基于差集的高效用项集挖掘方法

黄 坤[1],吴玉佳[2],李 晶[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: 高效用项集挖掘已成为关联规则中的一个热点研究问题. 一些基于垂直结构的算法已用来挖掘高效用项集,此类算法的主要优点是将项集的事务和效用信息存储到效用列表中. 在求一个项集的超集所在事务可以通过对它的子集进行一次交集运算得到. 这种算法在稀疏数据集中非常的有效. 但在稠密数据集中存在一个问题,即列表中存储的事务太多,在计算用于剪枝的效用上界时,需要耗费大量的存储空间,同时也影响运行速度. 并且在现有的算法中,缺乏针对稠密数据集的高效用项集挖掘算法,往往需要设置很高的最小效用阈值,影响算法的运行效率. 针对此问题,提出一个新的算法 D-HUI(mining High Utility Itemsets using Diffsets)以及一个新的数据结构—项集列表,首次在高效用项集挖掘中引入差集的概念. 利用事务的差集求项集的效用上界,减少计算量以及存储空间,从而提高算法的运行效率. 实验结果表明,提出的算法在稠密数据集中,执行速度更快,内存消耗更少.

关键词: 关联规则; 高效用项集; 稠密数据集; 垂直结构; 差集
中图分类号: TP311 **文献标识码:** A **文章编号:** 0372-2112 (2018)08-1804-11
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2018.08.002

Mining High Utility Itemsets Using Diffsets

HUANG Kun[1], WU Yu-jia[2], LI Jing[2]
(1. *China Ship Development and Design Center, Wuhan, Hubei* 430072, *China*;
2. *School of Computer, Wuhan University, Wuhan, Hubei* 430072, *China*)

Abstract: High utility itemsets mining (HUIM) has become an emerging topic in association rules. Some algorithms based on vertical data structure have been used for mining high utility itemsets(HUIs), and the main advantage of the algorithms are to maintain transaction and utility information of itemsets in some utility lists(ULs). The transactions of superset of an itemsets can be calculated by its subset doing an intersection. These algorithms are very effective in sparse datasets. However, in the dense datasets, a problem is that: too many transactions maintained in ULs, not only required a lot of memory space, but also affected the runtime when computing the upper bound of utility in order to prune search space. Few of existing HUIM algorithm focused on dense datasets and it often need to set a high minimum threshold utility which affect the running efficiency of the algorithm. To solve this problem, propose a new algorithm D-HUI(mining High Utility Itemsets using Diffsets) and a new data structure, namely Itemset Lists (ILs). Introduce the concept of diffsets in the HUIM. Calculate upper bound of utility by using diffsets of transaction for pruning search space. The runtime and memory consumption are reduced, and the running efficiency of the algorithm is improved. Experimental results show that the proposed algorithm in the dense datasets outperforms state-of-the-art algorithms in terms of both running time and memory consumption.

Key words: association rules; high utility itemsets; dense datasets; vertical structure; diffsets

1 引言

数据挖掘中的一个重要任务是关联分析. 关联分析中最主要的步骤是频繁项集挖掘[1~4]. 然而,在频繁项集挖掘的框架中[5,6],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、权重等). 然而在一些实际应用中,有时不仅仅需要考虑项是否频繁出现,也需要考虑此项出现的次数、权重等因素. 因此,发现高效用项集(High Utility Itemsets, HUIs)迅速成为数据挖掘领域中的一个研究热点问题之一.

从另一个角度,频繁项集挖掘可以看成是高效用项集挖掘的一种特殊情况. 如果不考虑项的效用值的具体大小而只考虑项在事务中出现或不出现,或者设置效用值为 1,高效用项集挖掘就等同于频繁项集挖

收稿日期:2016-11-26;修回日期:2017-06-30;责任编辑:梅志强
基金项目:国家自然科学基金(No. 61303046)

掘. 通常情况下,因为附加条件较少,频繁项集挖掘比高效用项集挖掘的效率更高,但是缺点也较为明显,即无法扩展到更多的应用领域,因为很多应用领域是需要考虑项的出现次数、权重等因素.

所以,挖掘 HUIs 并不是一个简单的任务. 相对于频繁项集挖掘,它常常更加困难. 在频繁项集挖掘中,多数算法使用了向下闭性质,即如果一个项集是频繁的,它的超集也都是频繁的,反之,如果一个项集是非频繁的,则它的超集都是非频繁的[1,2]. 利用这个性质对搜索空间进行剪枝,能大大减少计算量并降低内存消耗. 然而,这个性质在高效用项集挖掘中却不能直接使用. 因为,如果一个项集是低效用项集,不能断定它的超集是否为低效用项集,或者一个项集是高效用项集,也不能断定它的超集是否为高效用项集. 这个问题给高效用项集挖掘带来了一个巨大挑战.

鉴于此,一些算法使用效用上界估计方法来对搜索空间进行剪枝,以提升高效用项集挖掘的性能[7~9]. Liu 等[7]提出 Two-Phase 算法,算法通过两个阶段来确定 HUIs. Yao 等[8]提出了 UMining 算法,使用一种估计方法来减少搜索空间. Li 等[9]提出一个孤立项丢弃策略,用于减少候选项集的数量. 在这些方法中,挖掘过程一般分为两个阶段,第一阶段,寻找潜在的 HUIs(效用上界大于或等于最小效用阈值,称为候选项集),第二阶段,根据这些候选项集,再次扫描数据库计算其真实的效用,以确定最终的 HUIs. 在这些方法中,使用包含每个候选项集的事务效用之和来满足向下闭性质,从而对搜索空间进行剪枝,以降低候选项集的数量. 虽然所有的 HUIs 都能够被发现,但这些方法经常会产生大量的候选项集,并且需要多次扫描数据库.

为了避免多次扫描数据库,Ahmed 等[10]提出一个基于树结构的算法 IHUP 用于挖掘高效用项集. 使用一个树结构来存储项的事务和效用信息. IHUP 获得比 IIDS 和 Two-Phase 更好的性能 Tseng 在 IHUP 的基础上,提出了 UP-Growth 算法[11]和 UP-Growth$^+$ 算法[12],并提出一些剪枝策略,减少候选项集的数量,从而提高算法的性能. 但是,以上两阶段算法都需要先产生候选项集,再根据候选项集重新扫描数据库计算其真实的效用,以确定最终的 HUIs,这对算法的挖掘性能产生了较大的影响.

为了解决这个问题,Liu 等[13]提出一种的用于挖掘 HUIs 的算法 HUI-Miner. 它和两阶段算法有着本质区别,是一种基于垂直数据结构的算法,它不需要产生候选项集而是直接产生 HUIs. 首先产生一系列称为效用列表(Utility Lists,ULs)的数据结构,用于存储项集的事务信息、项的效用信息以及能对搜索空间进行剪枝的额外剩余效用信息. 在产生完所有的 ULs 之后,通过扫描 ULs 的方式生成所有的 HUIs,而这一过程不需要产生任何候选项集. Philippe 等[14]在 HUI-Miner 的基础上提出了 FHM 算法,增加了一个新的策略 EUCP,用于减少连接操作的数量. Krishnamoorthy[15]等提出了 HUP-Miner 算法,同 HUI-Miner 一样,这也是一种基于垂直结构的算法,此算法应用两个新的剪枝策略,称为划分效用剪枝和前向效用剪枝,算法在效率上略胜于 HUI-Miner.

由于不产生任何候选项集,在许多数据集上,基于垂直结构的算法性能都优于 UP-Growth 等基于树结构的算法. 但是,这些基于垂直的算法不仅需要存储项集的事务和效用信息,也需要存储用于对搜索空间进行剪枝的额外剩余效用信息,这也降低了算法性能并占用了更多的内存资源. 尤其是在稠密数据集的情况下,这种情况更加的严重. 在计算项集的效用上界时,也需要增加更多的运算时间和内存消耗.

针对上述问题,提出一个新的数据结构—项集列表(Itemset Lists,ILs)和一个挖掘算法 D-HUI,在稠密数据集中挖掘 HUIs. 项集列表仅存储项集的事务和效用信息. 提出使用差集的方法计算项集的剩余效用,从而计算得到项集的效用上界估计值,减少运算时间. 算法 D-HUI,直接从项集列表中直接发现所有的 HUIs 而不产生任何候选项集.

2　相关工作与问题定义

在这部分,首先给出高效用项集挖掘的问题定义,然后介绍相关工作.

2.1　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 $i_k(1\leqslant k\leqslant m)$ 都有一个外部效用值 $\mathrm{EU}(i_k)$,在这里的外部效用为单位利润值,如表 1 所示. 一个项集 P 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 P 的长度. 长度为 k 的项集称为 k-项集.

表 1　利润表

Item	A	B	C	D	E	F	G
Profit	7	2	1	2	3	2	3

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 $T_{\mathrm{id}}(1\leqslant \mathrm{id}\leqslant n)$ 都是 I 的一个子集,包含一个或多个项,具有一个唯一的标识符 T_{id}. 每个事务 T_{id} 中的一个项 i_k 具有一个内部效用值 $\mathrm{IU}(i_k,T_{\mathrm{id}})$,在这里内部效用值为项在事务中的数量,如表 2 所示.

表 2　事务数据库

TID	Transaction	TU
T_{01}	$(A,1)$ $(C,10)$ $(E,2)$	23
T_{02}	$(A,2)$ $(B,1)$ $(C,6)$ $(E,1)$ $(G,5)$	40
T_{03}	$(B,1)$ $(D,4)$ $(F,1)$	12
T_{04}	$(B,3)$ $(C,11)$ $(D,3)$ $(E,1)$	26
T_{05}	$(A,1)$ $(B,2)$ $(C,3)$ $(E,1)$ $(G,2)$	23

定义 1 项 i_k 在事务 T_{id} 中的效用 $U(i_k,T_{id})$,定义为 $U(i_k,T_{id})=EU(i_k)\times IU(i_k,T_{id})$. 例如,在表 2 中,$U(\{A\},T_{01})=7$.

定义 2 项集 P 在事务 T_{id} 中的效用 $U(P,T_{id})$,定义为 $U(P,T_{id})=\sum_{i_k\in P\wedge P\subseteq T_{id}}U(i_k,T_{id})$. 例如,$U(\{AC\},T_{01})=U(\{A\},T_{01})+U(\{C\},T_{01})=17$.

定义 3 项集 P 在数据库 D 中的效用 $U(P)$,定义为 $U(P)=\sum_{P\subseteq T_{id}\wedge T_{id}\subseteq D}U(P,T_{id})$. 例如,$U(\{AC\})=U(\{AC\},T_{01})+U(\{AC\},T_{02})+U(\{AC\},T_{05})=47$.

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

定义 5 事务 T_{id} 的事务效用记为 $TU(T_{id})$,为该事务中所有项的效用之和,定义为 $U(T_{id})=\sum_{P=T_{id}\wedge T_{id}\subseteq D}U(P,T_{id})$. 例如,$TU(T_{01})=U(\{ACE\},T_{01})=23$.

定义 6 项集 P 的事务加权效用是所有包含项集 P 的事务效用之和,记为 $TWU(P)$,定义为 $TWU(P)=\sum_{P\subseteq T_{id}\wedge T_{id}\subseteq D}TU(T_{id})$. 例如,项集 $TWU(\{AC\})=TU(T_{01})+TU(T_{02})+TU(T_{05})=86$. 如果 $TWU(P)\geqslant$minutl,则称 P 为一个高事务加权效用项集(HTWUI).

性质 1 事务加权向下闭性质[8~10,12](TWDC). 规定如下:对于任何一个项集 P,如果 P 不是 HTWUI,则 P 的超集都不是高效用项集.

定义 7 设项在事务中以字母顺序排列,令 T_{id}/P 中是在事务 T_{id} 的项集 P 后的一组项. 例如,在表 2 中,$T_{01}/AC=E$,$T_{02}/AC=EG$.

定义 8 给定一个项集 P,它在事务 T_{id} 中的剩余效用记为 $RU(P,T_{id})$,包含了在事务 T_{id} 中项集 P 后的所有项的效用之和,记为 $RU(P,T_{id})=\sum_{p_i\in T_{id}/P\wedge T_{id}\subseteq D}U(p_i,T_{id})$. 例如,$RU(\{AC\},T_{01})=6$.

定义 9 给定一个项集 P,它在数据库 D 中的剩余效用记为 $RU(P)$,定义为 $RU(P)=\sum_{P\subseteq T_{id}\wedge T_{id}\subseteq D}RU(P,T_{id})$. 例如,$RU(\{AC\})=33$.

定义 10 项集 P 在数据库 D 中所在的事务的集合记为 $t(P)$,定义为 $t(P)=\{T_{id}\mid P\subseteq T_{id}\wedge T_{id}\subseteq D\}$. 例如,$t(AC)=t\{1,2,5\}$,$t(C)=t\{1,2,4,5\}$.

性质 2 事务集合向下闭性质,设项集 P 的一个超集为 P',即 $P\subseteq P'$,则 $t(P')\subseteq t(P)$.

证明:

$$\begin{aligned}t(P')&=\{T_{id}\mid P'\subseteq T_{id}\wedge T_{id}\subseteq D\}\\&\subseteq\{T_{id}\mid P\subseteq T_{id}\wedge T_{id}\subseteq D\}\\&=t(P)\end{aligned}$$

性质 3 一个项集的效用和剩余效用之和如果小于用户指定最小效用阈值 minutl,即 $U(P)+RU(P)<$minutl,则项集 P 及其超集 P'都不是高效用项集.

证明:

$$\begin{aligned}U(P',T_{id})&=U(P,T_{id})+U(T_{id}/P,T_{id})\\&\leqslant U(P,T_{id})+\sum_{p_i\in T_{id}/P\wedge T_{id}\subseteq D}U(p_i,T_{id})\\&=U(P,T_{id})+RU(P,T_{id})\end{aligned}$$

根据性质 2,有 $t(P')\subseteq t(P)$,则

$$\begin{aligned}U(P')&=\sum_{P'\subseteq T_{id}\wedge T_{id}\subseteq D}U(P',T_{id})\\&\leqslant\sum_{P'\subseteq T_{id}\wedge T_{id}\subseteq D}(U(P,T_{id})+RU(P,T_{id}))\\&\leqslant\sum_{P\subseteq T_{id}\wedge T_{id}\subseteq D}(U(P,T_{id})+RU(P,T_{id}))\\&=U(P)+RU(P)\end{aligned}$$

2.2 相关工作

数据挖掘中的一个重要任务是关联分析. 在传统的关联分析算法中主要是以挖掘频繁项集作为主要目标,如 Han 等人提出了 FP-Growth 算法[1]及 Zaki 等人提出 Eclat 算法[2],都是用于挖掘事务数据库中频繁出现的项的集合(频繁项集).

但是,在大数据时代,数据量越来越大,相关应用的要求的也在逐渐提高,频繁项集已经不能满足更多的需求. 例如,如果需要考虑事务中项的数量以及重要性(如单位利润、价格、权重等). 这些因素在频繁项集的框架中并没有被考虑. 频繁项集仅考虑项在事务中是否出现,不考虑重要性或数量等因素,而高效用项集挖掘正好能解决此问题. 所以,为了适应更多的应用需求,高效用项集挖掘迅速成为数据挖掘领域中的一个研究热点问题之一.

在 2005 年,Liu 等[7]提出 Two-Phase 算法,算法通过两个阶段来确定高效用项集. 2006 年,Yao 等[8]提出了 UMining 算法,使用一种估计方法来减少搜索空间. 接着在 2008 年,Li 等[9]提出 IIDS 算法用于减少候选项集的数量. 在这些方法中,首先寻找候选项集,然后根据这些候选项集,再次扫描数据库计算其真实的效用,以确定最终的 HUIs. 在这些产生-测试的方法中,存在一个问题:虽然所有的 HUIs 都能够被发现,但这些方法经常会产生大量的候选项集,并且需要多次扫描数据库.

针对此问题,在 2009 年,Ahmed 等[10]提出一个基于树结构的算法 IHUP 用于挖掘高效用项集. 使用一个树结构来存储事务和项的效用信息. IHUP 获得比产生-测试的方法更好的性能. 2010 年,Tseng 在 IHUP 的基础上,提出了 UP-Growth 算法[11]和 UP-Growth$^+$算法[12],并提出一些剪枝策略,减少候选项集的数量,从而提高算法的性能. 此外,Tseng 等[16,17]也提出了先挖掘闭(Closed)项集的方式来无损的挖掘完全 HUIs,也取得

了较好的效果. 这些基于树结构的算法,首先需要将事务和效用信息存储到一个树上,然后再产生候选项集,最后确定 HUIs.

在 2012 年,Liu 等[13]提出一种全新的用于挖掘高效用项集的算法 HUI-Miner. HUI-Miner 和两阶段算法有着本质区别,HUI-Miner 是一种基于垂直数据结构的算法,它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 ULs 的数据结构,包括三个字段:事务、项的效用和剩余效用. 其中,剩余效用是为了对搜索空间进行剪枝的额外高估效用信息. 在产生完所有的 ULs 之后,通过扫描 ULs 的方式生成所有的 HUIs,而这一过程不需要产生任何候选项集. 由于不产生任何候选项集,在一些数据集上,HUI-Miner 算法性能在运行时间和内存消耗上都优于 UP-Growth 和 UP-Growth$^+$算法. 2014 年,Philippe 等[14]在 HUI-Miner 的基础上提出了 FHM 算法,增加了一个新的策略 EUCP,用于减少连接操作的数量. 2015 年,Krishnamoorthy[15]等提出了 HUP-Miner 算法,和 HUI-Miner 一样,这是一种基于垂直结构的算法,此算法应用两个新的剪枝策略,称为划分效用剪枝和前向效用剪枝,算法在效率上略胜于 HUI-Miner. 在 2015 年和 2016 年,Sahoo 和 Wu 分别实现了基于垂直结构的闭(Closed)项集的挖掘 HUIs 的算法 CHUM[18]和 CHUIMiner[19]. Guo 提出了 HUIT-WU[20]算法,结合了垂直结构和树结构算法.

3　提出的方法

在这部分,详细介绍提出的数据结构和算法. 提出的数据结构 ILs 中仅存储项的事务和效用信息,不存储额外的用于剪枝的高估效用信息,与文献[13]提出的效用列表有本质的不同. 扫描一次事务数据库,产生一个事务项效用表(Transaction Item and Utility table,TIU)表,计算得到每个项集的效用上界,有效的对项集进行剪枝,在产生所有的项集列表后,计算得到完全高效用项集.

3.1　初始项集列表

首先,扫描一次原始数据库,如表 1 和表 2 所示. 根据定义 6,利用表 1 和表 2 计算得到每个项的 TWU 值,各项的 TWU 值如表 3 所示.

表 3　TWU 表

Item	A	B	C	D	E	F	G
Profit	86	101	112	38	112	12	63

由表 3 可以看到,项 F 的 TWU 值为 12,设最小阈值 minutl = 18.6(15%),则 TWU($\{F\}$) < minutl. 根据性质 1,项 F 和它的超集都不是高效用项集. 称这样的项为无用项,而 TWU 值高于或等于 minutl 的称为有用项.

定义 11　(有用项和无用项)给定一个项 i_p,如果 TWU(i_p) ≥ 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[11,12].

同时,根据定义 1,计算每个项的效用值,得到一个 TIU 表,它存储了一个事务所包含的有用项及对应的效用值. 如表 4 所示.

表 4 中,每个事务的项及其在此事务中的对应效用值分别存储. 容易得到一个项或项集在某事务的剩余效用值. 例如,在事务 T_{01} 中,项 A 的剩余效用 RU(A,T_{01}) = 16,项集$\{AC\}$的剩余效用 RU(AC,T_{01}) = 6.

表 4　TIU 表

Trans	Items	Utility
T_{01}	A,C,E	7,10,6
T_{02}	A,B,C,E,G	14,2,6,3,15
T_{02}	B,D	2,8
T_{04}	B,C,D,E	6,11,6,3
T_{05}	A,B,C,E,G	7,4,3,3,6

记函数 RU(P,T_s)用于求一个项集 P 在事务 T_s 中的剩余效用值. 此函数的执行如下:

(1)获取项集 P 中的最后一个项 last;

(2)对于每个项,扫描 TIU 表,获得 last 在事务 T_s 中的位置 pos;

(3)将 pos 后面的效用值累加,得到项集 P 的 RU 值.

在使用策略 1 之后,去掉无用项,得到初始项集列表,如图 1 所示. 初始项集列表中存储每个项所包含的事务以及对应的效用. 例如,项 A 在事务 T_{01},T_{02},T_{05} 中出现,对应的效用分别为 7,14,17.

$\{A\}$	$\{B\}$	$\{C\}$	$\{D\}$	$\{E\}$	$\{G\}$
1 7	2 2	1 10	3 8	1 6	2 15
2 14	3 2	2 6	4 6	2 3	5 6
5 7	4 6	4 11		4 3	
	5 4	5 3		5 3	

图1　初始项集列表

然后根据 TIU 表求得每个可用项的 RU 值. 使用函数 RU(P,T_s)求各个项的剩余效用,得到 RU(A):58,RU(B):64,RU(C):42,RU(D):3,RU(E):21,RU(G):0.

3.2　挖掘 2-项集列表

在产生初始项集列表之后,各项以 TWU 升序排列. 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 2 和 5,将对应的效用值进行累加,得到 16 和 11,2-项集列表如图 2 所示.

项集列表仅存储了项的事务和效用信息,没有存储用于剪枝的高估效用信息. 在这里,需要计算得到每个项的剩余效用值. 如果使用函数 RU(P,T_s)扫描得到

{AB}	{AC}	{AE}	{AG}	{BC}	{BD}	{BE}	{BG}
2 16	1 17	1 13	2 29	2 8	3 10	2 5	2 17
5 11	2 20	2 17	5 13	4 17	4 12	4 9	5 10
	5 10	5 10		5 7		5 7	

{CD}	{CE}	{CG}	{DE}	{EG}
4 17	1 16	2 21	4 9	2 18
	2 9	5 9		5 9
	4 14			
	5 6			

图2 产生的2-项集列表

每个项的剩余效用也是一种可行的办法,但这种办法的效率并不高,因为在稠密数据集中,每个项的事务数量非常庞大,需要扫描的事务数非常多,所以效率较低.

针对此问题,在本文中,引入事务差集的概念求项集的剩余效用值.

定义 12 $t\{A\}$、$t\{B\}$是事务集合,由属于$t\{A\}$,而不属于$t\{B\}$的项构成的集合,称之为$t\{A\}$与$t\{B\}$的差集,记作$t\{A\}\backslash t\{B\}$.

例如,事务$t\{1,2,4,5\}$与事务$t\{2,3,4,5\}$的差集的结果为$t\{1\}$.

定义 13 两个k-项集$Px=\{p_1,p_2,\cdots,p_{k-1},x\}$和$Py=\{p_1,p_2,\cdots,p_{k-1},y\}$,包含共同的前缀项集$P=\{p_1,p_2,\cdots,p_{k-1}\}$. 定义$Pxy$的事务差集(diffset)$d(Pxy)$由式(1)计算得到.

$$d(Pxy)=t(Py)\backslash t(Px) \tag{1}$$

其中,如果共前缀项集$P=\{p_1,p_2,\cdots,p_{k-1}\}$为空集,则式(1)可简化表示为式(2)

$$\mathrm{d}(xy)=t(y)\backslash t(x) \tag{2}$$

例如,在表2中,$t(A)=t\{1,2,5\}$,$t(AB)=t\{2,5\}$,$t(AC)=t\{1,2,5\}$,则$d(ABC)=t(AC)\backslash t(AB)=t\{1,2,5\}\backslash t\{2,5\}=t\{1\}$.

性质 4 RU(Pxy)等于RU(Py)减去RU($d(Pxy)$),如式(3)所示.

$$\mathrm{RU}(Pxy)=\mathrm{RU}(Py)-\mathrm{RU}(Pxy,d(Pxy)) \tag{3}$$

证明:

$$\begin{aligned}
\mathrm{RU}(Pxy)&=\sum_{Pxy\subseteq T_{id}\wedge T_{id}\subseteq D}\mathrm{RU}(Pxy,T_{id})\\
&=\sum_{(t(Px)\cap t(Py))\subseteq T_{id}\wedge T_{id}\subseteq D}\mathrm{RU}(Pxy,T_{id})\\
&=\sum_{(t(Py)\cap[\overline{t(Py)}\cup t(Px)])\subseteq T_{id}\wedge T_{id}\subseteq D}\mathrm{RU}(Pxy,T_{id})\\
&=\sum_{(t(Py)\cap\overline{[t(Py)\cap t(Px)]})\subseteq T_{id}\wedge T_{id}\subseteq D}\mathrm{RU}(Pxy,T_{id})\\
&=\sum_{(t(Py)\backslash[t(Py)\cap t(Px)])\subseteq T_{id}\wedge T_{id}\subseteq D}\mathrm{RU}(Pxy,T_{id})\\
&=\sum_{Py\subseteq T_{id}\wedge T_{id}\subseteq D}\mathrm{RU}(Py,T_{id})\\
&\quad-\sum_{(t(Py)\backslash t(Px))\subseteq T_{id}\wedge T_{id}\subseteq D}\mathrm{RU}(Pxy,T_{id})\\
&=\sum_{Py\subseteq T_{id}\wedge T_{id}\subseteq D}\mathrm{RU}(Py,T_{id})\\
&\quad-\sum_{d(Pxy)\subseteq T_{id}\wedge T_{id}\subseteq D}\mathrm{RU}(Pxy,T_{id})\\
&=\mathrm{RU}(Py)-\mathrm{RU}(Pxy,d(Pxy))
\end{aligned}$$

例如,求项集$\{BC\}$的剩余效用值RU(BC),在初始项集列表中,已经得到RU(C)=42,根据式(3),RU(BC)的值等RU(C)减去RU(BC,d(BC)). 这里的$d(BC)$根据式(2)得到$t\{1,2,4,5\}\backslash t\{2,3,4,5\}=t\{1\}$. 所以,RU($BC$,$d(BC)$)=RU($BC$,$t\{1\}$),根据函数RU($P$,$T_s$)扫描事务$T_{01}$易得RU($BC$,$t\{1\}$)=6,这里仅需要扫描事务$T_{01}$.

所以,第一步计算得到两个项集的差集,再扫描差集的剩余效用,就可以利用前面得到的子集剩余效用,根据式(3)计算得到项集的剩余效用值RU(BC)=RU(C)−RU(BC,$d(BC)$)=42−6=36. 扫描事务的次数从3次减少到1次,大大提高了运行效率. 通过上述方法,可以计算得到所有2-项集的剩余效用值.

定义 14 (有用项集和无用项集)给定一个项集P,根据性质3,如果$U(P)+\mathrm{RU}(P)\geqslant$minutl,则称其为有用项集;否则,称其为无用项集.

策略 2 删除项集列表中的无用项集.

理论说明 将效用上界小于用户指定最小阈值minutl的项集列表从全局项集列表中移除. 根据性质3,无用项集及其超集都不是高效用项集,可以删除,而不会影响高效用项集的发现.

例如,项集$\{DE\}$的效用为$U(DE)=9$,RU(DE)=0,则$U(DE)+\mathrm{RU}(DE)=9<$minutl,根据性质性3,项集$\{DE\}$及其超集都不是高效用项集.

3.3 挖掘 k-项集列表($k>2$)

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2所示的2-项集列表,进行事务交集运算,生成3-项集列表. 例如,项集列表$\{ABC\}$根据项集列表$\{AB\}$、$\{AC\}$和$\{A\}$产生,在产生k-项集列表($k>2$)时,需要减去重复效用值,文献[13]有详细说明. 项集$\{AB\}$和项集$\{AC\}$的共同事务为$t\{2,5\}$,将对应的效用值累加,但同时要减少这两个项集的共同前缀项集$\{A\}$在事务2和5的效用,最后得到项集$\{ABC\}$的事务对应的效用. 生成所以$k(k>2)$项集都需要减去重复效用.

算法Construct是用于生成所以k-项集列表的算法. 其伪代码如算法1所示.

算法 1 Construct

Input: P. IL: the ILs of itemset P;
Px. IL: the ILs of itemset Px;

Py. IL:the ILs of itemset *Py*.
Output: *Pxy*. IL:the ILs of itemset *Pxy*.
1. *Pxy*. IL = null;
2. RU = *Py*. RU-RU(*Py*. l,*t*|*Py*|*t*|*Px*|);
3. froeach element *Ex* ∈ *Px* do
4. if ∃ *Ey* ∈ *Py*. IL and *Ex*. *n* = *Ey*. *n* then
5. if *P*. IL is not empty then
6. Search *E* ∈ *P*. IL that *E*. *n* = = *Ex*. *n*;
7. *Exy* = ⟨*Ex*. *n*,*Ex*. *u* + *Ey*. *u*-*E*. *u*⟩;
8. else
9. *Exy* = ⟨*Ex*. *n*,*Ex*. *u* + *Ey*. *u*⟩;
10. end
11. *Pxy*. RU = RU;
12. append *Exy* to *Pxy*. IL;
13. end
14. end
15. return *Pxy*. IL;

在算法 1 中,*Py*. *l* 表示 *Py* 的最后一项,*n* 表示项集所在的事务,*u* 表示项集的效用值。

至此,文章已介绍如何构建项集列表以及两种剪枝策略的原理. 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 D-HUI.

3.4　算法 D-HUI

本文提出的算法 D-UHI,通过扫描一次事务数据库,得到一个事务项效用表 TIU. 在不产生候选项集的情况下,直接产生完全高效用项集. 效用项集挖掘算法 D-UHI 包括两个算法:前面描述的算法 1:Construct 和算法 2:D-Miner.

算法 2　D-Miner

Input: *P*. IL:the ILs of itemset *P*,initially empty;
Ils:the set of ILs *P*′1-extensions;
Minutl:the minimum utility threshold.
Output: all the HUIs with *P* as prefix.
1. froeach IL *X* in ILs do
2. if SUM(*X*. *u*) ≥minutl then
3. output the extension associated with *X*;
4. end
5. if SUM(*X*. *u*) + *X*. RU≥minutl then
6. *ex*Ils = null;
7. froeach IL *Y* after *X* in ILs do
8. *ex*ILs + = Construct(*P*. IL,*X*,*Y*);
9. end
10. D-Miner (*X*,*ex*ILs,minutl)
11. end
12. end

算法 D-UHI 容易实现,首先,确定每个项集的效用是否不小于阈值 minutl,如果阈值不小于 minutl,则输出此高效用项集.

3.5　算法复杂度分析

设数据库 *D* 有 *m* 个事务,其中最大的事务包含 *n* 个项. 则算法的第一步是计算 TU 值和每个项的 TWU 值,需要扫描一次数据库,时间复杂度为 $O(mn)$. 两个长度分别为 *m* 和 *n* 的事务进行交集运算,它的算法复杂度为 $O(m+n)$. 仅需要在初始项集列表中使用函数 RU(*P*,*Ts*)求各个项的剩余效用,复杂度为 $O(n)$. 而 $k(k\geq 2)$项集的剩余效用是根据 *k*-1 项集的剩余效用和事务差集计算得到. 所以,本文提出的算法 D-UHI 的总体算法复杂度为 $\max\{O(mn)+O(n)+O(\sum_{i=2}^{k-1}|\mathrm{IL}_i|^2)\}$,其中$|\mathrm{IL}_i|$,表示项集列表 ILs 的个数,*k* 表示项集列表 ILs 的维数.

4　实验评估

在这一部分,为了测试本文提出的算法 D-HUI 的性能,对比实验分别采用了 FHM[14]、HUI-Miner[13]、HUP-Miner[15]、UP-Growth[11]和 UP-Growth$^+$[12]等五个算法进行对比,对比算法的 java 源代码来自 UP-Miner[21]和 SPMF[22]. 实验使用的计算机:3.6GHz Intel i3-4160 处理器;16G 内存,Windows 10 操作系统;D-HUI 算法是用 java 语言实现.

4.1　实验数据集

实验数据集来自 FIMI[23]及 NU-MineBench[24]. 实验挑选的 8 个数据集都没有提供对应的数量和单位利润. 在这里,以产生随机数据的方式生成项在事务中的数量和单位利润数据. 其中,项的数量服从均匀分布,取值范围 1 到 10;单位利润服从对数正态分布,取值范围为 0.01 到 10.

实验数据集如表 5 所示. 表 5 有 6 列,第 1 列是数据集名称,第 2 列是数据集的事务总数,第 3 列是项的总数,第 4 列是每个事务包含项的平均长度,第 5 列是事务包含最大项的长度,最后一项表示数据集的密度. 其中 chess 数据集密度最大,达到了 49.33%,而 kosarak 是 8 个数据集中最稀疏的数据集,密度仅有 0.02%.

表 5　数据集特点

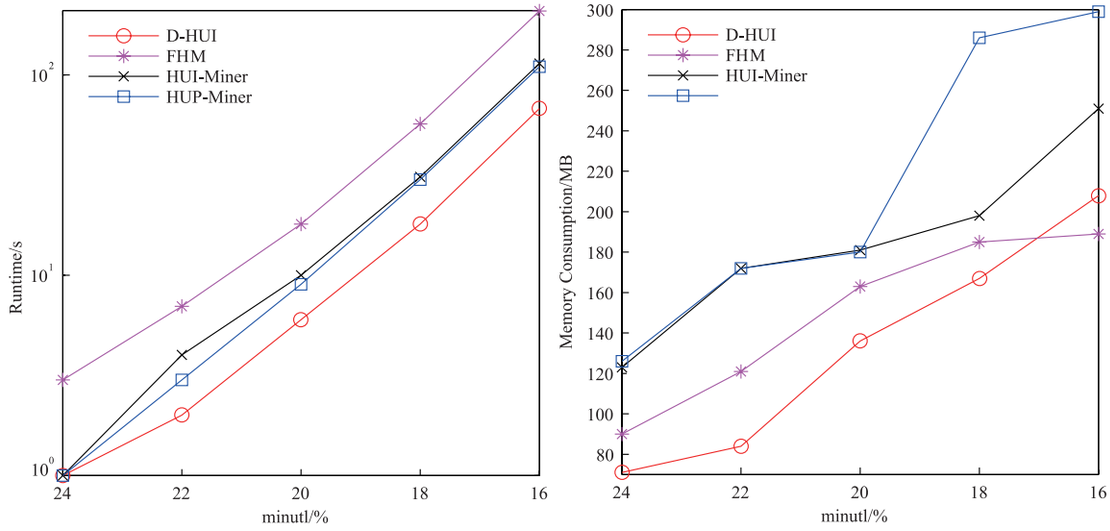
Dataset	Trans	Items	Avg Len	Max Len	Density
chess	3196	75	37	37	49.33%
connect	67557	129	43	43	33.33%
mushroom	8124	119	23	23	19.33%
accidents	340183	468	33.8	51	7.22%
T40I10D100K	100000	942	39.6	77	4.20%
pumsb	49046	2113	74	74	3.50%
T10I4D100K	100000	870	10.1	29	1.16%
kosarak	990002	41270	8.1	2498	0.02%

4.2 稠密数据集

首先在四个较稠密的数据集上进行实验,实验结果显示,本文提出的算法在运行时间上和内存消耗上都优于其它5个算法.

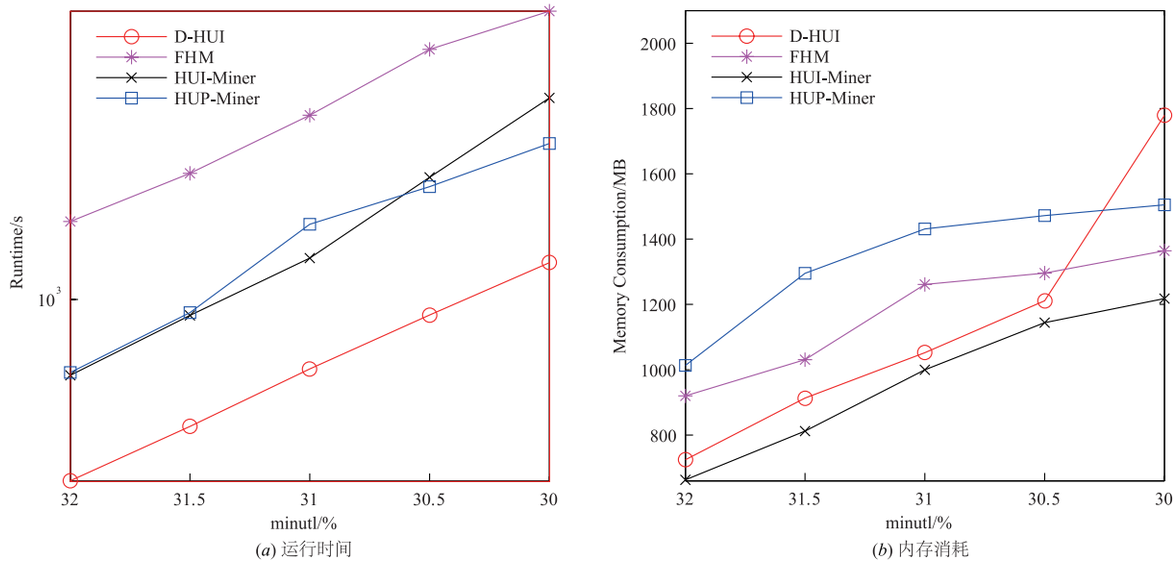
图3是chess数据集的实验结果,显示在不同的最小阈值产生情况下,6种算法的运行时间和内存消耗情况. 图3(*a*)显示的是运行时间,可以看到,D-HUI算法比HUI-Miner算法要快一倍左右,主要原因是,chess是一个非常稠密的数据集,D-HUI算法直接通过差集计算各项集的剩余效用,从而减少运行时间. 在chess数据集中,UP-Growth和UP-Growth$^+$等两个算法,在最小阈值为24%的情况,运行时间已经超过3.8小时,消耗内存超过2600M. 图3(*b*)显示的是内存消耗,从图中可以看到,多数情况下D-HUI比其它几个算法消耗的内存要少. 主要原因是,chess数据集的密度较大,达到了49.33%,FHM,HUI-Miner和HUP-Miner都需存储包括三个字段的ULs,这影响了系统的内存消耗.

图4是connect数据集的实验结果,connect数据集的密度达到了33.33%,包括67557个事务,项的平均长度为43. 在这个数据集上,D-HUI算法表现出了优越的性能,运行时间大大小于其它几个算法. 内存消耗不及HUI-Miner,但优于另外四个算法. 在这个算法中,UP-Growth和UP-Growth$^+$等两个算法,在最小阈值为32%的情况下,运行时间超过5个小时.



(*a*) 运行时间 (*b*) 内存消耗

图3 chess数据集上的运行时间和内存消耗情况



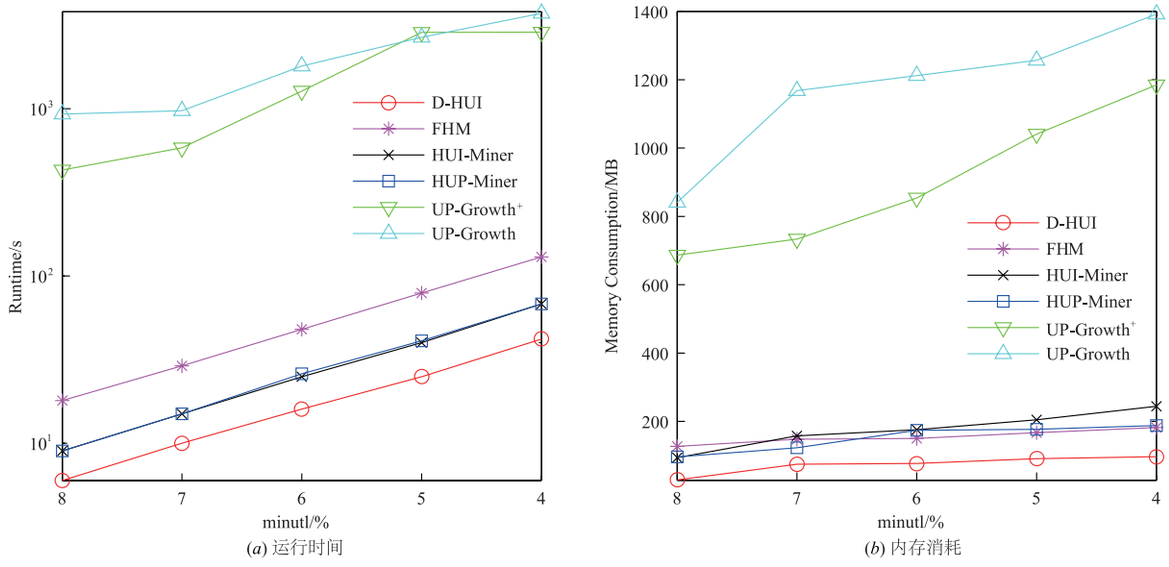
(*a*) 运行时间 (*b*) 内存消耗

图4 connect数据集上的运行时间和内存消耗情况

图 5 是 mushroom 数据集的实验结果,mushroom 的数据密度为 19.33%,有 8124 个事务,它的稠密程度不及前面两个数据集. 在这个数据集上,D-HUI 算法比 HUI-Miner 算法和 HUP-Miner 算法要快 50% 左右,比 UP-Growth 和 UP-Growth$^+$ 等两个算法要快 1 到 2 个数量级. 内存消耗也少于其它 5 个算法.

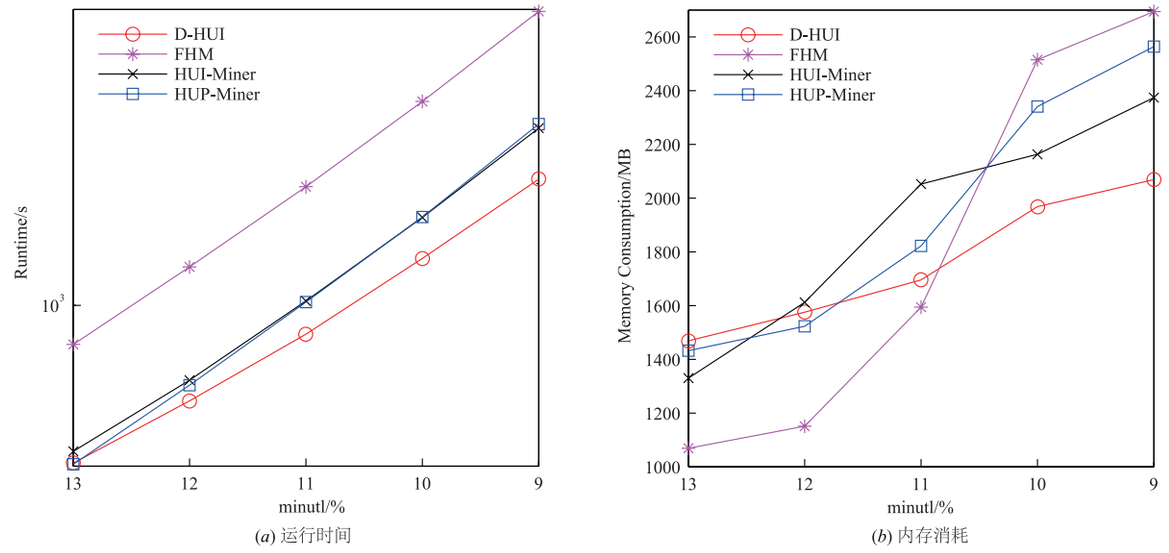
图 6 是 accidents 数据集的实验结果,这是一个非常大的数据集,事务数达到 340183 个,数据密度为 7.22%. 在最小阈值为 13% 的情况下,D-HUI 算法的运行时间为 407 秒,而 UP-Growth 和 UP-Growth$^+$ 等两个算法运行时间分别达到 21153 秒和 20799 秒,而 HUP-Miner 的表现较好,为 404 秒. 但随着最小阈值的提高,HUP-Miner 的表现就不及 D-HUI 算法.

实验结果显示,在四个密度的稠密的数据集中,提出的算法 D-HUI 比 FHM、HUI-Miner、HUP-Miner、UP-Growth$^+$ 和 UP-Growth 等五个最新的算法在性能上要好,并且数据集越是稠密,D-HUI 表现出来的性优越. 最主要的原因如下:第一,提出的算法不产生候选项集,性能要超过 UP-Growth 和 UP-Growth$^+$ 算法;第二,提出的 ULs 不存储冗余信息,仅存储事务和项的效用信息,占用空间少,并且无需计算额外的高估效用值,减少了运行时间;第三,提出两种剪枝策略,减少了 ULs 的数量,不仅减少的内存的消耗,也使得参与高阶项集列表计算的项集列表数量变得更少,从而降低了算法的运行时间和内存消耗.



(*a*) 运行时间　　(*b*) 内存消耗

图5　mushroom数据集上的运行时间和内存消耗情况



(*a*) 运行时间　　(*b*) 内存消耗

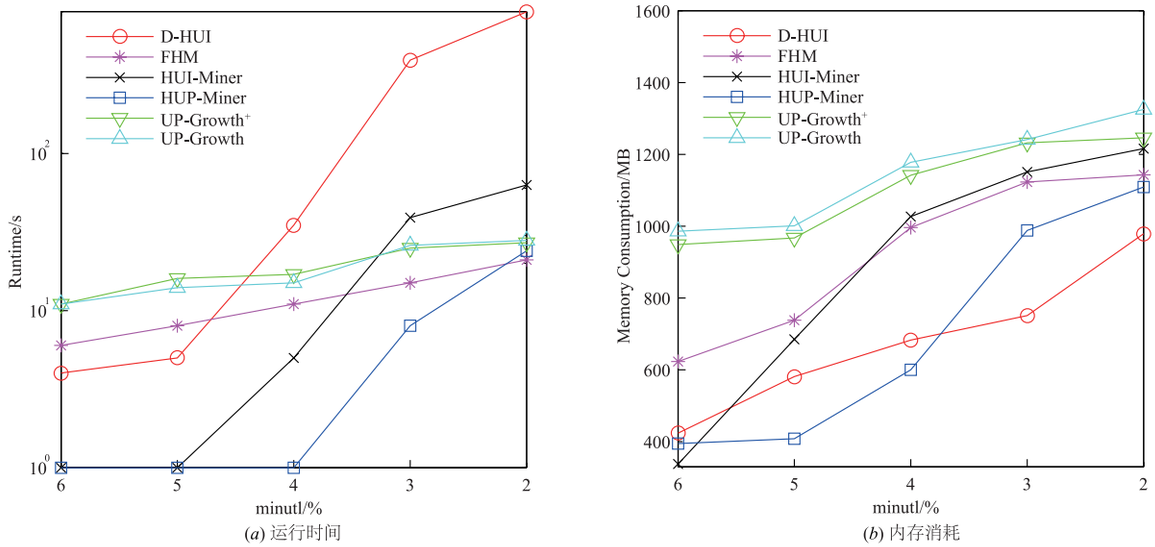
图6　accidents数据集上的运行时间和内存消耗情况

4.3 稀疏数据集

接下来在四个较为稀疏的数据集上进行实验,实验结果显示,本文提出的算法在运行时间上和内存消耗上在部分数据集是优于其它算法,但在非常稀疏的数据集如 kosarak,表现不及其它 5 个算法.

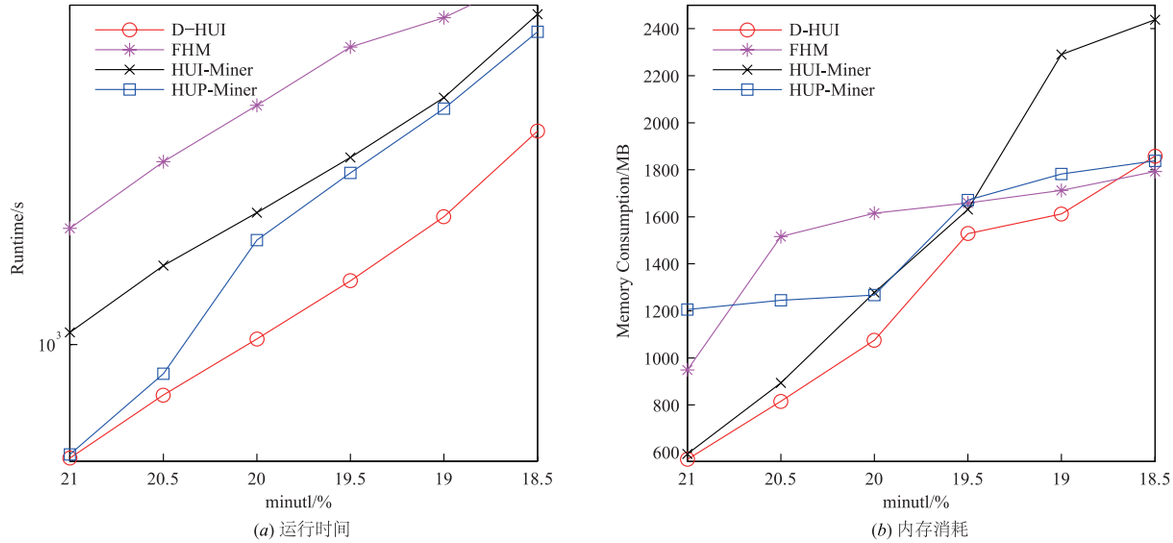
图7 是 T40I10D100K 数据集的实验结果,它事务数达到 100000 个,数据密度为 4.20%. 在最小阈值为 6% 和 5% 的情况下,D-HUI 算法在运行时间上优于 FHM、UP-Growth 和 UP-Growth$^+$等三个算法,但劣于 HUI-Miner 和 HUP-Miner.

图8 为 pumsb 数据集上的实验结果,此数据集的事务数为 49046,数据密度为 3.50%,它的稠密程度不及 T40I10D100K,但 D-HUI 算法在此数据上的性能却表现的更好,超过了其它 5 个算法. 主要原因是 pumsb 虽然稠密程度不及 T40I10D100K,但它的平均项的长度为 74,高于 T40I10D100K 的 39.6,数据更为集中,这体现出了 D-HUI 算法的优势.



(*a*) 运行时间 (*b*) 内存消耗

图7 T40I10D100K数据集上的运行时间和内存消耗情况



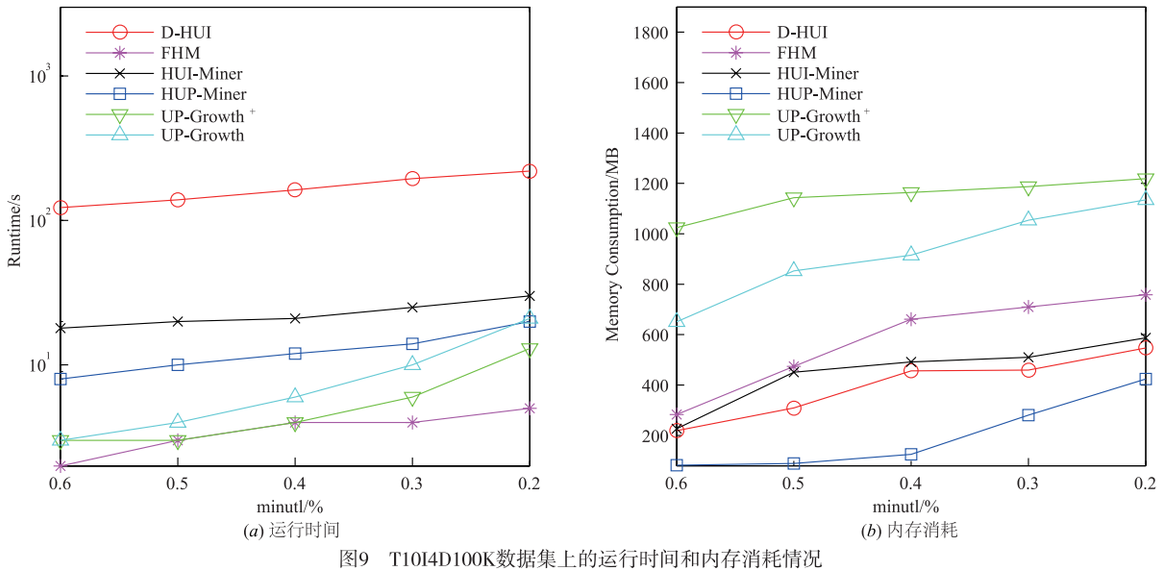
(*a*) 运行时间 (*b*) 内存消耗

图8 pumsb数据集上的运行时间和内存消耗情况

图9 是和图 10 分别是 T10I4D100K 和 kosarak 等两个数据集上的实验结果,这两个数据集非常稀疏,数据密度仅为 1.16% 和 0.02%. 在这两个数据集上,D-HUI 算法在运行时间上的表现都差于其它算法. 在稀疏数据集上,D-HUI 算法无优势.

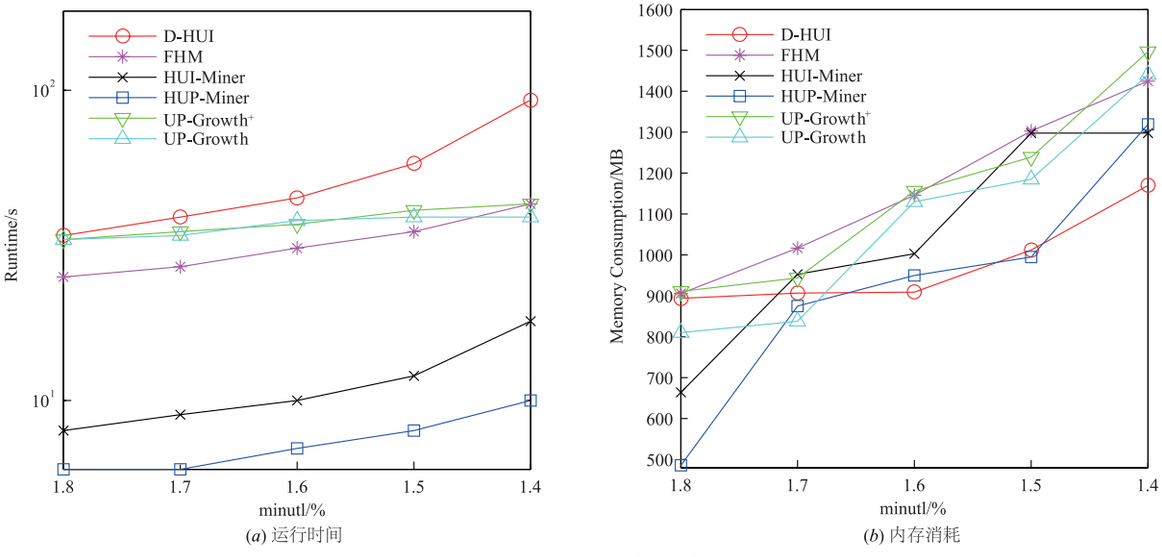
4.4 项的顺序

一个细节是项的执行顺序,不同的执行顺序对挖掘性能会带来不同的结果. 对于使用哪种顺序,文献[10]进行了深入的讨论. 基于树构建的算法,如 IUHP、UP-Growth$^+$等,通常是以项的 TWU 值降序进行排列的,文



(*a*) 运行时间　　(*b*) 内存消耗

图9　T10I4D100K数据集上的运行时间和内存消耗情况



(*a*) 运行时间　　(*b*) 内存消耗

图10　kosarak数据集上的运行时间和内存消耗情况

献[11]对此进行过研究. 基于垂直数据结构的 HUI-Miner 算法使用是以项集 TWU 值升序排列. 这种顺序能够导致减少搜索空间的范围,同时加快算法速度[10]. 本文提出的算法 D-HUI 以项的以 TWU 值升序排列.

5　结论

在本文中,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. 项集列表仅存储事务和项的效用信息,而无需存储额外的高估效用信息,减少了内存的消耗也提高了算法的运行速度. 并提出两种用于对项集列表进行剪枝的策略,应用这两种剪枝策略能减少项集列表的数量,从而减少参与计算的项集列表数量,不仅能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 D-HUI,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,提高了算法的性能.

参考文献

[1] J Han, J Pei, Y Yin, Mining frequent patterns without candidate generation[A]. Proc of the 2000 ACM SIGMOD International Conference on Management of Data[C]. Dallas, TX: ACM, 2000. 1 – 12.

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

[3] Ke-Chung Lin, I-En Liao, Tsui-Ping Chang, et al. A frequent itemset mining algorithm based on the Principle of Inclusion—Exclusion and transaction mapping[J]. Information Sciences, 2014, 276(C): 278 – 289.

[4] Yun Sing Koh, Sri Devi Ravana. Unsupervised rare pattern mining: A survey[J]. ACM Transactions on Knowledge Discovery from Data, 2016, 10(4): 45: 1 – 29.

[5] 刘旭, 毛国君, 孙岳, 等. 数据流中频繁闭项集的近似挖掘算法[J]. 电子学报, 2007, 35(5): 900 – 905.
LIU Xu, MAO Guo-jun, SUN Yue, et al. An algorithm to approximately mine frecuent closed Itemsets from data streams[J]. Acta Electronica Sinica, 2007, 35(5): 900 – 905. (in Chinese)

[6] 周秀梅, 黄名选. 基于项权值变化的完全加权正负关联规则挖掘[J]. 电子学报, 2015, 43(8): 1545 – 1554.
ZHOU Xiu-mei, HUANG Ming-xuan. All-weighted positive and negative association ruies mining based on dynamic item weight[J]. Acta Electronica Sinica, 2015, 43(8): 1545 – 1554. (in Chinese)

[7] Y Liu, W Liao, A Choudhary. A two-phase algorithm for fast discovery of high utility itemsets[A]. Proc of the 9th Pacific-Asia Conference on Knowledge Discovery and Data Mining[C]. Hanoi, Vietnam: Springer, 2005. 689 – 695.

[8] H Yao, H J Hamilton. Mining itemset utilities from transaction databases[J]. Data and Knowledge Engineering, 2006, 59(3): 603 – 626.

[9] Y C Li, J S Yeh, C C Chang. Isolated items discarding strategy for discovering high utility itemsets[J]. Data and Knowledge Engineering, 2008, 64(1): 198 – 217.

[10] C F Ahmed, S K Tanbeer, B S Jeong, 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.

[11] V S Tseng, C W Wu, B E Shie, et al. Up-growth: an efficient algorithm for high utility itemset mining[A]. Proc of the 10th Conference Knowledge Discovery and Data Mining[C]. Washington, DC: ACM, 2010. 253 – 262.

[12] V S Tseng, B E Shie, C W Wu, 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.

[13] M C. Liu, J F Qu. Mining high utility itemsets without candidate generation[A]. Proc of the 12th International Conference on Information and Knowledge Management[C]. Maui, HI, USA: ACM, 2012. 55 – 64.

[14] Fournier-Viger P, C W Wu, Zida S, et al. FHM: Faster high-utility itemset mining using estimated utility co-occurrence pruning[A]. Proc of the 21th International Symposium on Methodologies for Intelligent Systems[C]. Roskilde, Denmark: Springer, 2014. 83 – 92.

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

[16] C W Wu, P Fournier-Viger, P S Yu, et al. Efficient mining of a concise and lossless representation of high utility itemsets[A]. Proc of the 11th IEEE International Conference on Data Mining[C]. Vancouver, Canada: IEEE, 2011. 824 – 833.

[17] V S Tseng, C W Wu, Philippe Fournier-Viger 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.

[18] J Sahoo, A K Das, A Goswami. An efficient fast algorithm for discovering closed + high utility itemsets[J]. Applied Intelligence, 2016, 45(1): 44 – 74.

[19] C W Wu, Philippe Fournier-Viger, J Y Gu, et al. Mining closed + high utility itemsets without candidate generation[A]. The 2015 Conference on Technologies and Applications of Artificial Intelligence[C]. Taiwan: TAAI, 2015. 1 – 8.

[20] Guo S M, Gao H. HUITWU: An efficient algorithm for high-utility itemset mining in transaction databases[J]. Journal of Computer Science and Technology, 2016, 31(4): 776 – 786.

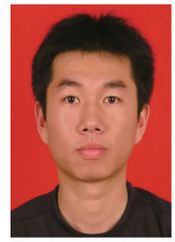
[21] Tseng V S, Wu C W, Lin J H, et al. UP-Miner: A utility pattern mining toolbox[A]. IEEE International Conference on Data Mining Workshop[C]. Atlantic, USA: IEEE, 2015. 1656 – 1659.

[22] Fournier-Viger P, Lin C W, Gomariz A, et al. The SPMF open-source data mining library version 2[A]. Machine Learning and Knowledge Discovery in Databases[C]. Berlin, Germany: Springer International Publishing, 2016. 36 – 40.

[23] B Goethals, MJ Zaki. Frequent Itemset mining implementations repository[DB/OL]. http://fimi.ua.ac.be/, 2016 – 06 – 30.

[24] J Pisharath, Y Liu, J Parhi, et al. NU-MineBench Version 3.0.1[DB/OL]. http://cucis.ece.northwestern.edu/projects/DMS/MineBench.html, 2016 – 06 – 30.

作者简介



黄 坤 男, 1979 年 10 月生于湖北孝感, 中国舰船研究设计中心高级工程师, 研究方向电子信息系统.
E-mail: hkcfan@163.com

吴玉佳 男, 1986 年 11 月生于湖北广水. 武汉大学计算机学院博士生. 研究方向为数据挖掘与机器学习.
E-mail: wuyujia@whu.edu.cn