流频繁模式挖掘的任务[3] 是在一定相对误差范围内, 近似算法利用有限存储空间对模式频率进行估计, 从而获得满足支持度要求的频繁模式集.

定义 1[12] **(事务投影).** 将事务 $TA=\{a_1,a_2,\cdots,a_k\}$ 中元素按全序 $\prec$ 重新排列后的新数据序列 $TA=\{a'_1,a'_2,\cdots,a'_k\}$ 称为事务 TA 的投影. 如果没有特殊说明, 不区分事务和事务投影.

定义 2[12]**.** 对于事务 $TA=\{a_1,a_2,\cdots,a_k\}$ 与模式 $P=\{b_1,b_2,\cdots,b_m\}$ ($m\leq k$), 如果 $\forall i$, $1\leq i\leq m$, 总有 $a_i=b_i$ 成立, 则称模式 P 为 TA 的 m-前缀模式.

根据数据流的概念漂移特性和时效性可知, 数据流中蕴含的知识将持续发生改变, 同时知识的重要性也将不断进行衰减. 因此, 需要能够区分当前事务和历史事务, 并不断减弱历史事务对挖掘结果产生的影响. 为此, 本文在时间衰减模型[12] 下考虑数据流的频繁模式挖掘问题. 在该模型中各事务包含的模式的支持数随着时间 t 变化且以衰减函数 $f(t)=2^{-\lambda t}$ 进行衰减, 其中, $\lambda>0$. 显然新产生的事务相对于旧事务而言, 更能影响频繁模式挖掘结果.

定义 3 (衰减支持数). 对模式 P, 令其出现的时间集合为 $Time(P)$. 则当前时刻 t_c, 该模式的衰减支持数定义为: $F(P,t_c)=\sum_{T_j\in Time(P)}2^{-\lambda(t_c-T_j)}$.

为了挖掘截至当前时刻数据流上真正的频繁模式, 定义了完全衰减支持数的概念, 代表当前时刻模式真实的衰减支持数信息. 实际情况中, 有限的内存和处理速度, 使得算法难以获取数据流上完整的模式信息. 为此, 提出了观测衰减支持数的概念, 代表当前时刻算法维护的模式衰减支持数信息.

定义 4 (观测衰减支持数). 截至当前时刻 t_c, 令算法维护的模式 P 的出现时间集合为 $Time_o(P)$, 则此时 P 的观测衰减支持数为

$$D(P,t_c)=\sum_{T_j\in Time_o(P)}2^{-\lambda(t_c-T_j)}$$

定义 5 (完全衰减支持数). 截至当前时刻 t_c, 令模式 P 完整的出现时间集合为 $Time_w(P)$, 则当前时刻 t_c, P 的完全衰减支持数为

$$D_a(P,t_c)=\sum_{T_j\in Time_w(P)}2^{-\lambda(t_c-T_j)}$$

根据定义 5, 将模式按频繁程度分为 3 种类型:

定义 6 (频繁模式, 潜在模式和疏模式). 如果模式 P 的完全衰减支持数大于或等于 D_m, 则定义为频繁模式 ($FPattern$); 如果完全衰减支持数小于 D_l, 则定义为疏模式 ($SPattern$); 如果介于两者之间, 则称为潜在模式 ($PPattern$); $PPattern$ 和 $FPattern$ 统称为非疏模式 ($fSPattern$).

3　频繁模式挖掘

3.1　模式在线维护的数据结构

通过对文献 [12] 中滑动窗口模式树 SW-tree 加以改进, 提出了一种全局模式 Global-tree 来增量维护数据流上的模式信息. 具体改进如下:

1) Global-tree 上除根节点外的节点包括 6 个数据域: $item$, $D(P,t)$, t_P, t_l, $status$, $nodelink$. 其中, t_P 为模式 P 最近一次在 Global-tree 上的创建时间; t_l 为模式 P 最近一次出现的时间; $status$ 表示该模式的 4 种状态, 分别为待定 (0)、非疏 (1)、疏 (-1) 或待删除 (-2). 与 SW-tree 不同的是, Global-tree 在 SW-tree 原有 4 项的基础上增加了 t_P, $status$ 两项.

2) Global-tree 的频繁数据项头表 ItemHead-Table 中, 每一个表项包含两个数据域: $item$ 和 $head\ of\ nodelink$, 在 SW-tree 原有 4 项的基础上, 删除了 tid 和 $count$ 两项.

除改进部分外, 其他均保持不变. 下面通过一个例子来说明 Global-tree, 为便于与 SW-tree 进行比较, 采用与文献 [12] 相同的例子, 如图 1 所示.

性质 1. 全局模式树 Global-tree 上任何节点的观测衰减支持数、完全衰减支持数与最近一次模式出现时间均不小于其子孙节点; 最近一次模式创建时间均不大于其子孙节点.

证明. Global-tree 采用前缀子模式复用方式存储数据流上的模式, 因此, 对任意的模式 P, P_1, 若 $P \subseteq P_1$, 则 P, P_1 共享同一个分支, 且 P_1 相对于 P 而言, 距离根节点更远. 因为, 事务包含模式 P 时, 并不一定包含 P_1; 但是, 包含模式 P_1 时, 则肯定包含 P. 所以, 没有引入模式删除机制, 当将事务的模式信息增量更新至 Global-tree 时, 总是更早、更多地更新离根节点近的节点. 从而子孙节点相对于祖先节点而言, 首次创建的时间 (等同于最近一次创建时间) 更晚, 最近一次出现的时间更早, 共享前缀子模式的机会更小 (模式观测衰减支持数和完全衰减支持数等价, 且与共享机会成正比). 接下来, 分析引入模式删除机制时的情况. 因为当 Global-tree 中的任意一个节点被删除时, 其子孙节点也会被同时删除; 而当其被建立时, 其子孙节点未必会同时建立 (约定: 对 Global-tree 中的任何节点, 除非被删除, 否则, 其中维护的 6 个数据域信息不会有任何损失, 并且只允许增量更新). 所以, 自从最近一次被删除以来, 子孙节点相对于祖先节点而言, 最近一次的创建时间更晚, 最近一次出现的时间更早, 共享前缀子模式的机会更小 (模式观测衰减支持数与共享机会成正比). □

3.2 模式在线维护理论

随着数据流的进行, Global-tree 中保存的疏模式数量将迅速增加, 导致内存存储开销过大, 所以有必要在不影响挖掘质量的前提下对其加以删除. 然而, 部分疏模式可能会在被删除之后的短期时间内立即出现, 甚至迅速成长为频繁模式. 如此一来, 不仅会降低挖掘质量, 而且还会增加不必要的时间开销. 所以, 应该适当保留具有一定成长潜力的疏模式. 下面, 在没有任何删除措施的情况下, 对疏模式的成长潜力加以分析:

1) 离群模式 ($OPattern$). 这些模式极少出现, 没有成长潜力, 无论何时都是 $SPattern$. 所以, 对任意时刻 t, $OPattern$ 的完全衰减支持数满足 $D_a(P,t) < D_l$. 假定模式 P 为 $OPattern$, 则对任意的模式 P_1, 若 $P \subseteq P_1$, 则 P_1 亦为 $OPattern$.

2) 退化模式 ($TPattern$). 这些模式在近期某时刻曾经是 $fSPattern$, 随后退化成为 $SPattern$, 但仍具备成长潜力. 所以, 给定当前时刻 t_c, 过去某时刻 t_0 ($0 < t_c - t_0 < T_\sigma$), 时间间隔 T_{Dl}, $TPattern$ 的完全衰减支持数满足: $D_a(P,t) < D_l$ ($t_c \le t < t_c + T_{Dl}$), $D_a(P,t_0) \ge D_l$, $D_a(P,t_c + T_{Dl}) \ge D_l$. 假定模式 P 为 $TPattern$, 则对任意的模式 P_1, 若 $P \subseteq P_1$, 则 P_1 为 $SPattern$ 中的任何一种.

3) 幼模式 ($YPattern$). 这些模式创建时间不长, 完全衰减支持数较小, 但具备成长潜力. 所以, 给定当前时刻 t_c, 首次创建时刻 t_f ($0 < t_c - t_f < T_\varepsilon$, T_ε 较小), 时间间隔 T_{Dl}, $YPattern$ 的完全衰减支持数满足: $D_a(P,t) < D_l$ ($t_f \le t < t_c + T_{Dl}$), $D_a(P,$

Time	Transaction	Projection
1	*dca*	*acd*
2	*ca*	*ac*
3	*bf*	*bf*
4	*ce*	*ce*
5	*ebc*	*bce*
6	*ecf*	*cef*
7	*cbf*	*bcf*

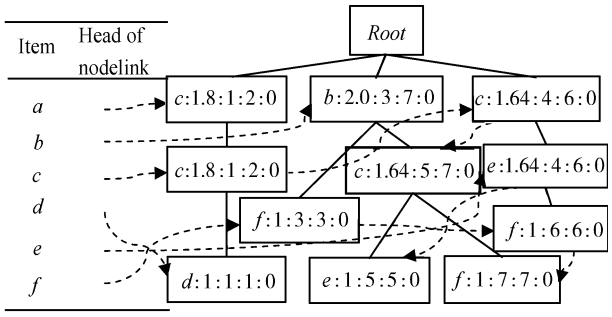


图 1 Global-tree 举例

Fig. 1 Example of Global-tree

$t_c + T_{Dl}) \geq D_l$. 假定模式 P 为 $YPattern$, 则对任意的模式 P_1, 若 $P \subseteq P_1$, 则 P_1 为 $YPattern$ 或 $OPattern$.

4) 异常模式 ($APattern$). 这些模式可能曾经是 $fSPattern$, 但在近期很长一段时间内, 一直都是 $SPattern$, 故而在未来时间间隔 T ($T >> T_{Dl}$) 内, 出现的可能性很低, 成长潜力非常小. 给定当前时刻 t_c, 过去某时刻 t_0 ($t_c - t_0 > T_\sigma$), $APattern$ 的完全衰减支持数满足: $D_a(P,t) < D_l$ ($t_c - T_\sigma \leq t < t_c + T$), $D_a(P,t_0) \geq D_l$, $D_a(P,t_c+T) \geq D_l$. 假定模式 P 为 $APattern$, 则对任意的模式 P_1, 若 $P \subseteq P_1$, 则 P_1 为 $OPattern$ 或 $APattern$.

可以看出, 在上述四种疏模式中, 离群模式和异常模式的成长潜力最小, 因此, 如果能够准确识别这两类模式, 便可立即加以删除. 对于另外两种疏模式, 需要在一定的观察周期 (定义为删除周期) T_{Dl} 内对其成长潜力加以考察, 经确认满足一定条件后方可加以删除.

首先, 针对退化模式和幼模式的特点, 定义合适的删除周期 T_{Dl}, 并在此基础上给出安全有效的删除策略 (见性质 2). 假设模式由疏至频繁所需的最短时间为 δ_1, 由频繁至疏所需的最短时间为 δ_2, 定义删除周期的取值区间为: $T_{Dl} \in [\min\{\delta_2,\delta_1\}, \max\{\delta_2,\delta_1\}]$.

性质 2. 对模式 P, 给定删除周期 T_{D1}. 令当前时刻 t_c, 模式完全衰减支持数满足 $D_a(P,t_c) < D_l$, 并且在未来时间段 $[t_c, t_c+T_{D1}]$ 内满足 $D_a(P,t) < D_l$. 倘若在时间段 $[t_c, t_c+T_{Dl}]$ 内, 没有模式 P 出现过, 则在时刻 t_c+T_{Dl}, 该模式可被删除.

性质 3. 对模式 P, 由 $fSPattern$ 衰退成 $SPattern$ 的最短时间为 $\delta_0 = \left\lfloor \frac{1}{\lambda}\log_2(\frac{D_l}{D_l-1}) \right\rfloor$; 由 $FPattern$ 衰退成 $SPattern$ 的最短时间为 $\delta_2 = \left\lfloor \frac{1}{\lambda}\log_2(\frac{D_m}{D_l}) \right\rfloor$; 由 $SPattern$ 成长为 $FPattern$ 的最短时间为 $\delta_1 = \left\lfloor \frac{1}{\lambda}\log_2(\frac{(1-2^{-\lambda})D_l-1}{(1-2^{-\lambda})D_m-1}) \right\rfloor$; 由 0 成长为 $fSPattern$ 的最短时间为

$$\delta_3 = \left\lfloor \frac{1}{\lambda}\log_2(\frac{1}{1-(1-2^{-\lambda})D_l}) \right\rfloor$$

其次, 给出离群、异常模式的识别依据 (性质 5).

性质 4. 假设模式 P 上一次被删除时刻为 t_m, 最近一次的创建时刻为 t_P. 如果当前时刻 t_c, P 为 $SPattern$, 且满足 $\delta_0+\delta_3 > t_c - t_P > 0$, 则 P 成为 $fSPattern$ 的最近时刻 t_0 满足: $t_c - t_0 \geq \delta_0 + 2T_{D1}$.

根据 T_{D1} 的定义可知, 时间间隔 $\delta_0 + 2T_{D1}$ 足以反映模式从疏成长为非疏, 而后又退化至疏的消长过程, 因此不妨设定 $T_\sigma \leq \delta_0 + 2T_{D1}$. 给定某疏模式 P, 根据退化模式的定义和性质 4 可知, 如果当前时刻 t_c, 模式 P 满足 $\delta_0+\delta_3 > t_c - t_P > 0$, 则 P 必然不为 $TPattern$; 另外, 根据幼模式定义可知, 如果当前时刻 t_c, 模式 P 满足 $t_c - t_P \geq T_\varepsilon$, 则该模式必定不为 $YPattern$. 综上分析, 可以得到如下结论:

性质 5. 假设模式 P 最近一次的创建时刻为 t_P. 如果当前时刻 t_c, P 为 $SPattern$, 且满足 $\delta_0+\delta_3 > t_c - t_P \geq T_\varepsilon$, 则 P 为 $OPattern$ 或 $APattern$.

性质 2～5 给出了疏模式的删除依据, 作为前提还需给出疏模式的识别依据. 引入周期删除机制以后, 定义观测衰减支持数阈值函数 ζ, 用以判定时刻 t 模式的完全衰减支持数是否小于 D_l.

定义 7. 令 P 最近一次的创建时刻为 t_P, 当前时刻 t_c, ζ 定义为: $\zeta(t_P,t_c) = (1-2^{-\lambda(t_c-t_P+1)})D_l$.

性质 6. 假设模式 P 上一次被删除的时刻为 t_m, 最近一次的创建时刻为 t_P. 对于当前时刻 t_c, 如果满足下列任何一种情况, 则必有 $D_a(P,t_c) < D_l$.

1) $D(P,t_c) < \zeta(t_P,t_c)$;

2) 如果 $t_c = NT_{D1}(N$ 为自然数), $0 < t_c - t_P < T_{D1}$, 且 $\zeta(t_P,t_c) \leq D(P,t_c) \leq (1-2^{-\lambda T_{Dl}})D_l$;

3) 如果 $t_1 = NT_{D1}(N$ 为自然数), $0 < t_P - t_1 < T_{D1}$, 且 $D(P,t_c) \leq (1-2^{-\lambda(t_c-t_l)})D_l$;

4) 对任意的模式 P, P_1 ($P_1 \subseteq P$), 如果存在 $D_a(P_1,t_c) < D_l$.

3.3 模式在线维护算法 DFPMiner

随着时间的推移, 新事务不断出现, 而历史事务随着时间衰减逐渐淡出. 为了实时维护数据流上最新的模式信息, 必须及时将其更新至 Global-tree 上; 另外, 为确保系统效率并节省内存开销, 还需要周期性地删除无用的疏模式信息. 为此, 提出模式在线维护算法 DFPMiner, 其具体过程如下:

步骤 1 (第 2 行). 对新到达事务 TA 进行吸收合并.

步骤 2 (第 3～12 行). 当到达删除周期时, 自顶向下获取 ItemHeadTable 中的数据项 $item$, 然后遍历 Global-tree 中所有与 $item$ 同名的节点 $node$, 进行如下操作:

1) 对 $node$ 状态进行更新 (第 6 行), 状态共有 4 种, 分别是待定、非疏模式、疏模式和待删除, 分别用 0、1、−1 和 −2 表示;

2) 根据更新后的 $node$ 状态采取相应措施:

a) 如果状态为 −2 且满足性质 2, 说明该模式为没有成长潜力的退化或幼模式, 可以安全删除 (第 7～9 行);

b) 如果状态为 −1 且满足性质 5, 说明该模式为离群或异常模式, 亦可安全删除 (第 10～12 行).

步骤 3 (第 13 ∼ 17 行). 当用户发出挖掘结果请求时, 在不改变 Global-tree 状态值的前提下输出 $fSPatterns$, 这是因为当前时刻 Global-tree 的模式信息不一定最新, 同时也为保持 DFPMiner 连续性考虑.

DFPMiner 算法的伪代码如下:

```
1  Get the arriving transaction T at time t_c;
2  Merge(T, Root, t_c);
3  If (t_c % T_D1) == 0 then
4      For each item in ItemHeadTable
5          For each node in Global-tree on the
           nodelink of item where node.item
           = item
6              Update(node.(item, D(P,t), t_p, t_l,
               status, nodelink), t_c);
7              If node.status == −2 then
8                  If t_c − node.t_l > T_D1 then
9                      Delete node in Global-tree;
10             Else If node.status == −1 then
11                 If δ_0 + δ_3 > t_c − node.t_P ≥ T_ε then
12                     Delete node in Global-tree;
13 If the request for result arrives then
14     If (t_c % T_D1) == 0 then
15         Use Global-tree to generate result;
16     Else
17         Keep the current Global-tree unchanged,
           and then use them to find the frequent
           patterns up to date to generate result.
```

4 算法分析

4.1 Global-tree 规模分析

首先, 分析算法实时维护的没有共享前缀子模式的疏模式集 C 的大小. 类似于文献 [14], 根据算法的删除周期 T_{D1}, 可以将数据流划分为若干宽度为 $l = T_{D1}$ 的桶. 令当前时刻 T 时, 桶的总数为 N, 约定 n_i $(1 \le i \le N)$ 表示当前集合 C 在桶 $N - i + 1$ 内创建的疏模式的个数, 令这些模式在桶 $N - i + 1$ 至 N 之间至少出现了 m_i 次, 因为同一个事务中所有的模式都共享相同的前缀子模式, 所以不失一般性, 假设单位时间内到达一个模式. 综上可得:

$$\sum_{i=1}^{j} m_i n_i \le jl,\ \ j = 1, 2, \cdots, N \tag{1}$$

事实上, 可用归纳法证明得到:

$$\sum_{i=1}^{j} n_i \le \left(\sum_{i=1}^{j} \frac{1}{m_i}\right) l,\ \ j = 1, 2, \cdots, N \tag{2}$$

观察到式 (2) 左边即为 $|C|$. 为分析 $|C|$ 的上限, 则需使 m_i 尽可能小. 根据性质 2 可知, 任意模式如果在连续的两个桶内没有出现过, 那么便会被删除. 所以 m_i 的最小值为 $2/i$. 根据不等式 (2), 可得

$$|C| = \left(\sum_{i=1}^{N} \frac{1}{m_i}\right) l \le 2l \log N = 2T_{Dl} \log\left(\frac{T}{T_{Dl}}\right)$$

上式表明, 时刻 T 算法所维护的没有共享前缀子模式的疏模式的规模为 $\mathrm{O}(2T_{Dl}\log(T/T_{Dl}))$. 假定模式最大长度为 $L_{\max}$, 那么算法维护的疏模式树 (Global-tree 中由疏模式构成的子树) 的规模为 $\mathrm{O}(2T_{Dl}L_{\max}\log(T/T_{Dl}))$.

其次, 分析算法实时维护的没有共享前缀子模式的非疏模式集 C_1 的大小. 类似于文献 [14] 的分析, 截至时刻 T, 数据流的衰减支持数为 (T 足够大):

$$W_T = \sum_{t=0}^{T} 2^{-\lambda t} = \frac{1 - 2^{-\lambda(T+1)}}{1 - 2^{-\lambda}} \approx \frac{1}{1 - 2^{-\lambda}}$$

根据阈值 D_l, 时刻 T 算法维护的没有共享前缀子模式的非疏模式的规模为 $\mathrm{O}(1/((1 - 2^{-\lambda})D_l))$; 进而, 维护的非疏模式树 (Global-tree 中由非疏模式构成的子树) 的规模为 $\mathrm{O}(L_{\max}/((1 - 2^{-\lambda})D_l))$.

综上, 当前时刻 T 算法实时维护的全局模式树规模为 $\mathrm{O}(2T_{Dl}L_{\max}\log(T/T_{Dl}) + L_{\max}/((1 - 2^{-\lambda})D_l))$.

4.2 算法正确性分析

为评价算法的正确性, 本文借鉴数据流频繁模式挖掘算法的衡量标准[12]:

$$C_{\text{overage}} = \frac{|C_M \cap C_T|}{|C_T|}$$

其中, C_{overage} 代表真实覆盖率, C_M 代表时间衰减模型下挖掘得到的频繁模式集, C_T 代表数据流上真实的频繁模式集 (非时间衰减模型下的频繁模式集).

由于数据流会发生概念漂移, 所以实时反映近期时间 T_{care} 内数据流的演变特征, 才是用户所重点关注的. 因此, 只有保证 T_{care} 内真正的频繁模式能够被输出, 那么才可认为算法真实覆盖率达到 100 %. 下面, 结合一种极端情况[12] 来分析算法的正确性. 据 T_{care} 可以将数据流划分为若干宽度为 $l = T_{\text{care}}$ 的桶, 如图 2 所示.

假定只有桶内阴影部分的 M 个事务包含模式 P. 如果记模式 P 在桶内的衰减支持数为 $D_l(P)$, 那么有: $D_l(P) = 2^{-\lambda(l-1)} + \cdots + 2^{-\lambda(l-M)}$. 由于模式 P 最早进入桶内, 因此模式 P 的支持数将被衰减至其最小值. 实际上 P 的支持数为 M, 如果 P 是频

繁的, 那么有 $M \geq D_m$, 因此, 为保证频繁模式挖掘算法的正确性, 必须保证 $2^{-\lambda(l-1)} + \cdots + 2^{-\lambda(l-M)} \geq D_l$ 成立. 由此得到衰减因子 λ 的取值区间:

$$0 < \lambda \leq \log_2(\frac{D_m}{D_l}) \cdot \frac{2}{2T_{\text{care}} - 1 - D_m}$$

综上所述, 只要衰减因子满足上述取值区间, 则必能保证算法的真实覆盖率为 100 %.

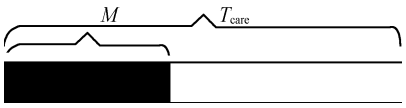


图 2　宽度为 T_{care} 的数据流

Fig. 2　The data stream whose width is T_{care}

4.3　算法误差分析

假设 P 最近一次被删除的时刻为 t_m, 最近一次的创建时刻为 t_P; 对于当前时刻 t_c, $t_c = NT_{Dl}$ ($N = 1, 2, \cdots$), P 的观测衰减支持数和完全衰减支持数分别为 $D(P, t_c)$ 和 $D_a(P, t_c)$. 定义相对误差为

$$error = \frac{D_a(P, t_c) - D(P, t_c)}{D_a(P, t_c)}$$

由于算法每次删除的是 $SPattern$, 因此删除措施对其造成的误差可以不予考虑. 下面仅考虑对 $fSPattern$ 造成的误差, 此时满足:

$$D_a(P, t_c) \geq D_l$$
$$D_a(P, t_c) \leq D(P, t_c) + 2^{-\lambda(t_c - t_m)} D_l$$

根据相对误差的定义, 可知:

$$error \leq \frac{2^{-\lambda(t_c - t_m)} D_l}{D_a(P, t_c)} \leq 2^{-\lambda(t_c - t_m)}$$

由于 $t_c - t_m \geq T_{Dl}$, 所以 $error \leq 2^{-\lambda T_{Dl}}$.

综上所述, 删除措施造成的相对误差不会超过 $2^{-\lambda T_{Dl}}$.

5　实验分析

5.1　实验设置

所有实验均在一台 Intel Core 2 Duo CPU 2.1 GHz 的 PC 机上进行; 实验程序采用 Visual C++ 实现. 实验数据包括: 1) 真实数据集 Connect 4[15−16]; 2) 由 illiMine 模拟数据产生器 (http://illimine.cs.uiuc.edu) 产生的若干人工数据集, 采用如下标记进行命名[12]: 'T' 和 'I' 分别表示数据集中事务的平均长度和最大潜在频繁模式的平均长度, 'D' 表示数据集中事务的数量.

为了验证 DFPMiner 算法的有效性, 本文选取 MSW[12] 算法进行比较. 算法的挖掘质量采用输出结果中频繁模式的纯度 (Pure rate, PR) 作为评价指标, 定义如下:

$$PR = \frac{|frePattern_{\text{true}}|}{|frePattern_{\text{out}}|} \times 100\,\%$$

对于 DFPMiner, $|frePattern_{\text{true}}|$ 和 $|frePattern_{\text{out}}|$ 分别表示 NDDFPMiner (即不采用模式周期删除机制的 DFPMiner) 和 DFPMiner 输出的频繁模式的数量. 对于 MSW, $|frePattern_{\text{true}}|$ 和 $|frePattern_{\text{out}}|$ 分别表示 NDMSW (即不采用模式周期删除机制的 MSW) 和 MSW 输出的未过期的频繁模式数量.

实验中, DFPMiner 的参数缺省设置如下: 数据项的个数设置为 10 000; 衰减支持数阈值 $D_l = 5$, $D_m = 20$; 模式删除周期 $T_{D1} = 5\,000$; 衰减因子 $\lambda = 0.0001$; $T_{\text{care}} = 15\,000$. MSW 采用文献 [12] 中的参数设置.

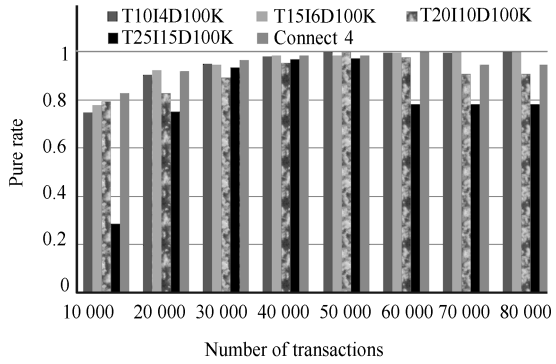
5.2　频繁模式挖掘质量

本小节利用数据集 T10I4D100K, T15I6D100K, T20I10D100K, T25I15D100K 和 Connect 4 对 DFPMiner 和 MSW 的挖掘质量进行对比分析.

图 3 (a) 显示, 数据流初期时, DFPMiner 的挖掘质量不太理想, 但从总的发展趋势上来看, 挖掘质量随着数据流的进行逐渐趋于稳定, 且大部分情况下保持在 90 % 以上. 图 3 (b) 显示, 大部分情况下 MSW 的挖掘质量均处于 80 % 到 90 % 之间, 不存在极端好坏的情况. 相比而言, 两者各有所长.

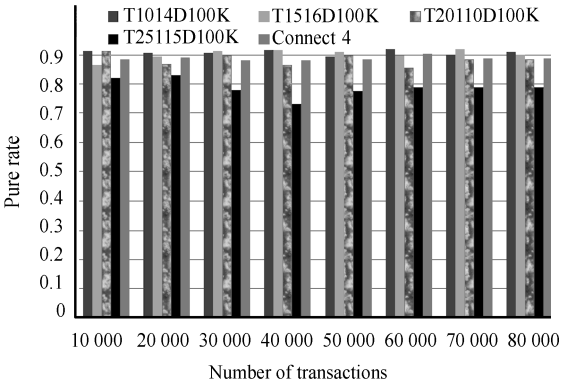
根据纯度 PR 的定义, 由于 $|frePattern_{\text{true}}|$ 相对固定, 所以, $|frePattern_{\text{out}}|$ 直接决定了挖掘质量的高低. 一般而言, 在 DFPMiner 输出的 $frePattern_{\text{out}}$ 中共包含两类模式: 1) 完全衰减支持数小于 D_m 的模式集; 2) 完全衰减支持数大于或等于 D_m 的模式集, 亦即 $frePattern_{\text{true}}$. 不难看出, 如果估计模式完全衰减支持数的准确性越高, 那么在 $frePattern_{\text{out}}$ 中第一类模式的个数也就越少, 从而挖掘质量也就越高.

令模式 P 最近一次的创建时刻为 t_P, 那么 $D(P, t_c)$ 代表时刻 t_P 之后 P 的衰减支持数, 而 $2^{-\lambda(t_c - t_P + 1)} D_l$ 代表时刻 t_P 之前 P 的最大可能的衰减支持数, 所以, DFPMiner 主要采用 $D(P, t_c) + 2^{-\lambda(t_c - t_P + 1)} D_l$ 来估计 $D_a(P, t_c)$. 不难看出, 估计误差上界为 $2^{-\lambda(t_c - t_P + 1)} D_l$. 一般情况下, 存活时间 $t_c - t_P$ 越长的模式, 越有可能成为频繁模式; 而频繁模式相对于其他模式而言, 其存活下去的可能性会更大. 所以, 对于 DFPMiner 输出的 $frePattern_{\text{out}}$ 而言, 该集合包含的模式的存活时间一般会比较长,



(a) DFPMiner 的挖掘纯度

(a) Pure rate of DFPMiner



(b) MSW 的挖掘纯度

(b) Pure rate of MSW

图 3 挖掘纯度

Fig. 3 Pure rate

而且很可能会随着数据流的进行逐渐增长; 又因为 $(t_c - t_P)$ 越长, $2^{-\lambda(t_c-t_P+1)}D_l$ 越小, 所以该集合包含的模式的估计误差很可能会逐渐减小. 根据上述分析, 便可以大致解释 DFPMiner 的挖掘质量变化规律: 在数据流初期时不太理想, 而随数据流的进行逐渐趋于理想、稳定.

对于 MSW, 根据文献 [12] 的定理 1 可知, 滑动窗口内模式真实衰减支持数的估计误差上界为常值 εM, 其中 M 为滑动窗口的大小. 因此, 对于 MSW 输出的 $frePattern_{\text{out}}$ 而言, 该集合包含的模式的估计误差随时间变化没有一般规律可循. 根据上述分析, 便可以解释 MSW 的挖掘质量变化情况: 挖掘质量随数据流进行会出现时好时坏的情况, 但整体上仍能基本保持稳定.

5.3 内存开销

内存开销通过内存中维护的全局模式树的大小来衡量. 首先, 统计 DFPMiner 和 MSW 之间的内存开销比 (Relative memory usage rate, RMUR), 以此来比较两者的空间复杂度. 假设 n_{DFP} 和 n_{MSW} 分别为 DFPMiner 和 MSW 算法维护的全局模式树的大小, 则内存开销比定义为: $RMUR = n_{\text{DFP}}/n_{\text{MSW}}$. 图 4 为算法处理数据集 T10I4D100K, T15I6D100K, T20I10D100K, T25I15D100K 和 Connect 4 的实验结果. 结果显示, $RMUR$ 基本保持在 0.5 到 0.6 之间, 说明 DFPMiner 的空间复杂度明显小于 MSW.

由于内存中需要维护的实际是频繁模式候选集, 所以内存开销的大小与模式周期删除策略的好坏有很大关联: 好的策略能够准确识别并删除预期难以成长为频繁的模式, 从而完整保留频繁模式候选集, 这样便可省去不必要的内存开销. DFPMiner 能够通过对模式成长潜力的理论分析, 准确识别频繁模式候选集; 而 MSW 尽管删除了滑动窗口内的不频繁模式和过期模式, 但由于缺乏对模式成长潜力的考量, 故而仍保留了许多预期难以成长为频繁的模式. 不难看出, MSW 花费的不必要的内存开销远大于 DFPMiner, 这就可能导致前者的内存开销远大于后者.

其次, 统计 DFPMiner 和 NDDFPMiner 的内存开销比 (Memory usage rate, MUR), 以此来衡量模式删除机制减少内存开销的效果. 假设 n_{total} 和 n_{retain} 分别为 NDDFPMiner 和 DFPMiner 算法实时维护的全局模式树的大小, 则内存开销比定义为: $MUR = n_{\text{retain}}/n_{\text{total}}$.

图 5 为算法处理数据集 T10I4D100K, T15I6D100K, T20I10D100K, T25I15D100K 和 Connect 4 的实验结果. 结果显示, MUR 不超过 0.60, 而且随数据流的进行急剧下降, 最低可达到 0.065. 这是由于当数据流中心发生偏移时, 会产生大量没有成长潜力的疏模式, 但模式删除机制能够及时将其删除, 从而减少了内存开销, 并准确地反映当前数据流的演变. 结合第 5.2 节分析可知, 模式删除机制能够在确保挖掘质量的同时显著减少内存开销.

5.4 频繁模式挖掘处理时间

算法效率通过执行时间来衡量. 首先, 统计 DFPMiner 和 MSW 之间的执行时间比 (Relative execution time rate, RETR), 以此来比较两者的时间复杂度. 假设 T_{DFP} 和 T_{MSW} 分别为 DFPMiner 和 MSW 的执行时间, 则执行时间比定义为: $RETR = T_{\text{DFP}}/T_{\text{MSW}}$. 图 6 为算法处理数据集 T10I4D100K, T15I6D100K, T20I10D100K, T25I15D100K 和 Connect 4 的实验结果. 结果显示, $RETR$ 总体上保持在 0.9 到 1 之间, 说明两者效率基本接近, 且 DFPMiner 略优于 MSW.

在整个时间开销中, 尽管 DFPMiner 用于搜索、遍历模式树的时间会比 MSW 短, 但是前者用于剪枝、重建模式树的时间却比后者长. 综合考虑, 两者

的整体时间开销相差不大.

然后, 统计 DFPMiner 和 NDDFPMiner 的执行时间比 (Execution time rate, ETR), 以此来衡量模式删除机制降低时间复杂性的效果. 假设 T_{total} 和 T_{retain} 分别为 NDDFPMiner 和 DFPMiner 的执行时间, 则执行时间比定义为: $ETR = T_{\text{retain}}/T_{\text{total}}$.

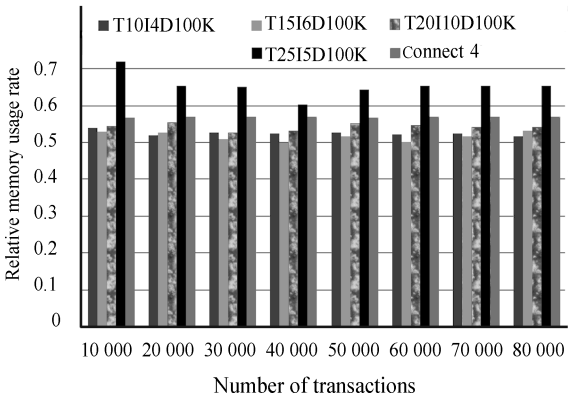


图 4　内存开销比

Fig. 4　Relative memory usage rate

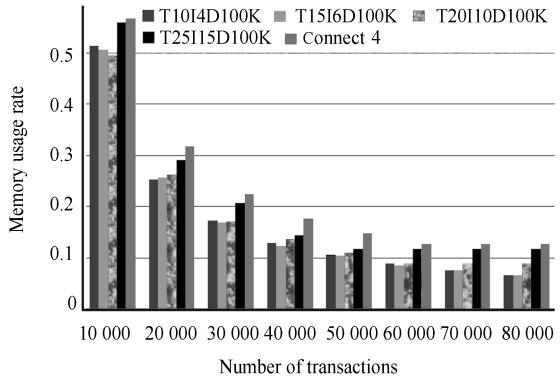


图 5　内存开销比

Fig. 5　Memory usage rate

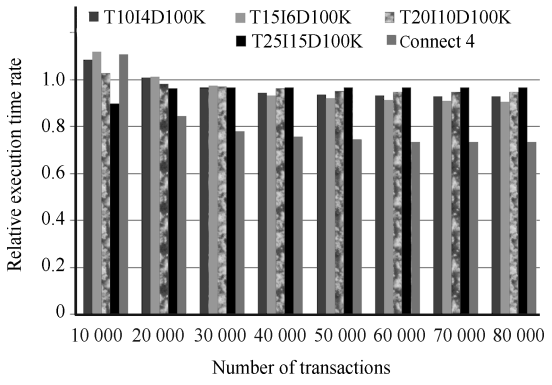


图 6　执行时间比

Fig. 6　Relative execution time rate

图 7 为算法处理数据集 T10I4D100K, T15I6D-100K, T20I10D100K, T25I15D100K 和 Connect 4 的实验结果. 结果显示, 尽管在数据流开始初期 ETR 大于 1, 但随着数据流的进行呈现下降趋势且可达到最低 0.368, 说明删除机制能够较显著地提高算法效率.

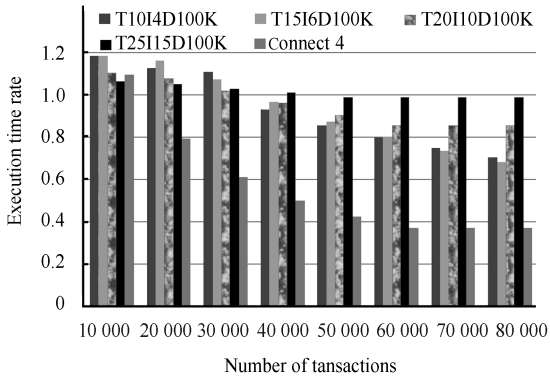


图 7　执行时间比

Fig. 7　Execution time rate

在数据流开始初期, 由于被删除模式的数量不够多, 所以 DFPMiner 用于搜索、遍历模式树的时间与 NDDFPMiner 大致相同, 但是前者却增加了用于剪枝、重建模式树的时间, 所以 ETR 大于 1. 随着时间的推移, 被删除模式的数量逐步增加, 导致前者在搜索、遍历中节省下来的时间开始大于增加的剪枝、重建时间, 所以使得 ETR 小于 1.

最后, 在不同事务数量、不同事务平均长度和不同最大潜在频繁模式平均长度的数据流上来检验 DFPMiner 算法的效率.

从 100 K 到 800 K 变化事务个数, 考察算法的效率. 图 8 (a) 为算法处理 T10I4D100K, T15I6D-100K 和 T20I10D100K 数据集的实验结果. 结果显示, 随着数据流的进行, 执行时间呈线性增长趋势, 且事务的平均处理时间不超过 0.3125 ms, 说明算法性能稳定、高效.

从 10 到 25 变化事务平均长度, 考察效率的变化. 图 8 (b) 为算法处理 T10I4D100K, T15I6D-100K 和 T20I10D100K 数据集 (实验中只处理其中 50 000 个事务) 的实验结果. 结果显示, 随着事务平均长度的增加, 执行时间呈现非线性增长趋势. 这是由于当平均事务长度增加时, 全局模式树的空间复杂度会随之增加, 从而导致维护时间代价相应增大, 因此效率会随之降低.

从 4 到 15 变化最大潜在频繁模式平均长度, 考察效率的变化. 图 8 (c) 为算法处理 T10I4D100K, T15I6D100K 和 T20I10D100K 数据集 (实验中只处理其中 50 000 个事务) 的实验结果. 结果显示, 随

着最大潜在频繁模式平均长度的增加, 执行时间呈现下降趋势. 这是由于全局模式树中的疏模式数量与最大潜在频繁模式平均长度成正比, 根据模式删除机制原理可知, 实时维护的模式树的空间复杂度与平均长度成反比, 所以平均长度越大, 维护代价也就越小, 效率也就随之提高.

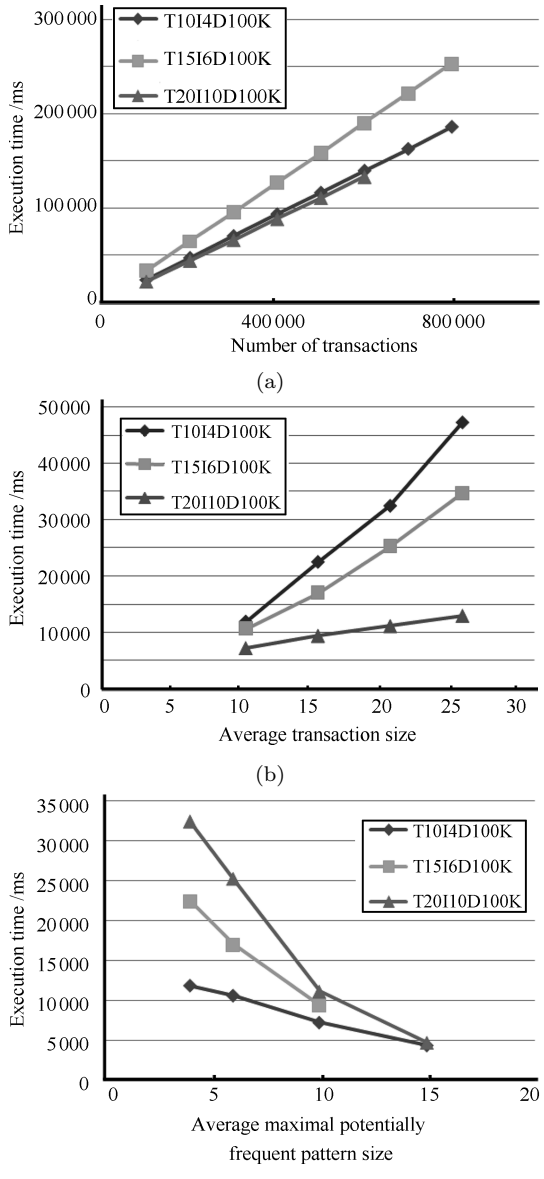


图 8 执行时间随事务数量 \事务平均长度 \最大潜在频繁模式平均长度变化

Fig. 8 Execution time vs. number of transactions\average transaction size\average maximal potentially frequent pattern size

5.5 参数敏感性分析

算法中一个重要的参数就是衰减因子 λ, 它反映了历史事务对当前挖掘结果的影响程度, 并直接影响到系统的运行性能: 当 λ 减小时, 则衰减速度慢, 使得模式的衰减支持数相应增加, 因此历史事务的影响程度也就相应提高; 倘若衰减支持数阈值 D_l 和 D_m 不变, 则会导致过去相同时间段内的非疏模式的数量相应增加, 最终加大了系统的内存开销和执行时间. 实验中, 通过设置 λ 在 0.00001 到 0.0005 之间取值, 来观察频繁模式挖掘质量、内存开销以及执行时间的变化情况 (实验中只处理其中 50 000 个事务).

图 9 显示算法处理 T10I4D100K, T15I6D100K, T20I10D100K, T25I15D100K 和 Connect 4 数据集的实验结果. 为便于考察变化规律, 图 9 (b) 和

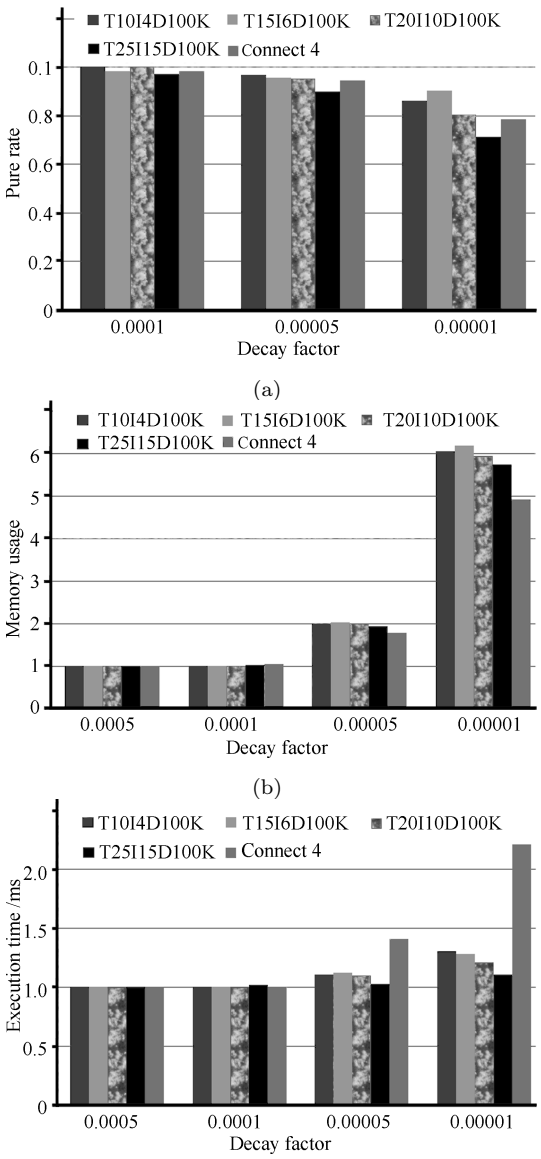


图 9 纯度 \内存开销 \执行时间随衰减因子变化

Fig. 9 Pure rate\memory usage\execution time vs. decay factor

图 9 (c) 中的实验数据, 均为真实实验结果同 $\lambda = 0.0005$ 时的真实实验结果的比值. 由图 9 观察发现, 当 λ 减小时, 挖掘质量随之呈现出较明显的下降趋势, 同时内存开销和执行时间则呈现出明显的非线性上升趋势, 这说明较小的 λ 会导致系统的挖掘质量和运行性能一致降低, 所以, 设置该参数时应在其取值范围之内尽可能大.

算法中另一个重要的参数就是观测衰减支持数阈值 D_l 和 D_m. D_l 的大小决定了算法实时维护的非疏模式的数量, 较大的 D_l 意味着较少的非疏模式, 因此系统的内存开销和执行时间均会随之减少; 另外, 当 D_l 变大时, 对于相同的 D_m 而言, NDDFPMiner 输出的频繁模式的数量保持不变, 而非疏模式的减少导致了 DFPMiner 输出的频繁模式的数量只可能减少, 又由于算法不会产生漏报, 所以算法的挖掘质量会随之提高. D_m 的大小对于非疏模式的数量没有影响, 仅决定了频繁模式的数量; 当 D_l 保持不变时, 较大的 D_l 会导致 NDDFPMiner 和 DFPMiner 输出的频繁模式的数量同时减少, 因此, D_m 与挖掘质量间不存在单调变化关联. 综上分析, 设置 D_m 在 10～50 之间取值, 主要观察挖掘质量的变化情况; 设置 D_l 在 0.1～10 之间取值, 重点考察内存开销以及执行时间的变化情况 (实验中只处理其中 50 000 个事务).

图 10 显示算法处理 T10I4D100K, T15I6D10-0K, T20I10D100K, T25I15D100K 和 Connect 4 数据集的实验结果. 为便于考察变化规律, 图 10 (b) 和图 10 (c) 中的实验数据, 均为真实实验结果同 $D_l = 0.1$ 时的真实实验结果间的比值. 图 10 (a) 显示, 挖掘质量在总体上保持平稳, 与 D_m 之间没有一般规律可循; 由图 10 (b) 和图 10 (c) 观察发现, 内存开销和执行时间则呈现出明显的非线性下降趋势, 加上前面的分析, 共同说明较大的 λ 有助于提高系统的运行性能和挖掘质量. 所以当其他参数取定时, 根据公式 $0 < \lambda \leq \log_2(D_m/D_l) \cdot 2/2T_{\text{care}} - 1 - D_m$, 设置该参数时应在其取值范围之内尽可能大.

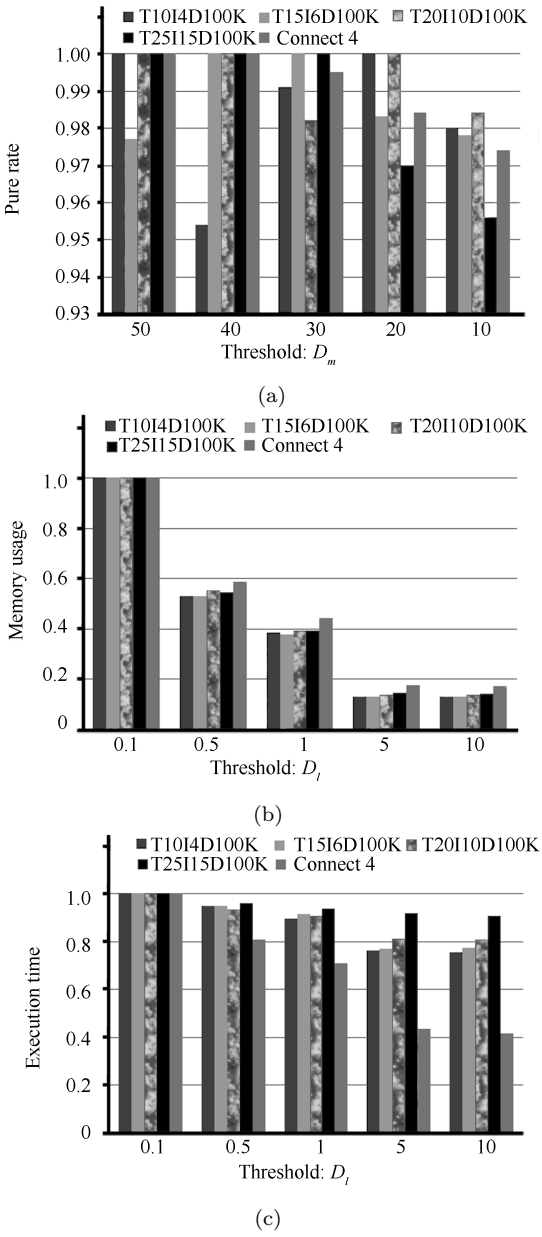


图 10　纯度 \内存开销 \执行时间随阈值变化

Fig. 10　Pure rate\memory usage\execution time vs. threshold

6　结论

针对数据流的时效性和流中心的偏移性特点, 提出了界标窗口模型与时间衰减模型相结合的数据流频繁模式挖掘算法. 定义了时间衰减模型下若干基本问题, 包括模式衰减支持数的概念、界标窗口内模式频繁程度的界定标准和模式树的结构特征. 在此基础上, 通过动态构建全局模式树, 用其中统计的模式的衰减支持数, 刻画其在界标窗口内的频繁程度; 根据模式统计特征的历史演变过程, 设计了界标窗口内疏模式的周期删除策略, 并结合理论推导出删除周期、剪枝阈值函数, 对预期难以成长为频繁的模式及时加以删除, 从而有效降低了空间开销. 一系列实验表明, 与现有同类算法 MSW 相比, 该算法挖掘精度高 (平均超过 90 %), 内存开销小, 速度上可以满足高速数据流的处理要求, 且可以适应不同事务数量、不同事务平均长度和不同最大潜在频繁模式平均长度的数据流频繁模式挖掘.

References

1 Babcock B, Babu S, Datar M, Motwani R, Widom J. Models and issues in data stream systems. In: Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on

Principles of Database Systems. Madison, USA: ACM, 2002. 1−16

2 Han J, Kamber M [Author], Fan Ming, Meng Xiao-Feng [Translator]. *Data Mining: Concepts and Techniques*. Beijing: China Machine Press, 2007
(Han J, Kamber M [著], 范明, 孟小峰 [译]. 数据挖掘: 概念与技术. 北京: 机械工业出版社, 2007)

3 Pan Yun-He, Wang Jin-Long, Xu Cong-Fu. State-of-the-art on frequent pattern mining in data streams. *Acta Automatica Sinica*, 2006, **32**(4): 594−602
(潘云鹤, 王金龙, 徐从富. 数据流频繁模式挖掘研究进展. 自动化学报, 2006, **32**(4): 594−602)

4 Manku G S, Motwani R. Approximate frequency counts over data streams. In: Proceedings of the 28th International Conference on Very Large Data Bases. Hong Kong, China: Morgan Kanfmann, 2002. 346−357

5 Arasu A, Manku G S. Approximate counts and quantiles over sliding windows. In: Proceedings of the 23rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. Paris, France: ACM, 2004. 286−296

6 Cormode G, Muthukrishnan S. What′s hot and what′s not: tracking most frequent items dynamically. In: Proceedings of the ACM SIGMOD/PODS Conference. San Diego, USA: ACM, 2003.

7 Jin C, Qian W, Sha C, Yu J X, Zhou A. Dynamically maintaining frequent items over a data stream. In: Proceedings of the 12th International Conference on Information and Knowledge Management. New Orleans, USA: ACM, 2003. 287−294

8 Chang J H, Lee W S. Finding recent frequent itemsets adaptively over online data streams. In: Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Washington D. C., USA: ACM, 2003. 487−492

9 Chang J H, Lee W S. estWin: adaptively monitoring the recent change of frequent itemsets over online data streams. In: Proceedings of the 12th International Conference on Information and Knowledge Management. New Orleans, USA: ACM, 2003. 536−539

10 Leung C K S, Khan Q I. DStree: a tree structure for the mining of frequent sets from data streams. In: Proceedings of the 6th International Conference on Data Mining. Hong Kong, China: IEEE, 2006. 928−932

11 Giannella C, Han J, Pei J, Yan X, Yu P S. Mining frequent patterns in data streams at multiple time granularities. *Data Mining: Next Generation Challenges and Future Directions*. Sri Lanka: AAAI Press, 2004. 191−212

12 Li Guo-Hui, Chen Hui. Mining the frequent patterns in an arbitrary sliding window over online data streams. *Journal of Software*, 2008, **19**(10): 2585−2596
(李国徽, 陈辉. 挖掘数据流任意滑动时间窗口内频繁模式. 软件学报, 2008, **19**(10): 2585−2596)

13 Mendes L F, Ding B, Han J. Stream sequential pattern mining with precise error bounds. In: Proceedings of the 8th IEEE International Conference on Data Mining. Pisa, Italy: IEEE, 2008. 941−946

14 Cao F, Ester M, Qian W, Zhou A. Density-based clustering over an evolving data stream with noise. In: Proceedings of the 6th SIAM International Conference on Data Mining. Bethesda, USA: SIAM, 2006. 328−399

15 Blake C L, Merz C J. UCI repository of machine learning databases [Online], available: http://www.ics.uci.edu/∼mlearn/ML repository.html, November 7, 2007

16 Coenen F. LUCS-KDD DN software [Online], available: http: // www.csc.liv.ac.uk/ ∼frans/KDD/software/LUCS_KDD_DN/, November 7, 2007

吴 枫 国防科学技术大学计算机学院博士研究生. 主要研究方向为数据挖掘, 网络安全和人工智能. 本文通信作者.
E-mail: wfttyy_2000@163.com
(**WU Feng** Ph. D. candidate at the School of Computer, National University of Defense Technology. His research interest covers data mining, network security, and artificial intelligence. Corresponding author of this paper.)

仲 妍 国防科学技术大学计算机学院博士研究生. 主要研究方向为高性能计算和数据挖掘.
E-mail: mandy_zh82 @163.com
(**ZHONG Yan** Ph. D. candidate at the School of Computer, National University of Defense Technology. Her research interest covers high performance computing and data mining.)

吴泉源 国防科学技术大学计算机学院教授. 主要研究方向为分布计算和人工智能.
E-mail: mandy_zh82@hotmail.com
(**WU Quan-Yuan** Professor at the School of Computer, National University of Defense Technology. His research interest covers distributed computing and artificial intelligence.)