文章编号：1000-8608(2017)05-0524-07

# 一种垂直结构的高效用项集挖掘算法

黄　坤*[1]，吴玉佳[2]

（1.中国舰船研究设计中心，湖北 武汉　430064；
2.武汉大学 计算机学院，湖北 武汉　430072）

**摘要：** 挖掘高效用项集已成为关联分析中的热点问题之一.多数高效用项集挖掘算法需要产生大量的候选项集，影响了算法性能.HUI-Miner是一个不需要产生候选项集就能发现事务数据库中所有高效用项集的算法.但其需要产生大量效用列表，不仅消耗了过多的存储空间，而且影响了算法的运行性能.针对此问题，提出一个新的数据结构，称为项集列表，用于存储事务和项的效用信息.提出3种剪枝策略，减少项集列表的数量，通过扫描一次事务数据库完成所有项集列表的构建.提出算法MHUI，直接从项集列表中挖掘所有的高效用项集而不产生任何候选项集.在3个不同的稀疏数据集上和最新的算法进行对比实验证明，MHUI算法的运行时间和内存消耗优于其他算法.

**关键词：** 数据挖掘；关联分析；频繁项集；高效用项集

**中图分类号：**TP312　　**文献标识码：**A　　**doi：**10.7511/dllgxb201705013

## 0　引　言

数据挖掘中的一个重要任务是关联分析.关联分析的应用非常广泛，它一般分为两个步骤：第一，从数据库中挖掘频繁项集；第二，发现关联规则.第二个步骤通常较为简单，所以，大多数学者都将研究重点放在第一个步骤上面[1-3].在频繁项集挖掘算法中，Agrawal等[1]提出的Apriori算法是此领域的开创性算法，其特点是简单、易实现，不足之处是需要多次扫描数据库，这影响了算法的性能.Han等[2]在随后提出了FP-Growth算法用于挖掘频繁项集，这是一种基于树结构的算法.FP-Growth算法通过扫描数据库将所有的信息存储到一个FP-Tree上，它的性能大大超过了Apriori，因为它寻找频繁项集时不需要产生任何候选项集.Zaki[3]提出一种基于垂直数据结构的频繁项集挖掘算法Eclat，将事务信息存储到列表上，通过对列表的交集运算，能非常快实现对支持度计数以及产生频繁项集.

然而，在这些频繁项集挖掘的框架中，没有考虑项在事务中的数量以及项的重要性（如单位利润、价格、重要性等）.在实际应用中，利益最大化对于销售经理来说是一个非常有吸引力的问题.因此，挖掘高效用项集迅速成为数据挖掘领域中的一个研究热点问题.但是，在频繁项集挖掘中，多数算法使用了向下闭性质[1-3]，即如果一个项集是非频繁的，则它的超集都是非频繁的.利用此性质能大大减少计算量并降低内存消耗.但是这个性质在高效用项集挖掘中却不能直接使用.因此，这个问题给高效用项集挖掘带来了一个巨大挑战.

鉴于此，一些算法使用估计效用上界的方法来对搜索空间进行剪枝，以提高效用项集挖掘的性能.Liu等[4]提出Two-Phase算法，算法通过两个阶段来确定高效用项集.Yao等[5]提出了UMining算法，使用一种估计方法来减少搜索空间.Li等[6]提出一个孤立项丢弃策略，用于减少候选项集的数量.虽然所有的高效用项集都能够被发现，但这些方法经常会产生大量的候选项集[4-11]，并且需要多次扫描数据库.

Ahmed等[7]提出一个基于树结构的算法IHUP用于挖掘高效用项集，获得了比IIDS和

**收稿日期：**2016-11-14；　**修回日期：**2017-07-23.
**基金项目：**国家自然科学基金资助项目(61303046).
**作者简介：**黄　坤*(1979-)，男，高级工程师，E-mail：hkcfan@163.com；吴玉佳(1986-)，男，博士生，E-mail：wuyujia@whu.edu.cn.

Two-Phase 更好的性能. Tseng 等提出了 UP-Growth算法[8]和 UP-Growth+算法[9]，减少候选项集的数量，从而提高算法的性能. 此外，Wu 等[10]和 Tseng 等[11]也提出了先挖掘闭项集的方式来挖掘完全高效用项集，取得了较好的效果. Liu 等[12]提出一种全新的用于挖掘高效用项集的算法 HUI-Miner. HUI-Miner 不需要产生候选项集，而是直接产生高效用项集. 首先产生一系列称为效用列表的数据结构，用于存储项集的事务信息、项的效用信息以及高估效用信息(剩余效用). 通过扫描效用列表的方式生成所有的高效用项集，而这一过程不需要产生任何候选项集，HUI-Miner 算法在运行时间和内存消耗上都优于上述算法.

HUI-Miner 算法产生高效用项集可以分为两个步骤：首先，将数据信息存储到效用列表上；然后再从效用列表上计算得到高效用项集. 而 HUI-Miner 产生的效用列表数量较多，消耗了存储空间并影响算法运行性能. 此外，由于该算法在效用列表中不仅存储了项集的事务和效用信息，也存储了用于对搜索空间进行剪枝的额外的剩余效用信息，这也降低了挖掘性能并占用了更多的内存资源. 针对上述问题，本文提出一个新的数据结构和一个挖掘算法 MHUI，以有效地挖掘高效用项集.

## 1　问题定义

给定一个有限的一组项 $I=\{i_1,i_2,\cdots,i_m\}$，其中每个项 $i_p(1\leqslant p\leqslant m)$ 都有一个利润值 $p(i_p)$. 一个项集 $X$ 由 $k$ 个项 $\{i_1,i_2,\cdots,i_k\}$ 组成，$i_j\in I$，$1\leqslant j\leqslant k$，$k$ 是项集 $X$ 的长度. 长度为 $k$ 的项集称为 $k$-项集. 一个事务数据库 $D=\{T_1,T_2,\cdots,T_n\}$，包含一组事务. 其中，每一个事务 $T_d(1\leqslant d\leqslant n)$ 都是 $I$ 的一个子集，具有一个唯一的标识符 $T_d$. 每个事务 $T_d$ 中的一个项 $i_p$ 具有一个数量 $q(i_p,T_d)$.

**定义 1**　项 $i_p$ 在事务 $T_d$ 中的效用 $u(i_p,T_d)$，定义为 $u(i_p,T_d)=p(i_p)\times q(i_p,T_d)$.

**定义 2**　项集 $X$ 在事务 $T_d$ 中的效用 $u(X,T_d)$，定义为 $u(X,T_d)=\sum_{i_p\in X\wedge X\subseteq T_d}u(i_p,T_d)$.

**定义 3**　项集 $X$ 在事务数据库 $D$ 中的效用 $u(X)$，定义为 $u(X)=\sum_{X\subseteq T_d\wedge T_d\subseteq D}u(X,T_d)$.

**定义 4**　给定项集 $X$ 及用户指定最小效用阈值 $minutl$，若 $u(X)\geqslant minutl$，则称 $X$ 为高效用项集.

**定义 5**　事务 $T_d$ 的事务效用记为 $TU(T_d)$，定义为 $u(T_d,T_d)$.

**定义 6**　项集 $X$ 的事务加权效用是所有包含项集 $X$ 的事务效用之和，记为 $TWU(X)$，定义为 $TWU(X)=\sum_{X\subseteq T_d\wedge T_d\subseteq D}TU(T_d)$.

**性质 1**　事务加权向下闭性质[4]，缩写为 TWDC. 规定如下：对于任何一个项集 $X$，如果 $TWU(X)<minutl$，则 $X$ 的超集都不是高效用项集.

例如：在表 1 中，$TWU(\{AB\})=TU(T_6)=28$. 设 $minutl=30$，则 $\{AB\}$ 和它的超集都不是高效用项集，因为 $TWU(\{AB\})<minutl$.

项的单位利润见表 2.

表 1　事务数据库示例

Tab. 1　An example transaction database

| $T_d$ | 事务 | $TU$ |
|---|---|---|
| $T_1$ | $(A,1)(C,10)$ | 18 |
| $T_2$ | $(A,2)(C,6)(E,2)(G,5)$ | 43 |
| $T_3$ | $(B,1)(D,4)(E,2)(F,1)$ | 18 |
| $T_4$ | $(B,4)(C,11)(D,3)(E,1)$ | 28 |
| $T_5$ | $(B,2)(E,1)(G,2)$ | 13 |
| $T_6$ | $(A,2)(B,2)(C,4)(H,2)$ | 28 |
| $T_7$ | $(B,1)(D,1)(F,1)$ | 6 |
| $T_8$ | $(A,1)(C,2)(E,3)$ | 19 |
| $T_9$ | $(B,1)(C,2)(D,2)$ | 8 |
| $T_{10}$ | $(C,1)(D,3)(G,4)$ | 19 |

表 2　项的单位利润

Tab. 2　Unit profit of item

| 项 | 利润 | 项 | 利润 |
|---|---|---|---|
| $A$ | 8 | $E$ | 3 |
| $B$ | 2 | $F$ | 2 |
| $C$ | 1 | $G$ | 3 |
| $D$ | 2 | $H$ | 2 |

## 2　方法介绍

在这部分，详细介绍提出的数据结构和算法. 扫描一次事务数据库，并应用 3 种剪枝策略，产生存储所有事务和项的效用信息的项集列表. 最后，扫描所有的项集列表，产生高效用项集.

### 2.1 初始项集列表

首先,扫描如表1所示的事务数据库一次,产生初始项集列表,如图1所示.初始项集列表中存储每个项所在的事务以及对应的效用.例如,项 $A$ 在事务 $T_1$、$T_2$、$T_6$、$T_8$ 中出现,对应的效用分别为8、16、16、8.

| $\{A\}$ | | $\{B\}$ | | $\{C\}$ | | $\{D\}$ | | $\{E\}$ | | $\{F\}$ | | $\{G\}$ | | $\{H\}$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 8 | 3 | 2 | 1 | 10 | 3 | 8 | 2 | 6 | 3 | 2 | 2 | 15 | 6 | 4 |
| 2 | 16 | 4 | 8 | 2 | 6 | 4 | 6 | 3 | 6 | 7 | 2 | 5 | 6 | | |
| 6 | 16 | 5 | 4 | 4 | 11 | 7 | 2 | 4 | 3 | | | 10 | 12 | | |
| 8 | 8 | 6 | 4 | 6 | 4 | 9 | 4 | 5 | 3 | | | | | | |
| | | 7 | 2 | 8 | 2 | 10 | 6 | 8 | 9 | | | | | | |
| | | 9 | 2 | 9 | 2 | | | | | | | | | | |
| | | | | 10 | 1 | | | | | | | | | | |

图1 初始项集列表

Fig. 1 Initial itemset lists

在图1中,初始项集列表$\{F\}$的加权事务效用之和为 $TWU(\{F\})=TU(T_3)+TU(T_7)=24$.

根据性质1,项 $F$ 和它的超集都不是高效用项集,因为 $TWU(\{F\})<minutl$.同理,项 $H$ 和它的超集都不是高效用项集,项 $F$ 和 $H$ 都是无用项,可以从初始项集列表中移除.

**定义7**(有用项和无用项) 给定一个项 $i_p$,如果 $TWU(i_p)\geqslant minutl$,则称其为有用项;否则,称其为无用项.

**策略1** 删除初始项集列表中的无用项.

### 2.2 2-项集列表

在产生初始项集列表之后,通过对初始项集列表进行交集运算,生成2-项集列表.例如项 $A$ 和项 $B$ 的共同事务为6,对应的效用分别为16和4,所以项集$\{AB\}$的事务为6,效用为20,2-项集列表如图2所示.

**性质2** 项集列表事务加权向下闭性质,缩写为ITWDC.规定如下:对于任何一个项集列表 $X$,如果项集列表 $X$ 的加权事务效用之和小于 $minutl$,则项集 $X$ 及其超集都不是高效用项集.

| $\{AB\}$ | | $\{AC\}$ | | $\{AD\}$ | | $\{AE\}$ | | $\{AG\}$ | | $\{BC\}$ | | $\{BD\}$ | | $\{BE\}$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 6 | 20 | 1 | 18 | | | 2 | 22 | 2 | 31 | 4 | 19 | 3 | 10 | 3 | 8 |
| | | 2 | 22 | | | 8 | 17 | | | 6 | 8 | 4 | 14 | 4 | 11 |
| | | 6 | 20 | | | | | | | 9 | 4 | 7 | 4 | 5 | 7 |
| | | 8 | 10 | | | | | | | | | 9 | 6 | | |

| $\{BG\}$ | | $\{CD\}$ | | $\{CE\}$ | | $\{CG\}$ | | $\{DE\}$ | | $\{DG\}$ | | $\{EG\}$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 10 | 4 | 17 | 2 | 12 | 2 | 21 | 3 | 14 | 10 | 18 | 2 | 21 |
| | | 9 | 6 | 4 | 14 | 10 | 13 | 4 | 9 | | | 5 | 9 |
| | | 10 | 7 | 8 | 11 | | | | | | | | |

图2 初始2-项集列表

Fig. 2 Initial 2-itemset lists

项集列表$\{AD\}$的结果为空集,它的超集都为空集,可以直接从2-项集列表中删除.计算每个2-项集列表的加权事务效用.其中,$TWU(\{AB\})=24$,$TWU(\{BG\})=13$,$TWU(\{DG\})=19$.它们可以从2-项集列表中删除,根据性质2,项集$\{AB\}$、$\{BG\}$和$\{DG\}$的超集也不是高效用项集,都称为无用项集.

**定义8**(有用项集和无用项集) 给定一个项集 $X$,如果 $TWU(X)\geqslant minutl$,则称其为有用项集;否则,称为无用项集.

**策略2** 删除项集列表中的无用项集.

图3所示为应用策略2之后的2-项集列表,2-项集列表的数量从最初的15个减少到11个.

| $\{AC\}$ | | $\{AE\}$ | | $\{AG\}$ | | $\{BC\}$ | | $\{BD\}$ | | $\{BE\}$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 18 | 2 | 22 | 2 | 31 | 4 | 19 | 3 | 10 | 3 | 8 |
| 2 | 22 | 8 | 17 | | | 6 | 8 | 4 | 14 | 4 | 11 |
| 6 | 20 | | | | | 9 | 4 | 7 | 4 | 5 | 7 |
| 8 | 10 | | | | | | | 9 | 6 | | |

| $\{CD\}$ | | $\{CE\}$ | | $\{CG\}$ | | $\{DE\}$ | | $\{EG\}$ | |
|---|---|---|---|---|---|---|---|---|---|
| 4 | 17 | 2 | 12 | 2 | 21 | 3 | 14 | 2 | 21 |
| 9 | 6 | 4 | 14 | 10 | 13 | 4 | 9 | 5 | 9 |
| 10 | 7 | 8 | 11 | | | | | | |

图3 应用策略2后的2-项集列表

Fig. 3 2-Itemset lists by applying Strategy 2

**定义 9**(前缀项集)　给定一个项集 $X$，由 $k$ 个项$\{i_s, i_{s+1}, \cdots, i_{s+k-1}\}$组成，所有的项按顺序排列. 项 $i_s$ 即为项集 $X$ 的前缀，排在项 $i_s$ 前的项集称为项集 $X$ 的前缀项集.

例如，项集$\{BC\}$的前缀项集为 $A$，项集$\{DE\}$的前缀项集为 $ABC$.

观察图 3 中的项集列表$\{DE\}$，如果直接使用表 1 中的事务效用 $TU$ 值来计算项集列表$\{DE\}$的 $TWU$ 值，则 $TWU(\{DE\})=46$. 此时项集列表$\{DE\}$不能删除，需要保留. 如果利用定义 9 删除前缀项集的效用，则项集$\{DE\}$的 $TWU$ 值的计算方法为 $TWU(\{DE\}) = 25$，它的值小于 $minutl$，根据性质 2，项集$\{DE\}$及它的超集都不是高效用项集，将项集列表$\{DE\}$从 2-项集列表中删除，进一步减少项集列表的数量.

**策略 3**　删除项集前缀效用.

图 4 所示为使用策略 3 之后的 2-项集列表，2-项集列表的数量进一步从 11 个减少到 10 个.

| $\{AC\}$ | | $\{AE\}$ | | $\{AG\}$ | | $\{BC\}$ | | $\{BD\}$ | | $\{BE\}$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 18 | 2 | 22 | 2 | 31 | 4 | 19 | 3 | 10 | 3 | 8 |
| 2 | 22 | 8 | 17 | | | 6 | 8 | 4 | 14 | 4 | 11 |
| 6 | 20 | | | | | 9 | 4 | 7 | 4 | 5 | 7 |
| 8 | 10 | | | | | | | 9 | 6 | | |

| $\{CD\}$ | | $\{CE\}$ | | $\{CG\}$ | | $\{EG\}$ | |
|---|---|---|---|---|---|---|---|
| 4 | 17 | 2 | 12 | 2 | 21 | 2 | 21 |
| 9 | 6 | 4 | 14 | 10 | 13 | 5 | 9 |
| 10 | 7 | 8 | 11 | | | | |

图 4　应用策略 3 后的 2-项集列表

Fig. 4　2-Itemset lists by applying Strategy 3

## 2.3　$k$-项集列表

为了构建 $k$-项集列表，可以根据 $k-1$ 项集列表进行事务交集运算构建. 例如根据图 4 所示的 2-项集列表，进行事务交集运算，生成 3-项集列表，如图 5 所示. 在产生 $k$-项集列表($k>2$)时，需要减去重复效用值，文献[12]有详细说明.

| $\{ACE\}$ | | $\{ACG\}$ | | $\{AEG\}$ | | $\{BCD\}$ | | $\{BCE\}$ | |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 28 | 2 | 37 | 2 | 37 | 4 | 25 | 4 | 22 |
| 8 | 19 | | | | | 9 | 8 | | |

| $\{BDE\}$ | | $\{CDE\}$ | | $\{CDG\}$ | | $\{CEG\}$ | |
|---|---|---|---|---|---|---|---|
| 3 | 16 | 4 | 20 | 10 | 19 | 2 | 27 |
| 4 | 17 | | | | | | |

图 5　初始 3-项集列表

Fig. 5　Initial 3-itemset lists

图 6 为使用策略 3 之后的 3-项集列表.

| $\{ACE\}$ | | $\{ACG\}$ | | $\{AEG\}$ | | $\{BCD\}$ | |
|---|---|---|---|---|---|---|---|
| 2 | 28 | 2 | 37 | 2 | 37 | 4 | 25 |
| 8 | 19 | | | | | 9 | 8 |

| $\{BDE\}$ | |
|---|---|
| 3 | 16 |
| 4 | 17 |

图 6　应用策略 3 后的 3-项集列表

Fig. 6　3-Itemset lists by applying Strategy 3

至此，文章已介绍如何构建项集列表以及 3 种剪枝策略的原理. 接下来，介绍如何从项集列表中直接生成所有的高效用项集挖掘算法 MHUI.

## 2.4　本文提出的算法 MHUI

本文提出的算法 MHUI，仅需扫描一次事务数据库，在不产生候选项集的情况下，直接产生所有的高效用项集. 效用项集挖掘算法 MHUI 表述如下.

Algorithm：MHUI

**Input**：DB：the transaction database；

　　$minutl$：the minimum utility threshold.

**Output**：all the HUIs.

**Step 1**：Produce the initial $ILs$ through scanning the DB. Calculate the initial transaction utility $tu$. Output the HUIs of 1-itemset.

**Step 2**：Delete the unpromise item from the initial $ILs$ and update $tu$.

1. $flag=0$；
2. repeat
3. 　　$flag=IL.size()$；
4. 　　$twu(i_p)=TWU(IL,tu)$；
5. 　　if $(twu(i_p))<minutl$
6. 　　delete $i_p$ from $IL$；
7. 　　end
8. 　　$tu=TU(IL)$；
9. until $IL.size()<flag$
10. Output the HUIs if $u(X)\geqslant minutl$

**Step 3**：$ILs$ of 2-itemsets are generated by initial $ILs$. Delete the unpromise itemset from the $ILs$. Output the HUIs of 2-itemsets.

11. $IL_k=List(IL)$；
12. 　　if $(twu(X))<minutl$
13. 　　delete $X$ from $IL_k$；
14. 　　end
15. Output the HUIs if $u(X)\geqslant minutl$

**Step 4**: *ILs* of *k*-itemsets obtained by *ILs* of (*k*−1)-itemsets

16. repeat
17. $k=k+1$;
18. $IL_k=List(IL_{k-1})$;
19. Output the HUIs if $u(X)\geqslant minutl$
20. until $IL_k.size()=0$

## 3 实 验

为评估提出的算法的性能，将其同最新的算法进行对比. 实验使用的计算机：3.3 GHz Intel i5-4590 处理器；8 GB 内存，Windows 7 操作系统；算法用 Java 语言实现.

### 3.1 实验数据集

实验数据集 Chain-store[13] 是一个真实数据集，采集于一家超市的销售数据. Chain-store 数据集提供了项在事务中的数量和单位利润，可直接使用. 数据集 retail 和 T10I4D100K 来自 FIMI 网站[14]. 3 个数据集的特点如表 3 所示.

表 3 数据集的特点

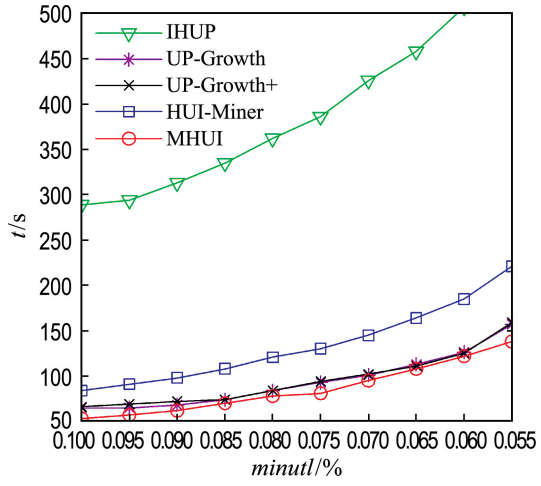
Tab. 3 Characteristics of datasets

| 数据集 | 事务数 | 项数 | 项平均长度 | 项最大长度 | 数据密度/% |
|---|---|---|---|---|---|
| T10I4D100K | 100 000 | 870 | 10.1 | 29 | 1.160 |
| retail | 88 162 | 16 470 | 20.6 | 152 | 0.125 |
| Chain-store | 1 112 949 | 46 086 | 14.4 | 340 | 0.031 |

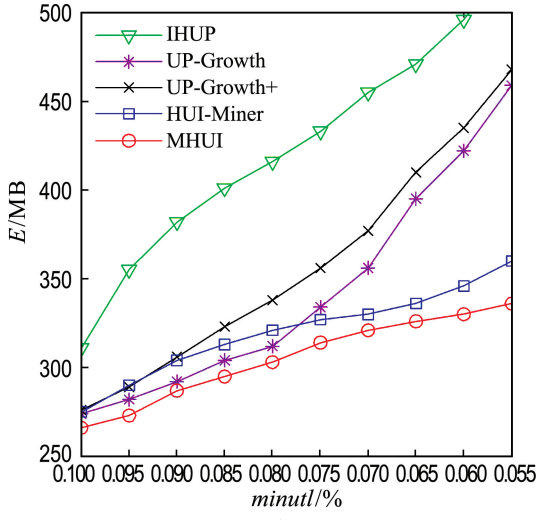
### 3.2 运行时间与内存消耗

为了测试本文提出的算法 MHUI 的性能，对比实验分别采用了 IHUP、HUI-Miner、UP-Growth 和 UP-Growth+等 4 个最新的算法.

图 7 是 Chain-store 数据集的实验结果，显示在不同的最小阈值的情况下，5 种算法的运行时间和内存消耗情况. 图 7(a)显示的是运行时间，从图中可以看到，MHUI 算法比 HUI-Miner 算法要快 1 倍左右，主要原因是，本文提出的剪枝策略，使得参与计算的项集列表数量减少. 图 7(b)显示的是内存消耗，可以看到，MHUI 和 HUI-Miner 比 UP-Growth+消耗的内存要少. 主要原因是，Chain-store 数据集所包含的项较多，达到 46 086 个，导致 UP-Growth+构建的 UP-Tree 规则较大，并且产生数量巨大的候选项集，消耗了大量的内存.




(a) 运行时间




(b) 内存消耗

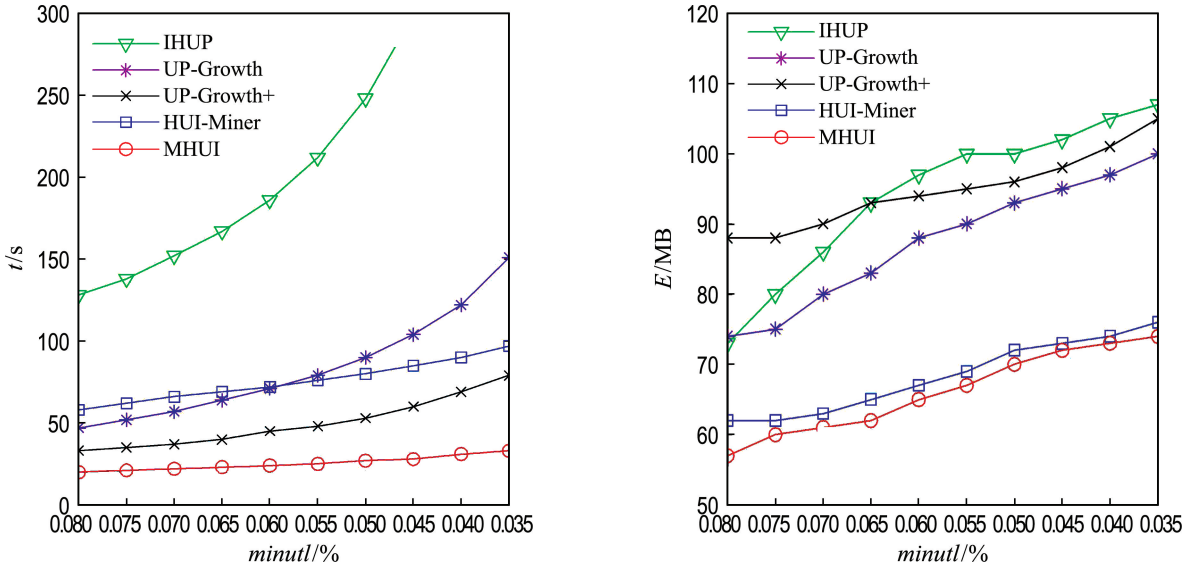
图 7 数据集 Chain-store 上的实验结果

Fig. 7 The experimental results on database Chain-store

图 8 显示了在 retail 数据集上，5 种算法在运行时间和内存消耗上的差异. MHUI 比 HUI-Miner、IHUP、UP-Growth 和 UP-Growth+等 4 个算法在内存消耗上优势明显，运行速度一般也要快 1 倍以上.

图 9 显示在 T10I4D100K 数据集上，5 种算法在运行时间和内存消耗上的差异. 算法 MHUI 在运行时间上和内存消耗上的表现都比另外 4 种算法更好. 其中，在运行时间上，MHUI 比 HUI-Miner 要快 60% 以上，比 UP-Growth 和 UP-Growth+要快 2 倍以上.

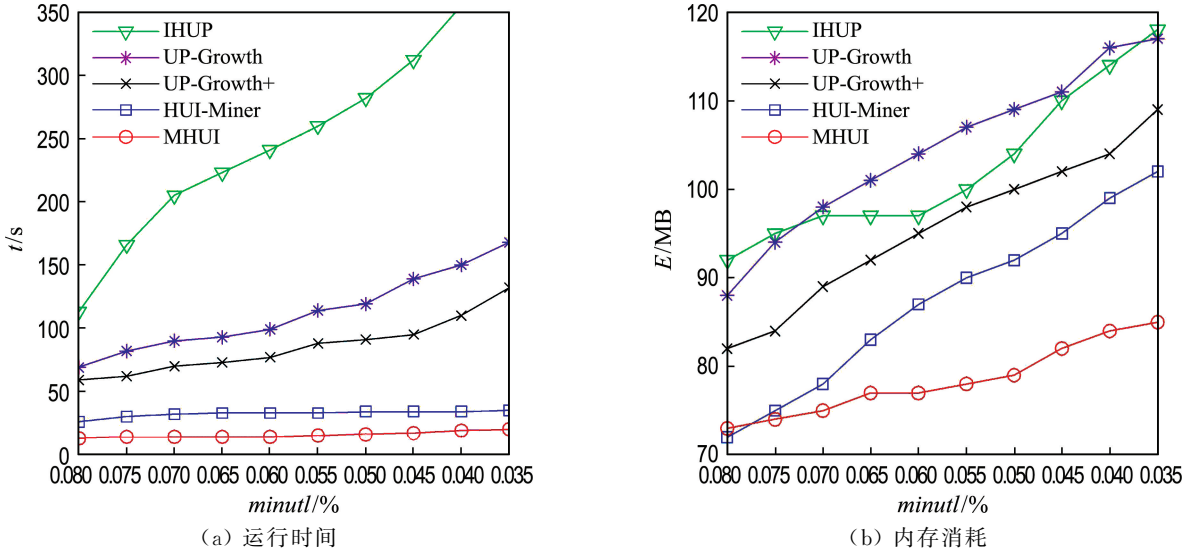
实验结果显示，在不同的数据集上，本文提出的算法在性能上好于最新算法. 主要的原因如下：第一，提出的算法不产生候选项集；第二，提出的项集列表不存储冗余信息，仅存储事务和项的效




（a）运行时间　　（b）内存消耗

图 8　数据集 retail 上的实验结果

Fig. 8　The experimental results on database retail




（a）运行时间　　（b）内存消耗

图 9　数据集 T10I4D100K 上的实验结果

Fig. 9　The experimental results on database T10I4D100K

用信息，占用空间少；第三，提出 3 种剪枝策略，减少了项集列表的数量，从而减少了算法的运行时间和内存消耗.

## 4　结　语

本文提出了一个新的数据结构——项集列表. 仅需扫描一次数据库，就能完成项集列表的构建. 并提出 3 种用于对项集列表进行剪枝的策略，应用这 3 种剪枝策略能减少项集列表的数量，减少算法的执行时间，也能降低内存的消耗. 最后，提出一个算法 MHUI，通过扫描项集列表，直接生成完全高效用项集，在这个过程中，不需要产生候选项集. 该算法运行速度快，且减少内存消耗，性能较好.

## 参考文献：

[1] AGRAWAL R, SRIKANT R. Fast algorithms for milling association rules in large databases [C] // **Proceedings of the 20th VLDB Conference**. Santiago: Morgan Kaufmann Publishers Inc., 1994:487-499.

[2] HAN Jiawei, PEI Jian, YIN Yiwen. Mining frequent patterns without candidate generation [C] // **Proceedings of the ACM SIGMOD International Conference on Management of Data.**

Dallas: ACM, 2000:1-12.

[3] ZAKI M J. Scalable algorithms for association mining [J]. **IEEE Transactions on Knowledge and Data Engineering**, 2000, **12**(3):372-390.

[4] LIU Y, LIAO W, CHOUDHARY A. A two-phase algorithm for fast discovery of high utility itemsets [C] // **Proceedings of the 9th Pacific-Asia Conference on Knowledge Discovery and Data Mining**. Vietnam:Springer, 2005:689-695.

[5] YAO Hong, HAMILTON H J. Mining itemset utilities from transaction databases [J]. **Data & Knowledge Engineering**, 2006, **59**(3, SI):603-626.

[6] LI Y C, YEH J S, CHANG C C. Isolated items discarding strategy for discovering high utility itemsets [J]. **Data & Knowledge Engineering**, 2008, **64**(1):198-217.

[7] AHMED C F, TANBEER S K, JEONG B S, *et al*. Efficient tree structures for high utility pattern mining in incremental databases [J]. **IEEE Transactions on Knowledge and Data Engineering**, 2009, **21**(12):1708-1721.

[8] TSENG V S, WU Chengwei, SHIE Baien, *et al*. UP-Growth: an efficient algorithm for high utility itemset mining [C] // **Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. Washington D C: ACM, 2010:253-262.

[9] TSENG V S, SHIE Baien, WU Chengwei, *et al*. Efficient algorithms for mining high utility itemsets from transactional databases [J]. **IEEE Transactions on Knowledge and Data Engineering**, 2013, **25**(8): 1772-1786.

[10] WU Chengwei, FOURNIER-VIGER P, YU P S, *et al*. Efficient mining of a concise and lossless representation of high utility itemsets [C] // **Proceedings - IEEE International Conference on Data Mining, ICDM**. Vancouver: IEEE, 2011:824-833.

[11] 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 and Data Engineering**, 2015, **27**(3):726-739.

[12] LIU Mengchi, QU Junfeng. Mining high utility itemsets without candidate generation [C] // **CIKM 2012 - Proceedings of the 21st ACM International Conference on Information and Knowledge Management**. Maui :ACM, 2012:55-64.

[13] PISHARATH J, LIU Y, PARHI J, *et al*. NU-MineBench Version 3.0.1 [DB/OL]. [2016-06-30]. http://cucis.ece.northwestern.edu/projects/DMS/MineBench.html

[14] GOETHALS B, ZAKI M J. Frequent itemset mining implementations repository [DB/OL]. [2016-06-30]. http://fimi.ua.ac.be/

# An algorithm of mining high utility itemsets with vertical structures

HUANG Kun[*1], WU Yujia[2]

(1. China Ship Development and Design Center, Wuhan 430064, China;
2. School of Computer, Wuhan University, Wuhan 430072, China)

**Abstract**: Mining high utility itemsets (HUIs) is one of popular tasks in field of association analysis. Most of HUIs mining algorithms need to generate a lot of candidate itemsets (CIs) which will affect the performance of algorithm. HUI-Miner can mine all the HUIs from a transaction database without generating CIs. However, this algorithm generates a large number of utility lists (ULs) and so many ULs not only consume too much storage space but also affect the operation performance. To solve this problem, itemsets lists (ILs), new data structures are proposed to maintain information of transaction and item utility. Three pruning strategies are proposed to reduce the number of ILs and can build the ILs just scanning the transaction database only once. A new algorithm namely MHUI is proposed which mines all the HUIs directly from the ILs without generating any CIs. The experimental results show that the proposed method outperforms the state-of-the-art algorithms in terms of runtime and memory consumption on three different sparse datasets.

**Key words**: data mining; association analysis; frequent itemsets; high utility itemsets