基于模式增长方式的高效用模式挖掘算法

王 乐 [1] 熊松泉 [1] 常艳芬 [1] 王 水 [1]

摘 要 高效用模式挖掘是数据挖掘领域的一个重要研究内容; 由于其计算过程包含对模式的内、外效用值的处理, 计算复杂度较大, 因此挖掘算法的主要研究热点问题就是提高算法的时间效率. 针对此问题, 本文给出一个基于模式增长方式的高效用模式挖掘算法 HUPM-FP, 该算法可以从全局树上挖掘高效用模式, 避免产生候选项集. 实验中, 采用 6 个典型数据集进行实验, 并和目前效率较好的算法 FHM (Faster high-utility itemset mining) 做了对比, 实验结果表明本文给出的算法时空效率都有较大的提高, 特别是时间效率提高较大, 可以达到 1 个数量级以上.

关键词 高效用模式, 频繁模式, 频繁项集, 数据挖掘

引用格式 王乐, 熊松泉, 常艳芬, 王水. 基于模式增长方式的高效用模式挖掘算法. 自动化学报, 2015, **41**(9): 1616−1626

DOI 10.16383/j.aas.2015.c150056

An Algorithm for Mining High Utility Patterns Based on Pattern-growth

WANG Le[1] XIONG Song-Quan[1] CHANG Yan-Fen[1] WANG Shui[1]

Abstract High utility pattern mining is an important research topic in data mining. Because of the additional inner/outer utility processing workload, its computational complexity increases, and the improvement of its temporal efficiency is vital. To address this issue, a new pattern-growth mining algorithm is proposed for high utility pattern mining named HUPM-FP. This algorithm can mine high utility patterns from a global tree without generating candidate itemsets. Six classical datasets were used in our experiments for comparing with the state-of-art algorithm faster high-utility itemset mining (FHM). The proposed HUPM-FP out-performed its counterpart significantly, especially for time efficiency, which was up to 1 order of magnitude faster.

Key words High utility pattern, frequent pattern, frequent itemset, data mining

Citation Wang Le, Xiong Song-Quan, Chang Yan-Fen, Wang Shui. An algorithm for mining high utility patterns based on pattern-growth. *Acta Automatica Sinica*, 2015, **41**(9): 1616−1626

频繁模式 (Frequent pattern, FP) 挖掘是数据挖掘中一项重要而基础的技术, 但是频繁模式挖掘仅仅考虑了一个项集 (或模式) 在数据集中出现的频率, 没有考虑项集中各项的外部效用值 (如重要度、利润、价格等) 和内部效用值 (数量)[1−6]; 例如对于一个购物单数据, 频繁模式挖掘只考虑一个购物单中有哪些商品, 而没有考虑一个购物单中商品的数量、价格或利润. 然而在现实的很多应用中, 这些效用值相关信息可以度量项集 (模式) 的成本、利润值或其他重要度等信息. 由此, 在频繁模式挖掘的基础上, 提出了高效用模式 (高效用项集) 挖掘, 即将事务项集中项的效用信息引入到模式挖掘中. 目前, 高效用模式挖掘也是数据挖掘领域的一个研究热点[7−14].

2004 年, 首先由 Yao 等提出了高效用模式挖掘的定义和数学模型[15]: 在一个数据集 D 上一个项集 X 的效用值 $U(X)$, 被定义为 X 在所有包含项集 X 的事务 t 上的效用值 $U(X, t)$ 的总和, 即; $U(X) = \sum_{x \in t} U(X, t)$, 例如: "啤酒"+"尿布" 组合在所有包含这 2 种商品的购物单中所产生的利润的总和, 即称为项集啤酒、尿布的效用值. 高效用项集挖掘的任务, 就是找出效用值不小于用户预先设定阈值 (最小效用值) 的所有项集. 同时, 在文献 [15] 中, Yao 等还提出一种挖掘算法, 该算法用项集的估计值来判断一个项集是否可能是高效用模式 (这里也称为候选项集); 然后, 再扫描数据集来统计这些候选项集的真实效用值来判断是否是高效用模式. 但是, 当用户设定的阈值比较低时, 或数据集中每个记录比较长, 该算法会产生过多的候选项集, 从而统计候选项集的真实效用值的时空代价都比较大.

2005 年, Liu 等[16] 提出了一个典型算法 Two-

收稿日期 2015-01-30 录用日期 2015-06-07
Manuscript received January 30, 2015; accepted June 7, 2015
宁波市自然科学基金和攻关项目 (2013A610115, 2014A610073, 2013C10010), 浙江省教育厅一般科研项目 (Y201432717), 宁波大红鹰学院大宗商品专项课题 (1320133004) 资助
Supported by Ningbo Natural Science Foundation & Research Program (2013A610115, 2014A610073, 2013C10010), General Scientific Research Fund of Zhejiang Provincial Education Department (Y201432717),and Bulk Commodity Special Fund of Ningbo Dahongying University (1320133004)
本文责任编委 周志华
Recommended by Associate Editor ZHOU Zhi-Hua
1. 宁波大红鹰学院信息工程学院 宁波 315175
1. School of Information Engineering, Ningbo Dahongying University, Ningbo 315175

phase, 并且给出一个 TWU (Transaction weighted utilization) 模型, 即一个项集的 twu 值不小于用户设定的最小效用值, 该项集就是一个候选项集, 相对文献 [15] 中的算法, 可以有效降低项集效用值的估计值. Two-phase 基于 Apriori 算法和 TWU 模型, 该算法分为两步: 首先, 挖掘出所有的候选项集; 然后, 扫描数据集来计算候选项集的真实效用值. 算法 Two-phase 的时空效率高于文献 [15] 中的算法, 算法 Two-phase 中的候选项集是采用算法 Apriori 的层次方式产生的, 其时间和空间代价也是比较大的.

Erwin 等[17] 提出了一个基于树结构的挖掘算法 CTU-mine, 采用模式增长的方式来产生候选项集, 然后同算法 Two-phase 相同, 还需要扫描数据集来计算候选项集的真实效用值. 该算法只是在挖掘稠密数据集或用户设定的阈值比较低的时候, 算法性能优于 Two-phase 算法.

Ahmed 等[18] 提出一个基于模式增长方式的挖掘算法 IHUP (Incremental high utility pattern), 该算法不需要多次扫描数据集来产生候选项集. 针对算法 IHUP, Tseng 等[19−20] 提出算法 UP-growth, 该算法主要是针对算法 IHUP 中建树方法的改进, 减少候选项集的个数, 算法的时间效率有较大的提升. 算法 IHUP 和 UP-growth 都是采用模式增长的方式产生候选项集, 而不能直接从树上得到高效用模式.

目前的算法主要通过两个阶段挖掘高效用模式, 首先, 采用层次方式或模式增长方式找出候选项集; 然后, 扫描数据集计算每个候选项集的真实效用值来判断该候选项集是否是高效用模式. 2012 年, Liu 等提出一个更高效的算法 HUI-miner (High utility itemsets-miner)[21], 该算法不产生候选项集, 也不需要多次扫描数据集就可以挖掘到高效用模式, 相对 UP-growth, 该算法的时空效率有较大程度的提高. 2014 年, 在算法 HUI-miner 基础上, Fournier-Viger 等[14] 提出策略来优化 HUI-miner, 并给出新的挖掘算法 FHM (Faster high utility itemset mining); 相对算法 HUI-miner, 该算法能有效降低挖掘过程中产生的项集个数, 从而算法的时间效率能提高到 6 倍.

尽管高效用模式挖掘算法的时空效率已有很多的提高, 但是算法的时间代价还是比较大, 挖掘算法时空效率的提升, 特别是时间效率的提高仍然为本领域的一个挑战. 本文给出一个新的算法 HUPM-FP (High utility pattern mining based on FP-growth), 该算法采用模式增长方式来挖掘高效用模式, 但是这里可以直接采用模式增长的模式挖掘到高效用模式, 而不是候选项集.

1 相关定义

设 $D = \{t_1, t_2, t_3, \cdots, t_n\}$ 是一个含有效用信息的数据集, 其中该数据集包含有 m 个不同的项 (即 $I = \{i_1, i_2, i_3, \cdots, i_m\}$), t_j $(j = 1, 2, 3, \cdots, n)$ 是数据集 D 中第 j 个事务项集 (j 被称为事务项集的 ID), 事务项集 t_j 的数据形式为 $\{(x_1, c_1), (x_2, c_2), \cdots, (x_v, c_v)\}$ (v 为事务项集中项的个数或事务项集的长度), 其中 c_k $(k = 1, 2, \cdots, v)$ 是项 x_k 在事务 t_j 中的数量, 记为 $q(x_k, t_j)$ 也被称为项 x_k 的内部效用值; 项集 I 中项 i_r $(r = 1, 2, 3, \cdots, m)$ 的权重值记为 $p(i_r)$, 该权重值也被称为项 i_r 的外部效用值. $|D|$ 表示数据集 D 中事务个数或数据集的大小.

定义 1. 项 x 在事务 t 中的效用值, 记为 $U(x, t)$, 其定义如下:

$$U(x, t) = p(x)q(x, t) \tag{1}$$

定义 2. 项集 X 在事务 t 中的效用值, 记为 $U(X, t)$, 其定义如下:

$$U(X, t) = \begin{cases} 0, & \text{若 } X \not\subseteq t \\ \displaystyle\sum_{x \in X} U(x, t), & \text{若 } X \subseteq t \end{cases} \tag{2}$$

定义 3. 项集 X 在数据集 D 中的效用值, 记为 $u(X)$, 其定义如下:

$$U(X) = \sum_{t \in D} u(X, t) \tag{3}$$

定义 4. 一个事务 t 的效用值, 记为 $tu(t)$, 其定义如下:

$$tu(t) = \sum_{x \in t} U(x, t) \tag{4}$$

定义 5. 在一个数据集 D 中, 一个项集 X 的事务权重效用值 twu, 记为 $twu(X)$, 定义如下:

$$twu(X) = \sum_{t \in D \wedge t \supseteq X} tu(t) \tag{5}$$

其中包含该项集的所有事务效用值的和.

定义 6. 一个项集/项 X 的 twu 值不小于预定义的最小效用值, 则该项集/项就是一个候选项集/项; 否则, 是非候选项集/项.

$$twu(X) = \sum_{t \subset D \wedge t \supseteq X} tu(t) \tag{6}$$

定义 7. 在一个数据集中, 一个项集 X 的支持数 (Support number, SN) 是指那些包含项集 X 的事务项集个数, 即为项集 X 在数据集中出现的次数.

性质 1. 项集的事务权重效用值满足闭包属性: 任一个候选项集的非空子集也是一个候选项集, 任一个非候选项集的超集也是一个非候选项集.

2 HUPM-FP 算法

HUPM-FP 算法采用模式增长的方式进行高效用模式挖掘, 该算法的主要工作分为 2 步: 1) 将事务项集压缩到一棵树上, 该树上存储维护整个数据集上高效用模式的信息, 该树也称为一棵全局树; 2) 从全局树上挖掘所有高效用模式.

2.1 全局树的构建

以表 1 和表 2 中的数据为例说明全局树的构建过程, 这里设定最小阈值为 40. 构建步骤如下:

步骤 1. 扫描一遍数据集, 统计各项的效用值 (Uitility)、事务权重效用值 (twu)、支持数 (SN), 如图 1 (a) 中的头表 H 所示.

步骤 2. 将头表 H 中 twu 值小于最小阈值 40 的项删除, 剩余项按 SN 从大到小排序 (这里也可以按 utility 或 twu 值排序), 并将头表中所有项的 twu 值重新赋值为 0. 如图 1 (b) 中的头表 H_1 所示.

步骤 3. 第 2 次扫描数据集, 将每个事务项集中不在头表 H_1 中的项删除, 并按头表中项的顺序排序, 然后依次添加到一棵树上; 并更新头表中相应项的 twu 值.

如表 1 中的第一个事务项集经过删除不在头表 H_1 中的项和排序后, 得到(c, 3), (d, 3), (b, 3), (e, 1); 然后添加到树 T 上的结果如图 1 (c) 所示, 其中添加的最后一个节点存储该项集的信息, 如 uList = {6, 3, 12, 3} 表示该项集中项 c、d、b 和 e 的效用值, SN = 1 表示次数; 当在添加该项集中项 c 时, 将

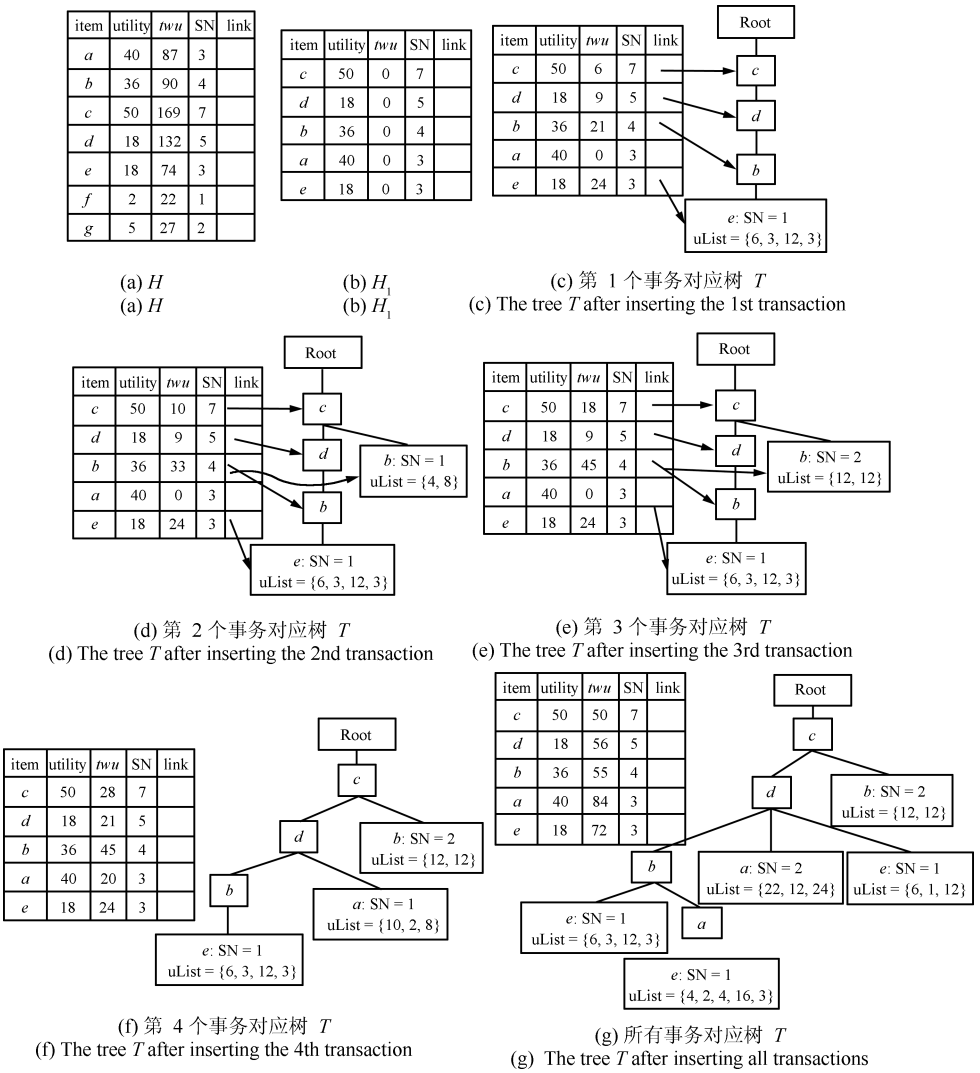


图 1 全局树构建实例

Fig. 1 An example of constructing a global tree

项 c 的效用值累加到头表项 c 对应的 twu 值中, 当添加到项 d 时, 将之前项 c 和项 d 的效用值和累加到头表项 d 对应的 twu 值中, 即当处理到每项 x 时, 将之前所有项和当前项 x 的效用值和累加到头表项 x 对应的 twu 值中, 头表中 twu 值更新后的结果如图 1 (c) 中的头表所示. 添加表 1 中第 2 个事务项集后的结果如图 1 (d) 所示. 当添加第 3 个事务的时候, 因为该项集对应的最后一个节点在树上也已经存在了, 所以这里只需要将各项对应的效用值累加到 uList 的对应项上, SN 增加 1 即可, 结果如图 1 (e) 所示. 第 4 个事务项集添加到树上后的结果如图 1 (f) 所示, 将所有事务项集添加到这棵树上后的结果如图 1 (g) 所示, 更新后的头表如图 1 (g) 中的图表所示. 其中树上每增加一个新节点时, 都将新节点的地址维护在头表对应项的 link 中, 便于以后检索树上该项对应的所有节点, 如图 1 (c) ∼ (e) 所示, 为了图例看起来清晰, 图 1 (f) ∼ (g) 上没有画 link 对应的线.

表 1　高效用数据集实例

Table 1　An example of high-utility transaction dataset

TID	Transaction
1	(b, 3), (c, 3), (d, 3), (e, 1)
2	(b, 2), (c, 2), (g, 3)
3	(b, 3), (c, 4), (f, 1)
4	(a, 1), (c, 5), (d, 2)
5	(a, 2), (b, 1), (c, 2), (d, 2), (e, 3)
6	(c, 3), (d, 1), (g, 2)
7	(a, 2), (c, 6), (d, 10)

表 2　利润表 (外部效用值)

Table 2　A profit table

Item	Profit
a	8
b	4
c	2
d	1
e	3
f	2
g	1

2.2　从全局树上挖掘高效用模式的算法

在第 2.1 节中, 将一个数据集维护在一棵全局树上, 并得到一个头表后, 算法 HUPM-FP 就可以采用模式增长方式从全局树上挖掘高效用模式, 算法详细过程如算法 1 所示.

算法 1. Ming(T, H, X)

输入. 全局树 T, 头表 H, 初始值为空的项集 X.

输出. 高效用模式集 HUPs.

Begin

1)　For each item Q in H do

　　/* 从头表 H 的最后一项开始处理 */

2)　　If ($Q.twu \geq \min Uti$) then

3)　　$X = X \cup Q$;

　　/* 判断项集 X 是否是高效用模式 */

4)　　If ($Q.Ut \geq \min Uti$) then

5)　　将项集 X 添加到 HUPs ;

6)　　End if

　　/* 因为项集 X 的 twu 值不小于最小效用值, 可以为项集 X 产生一个子树 sT 和子头表 sH*/

7)　　subTree(sT, sH, Q.link)

8)　　If (sH 不为空) then

9)　　Ming(sT, sH, X);

10)　　End if

11)　　将项 Q 从项集 X 中删除;

12)　End if

13)　将树 T 上所有节点 Q 上的 Ulist 和 SN 移到或累加到父节点;

14) End for

End

算法 1 中, 处理树 T 上项 Q 时, 为其构建子树步骤如下 (下文也将通过一个实例来说明子树的构建过程):

步骤 1. 扫描树 T 上所有节点 Q 到根节点路径上项, 并从节点 Q 上得到路径上各项的效用值和项 Q 在该路径上的效用值, 即可以计算出项 Q 和其余各项组合项集的效用值 (Utility)、事务权重效用值 (twu)、支持数 (SN), 并将其保存子头表 sH 中.

步骤 2. 将头表子头表 sH 中 twu 值小于最小阈值的项集删除, 剩余项按 SN 从大到小排序, 并将头表中所有项的 twu 值重新赋值为 0.

步骤 3. 再次扫描树 T 上每个节点 Q 到根节点路径上项, 即可得到一项集 X, 删除 X 中不在头表 sH 中的项, 并按头表 sH 排序, 并从相应节点 Q 上得到处理后 X 中各项的效用值和支持数, 同构建全局, 将处理后的 X 添加到一棵新树 sT 中, 同时修改子头表中相应项的 twu 值, 树 T 上所有节点 Q 处理完, 即得到新的子树和子头表.

算法 1 是从头表的最后一项依次往上处理所有项, 每处理完一项, 该算法就挖掘到包含该项的所有高效用模式, 因此在此后的挖掘过程中, 这些已经被处理过的项的效用信息就不需要了, 而头表 (以及算法 1 中创建的子头表, 下面详细介绍子树和子头表的创建过程) 中的 twu 值只包含没有被处理过项的效用信息, 因此算法 1 第 2) 行可以直接判断是否会存在包含项 Q 的高效用模式, 如果存在, 接下来判

断项集 X 是否是一个高效用模式.

在头表和子头表的创建过程中, 头表中的字段 Utility 存放项集 X 的效用值, 因此在算法 1 的第 4) 行中可以直接判断项集 X 是否是高效用模式.

全局树和子树在创建过程中, 是将一个项集 Y(或包含项相同的项集) 的效用信息存放到该项集在树上的最后一个节点上, 以便当创建新的子树时, 可以从树上得到项集 Y 中每项的效用值; 同时为了处理上层节点的时候也能得到效用值信息, 因此当算法 1 处理完树上最下层节点后, 将效用信息传递到父节点 (如图 2 (d) 所示, 当项 b 被处理后, 节点 b 上的效用信息被上传到父节点, 上传后的结果如图 2 (e) 所示).

这里以图 1 (f) 中的全局树和头表为例, 说明算法 1 构建子树和子头表的过程. 当处理头表 (图 1 (f)) 中项 e 的时候, 因为头表中项 a 的 twu 值不小于最小阈值 40, 因此可以为项集 $X = \{e\}$ 创建子树和子头表, 构建步骤如下:

步骤 1. 分别从树 T (图 1 (f)) 中可以得到包含项 a 的 3 个项集及效用值 ($\{c, d, e\}$, uList=$\{6, 1, 3\}$, SN=1; $\{c, d, b, e\}$, uList=$\{6, 3, 12, 3\}$, SN = 1; $\{c, d, b, a, e\}$, uList=$\{4, 2, 4, 16, 3\}$, SN = 1), 统计项 e 和其余各项组合的效用值 (Utility)、事务权重效用值 (twu)、支持数 (SN), 如图 2 (a) 中的头表 sH_1 所示; 同时也按第 2.1 节中步骤 2 处理头表 sH_1, 得到如图 2 (b) 所示的头表.

步骤 2. 如果新头表不为空, 则创建子树. 依次处理步骤 1 中的 3 个项集, 首先将项集 X (X 指步骤 1 中得到的 3 个项集) 按 sH_1 中的顺序排序 (这里不需要再考虑项 e); 然后, 同第 2.1 节中的步骤 3 的方法将有序的项集添加到一棵树上, 和第 2.1 节中的区别是, 这里的最后一个节点上还存放项集 $\{e\}$ 在 X 中的效用值, 这里记为 Ux, 第一个项集 $\{c, d, e\}$添加到树上后的结果如图 2 (c) 所示; 当添加项 c 时, 需要将项 c 和 e 在项集 $\{c, d, e\}$ 的效用值和累加到子头表 twu 中; 当添加项 d 时, 需要将项集 $\{c, d\}$ 和项 e 在项集 $\{c, d, e\}$ 的效用值和累加到子头表 twu 中; 更新后的头表如图 2 (c) 所示. 同样的方法将其余 2 个项集依次添加到树上, 并更新头表, 得到的结果如图 2 (d) 所示. 当子树创建以后, 按算法 1 递归的处理子树和子头表 (如算法 1 的第 9 行).

2.3 算法分析

HUPM-FP 将项集效用信息存放到最后一个节点, 这也保证算法 HUPM-FP 可以从树上得到高效用模式的真实效用值, 而不是估计的效用值, 因此不需要产生候选项集. 同时在创建子树的时候, 可以得到相关项集中每项的效用值 (如图 2 中子树的创建), 而不是估计值, 这也保证能计算到头表中各项 (或项集) 的真实效用值和新的 twu 值 (新的 twu 值不包含已经处理过的项和不在头表中项的效用值), 因此, 头表中的 twu 值都比原始创建的 twu 小, 如图 1 (f) 和图 1 (b) 中的头表 twu 值所示, 最终使算法减少创建子树和子头表的次数.

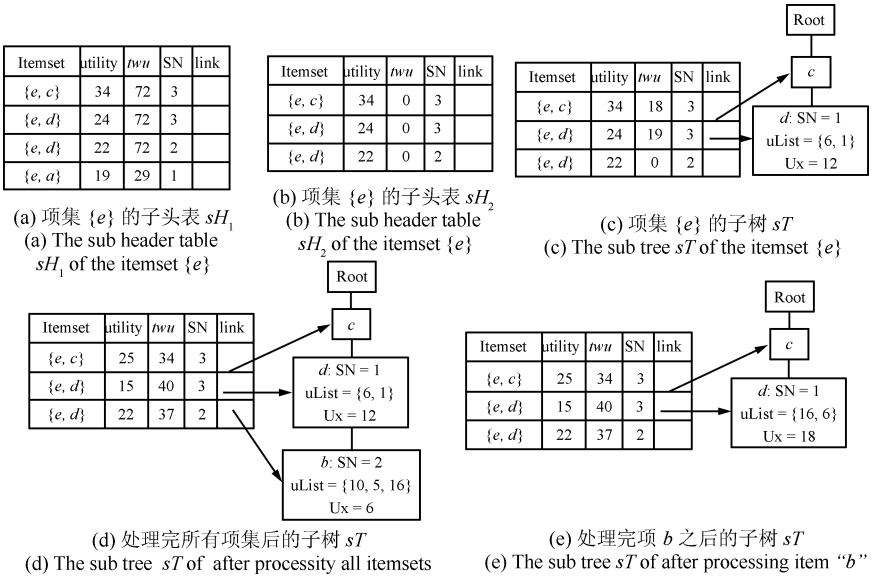


图 2 子树构建实例

Fig. 2 An example of constructing a sub-tree

根据第 2.1 节中全局树的创建过程, 没有丢失任何项的效用值, 可以从全局树上计算到任何项或项集的效用值. 当为项集或项 X 创建子树的时候, 从全局树上得到了所有包含 X 的项集及各项的效用值, 根据第 2.2 节中子树的创建过程, 也可以得到任何包含项集 X 的项集 Y 的效用值.

以上原因是算法 HUPM-FP 时间效率提高的主要原因. 本文第 3 节给出了实验结果.

以下证明算法 HUPM-FP 能挖掘到所有的高效用模式. 首先假设项集 Z 是任一个高效用模式. 因为 Z 是一个高效用模式, 则其任一非空子集的 twu 值都不会小于最小阈值, 根据算法 HUPM-FP, 项集 Z 中的所有项都会出现在全局树对应的头表中, 当项集 Z 中的第一个项 Z_1 被处理, 并创建子头表的时候, 因为项集 Z_1 和 Z 中其余各项的组合的 twu 值都不小于最小阈值, 因此这些组合的项集都会出现子头表中; 当处理新头表中项 Z_1 和 Z_2 组合, 并为其创建新子头表时, 同样项集 Z_1, Z_2 和 Z 中其余各项的组合的 twu 值都不小于最小阈值, 项集Z_1, Z_2和 Z 中其余各项的组合对出现在新的头表中, 依次处理下去, 一定能得到项集 Z 是高效用模式. 因此算法 HUPM-FP 能挖掘到所有的高效用模式.

3　实验

3.1　实验对比算法和实验平台

由于算法 FHM 的时空效率高于算法 UP-growth 和 HUI-miner 等算法, 是目前算法效率较高的算法, 因此本文新提出的算法 HUPM-FP 仅和 FHM 做了对比分析. 这两个算法都是采用 Java 编程语言实现, 实验平台: Windows 7, 10 GB Memory (Java heap size is 1.5 GB), Intel Core i5-3320 CPU 2.60 GHz.

3.2　实验数据

本节采用了 6 个经典的数据集对算法进行实验分析, 这 6 个数据集的特征如表 3 所示, 其中表 3 中 T10.I4.D100K 是由 IBM 数据产生器合成的数据集, 其他 5 个数据集都是真实数据集. 表 3 中前 5 个数据集不包含项的内部和外部效用值, 在本节的实验中, 也采用文献 [14, 17−19] 中方法, 事务项集中项的个数 (内部效用值) 都是随机产生的一个小于 10 大于 0 的整数; 项的单位效用值 (外部效用值) 也是随机产生的一个数值 (大于等于 0.0100, 小于等于 10.0000); 原始数据集 Chess、Mushroom、T10.I4.D100K 和 Retail 是从 FIMI 网站下载[22]. 数据集 Chain-store 是 California 一家连锁超市产生的数据[23], 该数据集包含内部效用值和外部效用值. 数据的稀疏度值越大表示数据集越稠密, 越小表示数据集越稀疏.

算法 HUPM-FP 和 FHM 都是精确的挖掘算法, 能从数据集中挖掘出所有的高效用模式, 两个算法挖掘的结果都是相同的, 因此本节实验只从算法的时空效率上进行了比较.

3.3　算法的时空效率

在高效用模式挖掘算法中, 算法运行时间会随着最小效用值 (最小阈值) 的降低而增加. 第一个实验测试了不同最小阈值下, 算法的运行时间和内存消耗对比. 图 3 给出了两个对比算法分别在 6 个不同数据集上运行时间对比. 由图 3 所示, 算法 HUPM-FP 的时间效率可以提高 1 个数量级以上, 甚至在部分数据集上 (如 T10.I4.D100K、Chain-store、Chess 和 Mushroom), 算法 FHM 在较低的阈值下都发生了内存溢出.

本文给出的算法 HUPM-FP 的时间消耗可以分为两部分, 一部分是创建全局树的时间消耗, 另外是从全局树上挖掘高效用模式的时间消耗. 用数据集 Retail、Chain-store、Connect 进行测试时, 随着最小阈值降低, 尽管高效用模式个数有增加, 但是总体个数不多, 没有引起从全局树上挖掘高效用模式的时间消耗的突增, 因此在这三个数据集上随着阈

表 3　数据集特征

Table 3　Dataset characteristics

数据集	项数	事务项集平均长度	事务项集总数	稀疏度 (%)
Chess	76	37	3 196	48.68
Mushroom	119	23	8 124	19.33
Connect	129	43	67 557	33.33
Retail	16 470	10.3	88 162	0.06
T10.I4.D100K	1 000	10	100 000	1
Chain-store	46 086	7.2	1 112 949	0.0156

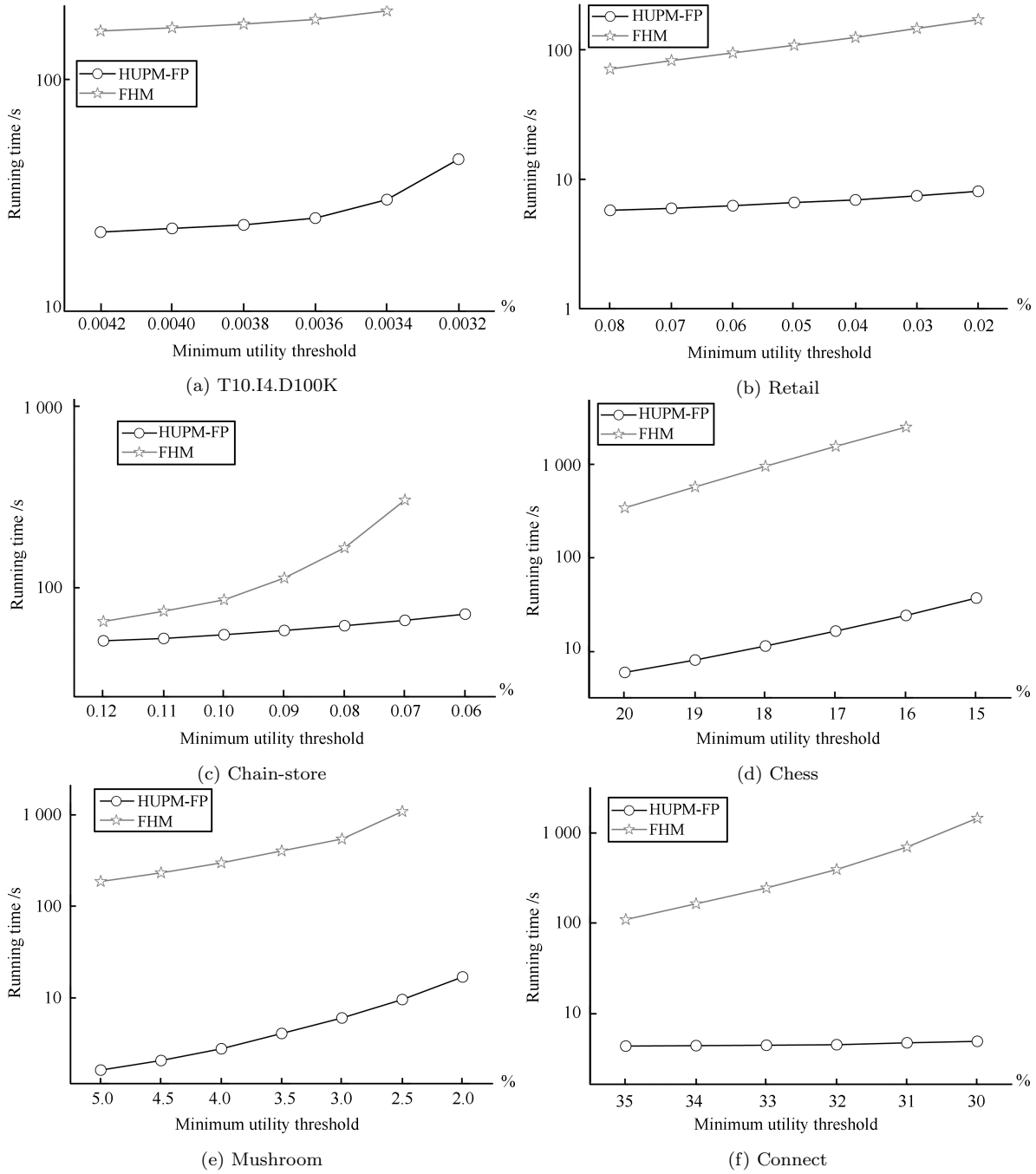


图 3 运行时间

Fig. 3 Running time

值的降低, 算法 HUPM-FP 时间效率变化比较平稳. 根据数据稀疏程度越稠密和最小阈值越小, 算法 FHM 挖掘过程中产生的项集的个数会越多, 从而导致算法的时间消耗越多, 如在稠密数据集上 Connect 进行测试时, 尽管在不同的阈值下产生的高效用模式个数比较少 (如取 35 % 时, 高效用模式个数为 0), 但是算法 FHM 在执行过程中需要计算大量项集的效用值, 导致算法时间消耗较大.

在大型数据集 Chain-store 上进行测试时, 因为该数据集是测试数据集中最稀疏的, 当最小阈值在 0.12 %～0.08 % 之间, 算法 FHM 过程中处理的项集个数较少, 同时该数据集平均长度是几个数据集中最小的, 因此时间效率相对其他数据集差异度不大, 如最小阈值为 0.12 %, 算法 HUPM-FP 所需要时间为 51 秒, 而算法 FHM 需要 66 秒; 但是随着最小阈值的降低, 算法的时间效率提高程度会明显

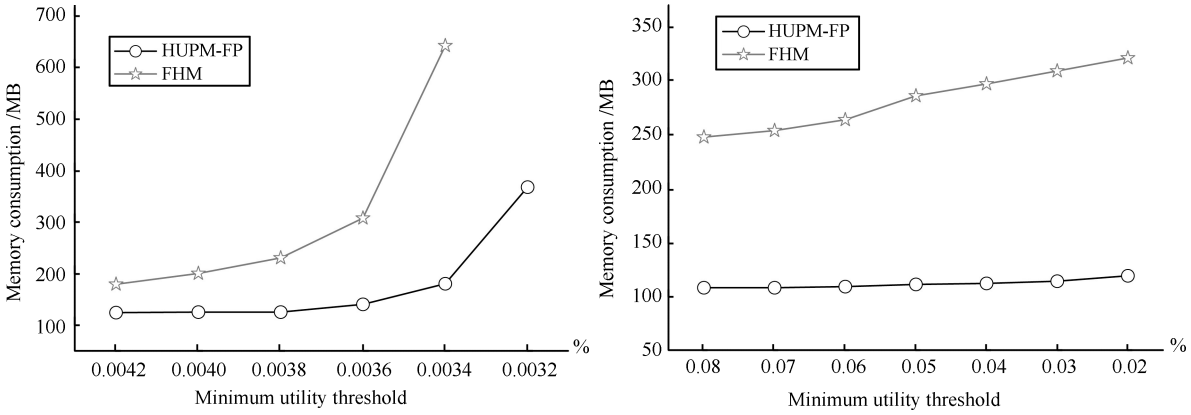
增大 (如最小阈值为 0.07%, 算法 HUPM-FP 所需要时间为 67 秒, 而算法 FHM 需要 307 秒; 当最小阈值取 0.06% 时, 算法 FHM 已经发生内存溢出了, 因此在该实验中最小值只测试到 0.06%; 实际上最小阈值取 0.06% 时, 数据集中仅包含 196 个高效用模式, 可能不满足用户需要的模式个数).

由于 Java 虚拟机自行负责内存管理并对编程不透明, 这里得到的内存开销存在一定的误差. 本节实验中的内存消耗值的提取方法如下: 在算法运行过程中, 插入多个测试点, 在每个测试点上, 先强行删除算法运行过程中产生的垃圾节点, 再提出当时的内存消耗值, 并以这些值中的最大者作为当前算法的内存消耗值.

图 4 给出了两种算法分别在 6 个不同数据集上内存消耗对比. 由图 4 所示, 在不同的最小阈值下, 相对算法 FHM, 算法 HUPM-FP 的空间效率有一定的提高. 除在数据集 T10.I4.D100K 上, 算法 HUPM-FP 随着最小阈值的降低内存消耗基本上没有变化, 这是因为算法在创建全局树后的内存消耗最大, 但是在创建完全局树后, 就可以将原始数据集从内存中移除, 而算法之后创建子树所用空间不大于原始数据集所占用的空间, 因此算法消耗最大内存仍为创建全局树后所有的内存空间, 即算法 HUPM-FP 随着最小阈值的降低内存消耗基本上没有变化. 而在数据集 T10.I4.D100K 上, 算法 HUPM-FP 创建子树所用空间大于原始数据集所占用的空间, 而且随着最小阈值的降低, 创建子树所用空间增加, 因此在该数据集上, 随着最小阈值的降低该算法所用空间增加. 同时从算法在同样情况下是否发生内存溢出也反映出本文提出的算法 HUPM-FP 的空间效率较高.

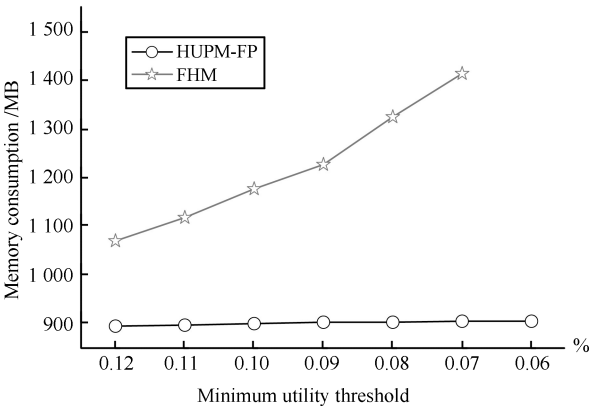
3.4　算法扩展性

第 2 个实验测试了在不同数据量下算法的运行时间和内存消耗性能, 实验结果如图 5 和图 6. 这里分别取了一个稠密数据集 Connect 和稀疏数据集 Chain-store 进行了实验. 在 Chain-store 上, 分别取原始数据集中前 200 000 条、400 000 条、600 000 条、800 000 条和 1 000 000 条数据进行了测试, 其中最小阈值设定为 0.1%; 在 Connect 上, 分别取原始数据集中前 20 000 条、30 000 条、40 000 条、50 000 条和 60 000 条数据进行了测试, 最小阈值设定为 32%. 由图 5 和图 6 所示, 随着数据量的增加, 本文给出的算法 HUPM-FP 的时空效率增加平稳; 并且总体时空效率都优于算法 FHM.

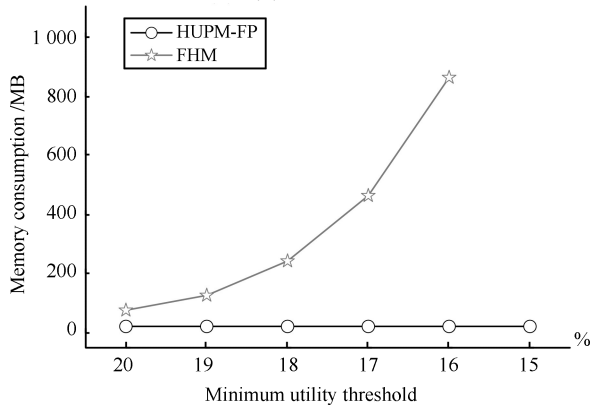


(a) T10.I4.D100K

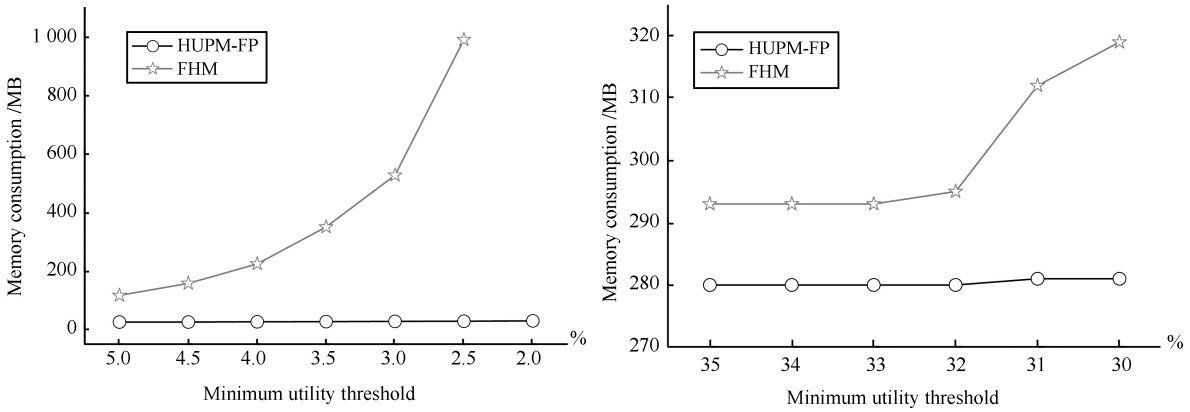
(b) Retail



(c) Chain-store



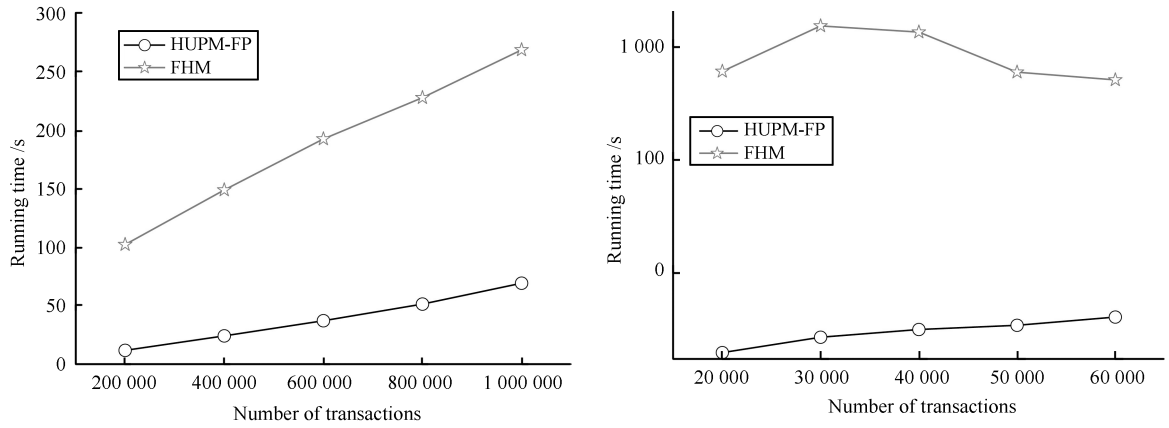
(d) Chess



(e) Mushroom (f) Connect

图 4 内存消耗

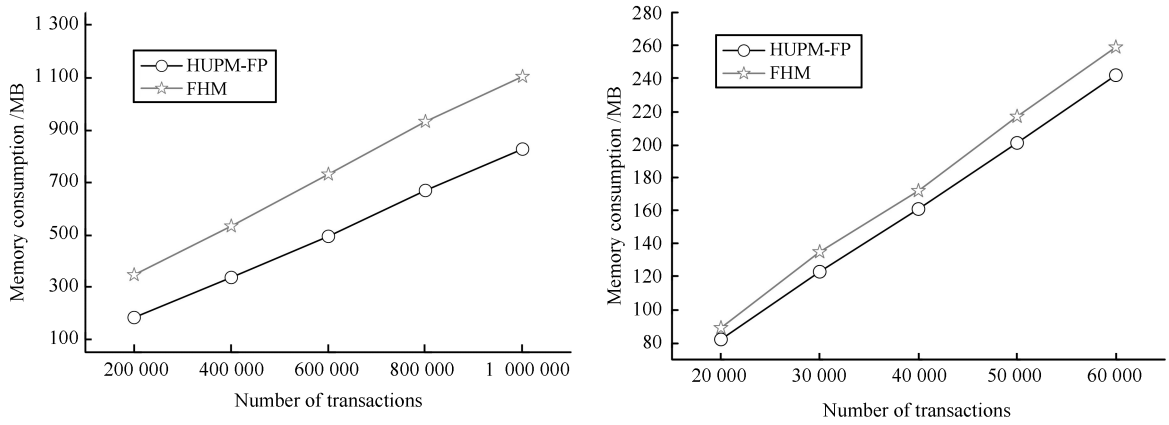
Fig. 4 Memory consumption



(a) Chain-store (b) Connect

图 5 运行时间

Fig. 5 Running time



(a) Chain-store (b) Connect

图 6 内存消耗

Fig. 6 Memory consumption

算法 FHM 同 Apriori 算法采用层次方式产生高效用模式, 该算法的时间消耗主要受所有层中产生项集数量的影响和数据集大小的影响, 因此算法过程中产生的项集个数越多, 算法的时间消耗越多; 对于同样数量的项集个数, 数据量越多, 算法时间消耗越多. 对于图 6 (b) 中的实验, 由于数据集中数据分布不均匀, 这导致了随着数据量的增加, 在同样最小阈值的情况下, 算法过程中产生的项集个数没有单调的增加, 当事务数增加到 30 000 时, 过程中产生的项集个数 171 162 为最多, 而当事务数增加到 60 000, 又降低到 27 021 个项集 (见表 4 所示), 因此当事务个数为 30 000 时, 算法 FHM 的时间消耗比较大. 通过对算法的内存消耗对比, 发现两个算法的内存消耗都随着数据量的增加成线性增加 (该测试结果不包含垃圾节点占用空间, 垃圾节点累积到一定程度, 系统会自动回收垃圾所占用的空间).

表 4　数据集 Connect 上的高效用模式个数

Table 4　Number of high utility pattern on Connect

数据量	高效用模式个数	FHM 算法过程中产生的模式个数
20 000	6 186	105 094
30 000	17 048	171 162
40 000	8 476	111 238
50 000	644	38 353
60 000	150	27 021

4　结论和建议

本文提出一个新的高效用模式挖掘算法 HUPM-FP, 该算法主要采用模式增长的方式来直接挖掘高效用模式, 而不是从树上先得到候选项集, 因此本文算法的时空效率都得到了提升. 并采用 6 个典型数据集 (包含真实数据集、合成数据集、稀疏数据集合稠密数据集) 进行了实验验证, 实验结果也表明了本文给出的算法时间和空间效率都有提高, 特别是时间效率提高较大; 其中在 4 个数据集的实验中, FHM 算法都发生了内存溢出, 而本文给出的算法仍然能有效地执行, 这也说明了本文算法的空间效率优于算法 FHM.

同时通过实验, 发现 HUPM-FP 的主要时间和内存消耗是在维护全局树和子树上, 特别是当指定最小阈值较大时, 时间和内存空间消耗仍然还是比较大, 因此下一步还可以针对此问题进行优化.

References

1 Zhang Xiao-Jian, Wang Miao, Meng Xiao-Feng. An accurate method for mining top-k frequent pattern under differential privacy. *Journal of Computer Research and Development*, 2014, **51**(1): 104−114
(张啸剑, 王淼, 孟小峰. 差分隐私保护下一种精确挖掘 top-k 频繁模式方法. 计算机研究与发展, 2014, **51**(1): 104−114)

2 Mao Yu-Xing, Shi Bai-Le. An incremental method for mining generalized association rules based on extended canonical-order tree. *Journal of Computer Research and Development*, 2012, **49**(3): 598−606
(毛宇星, 施伯乐. 基于扩展自然序树的概化关联规则增量挖掘方法. 计算机研究与发展, 2012, **49**(3): 598−606)

3 Wu Feng, Zhong Yan, Wu Quan-Yuan. Mining frequent patterns over data stream under the time decaying model. *Acta Automatica Sinica*, 2010, **36**(5): 674−684
(吴枫, 仲妍, 吴泉源. 基于时间衰减模型的数据流频繁模式挖掘. 自动化学报, 2010, **36**(5): 674−684)

4 Li Hai-Feng, Zhang Ning, Zhu Jian-Ming, Cao Huai-Hu. Frequent itemset mining over time-sensitive streams. *Chinese Journal of Computers*, 2012, **35**(11): 2283−2293
(李海峰, 章宁, 朱建明, 曹怀虎. 时间敏感数据流上的频繁项集挖掘算法. 计算机学报, 2012, **35**1333(11): 2283−2293)

5 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)

6 Chen Yin, Shan Si-Qing, Liu Lu, Li Yan. Minimum-redundant and lossless association rule-set representation. *Acta Automatica Sinica*, 2008, **34**(12): 1490−1496
(陈茵, 闪四清, 刘鲁, 李岩. 最小冗余的无损关联规则集表述. 自动化学报, 2008, **34**(12): 1490−1496)

7 Krishnamoorthy S. Pruning strategies for mining high utility itemsets. *Expert Systems with Applications*, 2015, **42**(5): 2371−2381

8 Lan G C, Hong T P, Tseng V S, Wang S L. Applying the maximum utility measure in high utility sequential pattern mining. *Expert Systems with Applications*, 2014, **41**(11): 5071−5081

9 Lin C W, Hong T P, Lan G C, Wong J W, Lin W Y. Efficient updating of discovered high-utility itemsets for transaction deletion in dynamic databases. *Advanced Engineering Informatics*, 2015, **29**(1): 16−27

10 Lin C W, Lan G C, Hong T P. Mining high utility itemsets for transaction deletion in a dynamic database. *Intelligent Data Analysis* , 2015, **19**(1): 43−255

11 Manike C, Om H. Sliding-window based method to discover high utility patterns from data streams. *Computational Intelligence in Data Mining*. India: Springer, 2015. 173−184

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

13 Zihayat M, An A J. Mining top-k high utility patterns over data streams. *Information Sciences*, 2014, **285**: 138−161

14 Fournier-Viger P, Wu C W, Zida S, Tseng V S. FHM: faster high-utility itemset mining using estimated utility co-occurrence pruning. *Foundations of Intelligent Systems*. Switzerland: Springer, 2014. 83−92

15 Yao H, Hamilton H J, Butz G J. A foundational approach to mining itemset utilities from databases. In: Proceedings of the 4th SIAM International Conference on Data Mining (ICDM 2004). Lake Buena Vista, FL, United States: Springer, 2004. 482−486

16 Liu Y, Liao W K, Choudhary A. A two-phase algorithm for fast discovery of high utility itemsets. *Advances in Knowledge Discovery and Data Mining: Proceedings of the 9th Pacific-Asia Conference, PAKDD 2005, Hanoi, Vietnam*. Berlin Heidelberg: Springer-Verlag, 2005. 689−695

17 Erwin A, Gopalan R P, Achuthan N R. CTU-mine: an efficient high utility itemset mining algorithm using the pattern growth approach. In: Proceedings of the 7th IEEE International Conference on Computer and Information Technology. Aizu-Wakamatsu, Fukushima, Japan: IEEE, 2007. 71−76

18 Ahmed C F, Tanbeer S K, Jeong B S, Lee Y K. Efficient tree structures for high utility pattern mining in incremental databases. *IEEE Transactions on Knowledge and Data Engineering* , 2009, **21**(12): 1708−1721

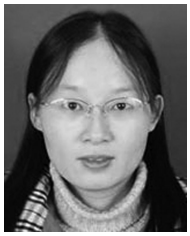
19 Tseng V S, Shie B E, Wu C W, Yu P S. Efficient algorithms for mining high utility itemsets from transactional databases. *IEEE Transactions on Knowledge and Data Engineering* , 2013, **25**(8): 1772−1786

20 Tseng V S, Wu C W, Shie B E, Yu P S. UP-growth: an efficient algorithm for high utility itemset mining. In: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Washington D. C., United States: ACM, 2010. 253−262

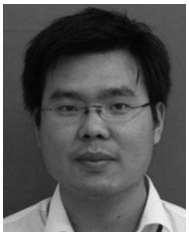
21 Liu M C, Qu J F. Mining high utility itemsets without candidate generation. In: Proceedings of the 21st ACM International Conference on Information and Knowledge Management (CIKM 2012). Maui, HI, United States: Association for Computing Machinery, 2012. 55−64

22 Generator I D. IBM data generator [Online], available: http://www.almaden.ibm.com/software/quest/Resources/index.shtml, October 1, 2010

23 Pisharath J, Liu Y, Ozisikyilmaz B, Narayanan R, Liao W K, Choudhary A, Memik G. NU-MineBench version 2.0 scource code and datasets [Online], available: http://cucis.ece.northwestern.edu/projects/DMS/MineBench.html, July 1, 2011



王 乐 宁波大红鹰学院信息工程学院讲师. 2013 年获得大连理工大学博士学位. 主要研究方向为数据挖掘.
E-mail: wangleboro@163.com
(**WANG Le** Lecturer at the School of Information Engineering, Ningbo Dahongying University. She received her Ph. D. degree in computer application from Dalian University of Technology in 2013. Her main research interest is data mining.)



熊松泉 宁波大红鹰学院信息工程学院副教授. 2012 年获得同济大学硕士学位. 主要研究方向为数据挖掘和嵌入式系统.
E-mail: yisan-sky@qq.com
(**XIONG Song-Quan** Associate professor at the School of Information Engineering, Ningbo Dahongying University. He received his master degree in computer application from Tongji University in 2012. His research interest covers data mining and embedded systems.)

常艳芬 宁波大红鹰学院信息工程学院讲师. 2009 年获得同济大学硕士学位. 主要研究方向为数据挖掘和软件工程.
E-mail: cyf511@163.com
(**CHANG Yan-Fen** Lecturer at the School of Information Engineering, Ningbo Dahongying University. She received her master degree in computer application from Tongji University in 2009. Her research interest covers data mining and software engineering.)

王 水 宁波大红鹰学院信息工程学院教授. 1989 年获得兰州大学学士学位. 主要研究方向为数据挖掘和软件工程. 本文通信作者.
E-mail: seawan@163.com
(**WANG Shui** Professor at the School of Information Engineering, Ningbo Dahongying University. He received his bachelor degree in theoretical physics from Lanzhou University in 1989. His research interest covers data mining and software engineering. Corresponding author of this paper.)