第37卷第1期 东北大学学报(自然科学版) Vol.37,No.1
2016年1月 Journal of Northeastern University(Natural Science) Jan. 2016

doi:10.3969/j.issn.1005-3026.2016.01.006

基于索引效用的 Top-k 高效用项集挖掘方法

林树宽,王晓丛,乔建忠,王 蕊
(东北大学 信息科学与工程学院,辽宁 沈阳 110819)

摘 要:已有的 Top-k 高效用项集挖掘为了保持向下封闭性,利用项集的事务效用代替其真实效用,使得项集效用被估计得过大,导致剪枝效果不好,挖掘效率较低.针对这一问题,提出了索引效用的概念,在此基础上建立两级索引,并进行索引剪枝,增强了挖掘中剪枝的效果,提高了 Top-k 高效用项集挖掘的效率;此外,通过建立效用矩阵,支持对项集效用的快速计算,进一步提高了挖掘效率.不同类型数据集上的实验验证了所提出的 Top-k 高效用项集挖掘方法的有效性和高效性.

关 键 词:项集效用;索引效用;Top-k 高效用项集;尾超项集;效用矩阵
中图分类号:TP 391 **文献标志码**:A **文章编号**:1005-3026(2016)01-0024-05

A Top-*k* High Utility Itemset Mining Method Based on the Index Utility

LIN Shu-kuan, *WANG Xiao-cong*, *QIAO Jian-zhong*, *WANG Rui*
(School of Information Science & Engineering, Northeastern University, Shenyang 110819, China. Corresponding author: LIN Shu-kuan, E-mail: linshukuan@ise.neu.edu.cn)

Abstract: The existing methods of Top-*k* high utility itemset mining substitute the transaction utilities of itemsets for their real utilities in order to keep the downward closure property. This makes the utilities of itemsets be estimated too large, resulting in bad pruning effect and low mining efficiency. To solve this problem, the concept of the index utility was proposed. On this basis, the two-level index was built and pruned, by which the pruning effect was strengthened and the efficiency of Top-*k* high utility itemset mining was enhanced. Moreover, the fast calculation of itemset utilities was supported by building the utility matrix. Therefore, the mining efficiency was further enhanced. The experiments on different types of datasets validate the effectiveness and the efficiency of the proposed method.

Key words: itemset utility; the index utility; Top-*k* high utility itemset; ending super itemset; utility matrix

频繁项集挖掘是数据挖掘领域的研究热点,在顾客购买行为分析[1-2]、网络入侵检测[3-4]等许多领域有着广泛的应用.传统的频繁项集挖掘[5-8]只依据项集出现的次数决定其频繁性,没有考虑项集的重要程度.近年来,高效用项集挖掘越来越受到关注[9-13],这里,效用是综合考虑项集出现次数和重要程度(对应单价或权重)的衡量指标.给定最低效用阈值,效用高于该阈值的项集称为高效用项集.然而,在实际挖掘过程中,指定大小合适的效用阈值并不容易,给定的阈值过大或过小,都会影响高效用项集挖掘的效果.为此,本文对 Top-k 高效用项集挖掘方法进行研究,用户无需指定效用阈值,Top-k 高效用项集挖掘将给出效用最高的前 k 个项集.然而,Top-k 高效用项集挖掘失去了向下封闭性,使挖掘中的剪枝过程面临挑战.已有的 Top-k 高效用项集挖掘方法[14]为了保持向下封闭性,用项集所在的事务效用代替其真实效用对候选项集进行剪枝,使得项集的效用被估计得过高,剪枝效果不明显,导致挖掘效率低.针对这一问题,本文提出了基于

收稿日期:2014-10-31
基金项目:国家自然科学基金资助项目(61272177).
作者简介:林树宽(1966-),女,吉林长春人,东北大学教授;乔建忠(1964-),男,辽宁兴城人,东北大学教授,博士生导师.

索引效用的 Top－k 高效用项集挖掘方法，基于所提出的索引效用建立索引并进行剪枝，提高了 Top－k 高效用项集挖掘的效率.

1　问题定义

本文挖掘的数据对象为效用事务数据库，表 1 是效用事务数据库的一个例子.

表 1　效用事务数据库 D

Table 1　A utility transaction database D

事务标识	事务中的项
T_1	(A,2,4)(B,3,4)(C,2,4)(D,1,5)
T_2	(A,1,5)(D,2,4)
T_3	(B,2,4)(C,1,3)(D,1,2)
T_4	(B,1,1)(C,2,5)(E,1,3)
T_5	(B,1,2)(E,1,1)(F,2,1)
T_6	(F,2,5)(G,2,1)

表 1 所示的效用事务数据库共包括 6 个事务 $T_1 \sim T_6$ 及 7 个项 A～G，每个事务 T 中包含的项 i 描述为一个三元组：$(i,\text{count}(i,T),w(i,T))$，其中，$i$ 为项的名称，$\text{count}(i,T)$ 为项 i 在事务 T 中的发生计数，$w(i,T)$ 为项 i 在 T 中的权重（即重要程度）. 在实际应用中，同一个项在不同的事务中可能具有不同的权重，本文充分考虑这种情况，项目的权重不是固定的，是与事务相关的可变量.

在高效用项集挖掘中，由项构成的集合称为项集，项集包含的项数称为项集的长度，长度为 l 的项集称为 l－项集.

定义 1　（项在事务中的效用）项 i 在事务 T 中的效用 $\text{uoi}(i,T)$ 定义为项 i 在事务 T 中出现的次数 $\text{count}(i,T)$ 与项 i 在 T 中的权重 $w(i,T)$ 的乘积，即

$$\text{uoi}(i,T)=\text{count}(i,T)\times w(i,T)\ . \tag{1}$$

定义 2　（项集在事务中的效用）项集 S 在事务 T 中的效用 $\text{uois}(S,T)$ 定义为项集 S 包含的所有项在事务 T 中的效用之和，即

$$\text{uois}(S,T)=\sum_{i\in S}\text{uoi}(i,T)\ . \tag{2}$$

定义 3　（项集的效用）项集 S 的效用 $\text{uois}(S)$ 即为 S 在整个效用事务数据库 D 中的效用，定义为 S 在所有包含 S 的事务中的效用之和，

$$\text{uois}(S)=\sum_{S\subseteq T\cap T\subset D}\text{uois}(S,T)\ . \tag{3}$$

Top－k 高效用项集挖掘就是在效用事务数据库中挖掘出效用最高的前 k 个项集.

定义 4　（边界阈值）在 Top－k 高效用项集挖掘的过程中，设当前的 Top－k 高效用项集为 $K=\{S_1,\ S_2,\ \cdots,\ S_k\}$，则对于 $1\leqslant p\leqslant k$，$\min(\text{uois}(S_p))$ 称为当前的边界阈值.

初始的边界阈值设为 0，随着 Top－k 高效用项集挖掘的进行，边界阈值将被不断提高.

定义 5　（项集的事务效用）若将一事务 T 中包含的所有项在该事务中的效用之和称为该事务的效用，表示为 $\text{uot}(T)$，则对于一项集 S，包含 S 的所有事务的效用之和称为项集 S 的事务效用 $\text{tuos}(S)$，即

$$\text{tuos}(S)=\sum_{S\subseteq T}\text{uot}(T)\ . \tag{4}$$

已有 Top－k 高效用项集挖掘方法利用项集事务效用的向下封闭性，对候选项集进行剪枝[14]. 项集的事务效用将项集的真实效用扩大化，基于此进行剪枝效果不明显，导致挖掘效率低. 针对这一问题，本文提出了基于索引效用和效用矩阵的 Top－k 高效用项集挖掘方法，一方面，通过建立效用矩阵，可对项集的效用进行快速计算，避免了已有挖掘方法中对效用事务数据库的多次扫描，提高了挖掘的效率；另一方面，引入索引效用的概念，并基于此建立索引及对索引进行剪枝，进一步提高了挖掘的效率.

2　效用矩阵的建立

本文基于效用矩阵实现项集效用的快速计算. 对于具有 n 个项 $i_1,i_2,\cdots,i_n$ 和 m 个事务 $T_1,T_2,\cdots,T_m$ 的效用事务数据库 D，效用矩阵 $\boldsymbol{M}$ 是一个 $(m+1)$ 行、$(n+2)$ 列的矩阵，$1\sim m$ 行分别对应 D 中的每一个事务，$1\sim n$ 列分别对应 $1\sim n$ 项. 对于 $m\geqslant p\geqslant 1$，$n\geqslant q\geqslant 1$，元素 M_{pq} 为项 i_q 在事务 T_p 中的效用 $\text{uoi}(i_q,T_p)$，第 $(m+1)$ 行第 q 列的元素为 1－项集 i_q 的效用 $\text{uois}(i_q)$，第 p 行第 $(n+1)$ 列的元素为事务 T_p 的效用 $\text{uot}(T_p)$，第 p 行第 $(n+2)$ 列的元素为事务 T_p 所包含的项数.

效用矩阵中的 $1\sim n$ 列确定了效用事务数据库中各项之间的一种顺序，本文称为矩阵列序，表示为 $\angle$. 为方便起见，本文约定所有项集中各项均按照矩阵列序排列.

通过一次扫描效用事务数据库，可生成效用矩阵以及各项所在的事务编号集合和最长事务长度. 具体生成过程见算法 1.

算法 1　效用矩阵生成

输入：效用事务数据库 D.

输出：效用矩阵 $\boldsymbol{M}$，各项所在的事务编号集

合 I,最长事务长度 maxLength.

1) 矩阵 $\boldsymbol{M}$ 的每一元素初始化为 0;
2) $I=\varnothing$; maxLength = 0;
3) for $q=1$ to n, $I_q=\varnothing$;
4) for D 中每个事务 T_p
5) for T_p 中的每一项 i_q
6) $\boldsymbol{M}_{pq}=\mathrm{uoi}(i_q,T_p)$;
7) $I_q=I_q\cup\{p\}$;
8) $\boldsymbol{M}_{p(n+1)}=\boldsymbol{M}_{p(n+1)}+\boldsymbol{M}_{pq}$;
9) $\boldsymbol{M}_{p(n+2)}=\boldsymbol{M}_{p(n+2)}+1$;
10) if $\boldsymbol{M}_{p(n+2)}>$ maxLength then maxLength $=\boldsymbol{M}_{p(n+2)}$;
11) for $q=1$ to n
12) 计算 $\boldsymbol{M}_{(m+1)q}$;
13) $I=I\cup I_q$;

对于具有 n 个项 $i_1,i_2,\cdots,i_n$ 和 m 个事务 $T_1,T_2,\cdots,T_m$ 的效用事务数据库 D,建立效用矩阵后,可以得到 $(n+m)$ 个效用值,包括 n 个 1 – 项集的效用 $u_1=\boldsymbol{M}_{(m+1)1}$, $u_2=\boldsymbol{M}_{(m+1)2}$, $\cdots$, $u_n=\boldsymbol{M}_{(m+1)n}$,以及 m 个项集在相应事务中的效用 $u_{n+1}=\boldsymbol{M}_{1(n+1)}$, $u_{n+2}=\boldsymbol{M}_{2(n+1)}$, $\cdots$, $u_{n+m}=\boldsymbol{M}_{m(n+1)}$,将这 $(n+m)$ 个效用和相应的项集按照降序进行排列,截取前 k 个项集可得到当前的 Top – k 高效用项集以及当前的边界阈值.

3 建立索引

本文建立两级索引并提出索引效用的概念,在此基础上,对索引进行剪枝,从而减少项集效用的计算量,提高挖掘效率.

定义 6 (项集在事务中的尾超集)对于项集 $S=i_1i_2\cdots i_r$ 和事务 T,若 $S\subseteq T$,令 $S_1=\{i\mid i_r\angle i$ 且 $i\in T\}$,$S'=S\cup S_1$,则 S' 称为项集 S 在事务 T 中的尾超集,表示为 Esuper(S,T).

定义 7 (索引效用)对于项集 S,其索引效用 uoin(S)定义为 S 在所有包含它的事务中的尾超集的效用之和,即

$$\mathrm{uoin}(S)=\sum_{S\subseteq T\cap T\subset D}\mathrm{uois}(\mathrm{Esuper}(S,T),T)\ . \tag{5}$$

按照定义 7,借助于算法 1 生成的效用矩阵和各个项所在的事务编号集合,可快速计算所有 1 – 项集和 2 – 项集的索引效用,为建立两级索引提供支持.

本文所建的两级索引结构如图 1 所示,其中,一级索引对应 1 – 项集,二级索引对应以 1 – 项集为首项的 2 – 项集(是以 1 – 项集为起始的超集).一级索引节点的指针 P 指向以相应的 1 – 项集为首项的二级索引节点,二级索引节点的指针指向以相应的 2 – 项集为起始的 l – 项集($l\geqslant 2$,l – 项集是以 2 – 项集为起始的超集)链成的链表.索引节点和链表节点分别随着挖掘过程中的剪枝和迭代发生变化.索引效用 uoin 是相应索引项集的索引效用.flag 是为了减少项集效用计算量而设置的一个标志变量,其值可根据索引项集的索引效用设为 0 或 1,若 flag = 0,表明该索引节点指向的所有项集,其效用均无需进行计算,从而提高挖掘效率.

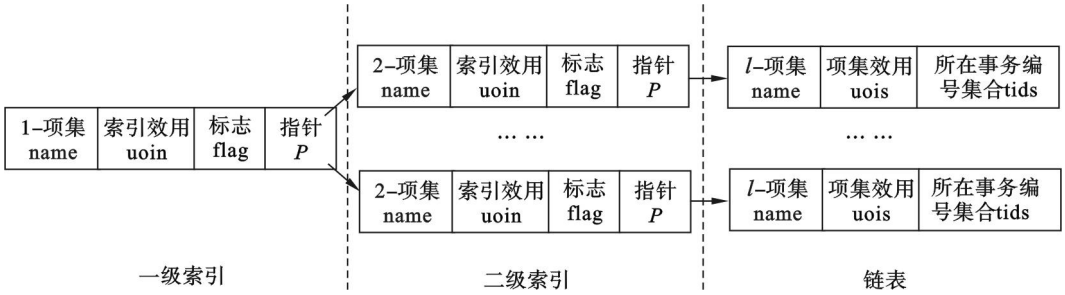

图 1 两级索引结构

Fig. 1 The structure of the two-level index

定理 1 对于任一项集 S 和以 S 为起始的任一超集 Super(S), Super(S) 的效用 uois(Super(S))不超过 S 的索引效用 uoin(S),即uois(Super(S))$\leqslant$uoin(S)

证明 略

由定理 1,若项集 S 的索引效用不超过边界阈值,则以 S 为起始的所有超集均不可能成为 Top – k高效用项集.本文将依据定理 1 构建两级索引,并在 Top – k 高效用项集挖掘的过程中,安全地对索引进行剪枝,从而提高挖掘效率.以定理 1 为依据,可按以下规则建立索引.

规则 1 若一级索引项集的索引效用小于边界阈值,则不建立该一级索引节点.

规则 2 若一级索引项集的索引效用不小于

边界阈值，但其所有二级索引效用均小于边界阈值，则置该一级索引节点中的标志变量 flag 为 0，表示以该一级索引为起始的所有项集均无需计算效用.

规则 3　若二级索引项集的索引效用为 0，则无需建立该二级索引节点.

规则 4　若二级索引项集的索引效用大于 0 而小于边界阈值，则置该二级索引节点中的标志变量 flag 为 0，表明该二级索引节点指向的链表中的所有项集效用均无需计算.

4　Top－k 高效用项集挖掘

本文在建立效用矩阵和两级索引的基础上，对 Top－k 高效用项集进行挖掘，具体过程如算法 2 所示.

算法 2　Top－k 高效用项集挖掘

输入：效用矩阵 $\boldsymbol{M}$，最长项集长度 maxLength，两级索引，初始的 Top－k 高效用项集集合.

输出：最终的 Top－k 高效用项集.

1）cLength = maxLength；
2）while（cLength > 1）do{
3）　查找效用矩阵第$(n+2)$列，得到 cLength－项集及其所在的事务编号集合；
4）　将 cLength－项集连同所在的事务编号集合按照索引插入相应链表；
5）　对于 flag 标志为 1 的索引节点，利用效用矩阵计算该节点指向的所有项集的效用；
6）　根据计算所得的项集效用提高边界阈值，并修改 Top－k 高效用项集集合；
7）　利用新的边界阈值对索引进行剪枝；
8）　将 cLength－项集分裂为（cLength－1）－项集，并将 cLength－项集从索引中删除；
9）　cLength－－；
10）}

从算法 2 可以发现，与已有的同类算法不同，算法 2 从高阶项集不断向低阶项集分裂，固定经过（maxLength－1）次循环，其中，maxLength 为效用事务数据库中最长的项集长度. 每次循环中，对于当前长度为 cLength 的项集，都要经过项集按索引插入链表、项集效用计算、根据新的项集效用提高边界阈值并对索引进行剪枝的过程，最终得到全部的 Top－k 高效用项集. 与已有的 Top－k 高效用项集挖掘方法相比，算法 2 的高效性体现在：① 迭代过程从长项集向短项集分裂，可尽早获得较大边界阈值，剪枝效果更好；② 效用矩阵的建立，减少了对效用事务数据库的扫描次数，并可支持项集效用的快速计算；③ 基于项集的索引效用对索引剪枝的过程大大减少了需计算效用的项集数量.

5　实验及其评价

为验证所提出的 Top－k 高效用项集挖掘方法的性能，本文在两个不同类型的数据集上进行了实验. 一个是模拟生成的数据集 TL10，事务的最大长度较小，是一个相对稀疏的数据集；另一个是真实数据集 Mushroom，事务的最大长度较大，是一个较稠密的数据集. 本文方法（index and matrix based Top－k utility，IMTKU）将与文献[14]中的方法（up－growth Top－k utility，UTKU）进行比较.

图 2 给出了两种方法在数据集 TL10 上的挖掘时间比较（实验中 k 的取值分别为 1，10，100，500，1 000）. 可以看出，IMTKU 的挖掘效率优于 UTKU，且随着 k 值的增大，优势越来越明显. 这是因为，IMTKU 在挖掘过程中不断利用项集的索引效用对索引进行剪枝，使得需要计算效用的项集数量大为减少，同时，借助于效用矩阵加快了项集效用的计算时间. 而 UTKU 在挖掘过程中产生大量的候选项集，且利用项集的事务效用剪枝效果不明显，因此，需要计算效用的项集数量比较多. 当 k 值较大时，UTKU 方法中上述问题更加突出，使得两种方法的差距更加明显.

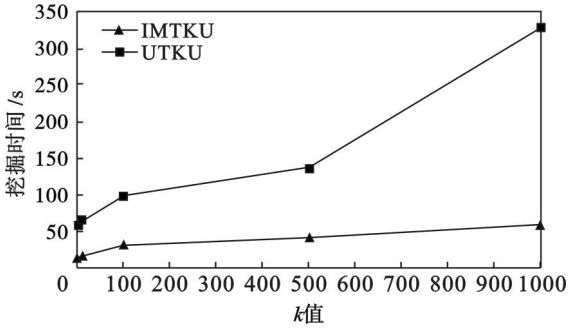

图 2　两种方法在数据集 TL10 上的挖掘效率比较
Fig. 2　The mining efficiency comparison of the two methods on TL10

图 3 给出了两种方法在数据集 Mushroom 上的挖掘时间比较（实验中 k 的取值分别为 1，10，100，500，1 000）. 可以看出，当 k 值较大时，IMTKU 具有明显的优势. 这是因为数据集

Mushroom 中最长事务长度较大,导致 IMTKU 在挖掘过程中迭代次数较多,当 k 值较小时,IMTKU 的挖掘效率较低. 但是,UTKU 用项集的事务效用估计项集的真实效用,剪枝效果不明显,当 k 值较大时,需要计算效用的候选项集数量大量增加,因此,UTKU 的挖掘时间迅速增加,超过 IMTKU 所需要的时间.

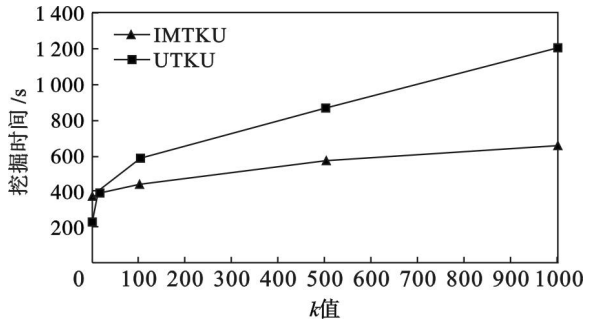

图 3 两种方法在数据集 Mushroom 上的挖掘效率比较

Fig. 3 The mining efficiency comparison of the two methods on Mushroom

图 4 给出了在数据集 TL10 上两种挖掘方法的扩展性实验结果. 从图 4 可以看出,随着效用事务数据库中事务数量的增多,两种方法的挖掘时间均随之增大,并呈线性增长,均具有较好的扩展性. 但由于建立了效用矩阵和索引并基于索引效用进行剪枝,IMTKU 的挖掘效率一直高于 UTKU,且数据库中的事务数量越多,优势越明显.

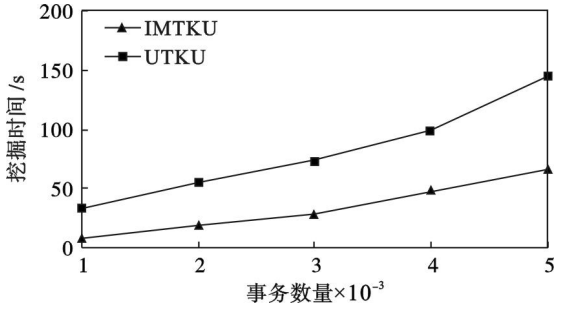

图 4 两种方法的扩展性

Fig. 4 The scalabilities of the two methods

6 结 论

1) 本文对 Top – k 高效用项集挖掘方法进行研究,针对已有方法利用项集的事务效用进行剪枝效率较低的问题,提出了索引效用的概念,在此基础上,建立两级索引并基于索引效用进行索引剪枝,大大减少了需计算效用的项集数量,提高了 Top – k 高效用项集挖掘的效率.

2) 本文通过建立效用矩阵,支持对项集效用的快速计算,同时只需对效用事务数据库进行一次扫描,进一步提高了挖掘效率.

3) 不同类型数据集上的实验表明,本文所提出的 Top – k 高效用项集挖掘方法对于事务长度较短的数据集和较大的 k 值具有非常明显的优势,并具有良好的扩展性.

参考文献:

[1] Han J, Kamber M, Pei J. Data mining: concept and technique [M]. 3rd ed. Beijing: China Machine Press, 2012.

[2] Brin S, Motwani R, Ullman J D, et al. Dynamic itemset counting and implication rules for market basket data[C]// Proceedings of ACM SIGMOD Conference on Management of Data. Tucson, 1997: 255 – 264.

[3] 毛国君,宗东军. 基于多维数据流挖掘技术的入侵检测模型与算法[J]. 计算机研究与发展,2009,46(4):602 – 609. (Mao Guo-jun, Zong Dong-jun. An intrusion detection model based on mining multi-dimension data stream[J]. *Journal of Computer Research and Development*, 2009, 46(4): 602 – 609.)

[4] 杨欢,张玉清,胡予濮,等. 基于权限频繁模式挖掘算法的 Android 恶意应用检测方法[J]. 通信学报,2013,34(Z1): 106 – 115. (Yang Huan, Zhang Yu-qing, Hu Yu-pu, et al. Android malware detection method based on permission sequential pattern mining algorithm[J]. *Journal on Communications*, 2013, 34(Z1): 106 – 115.)

[5] Agrawal R, Srikant R. Fast algorithms for mining association rules[C]// Proceedings of the 20th VLDB Conference. Santiago de Chile, 1994: 487 – 499.

[6] Agrawal R, Imielinski T, Swami A. Mining association rules between sets of items in large databases[C]// Proceedings of the ACM SIGMOD Conference on Management of Data. New York: ACM Press, 1993: 207 – 216.

[7] Pei J, Han J, Lu H, et al. H-Mine: hyper-structure mining of frequent patterns in large databases[C]// IEEE International Conference on Data Mining. Piscataway, 2001: 441 – 448.

[8] Han J, Pei J. Mining frequent patterns without candidate generation: a frequent-pattern tree approach [J]. *Data Mining and Knowledge Discovery*, 2004, 8(1): 53 – 87.

[9] Yao H, Hamilton H J, Geng L. A unified framework for utility-based measures for mining itemsets[C]// Proceedings of ACM SIGKDD 2nd Workshop on Utility-Based Data Mining. Philadelphia, 2006: 28 – 37.

[10] Tseng V, Wu C W, Shie B E, et al. UP-growth: an efficient algorithm for high utility itemset mining[C]// Proceedings of KDD'10. Washington DC, 2010: 253 – 263.

[11] Song W, Liu Y, Li J H. Vertical mining for high utility itemsets[C]// Proceedings of IEEE International Conference on Granular Computing. Hangzhou, 2012: 429 – 434.

[12] Liu J, Wang K, Fung B. Direct discovery of high utility itemsets without candidate generation[C]// Proceedings of ICDM'12. Brussels, 2012: 984 – 989.

[13] Lin C W, Hong T P, Lan G C, et al. Incrementally mining high utility patterns based on pre-large concept[J]. *Applied Intelligence*, 2014, 40(2): 343 – 357.

[14] Wu C W, Shie B E. Mining top-k high utility itemsets[C]// Proceedings of KDD'12. Beijing, 2012: 78 – 86.