计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2021.20200554
58(11): 2500-2514, 2021

基于滑动窗口模型的数据流闭合高效用项集挖掘

程浩东 韩 萌 张 妮 李小娟 王 乐
(北方民族大学计算机科学与工程学院 银川 750021)
(734811467@qq.com)

Closed High Utility Itemsets Mining over Data Stream Based on Sliding Window Model

Cheng Haodong, Han Meng, Zhang Ni, Li Xiaojuan, and Wang Le
(*College of Computer Science and Engineering, North Minzu University, Yinchuan* 750021)

Abstract It is a challenging task to mine high utility itemsets from the data stream, because the incoming data stream must be processed in real time within the constraints of time and storage memory. Data stream mining usually generates a large number of redundant itemsets. In order to reduce the number of these useless itemsets and ensure lossless compression of complete high utility itemsets, it is necessary to mine closed itemsets, which can be several orders of magnitude smaller than the collection of complete high utility itemsets. In order to solve the above problem, a high utility itemsets mining algorithm (sliding-window-model-based closed high utility itemsets mining on data stream, CHUI_DS) is proposed to achieve mining closed high utility itemsets on data stream. A new utility-list structure is designed in CHUI_DS, which is very effective in increasing the speed of batch insertion and deletion. In addition, effective pruning strategies are applied to improve the closed itemset mining process and eliminate potential low-utility candidates. Extensive experimental evaluation of the proposed algorithm on real datasets and synthetic datasets shows the efficiency and feasibility of the algorithm. In terms of speed, it is superior to the previously proposed algorithms that mainly run in batch mode. Moreover, it is suitable for sliding windows of different sizes, and has strong scalability in terms of the number of transactions.

Key words pattern mining; data stream mining; closed high utility itemsets; sliding window; utility list

摘 要 从数据流中挖掘高效用项集是一项具有挑战性的任务,因为传入的数据必须在时间和存储内存约束下进行实时处理.数据流挖掘通常会产生大量冗余的项集,为了减少这些无用的项集数量且保证无损压缩,需要挖掘闭合项集,它可以比全集高效用项集的集合小几个数量级.为了解决以上问题,提出一种基于滑动窗口模型的数据流闭合高效用项集挖掘(closed high utility itemsets mining over data stream based on sliding window model, CHUI_DS)算法. 在CHUI_DS中设计了一种新的效用列表结构,该结构在提升批次插入和删除的速度方面非常有效.此外,应用修剪策略来改进闭合项集挖掘过程,消除潜在的低效用候选对象.对真实数据集和合成数据集进行的广泛实验评估显示了该算法的效率

收稿日期:2020-07-14;**修回日期**:2021-01-18
基金项目:国家自然科学基金项目(62062004);宁夏自然科学基金项目(2020AAC03216);北方民族大学研究生创新项目(YCX20077)
This work was supported by the National Natural Science Foundation of China (62062004), the Natural Science Foundation of Ningxia Hui Autonomous Region of China (2020AAC03216), and the Graduate Innovation Project of North Minzu University (YCX20077).
通信作者:韩萌(2003051@nmu.edu.cn)

以及可行性.就速度而言,它优于先前提出的主要以批处理模式运行的算法.且它适用于不同大小的滑动窗口,在事务数量等方面具有较强的扩展性.

关键词 模式挖掘;数据流挖掘;闭合高效用项集;滑动窗口;效用列表

中图法分类号 TP311

高效用项集挖掘(high utility itemset mining, HUIM)是一项经过广泛研究的数据挖掘任务[1],它通过考虑一些其他因素扩展了频繁项集挖掘(frequent itemset mining, FIM)[2].FIM 仅将项集出现的频率作为其在数据库中的重要性度量,不考虑每个项目的效用价值.但是,HUIM 则同时会考虑项目的效用或实用价值.在高效用项集挖掘领域,交易事务中物品的数量称为内部效用,项目的利润被称为外部效用.项目的效用被定义为物品数量和利润(即内部效用和外部效用)的乘积.挖掘高效用项集(high-utility itemset, HUI)的问题可以描述为找到所有效用值大于或等于用户自定义阈值的项集,这个阈值称为最小效用阈值(*minutil*).较早的算法[3-6]被设计为在 2 个阶段挖掘高效用项集,其后又引入 1 阶段算法[7-14].

高效用项集在市场篮分析、产品推荐等实际应用中效果显著,但静态数据库的高效用项集包含了旧数据的影响.近年来,无线传感器网络、零售市场交易、通信网络和网站在线点击等场景都在生成数据流.挖掘来自数据流的项集更加有用,因为它包含基于最新数据且不受旧数据影响的项集或模式,用户可快速获取相关信息以采取适当的措施.例如,考虑在企业环境中部署多个 WiFi 接入点的场景[15],HUIM 可以用于该网络中的实时负载均衡.每个事务被建模为一天中访问点的使用情况,其中将访问点定义为项目,并且将连接到其的客户端负载表示为访问点的内部效用,将数据传输到服务器的成本定义为与其关联的外部效用.访问点的效用可以定义为负载和成本的乘积,近而实时识别高负载访问点集,此类信息可用于负载的重新分配或调控.

文献[16-18]中已经提出了算法用于从数据流中挖掘高效用项集,但这些算法均属于全集高效用项集挖掘算法.全集项集挖掘算法的固有问题是生成大量模式信息冗余的项集,难以被用户理解和运用.因此,一些压缩或紧凑的 HUI 表示方法被提出.静态高效用项集挖掘领域已提出一种称为闭合高效用项集(closed high utility itemset, CHUI)的概念[19],它的大小比高效用项集全集小几个数量级,且不丢失任何有用信息.相对于比较有限的内存空间,CHUI 挖掘能更好地满足用户需求,开展此类挖掘的研究具有重要意义.仍以 WiFi 场景为例,假设在相同日期内某一区域访问点 A 和 B 均处于高负载状态,且$\{A\}$,$\{A,B\}$都是 HUI,则$\{A,B\}$即为需要挖掘的 CHUI.显然该项集减少了冗余信息,在准确识别所有高负载访问点的基础上,后者为企业按区域优化网络配置提供更佳方案,向运营企业提供更有意义的决策信息.现有文献已提出多种有效挖掘 CHUI 的算法,比如 CHUD[19],CHUI-Miner[20],EFIM-Closed[21],CLS-Miner[22] 等.而后 Dam 等人[23]针对动态增量数据库提出新的闭合算法 IncCHUI.

类似于静态数据和增量数据,数据流中同样迫切需要进行 CHUI 挖掘.由于数据流的海量性等特点,造成对于它的挖掘和存储都有一定难度.同时,随着时间的不断推移,用户通常只关注数据流中近期有价值的信息.所以已提出的静态和增量闭合高效用项集挖掘算法并不能直接适用于数据流挖掘任务.

为此,本文设计了第 1 种基于滑动窗口的数据流闭合高效用项集挖掘算法,以解决数据流全集高效用项集挖掘带来的项集信息冗余问题,而这一主题在此之前尚未被探讨.这项工作的贡献总结为:

1) 提出一个名为 CH-List 的新颖数据结构,它可以有效地计算内存中项集的效用和支持数,而无需重复扫描原始数据流.CH-List 相较于传统闭合 EU_List[20],IUL(incremental utility-list)[23] 等结构,适用于数据流滑动窗口,在批次的插入和删除方面非常有效.

2) 提出了一种新的融合 CH-List 结构的算法,即基于滑动窗口模型的数据流闭合高效用项集挖掘(closed high utility itemsets mining over data stream based on sliding window model, CHUI_DS)算法.该算法采用分而治之的思想,通过创新一种事务重组方法以及改进的闭包挖掘技术,近而在数据流滑动窗口中挖掘出完整的 CHUI.

3) 对真实数据集和合成数据集进行了广泛的实验研究,以评估所提出算法的性能.实验结果表明:

CHUI_DS 的总体性能优于目前最先进的相关批处理和增量算法.此外,算法在不同窗口环境下的可扩展性也得到有效验证.

1 相关工作

高效用项集挖掘的缺点是算法返回的结果集数量巨大,这使得分析这些结果集成为一项艰巨的任务,同时它们包含很多冗余项集信息.针对以上问题,Tseng 等人[19]提出了一种紧凑且无损的高效用项集表示形式,如果项集的效用不小于用户指定的最小效用阈值,并且不存在相同支持数的真超集,则该项集称为闭合高效用项集.每个 CHUI 都由一种称为效用单元阵列的特殊结构进行注释,采用这种结构,高效用项集全集得以在无需访问原始数据库的情况下从该集合派生出来,因此 CHUI 的集合是无损的.另外,研究者提出了第 1 种发现静态数据库中 CHUI 的算法,名为 CHUD[19],以及一种名为 DAHU 的方法从完整的 CHUI 集中恢复所有 HUI. Wu 等人[20]提出了一种名为 CHUI-Miner 的新颖算法,该算法依赖 Utility-List 垂直列表结构以单阶段发现 CHUI.通过该结构的剩余效用(remaining utility, RU)属性,可以获得项集超集更严格的效用上限,从而可以较大范围地修剪搜索空间.Fournier-Viger 等人[21]提出了一种名为 EFIM-Closed 的闭合高效用项集挖掘算法.该算法包括 3 种修剪策略,分别为闭合跳跃、向前闭合检查和向后闭合检查,以修剪非闭合的 HUI.此外,它还引入了事务投影和事务合并机制.实验结果表明:与 CHUD 算法相比,EFIM-Closed 运行快 1 个数量级以上,并且消耗的内存少 1 个数量级.Dam 等人[22]提出一种称为 CLS-Miner 的算法,应用链估计效用共现修剪、较低分支修剪和按覆盖范围修剪等新颖剪枝策略减少搜索空间,提出的 CLS-Miner 算法优于 CHUD 和 CHUI-Miner 算法.

在增量数据环境中,同样针对上段描述的相关问题,研究者提出 IncCHUI[23]算法,其使用增量效用列表结构从数据库中挖掘 CHUI.IncCHUI 仅扫描原始或更新的数据库 1 次,以构造单个项的列表.使用称为 CHT 的散列表存储目前为止发现的 CHUI.对于在更新的数据库上挖掘时发现的每个闭合高效用项集 P,算法首先检查该项集是否已在 CHT 中,再决定是否需要将 P 插入表 CHT 中.这是目前首个对增量数据库进行闭合高效用项集挖掘的算法.

在数据流上挖掘 HUI 比在静态或增量数据库中更具挑战性.因为传入的数据流必须在时间和存储内存约束下进行实时处理.目前有 3 种主要的流处理模型[24-25]:1)阻尼窗口模型;2)地标窗口模型;3)滑动窗口模型.其中窗口是 1 组连续交易,被视为数据流中的单位.在滑动窗口模型[26-27]中,方法使用固定大小窗口中的最新数据来发现数据流中有意义的项集.该模型因能够强调最新数据并占用较小内存资源而被广泛用于数据流挖掘.为考虑资源受限环境中真实的数据流挖掘任务,Ahmed 等人[16]提出了基于滑动窗口的 HUIM.为了满足向下闭包特性,早前基于滑动窗口的研究采用了事务加权效用(transaction weighted utility, TWU)高估概念[28],缺点是导致生成大量候选项集,处理这些候选项集需要更多的执行时间.Ryang 等人[17]提出一种树结构的高效用挖掘算法 SHU-Grow,用于从滑动窗口模型中挖掘 HUI.此外,其还开发了 2 种技术:降低的全局估计效用(reducing global estimated utilities, RGE)和降低的局部估计效用(reducing local estimated utilities, RLE),以取代 TWU 高估方法来减少搜索空间和候选项集.上述在数据流上挖掘高效用项集的算法是 2 阶段的.随后 Jaysawal 等人[18]提出一种有效的单程 1 阶段算法 SOHUPDS 来在数据流上挖掘 HUI,借助 IUDataListSW 结构,存储项目在事务中的位置和实际效用,以及可以从项目扩展项集效用上限,它比之前的算法更加有效.

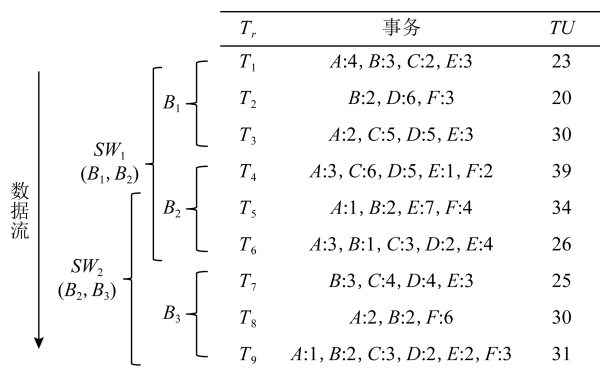
一些工作还提出了在数据流上挖掘 HUI 的衍生算法.Zihayat 等人[29]提出了一种称为 HUDS-tree 的树状结构,树中每个节点逐批维护高估的效用值.采用基于滑动窗口模型的 2 阶段算法 T-HUDS 挖掘前 k 个 HUI,它在第 1 阶段生成候选的高效用项集,在第 2 阶段验证项集的确切效用.Dawar 等人[15]提出一种称为 Vert_top-k_DS 的算法,该算法可在不生成任何候选项的前提下挖掘出前 k 个 HUI.其在动态阈值提升策略的基础上,利用 k-项集在公共批次的实际效用值增加窗口滑动时的初始阈值,是目前最先进的数据流 top-k 高效用项集挖掘算法.在文献[30-31]中提出的算法用于在数据流上挖掘高效用序列模式(high utility sequence pattern, HUSP),前者提出的 HUSP-Stream 是在数据流上查找 HUSP 的第 1 种方法,后者提出了一种名为 HUSP-UT 的新挖掘算法,实验性能大幅优于 HUSP-Stream.

2 定　　义

设由不同项构成的有限集合 $I^*=\{I_1,I_2,\cdots,I_m\}$,其中每个项 $I_i\in I^*$ 都与 1 个正数 $p(I_i,DS)$ 相关联,称为外部效用.令数据流 $DS=(T_1,T_2,\cdots,T_s)$是有限事务组成的序列,数据流 DS 中的每个事务 $T_r(1\leqslant r\leqslant s)$表示第 r 个到达的事务,其中每个事务 $T_r\in DS$ 是 I^* 的子集,并且具有唯一的标识符 r,称为其 Tid.在事务 T_r 中,每一项 $I_i\in I^*$ 都与 1 个正数 $q(I_i,T_r)$相关联,在 T_r 中称为其内部效用(例如购买数量).项集 P 是由 1 组属于 I^* 的项 $\{I_1,I_2,\cdots,I_k\}$构成的集合,P 的长度表示为 k,其中 $P\subseteq I^*$ 且 $1\leqslant k\leqslant m$.

在滑动窗口模型中,每个窗口 SW_k 由固定数量的 n 个最近的批次组成的有序集合,可以表示为 $SW_k=(B_j,B_{j+1},\cdots,B_n)$,该窗口大小($winSize$)为 n.批次 B_j 包含固定数量的事务,即 $B_j=(T_1,T_2,\cdots,T_r)$.窗口 SW_{k-1}滑动时,最旧的批次被从窗口移除出去,最新的一个批次将被添加进来,这个新生成的窗口即为 SW_k.其中 $B_{j+1},B_{j+2},\cdots,B_{n-1}$称为 SW_{k-1}和 SW_k 的公共批次($CommomBatches$).

图 1 显示了一个数据流的示例,示例中的每一行代表 1 个事务,事务中每个字母代表 1 个项目,并在右侧附其内部效用.表 1 列出了每个项目的外部效用.该数据流分为 3 个批次 B_1,B_2,B_3,其中每个滑动窗口由 2 个批次组成.示例中共存在 2 个滑动窗口,$SW_1=\{B_1,B_2\}$和 $SW_2=\{B_2,B_3\}$.在此,SW_1 是初始滑动窗口.随着新数据的到达,由于第 1 个窗口已满,通过移除最旧的批次并插入新批次后,SW_1 更新为滑动窗口 SW_2.



T_r	事务	TU
T_1	A:4, B:3, C:2, E:3	23
T_2	B:2, D:6, F:3	20
T_3	A:2, C:5, D:5, E:3	30
T_4	A:3, C:6, D:5, E:1, F:2	39
T_5	A:1, B:2, E:7, F:4	34
T_6	A:3, B:1, C:3, D:2, E:4	26
T_7	B:3, C:4, D:4, E:3	25
T_8	A:2, B:2, F:6	30
T_9	A:1, B:2, C:3, D:2, E:2, F:3	31

Fig. 1　Example of stream data

图 1　数据流示例

Table 1　Profit Table

表 1　外部效用表

项名	外部效用
A	2
B	1
C	3
D	1
E	2
F	4

定义 1. 项的效用.事务 T_r 中项 $I_i\in I^*$ 的效用表示为 $U(I_i,T_r)$,并定义为 $p(I_i,D)\times q(I_i,T_r)$.在一个批次 B_j 中,$U(I_i,B_j)=\sum_{T_r\in B_j}U(I_i,T_r)$.在一个窗口 SW_k 中,$U_{SW_k}(I_i)=\sum_{B_j\in SW_k}U(I_i,B_j)$.

定义 2. 项集的效用.在事务 T_r 中的 k 项集 $X=\{I_1,I_2,\cdots,I_k\}$的效用表示为 $U(X,T_r)$并定义为 $\sum_{I_i\in X}U(I_i,T_r)$.在一个批次 B_j 中,$U(X,B_j)=\sum_{T_r\in B_j}U(X,T_r)$.

在一个窗口 SW_k 中,$U_{SW_k}(X)=\sum_{B_j\in SW_k}U(X,B_j)$.

定义 3. 事务效用.事务 T_r 的事务效用被表示为 $TU(T_r)$,并且被定义为 $\sum_{\forall I_i\in T_r}U(I_i,T_r)$.

例如,T_1 的事务效用为 $TU(T_1)=U(\{ABCE\},T_1)=23$.

定义 4. 超集和子集.设非空项集 X 和 Y.X 是 Y 的子集,如果 $X\subseteq Y$,则等效地 Y 是 X 的超集.如果 $X\subset Y$,则 X 是 Y 的真子集,而 Y 是 X 的真超集.

定义 5. $TidSet$ 和支持计数.项集 X 的 $TidSet$ 表示为 $TidSet_{SW_k}(X)=\bigcup_{X\subseteq T_r\cap T_r\in SW_k}r$,并定义为窗口 SW_k 中包含 X 的所有事务的 Tid 集合.项集 X 的支持计数是 DS 中某窗口 SW_k 包含 X 的事务数,并表示为 $SUP_{SW_k}(X)$,用 $TidSet_{SW_k}(X)$表示为 $SUP_{SW_k}(X)=|TidSet_{SW_k}(X)|$.

例如,$SUP_{SW_1}(\{ACE\})=|TidSet_{SW_1}(\{ACE\})|=|\{1,3,4,6\}|=4$.

属性 1. 对窗口 SW_k 中的 k 项集 X,$SUP_{SW_k}(X)=|TidSet_{SW_k}(I_1)\cap TidSet_{SW_k}(I_2)\cap\cdots\cap TidSet_{SW_k}(I_k)|$.

属性 2. 对窗口 SW_k,令项集 Y 为 X 的真超集,则有 $TidSet_{SW_k}(Y)\subseteq TidSet_{SW_k}(X)$.

定义 6. 闭合项集.如果 SW_k 中不存在真超项集 $Y\supset X$ 使得 $SUP_{SW_k}(X)=SUP_{SW_k}(Y)$,则项集 X 在窗口 SW_k 中称为闭合项集.闭合项集的完整集合表示为 C_{SW_k}.

定义 7. 项集的闭包.令 Y 为项集 X 的超集.如果 Y 是闭合的且 $SUP_{SW_k}(Y)=SUP_{SW_k}(X)$,则 Y 称为 X 在 SW_k 中的闭包.X 的闭包定义为 $closure(X)=\bigcap_{r\in TidSet_{SW_k}(X)} T_r$.

例如,窗口 SW_1 中,$closure(\{AB\})=T_1\cap T_5\cap T_6=\{ABCE\}\cap\{ABEF\}\cap\{ABCDE\}=\{ABE\}$.

定义 8. 批次总效用.滑动窗口 SW_k 中的某个批次的总效用 $BTU(B_i)=\sum_{T_r\in B_i} TU(T_r)$.

定义 9. 总效用.滑动窗口 SW_k 的总效用表示为 $TotalUtility$,并定义为

$$TotalUtility_{SW_k}=\sum_{B_j\in SW_k} BTU(B_i).$$

定义 10. 高效用项集(HUI).如果窗口 SW_k 中的项集 X 的效用 $U_{SW_k}(X)\geqslant minutil$,则在 SW_k 中将 X 称为高效用项集.

例如,窗口 SW_1 中$\{BE\}$的效用为 $U_{SW_1}(BE)=U(BE,B_1)+U(BE,B_2)=9+25=34$.$SW_1$ 中事务的总效用为 172.若 $minutil=30.96$,则$\{BE\}$在 SW_1 中是高效用项集,因为其效用 $U_{SW_1}(BE)=34>minutil$.

定义 11. 闭合高效用项集(CHUI).如果项集 P 是 SW_k 中的高效用项集且满足 $P\subseteq C_{SW_k}$,则 P 为闭合高效用项集.

属性 3. 对于任何高效用项集 X,都存在一个闭合高效用项集 Y,使得 $Y=closure(X)$并且 $U(Y)\geqslant U(X)$.

属性 4. 对于不是闭合高效用项集的任何项集 X,其所有子集都是低效用的.

定义 12. 项集的事务加权效用(TWU).项集 X 的 TWU 是当前窗口 SW_k 下所有包含 X 的事务的事务效用之和,它表示为 $TWU_{SW_k}(X)$,定义为 $\sum_{X\subseteq T_r\cap T_r\in SW_k} TU(T_r)$.

例如,$TWU_{SW_1}(AC)=TU(T_1)+TU(T_3)+TU(T_4)+TU(T_6)=23+30+39+26=118$.

定义 13[28]. 事务权重使用率向下封闭(TWDC)属性.对于任何项集 X,如果其 $TWU<minutil$,则 X 的所有超集都是低效用的.扫描数据库 1 次后,将获得项目列表及其 TWU 值.通过 TWDC 属性,如果项目的 $TWU<minutil$,则其所有超集都不是 CHUI.这些项目称为无前途的项目.

定义 14. 窗口中的高事务加权效用项集.如果 $TWU_{SW_k}(X)\geqslant minutil$,则项集 X 称为窗口 SW_k 的高事务加权效用项集.

属性 5[28]. 使用 TWU 修剪.假设有一个项集 X 不是高 TWU 项集,则 X 及其超集均是低效用项集.

对于数据流中的滑动窗口 SW_k,问题是要在 SW_k 中找到所有 CHUI,其中 $minutil$ 是用户定义的参数.

3 CHUI_DS 算法

本节提出一种新的效用列表结构 CH-List 和基于滑动窗口模型的数据流闭合高效用项集挖掘算法 CHUI_DS,利用滑动窗口在数据流中挖掘闭合项集是为了在有限的内存资源下从连续数据中及时发现最新的无冗余项集.此外,算法开发了 2 种技术:基于批次剩余效用表(batch based remaining utility table, BRU_table)的公共批次事务重组方法和减少闭包计算的效用剪枝技术,以通过高估候选闭包项集的效用来减少搜索空间.在描述本节的数据结构之前,先介绍一些术语.

定义 15. 重组事务.若 $T_r(T_r\in SW_k)$中存在的项按其 TWU 被升序排列,则此时的事务称为重组事务 T'_r.

定义 16. 给定项集 X 和 $X\subseteq T'_r$,在 T'_r 中,X 之后的所有项的集合记为 T'_r/X.

定义 17. 交易中项集的剩余效用.在重组交易 T'_r 中,k 项集 $X=\{I_1,I_2,\cdots,I_k\}$的剩余效用表示为 $RU(X,T'_r)$,是 T'_r/X 中所有项目的效用之和.

定义 18. 项集的扩展.设 $X=\{x_1,x_2,\cdots,x_v\}$ $(X_i\in I^*,1\leqslant i\leqslant v)$和 $Y=\{y_1,y_2,\cdots,y_u\}$ $(y_j\in I^*,1\leqslant j\leqslant u)$为 2-项集.如果 $X\subset Y$ 并且每个项在 TWU 升序排序中都在 X 的项之后,则 Y 是 X 的扩展.例如,图 1 中$\{CAE\}$并非$\{CE\}$的扩展,而是$\{CA\}$的扩展.因为根据表 2 示出的滑动窗口 SW_1 中各项的 TWU,以 TWU 的升序排列的项的顺序为 $F\prec B\prec D\prec C\prec A\prec E$.在此,$\{A\}$排在$\{E\}$之前.

Table 2 *TWU* Information of Items in SW_1

表 2 SW_1 中项的 *TWU* 信息

项名	TWU
A	152
B	103
C	118
D	115
E	152
F	93

3.1 数据结构 BRU_table 和公共批次事务重组

本文提出一种名为 BRU_table 的双重散列表,该结构是在对当前窗口第 1 次事务扫描时被构建或更新的,包含了当前窗口的所有批次,并为每个批次给予一个唯一的编号,其值为该批次所包含的事务 Tid 和该事务对应的 BRU,BRU 初始值为每个事务的事务效用.随着窗口的滑动,批次更新的同时散列表的项也会随之更新.待新批次到来,在移除各项 CH-List 旧批次信息时同步移除其批次的 BRU_table 项.

BRU_table 作为临时存储结构,为新窗口的事务重组提供了必要信息.因为随着新批次的到来,项的 TWU 和顺序已经发生变化,根据属性 6,剩余效用修剪策略将无法正确对新搜索空间进行剪枝.需要根据当前事务元组信息和 TWU 顺序重构前一窗口和新窗口的 CH-List 公共批次事务元组.从更新后 TWU 最小的项开始遍历公共批次中所有 CH-List,并相应地更新 RU 值.在遍历时,使用 BRU_table 中存储的相关批次散列表及其初值来存储和计算具有相同 Tid 的 RU 值,并且每个散列项条目处的 BRU 值将减去该处当前 RU 值.不同于 IncCHUI[23] 的全局列表更新结构,本文提出的结构不需要对项的总顺序≺倒序遍历,且适用于滑动窗口机制.图 2 为一个 BRU_table 结构示意图,此时其存储的是第 1 个窗口 2 个批次的事务信息.

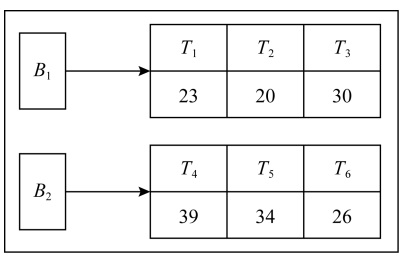


Fig. 2 BRU_table in SW_1 after the first scan

图 2 窗口 SW_1 首次扫描后的 BRU_table

3.2 CH-List 数据结构

各批次构成的 FIFO 队列效用信息由 CH-List 维护,该结构记录先前与当前窗口的项集有关效用信息.在提出的算法中,批次事务中的每个项(集合)都与一个 CH-List 列表相关联.项(集合)X 的 CH-List 包括 Utility-List 以及一种名为 $HistorySet(X)$ 的有序集合.X 的 Utility-List 由当前窗口各批次的事务元组组成.X 的 Utility-List 中的各个元组存储在以批次号(Bid)为键、批次元组为值的散列表中.每个元组代表重组事务 T_r 中 X 的效用信息,并具有 3 个字段:Tid,EU 和 RU.Tid 存储包含项集的事务标识符,EU 是项集的实际效用,RU 是项集的剩余效用.当新批次的事务到达时,按这些事务构造的元组会被添加到队列的尾部.一旦队列的大小超过窗口大小,则将属于第 1 批次的元组从项的 CH-List 中移除.可以看出,在滑动窗口滑动过程中,其效用信息更新是以批次为单位的.CH-List 相较于传统 EU-List[20] 数据结构的优点是它可以快速执行批次的插入和删除.例如,分别在图 3(a)(b)中示出了滑动窗口 SW_1 和 SW_2 中项{A}的 CH-List 变化情况.

$\{A\}_{SW_1}$

	Tid	EU	RU
B_1	1	8	6
	3	4	6
B_2	4	6	2
	5	2	14
	6	6	8
HistorySet(A)=∅			

(a) SW_1

$\{A\}_{SW_2}$

	Tid	EU	RU
B_2	4	6	0
	5	2	0
B_3	6	6	0
	8	4	0
	9	2	0
HistorySet(A)=∅			

(b) SW_2

Fig. 3 CH-List{A} in different windows

图 3 不同窗口的项{A}的 CH-List 变化

通过扫描滑动窗口 2 次来构造 CH-List 数据结构.在第 1 次扫描中,将计算项的 TWU 和事务效用.在第 2 次扫描期间,根据 TWU 的升序对每个事务中的项目进行排序,由上述过程产生重组事务.接着程序为每个项更新 CH-List 信息,由 2 个项组成的项集,其 CH-List 的 Utility-List 部分是通过将单个项的 Utility-List 相交而创建的.第 1 步是查找包含 2 个项的批次.例如 SW_1 中,要分别从 CH-List{C}和{A}构造项集{CA}的 CH-List.{C}和{A}的 CH-List 共同批次为 B_1,B_2.一旦确定了其共同出现的批次,就确定了包含这 2 个项的事务.项{C}和{A}分别出现在事务 T_1,T_3,T_4,T_6 中.图 4 中显示了项集{CA}的 CH-List.构造 k-项集的 CH-List 的过程与上述过程相似.

$\{CA\}_{SW_1}$

	Tid	EU	RU
B_1	1	14	6
	3	19	6
B_2	4	24	2
	6	15	8
HistorySet(CA)=∅			

Fig. 4 CH-List{CA} in SW_1

图 4 项集{CA}在 SW_1 的 CH-List

定义 19. 在项集 X 的 CH-List 中,其 EU 和 RU 值的总和分别表示为 $SumEU_{SW_k}(X)$ 和 $SumRU_{SW_k}(X)$,它们是将当前窗口 SW_k 下列表所含所有批次事务的 EU,RU 分别相加得到的.

属性 6. 如果 $SumEU_{SW_k}(X)+SumRU_{SW_k}(X)<minutil$,则 X 的所有扩展项集在 SW_k 中都是低效用项集.

证明. 设项集 Y 为 X 的扩展,$X\subset Y$,$TidSet_{SW_k}(Y)\subseteq TidSet_{SW_k}(X)$.令 Y/X 表示 Y 中在 X 之后的所有项的集合.根据定义 2 和定义 16,存在:

$$U_{SW_k}(Y)=\sum_{T'_r\in TidSet_{SW_k}(Y)}U(Y,T'_r),$$

且

$$U(Y,T'_r)=U(X,T'_r)+U((Y/X),T'_r)=U(X,T'_r)+\sum_{y\in(Y/X)}U(y,T'_r)\leqslant U(X,T'_r)+\sum_{y\in(T'_r-X)}U(y,T'_r)=U(X,T'_r)+RU(X,T'_r),$$

因此,

$$U_{SW_k}(Y)\leqslant\sum_{T'_r\in TidSet_{SW_k}(X)}U(X,T'_r)+RU(X,T'_r).$$

又依据定义 19,且 $U(X,T'_r)=EU(X,T'_r)$,有 $U_{SW_k}(Y)\leqslant SumEU_{SW_k}(X)+SumRU_{SW_k}(X)$. 证毕.

3.3 CHUI_DS

收到一个新批次 B_{new} 后,算法会检查该窗口 SW_k 是否为初次创建,若非创建的首个窗口,则更新前一个窗口 SW_{k-1} 中项的 CH-List,删除这些 CH-List 中最旧批次 B_{old} 存储的信息;同时清除 BRU_table 中关于最旧批次事务的相关值(算法 1 的行①~④).逐个遍历新批次的事务,对事务中的项进行扫描,检查它的 CH-List 是否为空.若为空,则算法创建其 CH-List 结构,并计算当前批次所在窗口 SW_k 下各项的 TWU.否则,为当前项更新该批次插入后的 TWU.本文的算法逐批维护事务中项的 TWU,以便在新批次到达并移除最旧的批次时可以迅速更新项的 TWU.同时新批次的各事务效用会被添加到 BRU_table 散列表中(算法 1 的行⑤~⑭).

若窗口批次数达到设定值,令 SW_k 中所有项属于集合 I.按≺对集合 I 中各项的 CH-List 排序(算法 1 的行⑰~⑲).接下来通过 BRU_table 及 CH-List 公共批次信息进行公共批次事务重组.在算法 2 中,按照项的 TWU 升序遍历各项在公共批次的所有事务,根据当前事务元组信息和各事务 BRU 重构事务元组,更新项的 CH-List,同时更新各事务的 BRU 值(算法 2 的行①~⑦).以图 1 为例,SW_1 与 SW_2 窗口公共批次 B_2 中{C},{D}的 CH-List 和 BRU_table 在重构过程中变化分别如图 5 和图 6 所示.

$\{C\}_{SW_2}$			
	Tid	*EU*	*RU*
B_2	4	18	21
	6	9	17
B_3	7	12	0
	9	9	0
HistorySet(C)=∅			

(a) SW_1

$\{D\}_{SW_2}$			
	Tid	*EU*	*RU*
B_2	4	5	16
	6	2	15
B_3	7	4	0
	9	2	0
HistorySet(D)=∅			

(b) SW_2

Fig. 5 *CommonBatches* changes during revised CH-List

图 5 SW_2 公共批次重组过程中 CH-List 变化

B_2	T_4	T_5	T_6
	39	34	26

⇨

B_2	T_4	T_5	T_6
	21	34	17

Fig. 6 BRU_table changes during revised CH-List in SW_2

图 6 SW_2 公共批次重组过程中 BRU_table 变化

紧接着返回算法 1,对 SW_k 中除公共批次之外的批次进行第 2 次扫描,以创建和更新 CH-List 中的元组信息.待 I 中所有项的 CH-List 创建完成,根据属性 5 筛选出所有高 TWU 的项组成集合 I'(算法 1 的行㉑~㉓).

算法 1. CHUI_DS 算法.

输入:批次 B_{old},B_{new};窗口大小 $winSize$;批次事务数 $number\_of\_batches$;列表 CH-List;窗口 SW_k;阈值 $minutil$;项的集合 I.

输出:数据流中所有 CHUI.

① if SW_k 不是首个滑动窗口 then
② 更新前一窗口 SW_{k-1} 中属于 I 的项的 CH-List;
③ 将最旧批次 B_{old} 从 BRU_table 中移除;
④ end if
⑤ for 事务 $T_r\in B_{new}$ do
⑥ 构造记录(Tid,TU)插入 $BRU\_table(B_{new})$;
⑦ for $item\in T_r$ do
⑧ if $item$ 的 CH-List 不存在 then
⑨ 创建 CH-List,更新 $item$ 的集合 I;
⑩ 首次计算 $item$ 的 TWU_{SW_k};
⑪ end if
⑫ 更新 $item$ 的 TWU_{SW_k};
⑬ end for
⑭ end for

⑮ 添加新批次 B_{new} 至当前滑动窗口 SW_k;
⑯ if $number\_of\_batches \geqslant winSize$ then
⑰ 设 I 为 SW_k 中所有单个项构成的集合;
⑱ 按 TWU 增序顺序对 I 中的项排序,并使用 $\prec$ 表示该顺序;
⑲ 根据 $\prec$ 对 I 中项的 CH-List 排序;
⑳ 调用算法公共批次事务重组(I 中所有项的 CH-List,BRU_table,SW_k);
㉑ 重组 SW_k 在批次 B_{new} 中的事务;
㉒ 更新各项 CH-List 中关于批次 B_{new} 的元组;
㉓ 筛选出所有高 TWU 的项,组成集合 I';
㉔ 调用算法 $CHUI\_DS\_Miner(\varnothing,\varnothing,I'$ 中所有项,$minutil$,$SW_k)$;
㉕ end if

算法 2. 公共批次事务重组算法.

输入:列表 CH-List、散列表 BRU_table、窗口 SW_k;

输出:公共批次事务重组后的 CH-List.

① for $CL(i)\in$CH-List 组成的集合 do
② for $CL(i)$ 在窗口 SW_{k-1} 和 SW_k 公共批次中的全部元组 do
③ 元组 $RU=BRU_table[Bid][Tid]-$元组 EU;
④ $BRU_table[Bid][Tid]=$元组 RU;
⑤ end for
⑥ end for
⑦ 返回 CH-List.

接下来就可以开始窗口 SW_k 的挖掘过程,调用算法 CHUI_DS_Miner 以使用有前途的项构成的集合 I' 生成 CHUI.该过程具有 5 个输入参数:1)项集 X;2)$HistorySet(X)$;3)X 的扩展($extendOfX$);4)$minutil$;5)窗口 SW_k.对于该算法发现的每个闭合项集 X,构造其 CH-List 来计算其效用,以确定其是否为 CHUI.如果实际效用和剩余效用之和小于 $minutil$,则将修剪搜索空间.CHUI_DS_Miner 的伪代码如算法 3 所示:

算法 3. CHUI_DS_Miner 算法.

输入:项集 X、集合 $HistorySet(X)$、X 的扩展集合 $extendOfX$、阈值 $minutil$、窗口 SW_k;

输出:窗口 SW_k 中所有 CHUI.

① for 项 $a\in extendOfX$ do
② 将 a 从 $extendOfX$ 移除;
③ $Y\leftarrow X\cup a$;
④ 初始化 $extendOfY=\{\}$;
⑤ 构造 $CH\text{-}List(Y)$;
⑥ $HistorySet(Y)=HistorySet(X)$;
⑦ if $SumEU_{SW_k}(Y)+SumRU_{SW_k}(Y)\geqslant minutil$ then
⑧ if $SubsumedCheck(Y,HistorySet(Y))=$false then
⑨ for 项 $Z\in extendOfX$ do
⑩ if $TidSet_{SW_k}(Y)\subseteq TidSet_{SW_k}(Z)$ then
⑪ $Y\leftarrow Y\cup Z$;
⑫ 构造 $CH\text{-}List(Y)$;
⑬ 将 Z 从 $extendOfX$ 移除;
⑭ if $SumEU_{SW_k}(Y)+SumRU_{SW_k}(Y)<minutil$ then
⑮ break;
⑯ end if
⑰ end if
⑱ end for
⑲ if Y 是高效用项集 then
⑳ 输出$(Y,SUP_{SW_k}(Y))$;
㉑ end if
㉒ $extendOfY=extendOfX$;
㉓ 调用算法 $CHUI\_DS\_Miner(Y,History\text{-}Set(Y),extendOfY,minutil,SW_k)$;
㉔ $HistorySet(X)\leftarrow HistorySet(X)\cup a$;
㉕ end if
㉖ end if
㉗ end for
㉘ 方法 $SubsumedCheck(Y,HistorySet(Y))$
㉙ for 项 $H\in HistorySet(Y)$ do
㉚ if $TidSet_{SW_k}(Y)\subseteq TidSet_{SW_k}(H)$ then
㉛ 返回 true;
㉜ end if
㉝ end for

当 $extendOfX\neq\varnothing$ 时,该过程选择 $extendOfX$ 中顺序最小的项 a 来创建项集 $Y=X\cup a$,并将 a 从 $extendOfX$ 中删除(算法 3 的行②③).然后,Y 的 CH-List 被构造过程为:1)通过将所有批次的 $TidSet_{SW_k}(X)$ 和 $TidSet_{SW_k}(a)$ 相交获得 $TidSet_{SW_k}(Y)$.2)对于每个事务 $T'_r\in TidSet_{SW_k}(X)\cap TidSet_{SW_k}(a)$,计算 T'_r 所属批次 Bid,构建元组($Tid(T'_r)$,EU,RU)插入到散列表的对应批次,由此获得 Y 的 CH-List.3)将 $HistorySet(Y)$ 初始化为 $HistorySet(X)$.

如果$SumEU_{SW_k}(Y)$和$SumRU_{SW_k}(Y)$的总和小于$minutil$,则Y与任何项$a\in extendOfY$的组合都是低效用的.若总和大于$minutil$,算法3需进一步调用过程$SubsumedCheck(Y,HistorySet(Y))$来检查$Y$是否已被挖掘到的CHUI所包含.

定义20. 闭合项集的包含.如果$Y\subset S$并且$SUP_{SW_k}(Y)=SUP_{SW_k}(S)$,则项集$Y$被包含在项集$S$中.例如$SW_1$中,$\{C\}$被$\{ACE\}$闭合包含,因为$\{C\}\subset\{ACE\}$并且$SUP_{SW_1}(\{C\})=SUP_{SW_1}(\{ACE\})$.

属性7. 在SW_k中,给定2个项集X和S,如果$X\subset S$且$SUP_{SW_k}(X)=SUP_{SW_k}(S)$,则$closure(X)=closure(S)$.

属性8. 在SW_k中,给定1个项集X和1个项$I_i\in I^*(1\leqslant i\leqslant m)$,则$TidSet_{SW_k}(X)\subseteq TidSet_{SW_k}(I^*)$和$I^*\in closure(X)$互为充要条件.

根据定义20,如果存在1个包含Y的已挖掘的闭合高效用项集S,则可以得出结论:Y没有闭合,并且$closure(S)=closure(Y)$.因此,可以安全地修剪项集Y并停止探索Y后续超集的搜索空间.否则,程序继续向下探索.

闭合包含检测过程的伪代码如算法3行㉘~㉝所示.该过程具有2个输入参数:项集Y,$HistorySet(Y)$.其执行过程为:对于$HistorySet(Y)$中的每个项H,如果$TidSet_{SW_k}(H)$中包含$TidSet_{SW_k}(Y)$,根据属性7和属性8,则该过程返回true,表明Y被已挖掘过的CHUI所包含,Y不是CHUI.若$TidSet_{SW_k}(Y)$没有包含在$HistorySet(Y)$任何项的$TidSet_{SW_k}$中,则该过程返回false,说明Y的闭包是闭合的.

假设Y通过了闭合包含检测过程,接下来计算Y的闭包,并将Y更新后的CH-List结构作为其闭包的CH-List(算法3的行⑨~⑱).其执行过程为:初始化$extendOfY$集合,对于$extendOfX$中的每个项Z,算法检查$TidSet_{SW_k}(Z)$中是否包含$TidSet_{SW_k}(Y)$.由属性8可知Z是否包含在Y的闭包中.若包含,则在$extendOfX$中移除Z,并将Z添加到Y,更新Y的CH-List.处理完$extendOfX$中的所有项后,根据此时$extendOfX$的值,更新$extendOfY$,返回更新后已经闭合的Y和$extendOfY$.

与DCI_CLOSED[32]和IncCHUI[23]相比,本文算法依据属性6,通过添加剩余效用剪枝策略修剪潜在低效用的闭包候选对象(算法3的行⑭),避免了构造低候选效用或非闭合项集的效用列表.

若Y的实际效用不小于$minutil$,则算法3输出Y为CHUI,因为Y满足条件:闭合项集、高效用项集.然后CHUI_DS_Miner算法调用自身以进一步递归探索搜索空间并找到作为Y后续超集的CHUI.递归过程完成后,将项a添加到$HistorySet(X)$中.待对所有项遍历完毕,得到该窗口SW_k中所有CHUI.

在滑动窗口模型中,最后$winSize-1$个批次在2个连续的窗口之间保持相同.即在对新的窗口扫描前,上一窗口项其CH-List最后$winSize-1$批次的元组将得到保存.因为其为挖掘下一窗口闭合项集提供必要事务计数信息.同时,由于进入新滑动窗口,项的TWU被重新计算,CH-List也需要更新.因此,相比于EU-List,CH-List取消了每个项的$PostSet$属性,CHUI_DS中的全局$extendOfX$结构在实现$PostSet$原有功能的基础上,仅在项进行排序后初始1次,节约了内存空间.

接下来,本文通过一个例子来说明算法的工作原理.考虑图1中所示的数据库.令$minutil=30$,窗口大小为2.第1个窗口由批次B_1和B_2组成.每条事务中的项均按照TWU的升序排序.对于第1个滑动窗口,项的排序为$F\prec B\prec D\prec C\prec A\prec E$.获取完整的滑动窗口后,本文的算法将构建单个项的CH-List,由于每个项的TWU全部大于30,因此将其全部加入$extendOfX$中.

依次向CHUI_DS_Miner传入$\varnothing$,$HistorySet(\varnothing)$,$extendOfX$等参数,开始遍历$extendOfX$,并首先处理其中的第1个项$\{F\}$.同时将$\{F\}$从$extendOfX$集合中去除,而后$\varnothing$和$\{F\}$进行组合得到$\{F\}$,并构建其组合后的CH-List.因为X此时为$\varnothing$,所以:

$$HistorySet(\{F\})=HistorySet(\varnothing)=\varnothing.$$

$$SumEU_{SW_1}(\{F\})+SumRU_{SW_1}(\{F\})=36+57=93>30.$$

下一步对$\{F\}$进行包含检测,由于$\{F\}$是初始结点,不会出现被前项包含的情况.因此可通过集合$extendOfX$计算$\{F\}$的闭包.首先循环$extendOfX$中的各项,此时因为$\{F\}$的Tid集合不含于$\{B\}$所在的集合,于是跳过本项直接进入下一循环.否则,基于$\{F\}$的事务集合构造$\{FB\}$的效用列表,并将$\{B\}$从$extendOfX$中删除.待遍历完最后1个元素,返回的结果是$\{F\}$,$extendOfX=\{B,D,C,A,E\}$.因为$U_{SW_1}(\{F\})=36$,所以$\{F\}$为找到的第1个CHUI,支持计数为3.之后递归进行搜索,此时的$X=\{F\}$,其继续和$\{B\}$联接,由于$SumEU_{SW_1}(\{FB\})+SumRU_{SW_1}(\{FB\})=32+22=54>36$,因此可以进行闭包计算,此时$Y=\{FB\}$.直到递归完$\{F,B,D,C\}$下的所有搜索空间,$HistorySet(\{FBD\})=\{C\}$,

紧接着遍历{F,B,D}的其余搜索扩展{A,E},由于2次执行SubsumedCheck均返回false,所以本次递归停止,开始回溯.此时$HistorySet(\{FB\})=\{D\}$.以同样的方法,算法完成对剩余所有搜索空间的搜索,最终完成对第1个滑动窗口的闭合高效用项集挖掘.

在下一批次B_3到达时,算法1构造滑动窗口SW_2.窗口SW_2中的项目TWU如表3所示.算法从第2个窗口中继续运行以挖掘CHUI,当没有新传入的批处理数据时算法终止.

Table 3 *TWU* Information of Items in SW_2

表3 SW_2中项的*TWU*信息

项名	*TWU*
A	160
B	146
C	121
D	121
E	155
F	134

4 实验与结果

本节提供了广泛的实验以评估算法的效率,包括其在3种情况下的性能:1)最小效用阈值;2)窗口大小和批次数;3)数据集的大小(可伸缩性测试).由于CHUI_DS是第1种用于数据流环境挖掘CHUI的算法,因此,针对最新的CHUI挖掘算法,评估其性能的可靠方法是将其与静态或增量数据库中现有的算法进行比较.对于情况1(变化最小效用阈值角度)和情况3(变化数据集大小的角度),本文的算法CHUI_DS以单窗口批处理方式运行,对比算法包括CHUI-Miner[20],EFIM-Closed[21],CLS-Miner[22],IncCHUI[23].CHUI-Miner,IncCHUI采用效用列表结构,IncCHUI是目前为止最先进的增量闭合高效用挖掘算法,具有与算法CHUI_DS类似的搜索技术.而EFIM-Closed利用数据库投影和事务合并技术,是目前对静态数据挖掘闭合高效用项集最高效的算法之一.情况2中,实验主要通过控制窗口和批次大小对算法的影响来验证算法性能,对比算法是去掉闭包计算过程剪枝策略的CHUI_DS,称为CHUI_DS(no_sup).表4列出了本文用于实验的数据集,包括稀疏和密集数据集.其中Connect数据集是真实的数据集,但具有合成效用值,内部效用值是通过在[1,10]中均匀分布生成的,其余数据集为真实效用值.

Table 4 Details of the Datasets

表4 实验数据集

数据集	事务数	平均事务长度	数据项个数	类型
Mushroom	8 124	23	119	密集
Foodmart	4 141	4.4	1 559	稀疏
Connect	67 557	43	129	密集
Retail	88 162	10.3	16 470	稀疏
Chainstore	1 112 949	7.3	46 086	稀疏
Accidents	340 183	33.8	468	密集

4.1 最小效用阈值的影响

本节在每个数据集上运行所有比较的算法,同时逐渐增大最小效用阈值,直到观察到明显的对比结果为止.执行时间包括读取输入数据、发现项集以及将结果写入输出文件的总时间.

在图7中,所有对比算法的运行时间都随着*minutil*的增加而逐渐降低至某个较低水平,因为算法的运行时间与产生的中间项集数量成正比,当阈值越大时,产生的中间项集越少,其运行时间就越短.在这个过程中,不同算法使用的数据库扫描方式、闭合项集搜索策略以及剪枝策略都不尽相同,造成下降速率也随之不同.在Connect数据集上,本文提出的算法运行时间大幅小于除EFIM-Closed之外的所有算法.尽管*minutil*较大时,EFIM-Closed比其余算法快;但当*minutil*较小时,EFIM-Closed需要增加大量执行时间,消耗的时间最大.这是因为Connect数据集平均事务长度在实验的所有数据集中最长,而EFIM-Closed需递归构建子投影数据库,这个过程依靠多次查找每条投影事务数组以寻找待扩展投影项索引,阈值低时的侯选待投影项数量巨大,近而导致EFIM-Closed在该数据集上表现差异较大.因此本文提出的算法在该数据集不同阈值下平均运行时间最低,总体表现优异.对于其他数据集,本文提出的算法比IncCHUI,EFIM-Closed都要略快,并且和CHUI-Miner拉开了较大差距.特别是在Connect,Mushroom,Chainstore数据集上,当阈值设置较低时运行时间大约比IncCHUI,EFIM-Closed缩短1/2.此外,当在Foodmart,Mushroom数据集上降低*minutil*时,本文算法的运行时间几乎相同.同时可以看到CHUI-Miner在绝大部分数据集上具有最大的时间消耗.原因可以解释如下.

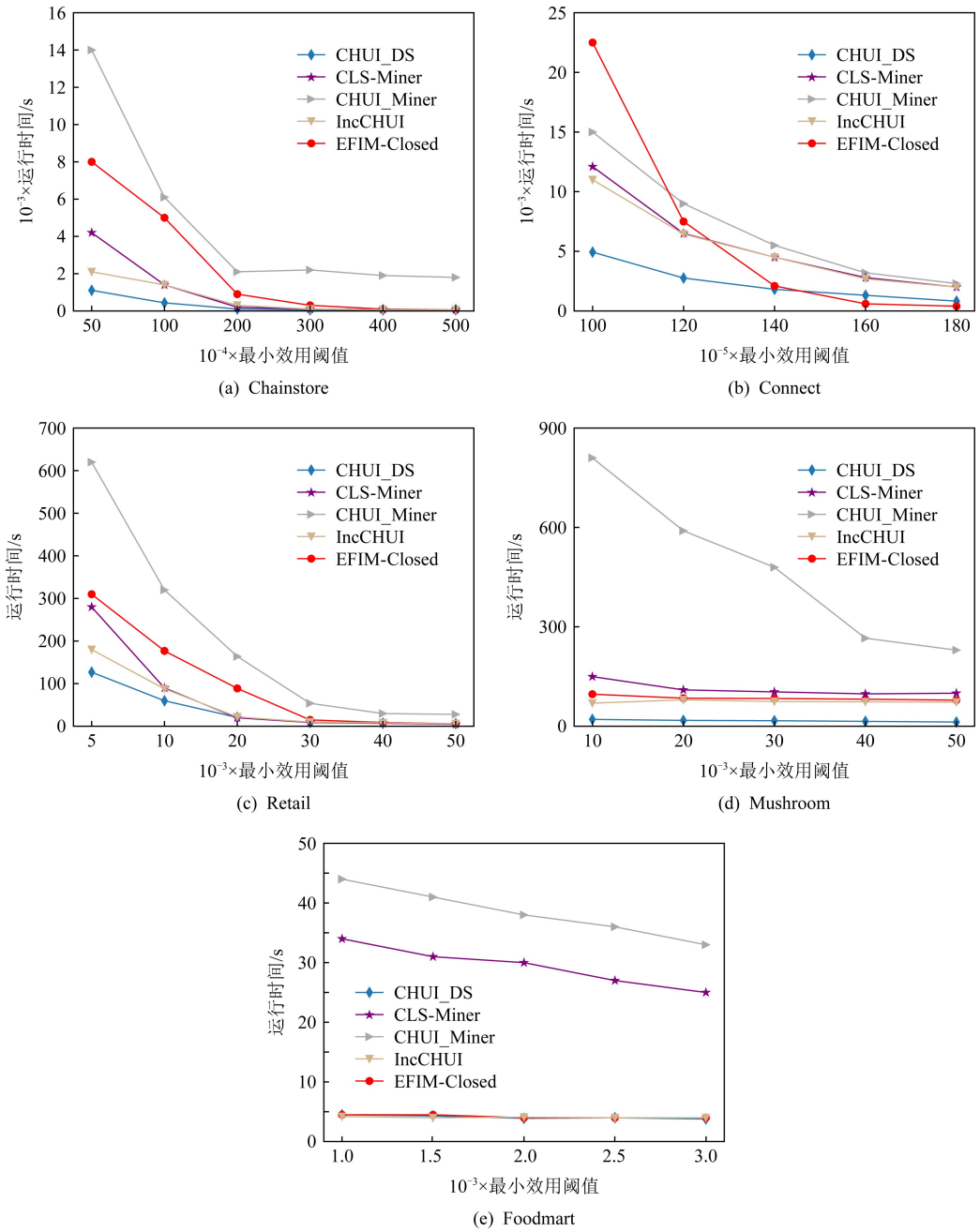


Fig. 7 Runtime comparison when varying utility threshold

图 7 不同效用阈值时的运行时间比较

尽管 CHUI-Miner, CLS-Miner, IncCHUI 和本文的算法采用相似的效用列表结构, 但是 CHUI-Miner 和 CLS-Miner 都使用传统的效用列表, 需要 2 次扫描数据库以构建效用列表, 且在闭包计算等过程缺乏修剪策略. 这也适用于 EFIM-Closed, 即使它采用了不同的技术来降低数据库扫描的成本. IncCHUI 其闭包计算过程并没有使用适用于非增量环境的修剪策略, 因此在阈值较低时慢于本文提出的算法.

4.2 不同窗口和批次大小的效率测试

在滑动窗口模型中, 算法使用 2 个参数: 1) 窗口中的批次数量; 2) 批次中的事务数量. 为了分析它们的影响, 本节使用密集的数据集“Accidents”和稀疏的数据集“Chainstore”在各种参数下进行实验. 其中 Accidents 的实验阈值设定为 1.5×10^{7}, Chainstore 的实验阈值为 9.5×10^{7}. 由于比较的算法是基于窗口的, 因此其挖掘性能通常取决于窗口大小和批次大小. 首先, 比较窗口中含不同批次时的总运行时间,

即固定 Accidents 每个批次的事务数为 3×10^4,Chainstore 每个批次的事务数为 1×10^5.2 个数据集的窗口大小参数设置,以及实验结果如图 8 所示:

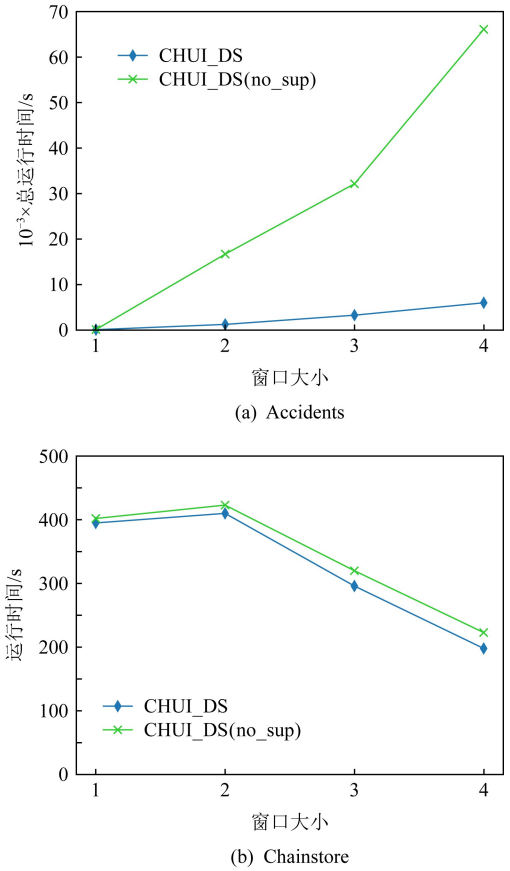


Fig. 8 Accumulated runtime comparison when varying window sizes

图 8 不同窗口大小下运行时间对比

图 8 表明,CHUI_DS(no_sup)在 2 个数据集中表现都要慢于 CHUI_DS.特别是在 Accidents 这种密集程度较大的数据集上,表现差距较大.原因在于本文是利用改进的闭包计算和递归搜索进行项集的闭合扩展,而密集数据集相比于稀疏数据集需要更多次深度搜索.若无有效的剪枝策略,在这个过程会产生大量候选项集,导致程序计算的成本大幅增加.与 CHUI_DS(no_sup)相比,CHUI_DS 在 CHUI-Miner,IncCHUI 的基础上对闭包挖掘过程额外增加了剪枝策略,同时利用提出的高效列表重组方法,使该剪枝策略能适用于任意滑动窗口,以此保证每个滑动窗口处理速度都得到提升.尽管公共批次列表的重组会产生一定时间消耗,但该过程并不需要重新扫描数据集,相较于剪枝掉的时空代价对算法效果影响有限.尤其是在阈值和数据集大小一定的情况下,加大窗口内批次数意味着事务数的增加,密集数据集下的候选项集将成数倍增大,对算法运行时间影响较大.由于 Chainstore 数据集的稀疏性,列表可复用率低,新增的中间候选项集数相对较少,由此其候选项集数量受窗口增大的影响较轻.尽管前期算法运行时间略微增加,但随着窗口的扩大,算法对整个数据集窗口滑动和垂直数据结构更新的次数明显减少,因此 2 种算法运行时间后期均出现下降.

考虑算法固定窗口内的批次数为 2,比较批次中事务数不同时的总运行时间,2 个数据集的实验参数设置及结果分别如图 9(a)(b)所示.实验逐渐增加每个批次中的事务数,运行时间的变化趋势和数据集的稀疏类型关系较大,具体表现为 Accidents 数据集运行时间逐渐变长,而 Chainstore 数据集运行时间变短.原因在于 Accidents 这样的密集型数据集,其往往会生成大量长度较长的高效用项集.和图 8 实验部分给出的原因相似,对于 Chainstore 稀疏数据集而言,因略微增大批次事务数而新增的中间候选项集数较小,批次中事务数越多从一定程度上减少滑动窗口的创建次数和挖掘算法的执行次数,

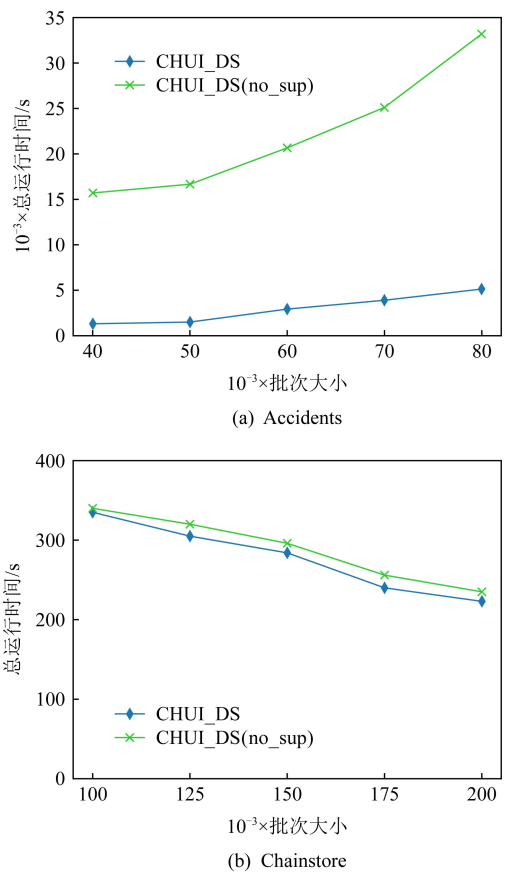


Fig. 9 Accumulated runtime comparison when varying batch sizes

图 9 批次内含不同事务数下运行时间对比

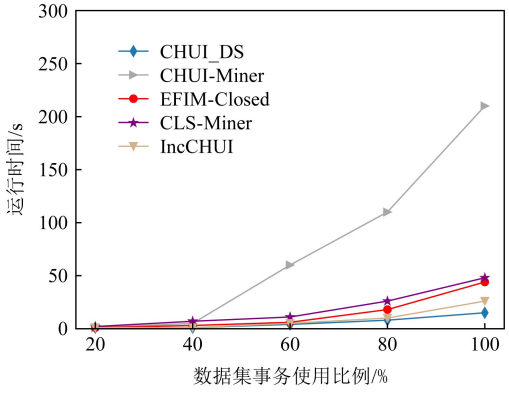
因此算法运行时间稳步下降.实验同时证明,与CHUI_DS(no_sup)相比,CHUI_DS具有更大的可扩展性,适用于大型窗口.

4.3 可扩展性测试

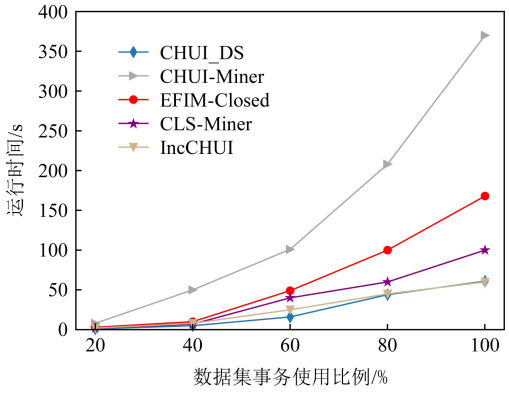
本节选取表4中的Mushroom,Retail数据集评估算法的可扩展性,以验证本文算法的运行时间和内存消耗没有呈指数增长.实验是根据总运行时间和内存进行的,采用4.1节实验中使用的最低*minutil*来执行相关工作中的算法.需要注意的是,IncCHUI在本实验部分以增量方式运行,即使用数据集中20%的事务作为原始数据集,然后分别对Mushroom,Retail数据集按5%,10%的插入率添加到原始数据集.

图10显示了运行时间评估的结果.IncCHUI的运行时间是处理每个递增部分的时间,其余算法的运行时间是累积的执行时间.本文的算法使用单窗口运行,因为其余算法均以批处理模式运行在静态数据集中.可以看到,当使用的数据集事务变多时,5种算法都需要更多的执行时间,其中在密集数据集Mushroom中CHUI_DS具有最佳的可伸缩性,在稀疏数据集Retail中CHUI_DS和IncCHUI性能均接近最优.CHUI_DS运行速度最快的原因是,与CLS-Miner和EFIM-Closed相比,CHUI-Miner仅采用了简单的*TWU*高估策略.尽管IncCHUI增加了相关修剪策略,但其以增量方式插入原始数据库,每次新数据到来都需要对项逆*TWU*顺序遍历,并依此进行先前全部效用列表事务元组的更新,且其需要判断闭合项集是否存在于已发掘的CHT表中,特别是当数据密集时时间开销会增加较快.

在内存使用方面,图11中实验结果表明:CHUI_DS在Mushroom数据集上的内存消耗比绝大部分算法都要低.在Retail数据集上,CHUI_DS随着其事务数量的变化内存总体呈增加趋势,相较于Mushroom数据集,算法产生更多的中间项集,内存占用普遍大于Mushroom数据集.但由于Retail数据集的高稀疏性,随着事务的阶段性增加,产生的中间项集数量和高*TWU*项在该阈值条件下并非一直大幅增多.在此基础上,非闭合的项集冗余比例在部分阶段略微增大,导致内存消耗呈阶段性增长.总体来说,CHUI_DS在各数据集整个挖掘过程的



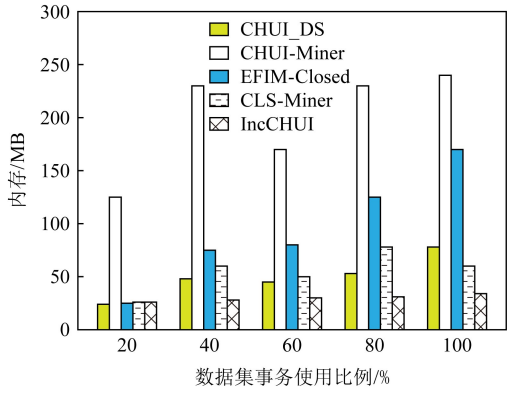
(a) Mushroom



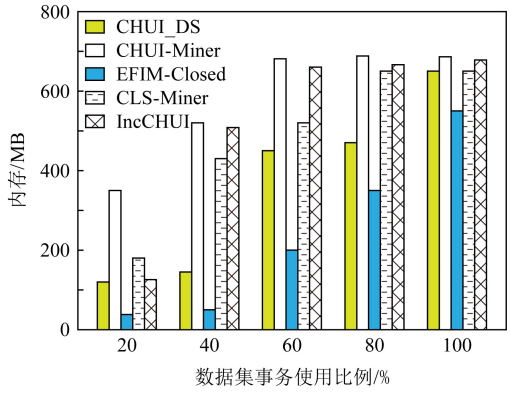
(b) Retail

Fig. 10 Scalability test of runtime

图10 运行时间可扩展性测试



(a) Mushroom



(b) Retail

Fig. 11 Scalability test of memory

图11 内存可扩展性测试

最大内存结果合理,运行时间和内存消耗均没有呈指数增长.通常,对于所有算法,内存消耗的趋势会根据所使用的数据集、算法采用的结构以及在挖掘过程中生成的候选总数而变化很大.例如,Retail 是一个稀疏的数据集.在此数据集上,CLS-Miner 消耗更多的内存,因为它需要存储其修剪策略的结构,例如项的覆盖范围和估计的效用共现结构.但是此数据集上 CLS-Miner 的运行时间非常低.在密集数据集 Mushroom 上,由于其覆盖修剪策略的有效性,CLS-Miner 的内存消耗较低,IncCHUI 在此数据集上的总体内存消耗几乎最少,这是因为其在此部分的实验以增量方式运行,只需要扫描数据库 1 次以构建全局列表,采用候选项集构建的增量修剪策略和高效用闭合项集更新机制来节约内存占用,而在 Retail 数据集上又高于 CHUI_DS.

综合来看,本文提出的算法对于不同类型数据集均有较强适应能力,总体性能良好.特别是在不同阈值、不同数据集大小等条件下的时间性能要优于先前提出的所有闭合高效用项集挖掘算法,内存占用相对其他算法处于较低水平.所提出的算法不仅对滑动窗口大小和批次事务数量具有良好的可伸缩性,而且几乎在每种情况下都保持较低的时间消耗.这些结果验证了该算法在各种现实世界数据流环境中的挖掘效率.因此,所提出的算法可以作为基础部分应用于专家和智能系统,通过滑动窗口参数设置以及对现实世界数据库特性的考虑,帮助用户更轻松地分析数据流并从中获取有意义的信息.

5 总 结

为了应对数据流的动态特性带来的挑战,本文提出了一种称为 CHUI_DS 的新算法,其可以有效地挖掘和维护数据流中闭合的高效用项集 CHUI,同时 CHUI_DS 也是第 1 种基于滑动窗口模型的数据流闭合高效用项集挖掘算法.首先,本文提出了一种新型效用列表结构,用于将数据流关键信息存储在项集中.此列表结构的一个重要方面是它能快速准确构建和更新数据流中的大量最新信息,适用于对数据流进行闭合高效用项集的挖掘任务.其次,本文基于该效用列表结构,提出数据流上无候选者产生的闭合高效用项集挖掘算法.第三,为了提高效率,本文对闭合项集挖掘过程进行改进,使用一个更加紧凑的效用剪枝策略.为了评估提出的算法,本文对现实和合成数据集进行了广泛的实验,并将其与现有算法进行了比较,该评估证实了本文算法的效率和可行性.

参 考 文 献

[1] Liu Ying, Liao Weikeng, Choudhary A. A fast high utility itemsets mining algorithm [C] //Proc of the 1st Int Workshop on Utility-based Data Mining. New York: ACM, 2005: 90-99

[2] Agrawal R, Srikant R. Fast algorithms for mining association rules [C] //Proc of the 20th Int Conf on Very Large Data Bases. San Francisco, CA: Morgan Kaufmann, 1994: 487-499

[3] Tseng V S, Shie B E, Wu Chengwei, et al. Efficient algorithms for mining high utility itemsets from transactional databases [J]. IEEE Transactions on Knowledge & Data Engineering, 2013, 25(8): 1772-1786

[4] Dawar S, Goyal V. UP-Hist tree: An efficient data structure for mining high utility patterns from transaction databases [C] //Proc of the 19th Int Database Engineering and Applications Symp. New York: ACM, 2015: 56-61

[5] Tseng V S, Wu Chengwei, Shie B E, et al. UP-Growth: An efficient algorithm for high utility itemset mining [C] //Proc of the 16th ACM SIGKDD Int Conf on Knowledge Discovery and Data Mining. New York: ACM, 2010: 253-262

[6] Yun U, Ryang H, Ryu K H. High utility itemset mining with techniques for reducing overestimated utilities and pruning candidates [J]. Expert Systems with Applications, 2014, 41(8): 3861-3878

[7] Liu Junqiang, Wang Ke, Fung B C M. Direct discovery of high utility itemsets without candidate generation [C] //Proc of the 12th IEEE Int Conf on Data Mining. Piscataway, NJ: IEEE, 2012: 984-989

[8] Liu Mengchi, Qu Junfeng. Mining high utility itemsets without candidate generation [C] //Proc of the 21st ACM Int Conf on Information and Knowledge Management. New York: ACM, 2012: 55-64

[9] Fournier-Viger P, Wu Chengwei, Zida S, et al. FHM: Faster high-utility itemset mining using estimated utility co-occurrence pruning [C] //Proc of the 21st Int Symp on Methodologies for Intelligent Systems. Berlin: Springer, 2014: 83-92

[10] Krishnamoorthy S. Pruning strategies for mining high utility itemsets [J]. Expert Systems with Applications, 2015, 42(5): 2371-2381

[11] Sahoo J, Das A K, Goswami A. An efficient approach for mining association rules from high utility itemsets [J]. Expert Systems with Applications, 2015, 42(13): 5754-5778

[12] Liu Junqiang, Wang Ke, Fung B C M. Mining high utility patterns in one phase without generating candidates [J]. IEEE Transactions on Knowledge & Data Engineering, 2016, 28(5): 1245-1257

[13] Zida S, Fournier-Viger P, Lin Chunwei, et al. EFIM: A fast and memory efficient algorithm for high-utility itemset mining [J]. Knowledge & Information Systems, 2017, 51(2): 595-625

[14] Prasad J B, Jen-Wei H. DMHUPS: Discovering multiple high utility patterns simultaneously [J]. Knowledge and Information Systems, 2018, 59(2): 337-359

[15] Dawar S, Sharma V, Goyal V. Mining top-k high-utility itemsets from a data stream under sliding window model [J]. Applied Intelligence, 2017, 47(12): 1240-1255

[16] Ahmed C F, Tanbeer S K, Jeong B S, et al. Interactive mining of high utility patterns over data streams [J]. Expert Systems with Applications, 2012, 39(15): 11979-11991

[17] Ryang H, Yun U. High utility pattern mining over data streams with sliding window technique [J]. Expert Systems with Applications, 2016, 57(9): 214-231

[18] Jaysawal B P, Huang J W. SOHUPDS: A single-pass one-phase algorithm for mining high utility patterns over a data stream [C] //Proc of the 35th ACM/SIGAPP Symp on Applied Computing. New York: ACM, 2020: 490-497

[19] Tseng V S, Wu Chengwei, Fournier-Viger P, et al. Efficient algorithms for mining the concise and lossless representation of high utility itemsets [J]. IEEE Transactions on Knowledge & Data Engineering, 2015, 27(3): 726-739

[20] Wu Chengwei, Fournier-Viger P, Gu Jiayuan, et al. Mining closed+ high utility itemsets without candidate generation [C] //Proc of Technologies & Applications of Artificial Intelligence. Piscataway, NJ: IEEE, 2015: 187-194

[21] Fournier-Viger P, Zida S, Lin Chunwei, et al. EFIM-Closed: Fast and memory efficient discovery of closed high-utility itemsets [C] //Proc of the 12th Int Conf on Machine Learning and Data Mining in Pattern Recognition. Berlin: Springer, 2016: 199-213

[22] Dam T L, Kenli L I, Fournier-Viger P, et al. CLS-Miner: Efficient and effective closed high-utility itemset mining [J]. Frontiers of Computer Science, 2019, 13(2): 357-381

[23] Dam T L, Ramampiaro H, Norvag K, et al. Towards efficiently mining closed high utility itemsets from incremental databases [J]. Knowledge-Based Systems, 2019, 165: 13-29

[24] Li Huafu, Lee S Y. Mining frequent itemsets over data streams using efficient window sliding techniques [J]. Expert Systems with Applications, 2009, 36: 1466-1477

[25] Han Meng, Wang Zhihai, Yuan Jidong. A method to set decay factor based on Gaussian function [J]. Journal of Computer Research and Development, 2015, 52(12): 2834-2843 (in Chinese)
(韩萌, 王志海, 原继东. 基于高斯函数的衰减因子设置方法研究[J]. 计算机研究与发展, 2015, 52(12): 2834-2843)

[26] Chen Hui, Shu L C, Xia Jiali, et al. Mining frequent patterns in a varying-size sliding window of online transactional data streams [J]. Information Sciences, 2012, 215: 15-36

[27] Pauray S M T. Mining top-k frequent closed itemsets over data streams using the sliding window model [J]. Expert Systems with Applications, 2010, 37(10): 6968-6973

[28] Liu Ying, Liao Weikeng, Choudhary A. A two-phase algorithm for fast discovery of high utility itemsets [C] // Proc of the 9th Pacific-Asia Conf on Advances in Knowledge Discovery and Data Mining. Berlin: Springer, 2005: 689-695

[29] Zihayat M, An Aijun. Mining top-k high utility patterns over data streams [J]. Information Sciences, 2014, 285(1): 138-161

[30] Zihayat M, Wu Chengwei, An Aijun. Efficiently mining high utility sequential patterns in static and streaming data [J]. Intelligent Data Analysis, 2017, 21(1): 103-135

[31] Tang Huijun, Liu Yangguang, Wang Le. A new algorithm of mining high utility sequential pattern in streaming data [J]. International Journal of Computational Intelligence Systems, 2019, 12(1): 342-350

[32] Lucchese C, Orlando S, Perego R. Fast and memory efficient mining of frequent closed itemsets [J]. IEEE Transactions on Knowledge & Data Engineering, 2006, 18(1): 21-36

Cheng Haodong, born in 1996. MSc candidate. Student member of CCF. His main research interests include data mining, high-utility pattern mining.

程浩东,1996 年生.硕士研究生,CCF 学生会员.主要研究方向为数据挖掘、高效用模式挖掘.

Han Meng, born in 1982. PhD, associate professor, master supervisor. Member of CCF. Her main research interests include data mining, machine learning.

韩　萌,1982 年生.博士,副教授,硕士生导师,CCF 会员.主要研究方向为数据挖掘、机器学习.

Zhang Ni, born in 1996. MSc candidate. Student member of CCF. Her main research interests includes data mining, pattern mining.

张　妮,1996 年生.硕士研究生,CCF 学生会员.主要研究方向为数据挖掘、模式挖掘.

Li Xiaojuan, born in 1994. MSc candidate. Student member of CCF. Her main research interests include data mining, ensemble classification of data streams.

李小娟,1994 年生.硕士研究生,CCF 学生会员.主要研究方向为数据挖掘、数据流的集成分类.

Wang Le, born in 1994. MSc candidate. Student member of CCF. Her main research interests include data mining, ensemble classification of data streams.

王　乐,1994 年生.硕士研究生,CCF 学生会员.主要研究方向为数据挖掘、数据流的集成分类.