定向挖掘 Top-k 高效用项集

珊 黄[a]，文生 甘[a, b, *]，金宝 苗[a]，旭明韩[c] 和 菲利普富尼耶-维热[d]

[a]暨南大学网络安全学院，广州 *510632*，中国[b]琶洲实验室，广州 *510330*，中国[c]暨南大学信息科学与技术学院，广州 *510632*，中国[d]深圳大学计算机科学与软件工程学院，深圳 *518060*，中国

arXiv:2303.14510v1 [cs.DB] 25 Mar 2023

文章信息

关键词：数据科学 数据挖掘 效用挖掘 目标项集 定向挖掘 Top-k

摘要

在数据中发现高重要性模式是一项新兴的数据挖掘任务，称为高效用项集挖掘（HUIM）。给定最小效用阈值后，高效用项集挖掘算法会提取所有效用值不低于该阈值的高效用项集（HUIs）。这有助于揭示大量有用信息，但并未充分考虑用户的具体需求。尤其是，用户往往希望关注包含某些特定项的模式，而不是找出所有模式。为解决这一难题，定向挖掘应运而生，其重点是用户偏好，但目前相关研究仍处于初步阶段。例如，有人提出了目标高效用项集查询算法（TargetUM），该算法利用字典序树查询包含目标模式的项集。然而，如果用户不熟悉待处理的数据库，就很难选定最小效用阈值。为此，本文将定向挖掘top-k 个高效用项集这一任务形式化，并提出一种基于TargetUM算法的高效算法TMKU，用于发现top-k 个目标高效用项集（top-k 个THUIs）。同时，本文采用了多种剪枝策略，以减少内存消耗和执行时间。大量实验表明，所提出的TMKU算法在真实数据集和合成数据集上均表现良好。

1. 引言

随着数字系统的广泛应用，大数据时代已经到来，海量数据不断生成。因此，如何从复杂数据中挖掘有用信息，已成为一个重要问题。数据挖掘软件能够帮助用户从数百万乃至数十亿条记录中发现关联关系，使决策者能够更快地做出更明智的决策 [10, 15, 18]。这项技术正在影响日常生活的诸多领域（例如市场分析[21], 生物医学 [31], 和网络安全管理 [4]）。一个典型的应用是购物篮分析 [25]。购物篮分析主要分析顾客的消费记录，以揭示产品和服务之间的重要关联关系，从而帮助商家重新安排货架上的商品，或设计用户感兴趣的促销组合。例如，如果超市的销售记录显示，顾客经常同时购买面包和牛奶，那么将二者放在同一货架上进行捆绑销售，就能增加销量和收入。

以往研究提出了许多频繁项集挖掘算法（FIM） [1, 25], ，其中最著名的是 Apriori [3]和 FP-growth [20]。传统的频繁模式挖掘算法仅使用支持度这一种度量指标，来寻找用户感兴趣的模式。应用频繁项集模型后，往往会发现某些项集的销售频率很高，但利润却很低。然而，有些销量不大的奢侈品利润丰厚，值得关注。为解决这一问题，研究者提出了效用导向模式挖掘（UPM） [19, 22, 34, 41]算法。效用的概念同时涵盖内部效用和外部效用。内部效用通常指某种商品出现的次数，而每件商品产生的单位利润则称为外部效用。近几十年来，效用导向模式挖掘受到了广泛研究 [16, 17], ，因为考虑每个项的盈利能力和相对重要性是有益的。一些有代表性的效用导向模式挖掘算法包括：基于效用列表结构的 HUI-Miner [28], ； 基于 UP-tree结构的 UP-Growth [39] ； 以及基于数据投影的 EFIM [45] 。

尽管高效用项集挖掘算法能够在特定的实际应用中收集一些有用信息，但并不适用于面向目标的挖掘任务。为此，提出了定向高效用项集查询（TargetUM）

*通讯作者：shuang9901@gmail.com（S. Huang）；wsgan001@gmail.com（W. Gan）；osjbmiao@gmail.com（J. Miao）；hanxuming@jnu.edu.cn（X. Han）；philfv@qq.com（P. Fournier-Viger）

算法 [30]，通过考虑用户需求来解决这一问题。以消费者购买行为为例：人们更关注购物清单上的商品，而较少关注其他商品。因此，零售商可以根据用户的不同需求提供不同的产品推荐，并为库存商品提供折扣。因此，面向目标的高效用项集挖掘（THUIM）能够过滤掉数据中的许多无关模式，从而精准地聚焦于包含感兴趣项的模式。然而，在这些效用驱动的数据挖掘算法中选择最小效用阈值既复杂又费力。尤其是当用户不熟悉某个数据库时，便难以了解项集的效用值以及事务的内容。效用阈值设得过高时，只能捕获少量目标高效用模式；而阈值设得过低时，则会产生大量对用户毫无价值的无关信息。因此，必须根据每个数据库中的项数和效用分布设置不同的效用阈值。一般而言，决策者总要反复测试并调整效用阈值，直到找到能够产生适量目标高效用项集的阈值，这一过程既耗时又不便。

因此，本文提出了定向挖掘 Top-k 高效用项集的问题，用户可以直接设定要挖掘的项集数量。本文设计了一种名为 TMKU（Top-k 高效用项集定向挖掘）的算法，以高效地完成这一任务。实验结果证明，TMKU 算法有效且高效。本研究的主要贡献概述如下：

- 本文探讨了两个重要方面：效用挖掘和面向目标的模式发现。据我们所知，TMKU 算法是首个将 Top-k 挖掘概念融入基于目标的效用挖掘的算法。
- 本文使用字典树结构查询目标项集，并采用一种名为 TopKMap 的新型数据结构，高效存储和检索 Top-k 目标项集，从而有助于减少查询时间和搜索空间。
- TMKU 算法利用了若干上界，并采用三种剪枝策略来缩减搜索空间。此外，该算法还采用两种目标效用提升策略，以快速提高阈值，确保发现所有 Top-k 目标高效用项集。
- 我们在真实数据集和合成数据集上进行了大量实验。实验结果表明，TKUM 高效且有效。

本文其余部分由六个章节组成。第2节简要回顾相关工作。第3 节介绍 top-k THUIM 问题及相关定义。第4 节介绍所提出的 TMKU 算法，以及所采用的主要策略和技术。第5节展示并评估实验结果。最后，第6节总结本文，并展望有前景的研究方向。

2. 相关工作

本节介绍与本文相关的研究，可分为三个部分：（1）高效用模式挖掘；（2）top-k 效用项集挖掘；（3）目标模式挖掘。

2.1. 高效用模式挖掘

频繁项集挖掘（FIM） [1, 3, 20] 数十年来一直受到广泛研究。然而，仅依赖频率无法为用户带来足够的收益。还应考虑数量和利润等因素。因此，Chen等人 [5] 提出了一项名为高效用项集挖掘（HUIM）的新任务。此后，效用挖掘研究迅速发展 [17, 27, 35, 41]。为便于讨论，高效用项集挖掘算法分为以下三类：

基于Apriori的算法：自 Agrawal等人 [2] 于1994年提出 Apriori性质以来，已有大量基于 Apriori 的算法问世。例如，Liu等人 [29] 提出了著名的 Two-Phase算法，以解决效用不同于频率、既不具备单调性也不具备反单调性所带来的难题。该算法在第一阶段使用一种称为事务加权效用（Transaction Weighted Utilization，TWU）的效用高估值来查找候选项集。随后，在第二阶段再次搜索数据库，以确定每个候选项集的精确效用值。IIDS算法 [26] 是 Two-Phase算法的改进版本，它通过剔除孤立项来缩小搜索

空间。然而，Apriori 类算法的共同缺点是会生成大量候选模式，从而导致巨大的计算成本和内存消耗。

基于树的算法：Tseng 等人 [39] 设计了 UP-tree 结构（一种效用模式树），并受 FP-Growth 启发提出了 UP-Growth 算法。此后，又出现了其他版本的基于树的算法 [34, 37]。通常，利用 UP 树可以避免许多无意义的数据库扫描。然而，处理大规模数据库时，这种结构会变得愈发复杂，并占用大量内存。

其他基于结构的算法：HUI-Miner [28]使用一种名为效用列表的新型数据结构，避免了生成大量候选项集的难题。此外，FHM 算法 [12] 通过使用更紧的上界降低连接操作的成本，因此性能优于 HUI-Miner。然而，这些算法对列表执行连接操作会耗费时间和内存。因此，Zida 等人 [45]提出了 EFIM 算法，并采用高效用数据库投影（HDP）和高效用事务合并（HTM）技术，以降低数据库遍历的高昂成本。基于效用列表的 CoUPM 算法用于基于相关效用的模式挖掘 [14]。总之，这些算法结合了各种策略，以尽可能高效地发现高效用项集。

2.2. Top-k 效用项集挖掘

尽管上述算法能够有效地找出所需的项集，但挖掘效率与最小效用阈值的选择密切相关。然而，确定合适的阈值并不容易。因此，许多 Top-k 模式挖掘算法被设计出来，旨在直接发现 Top-k 高效用项集，而不要求用户指定效用阈值。Top-k 高效用项集挖掘算法主要分为两类：一类是两阶段算法，另一类是单阶段算法。

两阶段算法：发现 Top-k 高效用项集的任务由 Wu 等人 [40] 通过 TKU 算法提出，该算法在速度方面优于高效用项集挖掘算法。TKU 算法是一种两阶段算法。在第一阶段，构建一棵 UP 树，并生成有希望的 Top-k 高效用项集。随后，在第二阶段，从中选出所需的 Top-k高效用项集。TKU 在搜索过程中采用多种策略过滤无望的候选项集 [38]，从而提高效率。随后，REPT [32]问世，它通过优化记录并预先计算各项的效用，以有效剪枝搜索空间并提高最小效用阈值。REPT 使用树结构和预评估矩阵来存储效用信息。然而，这两阶段算法仍会生成大量候选项集，导致运行时间过长且内存占用过高。

单阶段算法：对于 top-k 高效用项集挖掘，单阶段算法 TKO [38] 应运而生，用以解决两阶段算法的不足。TKO 利用了 HUI-Miner 的效用列表结构，实验结果表明，其性能优于 TKU 和 REPT 算法 [38]。类似地，另一种名为 KHMC [9]的单阶段算法也利用效用列表结构来挖掘 top-k 高效用项集。在 KHMC 中，采用了估计效用共现剪枝（EUCP）技术，该技术基于预先计算二项集的事务加权效用（TWU）。此外，该算法还引入了另一种名为提前终止的剪枝策略，以避免为无望项集完整构建效用列表。三种阈值提升策略能够显著缩小搜索空间，并提高算法效率。THUI 算法 [23] 通过引入叶节点项集效用（LIU）的概念而表现更佳。LIU 是一种三角矩阵，只需占用少量内存即可存储效用信息。此外，LIU-E 和 LIU-LB 阈值提升策略也加快了该算法的挖掘速度。THUI 的性能远超 TKO 和 KHMC，尤其是在稠密数据集或大型数据集上。

此外，还有各种其他的 Top-k模式挖掘问题及其变体，例如挖掘 Top-k 序列模式 [43], 在数据流中挖掘 Top-k 高效用项集 [7], 发现 Top-k 高效用序列模式 [43], 以及挖掘具有负效用值的 Top-k高效用项集 [36]。

2.3. 目标模式挖掘

上述算法旨在找出所有符合单一预设标准的项集。面向目标的查询算法通过过滤掉不必要的信息，为这一问题提供了另一种解决方案。用户无需搜索大量但大多无关紧要的项，而是可以输入任意目标，进而发现包含所需项的模式。早期研究中已经提出了若干基于频率的面向目标查询算法。这些交互式方法能够返回包含目标的结果。Kubat 等人 [24] 率先着手解决事务数据库中的目标查询处理问题。他们通过创建可逐步更新的项集树，实现了用于关联挖掘的目标查询处理算法。Fournier-Viger 等人 [11] 开发了内存高效项集树（MEIT），以进一步降低内存需求。该树经过优化，可在插入新事务时进行增量修改，并采用

节点压缩方法。针对大数据的多目标挖掘，Shabtay 等人 [33]提出了基于 FP-Growth 的引导式 FP-growth（GFP-growth）算法。特别是，许多实验表明，该算法在不平衡数据上表现出色。面向目标的挖掘也已用于研究和发现序列模式。Chueh等人 [8]提出了用于序列模式的目标定向挖掘算法，通过反转原始序列并将逆序序列与相关项集进行比较，加快了对目标项集的搜索。此外，该算法还应用聚类分析，自动为时间间隔序列模式挖掘任务设置时间划分值。Chand 等人 [6], 提出了一种新颖的面向目标的序列模式挖掘方法，该方法采用 RFM（最近一次消费时间、频率和消费金额）约束。因此，数据库投影次数减少，空间复杂度也随之降低。为去除高效用序列模式挖掘 [44, 13], 中的一些无用或无关模式，TUSQ 算法 [42] 首次将效用概念引入目标序列查询。与以往算法关注频率不同，该算法关注的是效用。近期提出了 TargetUM 算法 [30]，以弥补这一空白，并在高效用项集挖掘（HUIM）中执行面向目标的挖掘。

一般而言，TargetUM 算法为带有目标查询的高效用挖掘提供了一种集成方法，也是本研究的基础。然而，目前尚无研究将 Top-k 高效用方法与目标模式查询相结合。本文提出了目标定向效用挖掘问题，并引入 Top-k模式的概念，以避免生成大量高效用项集，并准确、快速地处理目标查询。

3. 预备知识与问题陈述

本节介绍重新设计的数据结构，并简要说明与目标定向 Top-k 高效用项集挖掘问题相关的关键定义。$\mathcal{D}$ 表示由多个事务 $\{t_1,t_2, t_3, \ldots, t_n\}$构成的事务数据库。每个事务都包含一个或多个来自集合 $\{i_1, i_2, i_3, \ldots, i_m\}$ 的项。事务 t_{id} 的事务标识符为 id。若一个项集包含 l 个项，则称其为 l-项集。事务中的每个项都可以标注一个正整数值，表示购买数量（也称为内部效用值或计数）。例如，$t_1 = \{(b,4),(d,3),(e,1)\}$表示项 b有四个单位、项 d有三个单位、项 e有一个单位。项集 $\{b,d,e\}$ 称为 3-项集。

表 1：示例数据库

T_{id}	事务	计数
t_1	$\{b,d,e\}$	{4,3,1}
t_2	$\{a,b,e\}$	{1,3,2}
t_3	$\{c,d\}$	{3,1}
t_4	$\{a,b,c,f,g\}$	{3,1,4,2,1}
t_5	$\{a,b\}$	{1,2}
t_6	$\{b,c,d,f\}$	{1,3,4,2}
t_7	$\{a,e,g\}$	{4,1,2}

表2：关于最小效用阈值 =$25 的高效用项集

项集	效用	项集	效用
$\{f,c,d,b\}$	$27	$\{c,d,b\}$	$25
$\{e,d,b\}$	$26	$\{d\}$	$32
$\{e,b\}$	$27	$\{d,b\}$	$43
$\{c,d\}$	$32	$\{b\}$	$33

定义1.（项集效用）$\mathcal{D}$中有两种效用值。一方面，在每个事务t_{id}中，项 $i \in t_{id}$ 的内部效用记为 $iu(i,t_{id})$。另一方面，项 i 的外部效用是一个正数，记为 $eu(i)$，用于表示其相对权重或重要性。因此，在事务t_{id}中，$u(i,t_{id})$表示项 i的效用，它是 $eu(i)$ 与 $iu(i,t_{id})$的乘积，即 $u(i,t_{id}) = eu(i) \times iu(i,t_{id})$)。对于项集 X 和包含 X 的事务 t_{id}（$X \subseteq t_{id}$），X 在 t_{id} 中的效用计算为 $u(X,t_{id}) = \sum_{i_\ell \in X \wedge X \subseteq t_{项}} u(i_\ell, t_{id})$。符号 $u(X)$ 表示 X 在 $\mathcal{D}$ 中的效用，其计算方式为所有包含 X的事务中X 的效用之和，即 $u(X) = \sum_{X \subseteq t_{id} \in \mathcal{D}} u(x,t_{id})$。

例如，示例数据库中所有项的外部效用为 {(a: 1), (b: 3), (c: 2), (d: 4), (e: 2), (f: 1), (g: 2)}。在表 1 中，$iu(b,t_2) = \$3$, $eu(b) = \$3$, 因此 $u(b,t_2) = iu(b,t_2) \times eu(b) = 3 \times \$3 = \$9$。$u(\{ae\},t_2) = u(a,t_2) + u(e,t_2) = 1 \times \$1 + 2 \times \$2 = \$5$, $u(\{abc\},t_4) = u(a,t_4) + u(b,t_4) + u(c,t_4) = 3 \times \$1 + 1 \times \$3 + 4 \times \$2 = \$14$。由于$d$ 出现在 t_1、t_3 和 t_6 中, $u(d) = u(d,t_1) + u(d,t_3) + u(a,t_6) = 3 \times \$4 + 1 \times \$4 + 4 \times \$4 = \$32$，并且 $u(\{ab\}) = u(\{ab\},t_2) + u(\{ab\},t_4) + u(\{ab\},t_5) = u(a,t_2) + u(b,t_2) + u(a,t_4) + u(b,t_4) + u(a,t_5) + u(6,t_5) = 1 \times \$1 + 3 \times \$3 + 3 \times \$1 + 1 \times \$3 + 1 \times \$1 + 2 \times \$3 = \$23$。

定义 2.（项集的事务加权效用）符号 $tu(t_{id})$ 表示事务t_{id} 的效用，其定义为 t_{id}中所有项的效用值之和。换言之， $tu(t_{id}) = \sum_{i_\ell \in t_{id}} u(i_\ell, t_{id})$。项集 X 在 $\mathcal{D}$ 中的事务加权效用记作 $TWU(X)$，它是所有包含 X 的 $\mathcal{D}$中 X 的效用值之和，即 $TWU(X) = \sum_{X \subseteq t_{id} \wedge t_{id} \in \mathcal{D}} u(X, t_{id}) = \sum_{X \subseteq t_{id} \wedge t_{id} \in \mathcal{D}} \sum_{i_\ell \in X} u(i_\ell, t_{id})$。

例如， $tu(t_1) = \$26$，且效用最高的事务为 t_6，其效用为 $tu(t_6)$。此外，计算得到的各事务效用为 {(t_1: \$26), (t_2: \$14), (t_3: \$10), (t_4: \$18), (t_5: \$7), (t_6: \$27), (t_7: \$10)}。

性质 1. *TWU* 符合向下闭合性。这意味着，如果某个项集的 *TWU(X)* 小于或等于效用阈值 xi，则任何包含 X 的其他项集都是低效用项集。

通过初次扫描数据库 $\mathcal{D}$，确定所有项的 TWU 值。性质 1 指出，当某个项 X 的 TWU(X) $< \xi$ 时， X 将被丢弃，以减少需要评估的项集数量。为实现快速挖掘，本文采用 TWU升序排序，并按照相同顺序处理事务数据库中的每个事务。所有 1-项集的事务加权效用按如下顺序排列： g (= \$28) $< f$ (= \$45)< a (= \$49) $< e$ (= \$50) $< c$ (= \$55) $< d$ (= \$63) $< b$(= \$92)。

定义 3.（剩余效用）如果存在项集 X，且 $X \subseteq t_{id}$，则 X 在事务 t_{id} 中的剩余项记为 t_{id}/X。项集 [28] 在事务 t_{id} 中的剩余效用，是 t_{id}/X 的效用值之和，记为 $ru(X, t_{id})$，其中 $ru(X, t_{id}) = \sum_{i_\ell \in t_{id}/X} u(i_\ell, t_{id})$。类似地， X 在 $\mathcal{D}$ 中的剩余效用记为 $ru(X)$，其中 $ru(X) = \sum_{X \subseteq t_{id} \in \mathcal{D}} ru(X, t_{id})$。

在表 1中， $\{bd\}$ 在事务 t_1 中的剩余效用定义为 $ru(\{bd\}, t_1) = u(e, t_1) = \$2.$

定义 4.（效用列表）在 HUI-Miner [28], 中，效用列表是为效用挖掘任务引入的。效用列表包含三个基本元素（t_{id}、*iutil*和 *rutil*）。 t_{id} 是包含 X的事务的标识符。元素 *iutil* 表示 X_i 在 t_{id}中的效用。*rutil* 表示 X_i 在 t_{id}中的剩余效用。

例如， $\{c\}$ 的效用列表为 {t_3, \$6, \$4}、{t_4, \$8, \$4} 和 {t_6, \$6, \$18}。 $\{f\}$ 的效用列表为 {t_4, \$2, \$2} 和 {t_6, \$2, \$0}。因此， $\{c, f\}$ 的效用列表为 {t_4, \$10, \$2} 和 {t_6, \$8, \$0}。

定义5。（目标高效用项集）设*targetUtil* 为用户指定的目标效用阈值，并记为 ξ。如果 X 的效用值不小于 ξ，则称其为该用户的高效用项集。用户设定的项集称为T'。之所以称其为目标项集，是因为它是用户希望关注的对象。如果 $u(X) > \xi, T' \subseteq X$，则进一步认定 X 是目标高效用项集（THUIs）之一 [30]。

例如，用户想查找与目标项集 $T' = \{b, d\}$相关的高效用项集。从表 2中，我们可以搜索满足 T'的高效用项集。因此，目标高效用项集的集合为 {$\{f, c, d, b\}, \{e, d, b\}, \{c, d, b\}, \{d, b\}$}。

定义6。（Top-k 目标高效用项集）受其他 top-k 模式挖掘任务的启发，我们将 top-k 目标高效用项集（top-k THUIs）定义为具有最高效用值的 k 目标项集。如果数据库中包含目标模式的项集相对少见，且用户设定的 k 值相对较大，则结果项集将少于 k 项。当效用相同的模式超过 k 个时，结果也只会按照挖掘顺序显示 k个项集。

例如，如果用户只想找出满足先前条件的前 3 个项集，那么我们将 k 设为 3。显然，前 3 个目标高效用项集为 $\{f, c, d, b\} = \$27$、 $\{e, d, b\} = \$26$ 和 $\{d, b\} = \$43$。

问题陈述：大多数效用挖掘算法旨在识别效用大于或等于固定效用阈值的项集。然而，实际上，用户并不想找出所有高效用项集。例如，用户可能只想要包含特定项或项集的前-k 个高效用项集。本文将所提出的模式挖掘问题称为前-k 个高效用项集的定向挖掘。

与其他高效用项集挖掘问题不同，本文面临的挑战是如何更好地处理目标模式与 top-k 模式相结合时的搜索空间。换言之，需要采用合理、高效的数据结构存储项集信息，并运用正确、快速的剪枝策略来提高挖掘效率。至此，相关定义和概念已介绍完毕，下一节将详细介绍所提出的 TMKU 算法。

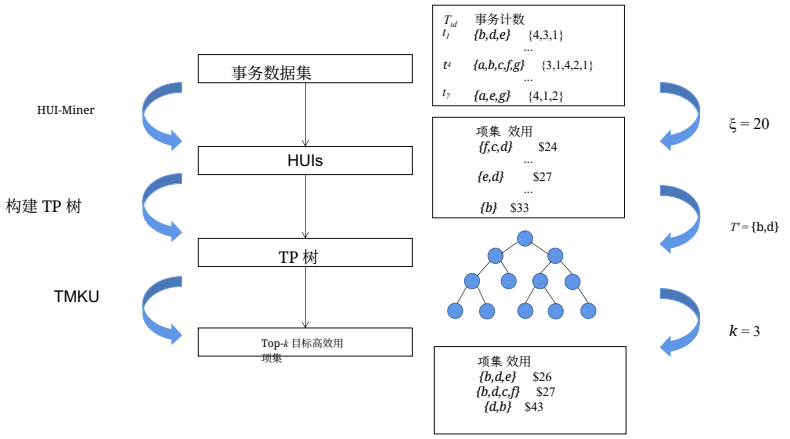


图 1： TMKU 的框架

4. TMKU 算法

定向挖掘 top-k 高效用项集的算法，简称 TMKU，将在本节中详细介绍。图 1展示了所提出的 TMKU 算法的框架图，说明了整个算法的基本步骤。TMKU 与 TargetUM 算法类似，采用字典树结构作为存储方式，然后利用筛选思想动态调整 η （效用阈值），以选出所需的 top-k 目标高效用项集。TMKU 算法通过调整目标模式和 k ，能够在短时间内查询不同的目标以及结果项集的数量。介绍完这些概念后，我们将详细说明搜索空间剪枝的执行方式以及 TMKU 算法的具体流程。

4.1. 构建 TP 树与查询机制

进行数据分析时，必须扫描数据库以处理信息，并获得符合用户需求的模式。使用目标模式树作为数据结构时，算法以节点表示项集，并可通过插入新节点来增量更新模式树。

MEIT [11] 是关于这一问题的早期研究。它采用紧凑的节点压缩机制来存储事务信息，并回答有关目标项集的查询，但其设计目的并非效用挖掘。后来，研究者提出了用于高效用项集挖掘的 TargetUM 算法。该算法使用字典树存储整个数据库中的高效用项集，并已证明可以减少内存消耗。因此，本文提出的算法也采用字典树结构，通过利用字符串的公共前缀提高效率，从而减少查询的开销时间。随后，在找出所有包含目标项或项集的高效用项集后，构建映射结构（TopKMap）来存储已发现的目标项集。该算法将第 k 个项集的效用值设为初始阈值，并根据最小阈值不断更新 TopKMap，直到发现前 k 个目标高效用项集。

定义 7。 （目标模式树）。该树中的每个节点由七个元素描述。节点 n （名称）会记录其父节点（父节点）。此外，与效用相关的三个元素分别是事务加权利用率（twu）、记录当前节点所表示项集效用的 $sumIu$ ，以及记录当前节点所表示项集剩余效用的 $sumRu$ 。当 n 是某个高效用项集的最后一项时，一个变量（$isEnd$）将设为 true；否则，$isEnd$ 将设为 false。最后一个元素表示指向下一个包含相同项的节点的链接（$link$）。

定义 8. （项头表）。每个项都有自己的项头表。表中还包含指向 TP 树中对应每个项的第一个链式节点的指针。该表有助于快速定位项节点的位置。

找到初始高效用项集 $\{f, c, d\}$ 后，由于节点 f 的事务加权效用 值最小，因此首先将其添加到这棵树中。此外，还会记录 f 的元素信息。然后，算法为 f 节点创建项头表。

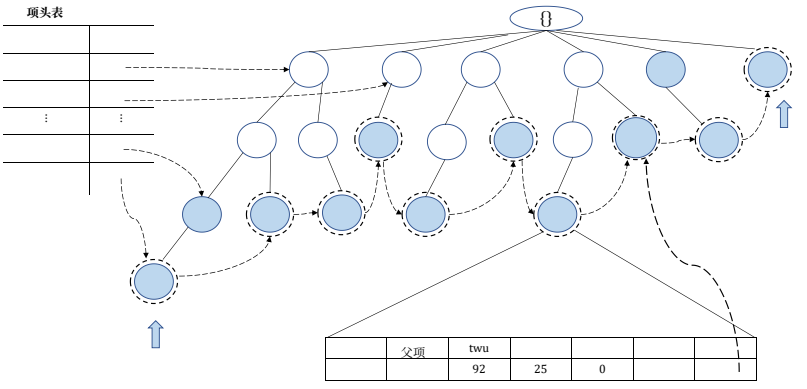


图 2： TP 树的构建

随后以相同方式插入其他节点。不过，每次插入节点时，算法都需要检查是否已存在同名节点。如果已存在，则不会再次创建该节点。但插入的项集会与已存在节点中的项共享一个公共前缀。将第二个高效用项集 $\{f, c, d, b\}$ 插入 TP 树时，它与第一个高效用项集 $\{f, c, d\}$ 共享前缀 (f, c, d)。随后创建节点 b，并记录其信息以更新项头表。此外，更新项 b 的项头表时，需要链接新节点。之后，最后一个 b 成为更新后的尾节点。存储在 TP 树中的其他高效用项集也必须按照相同流程插入，直至完成 TP 树的构建。TP 树的构建过程如图2所示。

4.2. 定向效用提升策略

如前所述，如果用户只设置 k 的值，top-k 高效用项集挖掘的任务就是找出 top-k 个高效用项集。由于用户未给出效用阈值，因此需要提出有效的效用提升策略，以高效地找出 top-k 个高效用项集。本文的 TMKU 算法采用两种策略（SUR 和 RIU）来实现这一目标。

策略 1.（SUR 策略）。SUR 策略 [43] 是一种经典的阈值提升策略，TMKU 算法在将结果存储到 TopKMap 时会使用该策略。TopKMap 可以动态存储 k项及其效用值。

起初，TopKMap 为空。每当发现一个目标高效用项集时，所设计的 TMKU 算法都需要考虑是否将其添加到 TopKMap 中。如果该目标高效用项集的效用值大于 η，则可以将其纳入 TopKMap。不过，当 TopKMap 中存储的项集数量超过用户设定的 k 值时，需要逐步删除并更新 TopKMap 中的某些项集。效用值较低的项集将被效用值较高的项集取代。

策略2.（RIU 策略）。RIU 提升策略 [38] 主要基于各项的实际效用值。如果目标高效用项集的数量超过 k，且对高效用项集按效用值排序后第 k 高的效用值仍高于 η，则应将 η 的值改为第 k 高的值。

RIU 阈值提升策略在 TMKU 算法中发挥着重要作用。通过扫描目标高效用项集，可以确定每个项的真实效用。为了更快地提升效用阈值，TMKU 使用第一个项的真实效用作为 η 的参考值。最后，结合 SUR 策略，可以快速挖掘出正确的前 k 个目标高效用项集。

4.3. 剪枝策略

前文已介绍 TP 树和项头表，它们能够加快处理速度。此外，TMKU 算法还采用了上一小节列出的若干效用阈值提升策略。

本小节将介绍 TMKU 中使用的一些剪枝策略。它们能够显著缩小算法的搜索空间并减少耗时。

策略 3. 如果 $X_i < \xi$ 的效用值总和，那么 X_i 不是高效用项集。

先前研究表明，如果 $TWU(X_i) < \xi$，则 $u(X_i)$ 也小于阈值，因为 $u(X_i)$ 必须小于 $TWU(X_i)$。因此，无需检查任何包含 X_i 的项集。每个项的效用和剩余效用都记录在其效用列表中。如果 X_i 的效用值总和（*sumIu*）小于用户定义的 ξ，那么 X_i 就不是我们要寻找的高效用项集。反之，它就是一个高效用项集。此外，效用列表还记录了剩余效用的总和，称为 *sumRu*，可巧妙地用于剪枝这些效用值，从而提高搜索效率。

策略 4. 在 TMKU 算法中，令 X' 和 ξ 分别为候选项集和当前最小效用值。假设 X' 的 *sumIu* 与 *sumRu* 之和小于 ξ。那么无需检查 X' 的超集和扩展项集。

换言之，*sumIu* 与*sumRu* 之和满足向下封闭性质。因此，不存在任何表示 X' 的扩展能够成为所需的 top-k 高效用项集。这种剪枝策略可以避免探索多个项集。值得注意的是，如果 X' 的*sumIu* 与*sumRu* 之和大于 ξ，那么 X' 的扩展就是一个潜在的高效用项集，需要进一步检查。

此外，我们可以发现，在构建 TP 树的过程中可能会生成越来越多的分支。然而，该任务旨在发现所有包含目标项集的 top-k 高效用项集，因此并非所有高效用项集都必须经过搜索。为了缩短运行时间，TMKU 算法需要在插入节点之前确定是否包含目标项集。因此，TMKU 会预先利用项头表剪除无用分支，并准确、快速地识别包含目标模式的高效用项集。随后，我们可以利用项头表快速查找目标模式树中的节点。

策略 5。 当将 T' 设为目标模式时，TMKU 算法会按照事务加权效用升序排列目标项集中的这些项。此时，选择事务加权效用 值最大的项 i，并使用项头表确定 i 在 TP 树及其分支中的位置。采用自底向上方法查询该高效用项集是否包含目标模式。判断目标模式中的项 j 是否位于分支中时，可以比较当前节点的事务加权效用 与事务加权效用(j)的值。如果当前节点的事务加权效用 小于事务加权效用(j)，则该分支不包含目标模式，可以直接剪枝。如果当前节点的事务加权效用 大于事务加权效用(j)，则需要进一步探索。如果二者相等，还需要检查当前节点的节点名称是否为 j，以避免替换具有相同事务加权效用 值的项。

例如，假设 {f, c, d, b} 是一个高效用项集，{a, b} 表示为 T'。首先，我们找出目标模式中事务加权效用 最高的项 b。然后，需要继续检查 a 是否属于该高效用项集，并依次将该高效用项集中其他项的事务加权效用 值与 a 的事务加权效用 进行比较。结果发现， d 和 c 的事务加权效用 都大于 a，但 f 的事务加权效用 小于 a。根据以上分析可知，{a, b} 不属于该高效用项集，因此停止搜索。

4.4. 提出的算法

根据上述介绍，TMKU 任务可总结如下。为发现包含目标项集的项集，算法 1 通过构建 TP 树和项头表来完成此任务。算法 2 介绍了如何搜索 TP 树以找出目标高效用项集，然后挖掘前 k项集；通过采用若干效用阈值提升策略，用户可以尽快获得所需结果。

（1）TMKU 的构建：此过程可分为两部分：TP 树的构建和项头表的处理。构建 TP 树是 TMKU 算法的基础。需要介绍几个不可或缺的参数，包括其最后一项 x、三个效用值列表（*IUs*、*Rus*、*TUs*），以及 x 的三个效用值。*mapItemNode* 用于存储每个项头表的头节点。在构建 TP 树的过程中，用于存储每个项集最后一项的节点称为尾节点，尾节点会不断更新。此外，还需要使用 *mapItemLastNode* 存储每个项头表的尾节点。首先，算法初始化 TP 树，即为这几个参数设置初始值（第

算法1： 构建过程

输入： X'：高效用项集的前缀；IUs：X' 的效用列表；RUs：X' 的剩余效用列表；TUs：X' 的事务加权效用 列表；x：高效用项集的最后一项；α：x 的效用；β：x 的剩余效用；θ：x 的事务加权效用；*mapItemNode* 存储每个项头表的头节点；*mapItemLastNode*：存储每个项头表的尾节点；ξ：目标效用阈值。

输出： *TP* 树；项头表。

```
1  初始化 TP 树，当前节点 =null，父节点 = null，节点列表 = root.childs;
2  将一个高效用项集插入 TP 树;
3  对于 每个项 e ∈ X′ 执行
4  |  搜索 e 的位置;
5  |  如果 currentNode ==null 那么
6  |  |  在 TP 树中创建一个新节点，并构建 e的项头表;
7  |  end
8  |  更新父节点，节点列表;
9  结束
10 当前节点 = x;
11 如果当前节点 !=空则
12 |  更新当前节点.sumIu，当前节点.sumRu，当前节点.TWU;
13 否则
14 |  为 x 构建新的项头表;
15 结束
16 当前节点.isEnd =true;
```

1 行）。基于使用 HUI-Miner 获得的高效用项集，一旦发现新的高效用项集，就应考虑将其插入 TP 树（第 2 行）。对于 X' 中的每个项 n （第 3 行），都可以轻松找到 n 在此 TP 树中的位置（第 4 行）。如果 TP 树中没有 n 节点，TMKU 算法就会创建一个新节点，并将其信息存储在项头表中（第 5–6 行）。之后，更新节点 *parentNode* 和 *listNodes* 的值（第 8 行）。处理完 X' 中的所有项后，X' 的 TP 树便完全构建完成（第 3–8 行）。然后，当先前存在的 x （通过二分搜索法找到）时（第 10 行），算法会更新当前节点的信息值（第 11 至 12 行）。如果它不存在，则创建 x 的项头表（第 14 行）。最后，将*currentNode.isEnd* 变量的值设为 true（第 16 行）。

(2) TMKU 的发现：处理目标查询的过程采用深度优先搜索。首先，该过程会判断接收到的前缀是否出现。因此，将节点记为 T'的最后一项，然后初始化 *posToMatch* 参数，以记录当前的两个项（第 1 行）。接着，遍历 T' 中的每一项。如果 *sumLu* 与 *sumRu* 之和不小于 ξ，根据策略 4，X 可能是一个目标高效用项集，因此可以进一步与其他项进行比较（第 2–25 行）。此外，根据策略 5，如果 *TWU(currentNode)* 小于 T' 中的当前项，则可以直接将其丢弃，并终止比较（第 9–10 行）。当 *posToMatch* 的值减小到 -1 时，即找到一个目标项集（第 19–20 行）。此时，算法使用递归方法输出该目标高效用项集。首先，探索 X 的所有后缀节点，并从每个节点获取扩展节点（第 22 行）。然后，通过筛除不包含目标项集的高效用项集，存储目标高效用项集。最后，如果剩余的后缀节点满足策略 4的条件，则需要对其进行探索。

之后，算法获得目标高效用项集。接下来，算法的任务是快速挖掘 top-k 个目标高效用项集。首先，将 TopKMap 初始化为空集，并使用第一个目标高效用项集的效用值初始化η （第 26 行）。然后，算法遍历目标高效用项集中的所有项集；如果某个项集的效用值高于 η，则将其添加到 TopKMap 中（第 27–29 行）。否则，移除该目标高效用项集（第 31 行）。当 TopKMap 的大小超过 k 时（第 33 行），将 η 更新为 TopKMap 中第 k 高的效用值（第 34 行）。此外，算法会更新 TopKMap，并在其中保留效用值最高的 k 个目标项集（第 35 行）。最后，成功挖掘并输出所有 top-k 个目标高效用项集（第 38 行）。

5. 性能评估

TMKU 算法通过在各种类型的数据集上进行一系列测试得到了评估。通过比较结果，证明了 TMKU 算法的正确性和效率。正如前文所述，TMKU 算法是首个

算法 2： 发现过程

输入： k：目标高效用项集的数量； T'：获取目标的项集； ξ：目标效用阈值； η：最小效用阈值。

输出： Top-k 个目标高效用项集。

```
posToMatch = |T'|, node = 最后一项的目标项;
当node != null 时
    如果 node.sumIu + node.sumRu ≥ ξ 则
        node.name = 当前高效用项集 X;
        posToMatch 减 1，更新currentNode = node.parent;
        当currentNode != null 时
            如果 posToMatch ≥ 0 则
                新项 y = T'[posToMatch];
                如果 currentNode.TWU <y.TWU 则
                    中断;
                end
                如果 currentNode.TWU == y.TWU AND currentNode == y 则
                    posToMatch减少 1;
                end
            end
            更新 x和 currentNode;
        end
        如果posToMatch ==-1 则
            如果node.sumRu ≥ ξ 且 currentNode.isEnd = true 则
                已发现一个高效用项集;
            end
            递归探索 X的所有后缀节点;
        end
    end
结束
初始化 TopKMap = ∅;
对于 THUIs 中的每个项集 t 执行
    如果 t.utility ≥ η 则
        TopKMap.add(t);
    else
        从 TargetUM 中跳过 t;
    end
    如果 TopKMap 的数量 > k 则
        将 η 更新为 TopKMap 中效用排名第 k 高的项;
        更新 TopKMap，并在 TopKMap 中保留效用值最高的 k 目标项集;
    end
结束
返回top-k 目标高效用项集
```

能够发现包含目标模式的前-k 个高效用项集的算法。所有现有的前-k 高效用项集挖掘算法都无法成功应对考虑用户需求这一挑战。因此，没有任何现有方法可用于比较分析和评估所提出的 TMKU 算法。

一方面，为了确定算法的准确性，我们比较了 TMKU 算法与 TargetUM 算法在各种数据集上经过后处理后的结果。另一方面，我们在六个数据集上进行了各种实验，以全面确定 TMKU 的效率，并使用不同的剪枝策略评估了三个变体。值得注意的是，TMKU$_{V1}$ 仅使用策略 3 和策略5，而 TMKU$_{V2}$ 使用策略 4 和策略 5。使用所有剪枝策略时，即得到 TMKU。因此，我们使用三个版本来评估 TMKU 的效率。

表 3数据集的特征

数据集	Size	事务数	最大长度	平均长度
foodmart	176	4141	4.4	14
零售	1845	24735	10.3	74
电子商务	1968	14976	11.6	29
Chainstore	81102	1112950	7.2	170
蘑菇	1309	8124	23	23
T10I4D100K	8267	100000	29	10.1

5.1. 实验设置

所有算法均使用 Java 编写。实验所用的个人计算机配备了 8.0 GB 随机存取存储器、64 位 Windows 10 和 AMD Ryzen 5 3600 处理器。

为评估 TMKU 算法的有效性和可扩展性，我们在多种数据集上进行了实验，其中包括真实数据库和合成数据库。所有数据集均来自 SPMF 数据挖掘库[1]。这些数据集具有不同的特征，因此观察算法在不同情境下的表现很有价值。表 3 列出了实验数据集的所有特征。下面将作简要介绍。

- **零售** 是一个真实数据集，每笔事务包含许多项。事务来自比利时的一家商店，效用值为合成数据。
- **foodmart** 是一个来自杂货店的事务数据集，其中项较为稀疏，且事务较短。
- **Chainstore** 是一个事务数据集，包含加利福尼亚州一家连锁商店的逾百万笔事务。
- **电子商务** 是一个事务数据集，包含一家在线零售商店的所有事务，并具有真实的效用值。
- **蘑菇** 是一个包含 8,000 多条事务的稠密型真实数据库，其中 最大长度 和平均长度 相同，效用值为合成数据。
- **T10I4D100K** 是由 IBM 数据生成器生成的合成数据，包含 8,124 条事务。

5.2. 项集实验

TMKU 算法的输出是排名前 k 的目标高效用项集。为评估结果的准确性，我们进行了实验，将 TMKU 的输出与 TargetUM 算法后处理后的结果进行比较。这里，u_1 表示 TMKU 结果集中的最低效用值，u_2 表示 TargetUM 算法后处理结果中的最低效用值。由于各数据集的数量和特征不同，我们针对不同数据集测试了多种 k值。

结果主要展示了改变 k值对比数据集中的最小效用所产生的影响。表 4 表明，u_1 和 u_2 具有相同的值，并且随着 k 增大，该值逐渐减小。此外，研究发现两种算法得到的结果相同。例如，我们随机选择西瓜作为零售数据库中用户选择的目标模式，因此将 T' 设为 {976}，它表示西瓜。然后，实验比较了不同 k 值下的最小效用值，以及 u_1 和 u_2。

研究发现，当 k 从 10 增加到 110 时，最小效用值从 4,325 降至 1,389。根据上述分析，本文所述的 TMKU 算法能够成功挖掘出排名前k 的目标高效用项集。表5 展示了六个数据库在不同 k 值下的候选项集数量。例如，我们为蘑菇数据库设置 $T' = \{110\}$，该值表示某种蘑菇的风味。当 k 从 1,000 增加到 6,000 时，候选项集的数量从 16,978 增加到 65,512。显然，随着 k值的增大，候选项集的数量也会增加。这符合排名前k算法的预期结果。

[1]http://www.philippe-fournier-viger.com/spmf/

表 4TMKU 在不同 k 值下的最小效用

数据集	u	通过改变 k 得到的最小效用					
		k_1	k_2	k_3	k_4	k_5	k_6
chainstore	u_1	1643,851	697,152	488,009	407,079	348,233	310,306
{16967}	u_2	1643,851	697,152	488,009	407,079	348,233	310,306
电子商务	u_1	976,618	954,154	940,042	929,362	920,962	913,834
{150561222}	u_2	976,618	954,154	940,042	929,362	920,962	913,834
零售	u_1	1,381	1,379	1,377	1,377	1,376	1,375
{976}	u_2	1,381	1,379	1,377	1,377	1,376	1,375
foodmart	u_1	6,266	4,319	5,541	6,530	7,420	7,978
{1340}	u_2	6,266	4,319	5,541	6,530	7,420	7,978
蘑菇	u_1	449,193	358,323	256,158	125,079	93,238	73,171
{110}	u_2	449,193	358,323	256,158	125,079	93,238	73,171
T10I4D100K	u_1	93,238	39,273	30,843	27,984	26,228	25,716
{71}	u_2	93,238	39,273	30,843	27,984	26,228	25,716

表 5TMKU 在不同 k 值下的候选项数量

数据集	# 通过改变 k 得到的候选项数量					
	k_1	k_2	k_3	k_4	k_5	k_6
Chainstore {16967}	75	193	233	254	261	267
电子商务 {150561222}	43,509	84,230	123,914	160,464	196,604	231,791
零售 {976}	4,430	7,771	10,557	13,064	15,193	17,242
foodmart {1340}	2,653	4,319	5,541	6,530	7,420	7,978
蘑菇 {110}	461	676	997	1248	1345	1456
T10I4D100K {71}	7,761	13,678	18,639	23,178	27,431	31,569

5.3 运行时间实验

我们还进行了实验，以比较算法的多个版本在 k 变化时的运行时间。图3 展示了该算法在六个数据集上不同 k 值对应的执行时间趋势。值得注意的是，排名前k 算法只有一个参数，因此性能可能会有所不同。一般而言，对于 TMKU 任务，很难确定一个合适的 k 值，以找到能够带来最佳性能的最优阈值。因此，在大多数情况下，TMKU 算法中的 k值完全由用户需求决定。本研究对于定向挖掘具有重要意义，它考虑了用户的主观需求，并为用户带来良好的体验。

为进行公平评估，实验记录了三种算法在不同数据集上的性能，如图3所示。结果表明，应用所有策略时，TMKU 算法的运行时间比其他版本更短。随着 k 增大，总运行时间也会增加。由于 TP 树的限制，三个版本的运行时间差异并不显著。不过，我们可以从图3 的（a）和（d）中观察到这种差异。foodmart 和 chainstore 数据集都较为稀疏，其中 chainstore 数据集尤其稀疏。因此，在不同的 k 设置下，可以清楚地观察到三个变体之间的运行时间差异。零售数据库与电子商务数据库之间的差别则不太明显。

总之，该算法在采用剪枝策略的稀疏数据集上表现更好。例如，在 chainstore数据集 中，当 $k = 210$ 时，采用全部剪枝策略的所提算法耗时明显少于 TMKU$_{V1}$ 和 TMKU$_{V2}$。TMKU 需要 1,010 秒才能发现所有排名前k 的目标高效用项集，而 TMKU$_{V1}$ 和 TMKU$_{V2}$ 分别需要 1,019 秒和 1,025 秒才能完成此任务。该结果表明，所有策略对于挖掘排名前k 的目标高效用项集均有效。此外，策略 3和策略 4 分别作用于项集的前缀和后缀，这对加快查询至关重要。策略 5 在目标模式 $|T'| > 1$ 时具有剪枝效果，如 TargetUM 任务中所述。

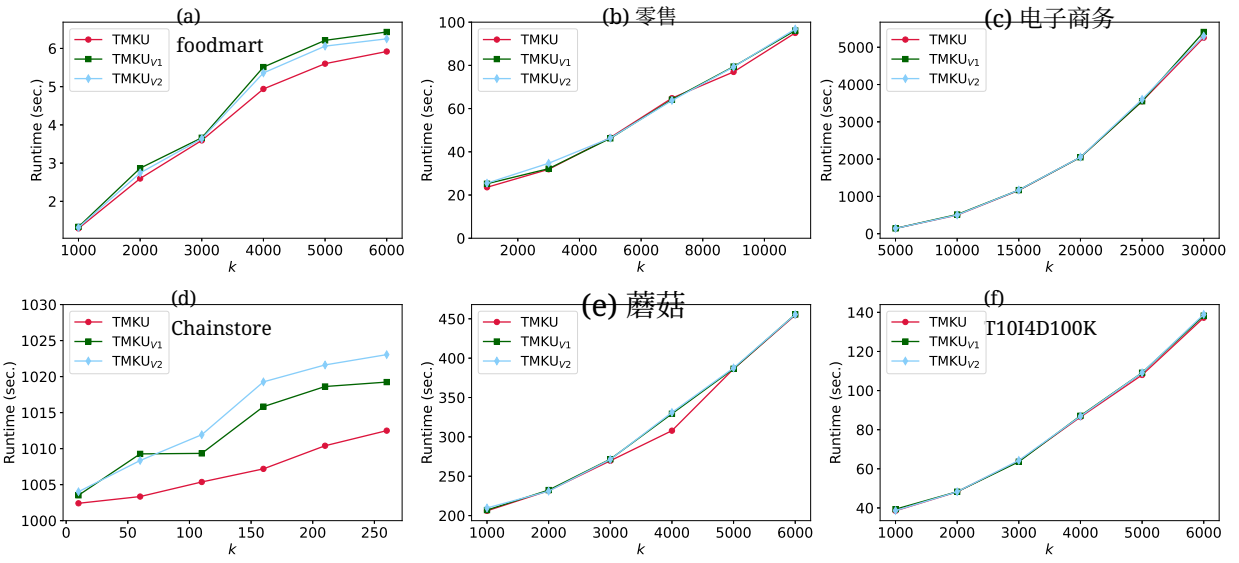


图3： 不同 k 下的运行时间。(a) foodmart (T' ={1340})。(b) 零售 (T' ={1370})。(c) 电子商务 (T' ={150561222})。(d) Chainstore (T' ={16967})。(e) 蘑菇 (T' ={110})。(f) T10I4D100K (T' ={71})。

5.4. 内存实验

我们还进行了额外测试，以检查在改变 k参数时，算法各个版本的内存使用量。图 4展示了详细结果。显然，采用全部剪枝策略的算法比其他变体消耗的内存更少。这是因为在 TP 树构建的第一阶段会消耗一些内存，而这些剪枝策略对内存的影响并不显著。此外，在所有三个版本中，TMKU 的整体内存使用量都会随着 k 的增大而增加，最终趋于稳定。由于通过策略 1 和策略 2 挖掘排名前k 的目标高效用项集需要保存项及其实际效用值，因此算法第二步需要更多存储空间。不过，在某些情况下，算法运行时的内存消耗量可能会略有波动。例如，对于 foodmart 数据集，随着 k 从 1,000 逐渐增至 6,000，内存使用量也逐步增加。然而，在 T10I4D100K 数据库中，当 k 设为 4,000 时，内存使用量略低于此前的水平。此外，对于 chainstore 这类事务数量较多且效用值较集中的数据集，内存使用量更加稳定。总体而言，TMKU 算法在内存使用方面表现相当出色。

5.5. 可扩展性实验

最后，为了更好地评估 TMKU 算法的性能，在 chainstore数据集上测试了 TMKU 的可扩展性。chainstore数据集的大小从 100K 线性增加到 1,100K。三个 TMKU 算法变体在执行时间和内存方面的实验结果如图5 所示，针对 k = 50的情况。总体而言，TMKU 算法在三个算法中表现最佳。可以清楚地看到，随着数据库中事务数量的增加，执行时间也会增加。例如，当 chainstore 数据库大小从 300K 增长到 900K 时，TMKU 的执行时间从 242.86 秒增加到 923.79 秒。与 TMKU$_{V1}$ 和 TMKU$_{V2}$相比，TMKU 所需时间最少。就内存使用而言，可以观察到，随着数据集大小的增加，内存消耗总体上也会增加。然而，由于内存消耗不稳定，TMKU 所采用策略的性能会出现波动。例如，当数据集大小为 500K 时，TMKU 的内存消耗与 TMKU$_{V1}$ 和 TMKU$_{V2}$相差甚微。图 5 清楚地表明，运行时间和内存消耗会随着数据集大小的增加而线性增加。此外，TMKU 算法比另外两个变体耗时更少、占用空间更小，因此具有良好的可扩展性。

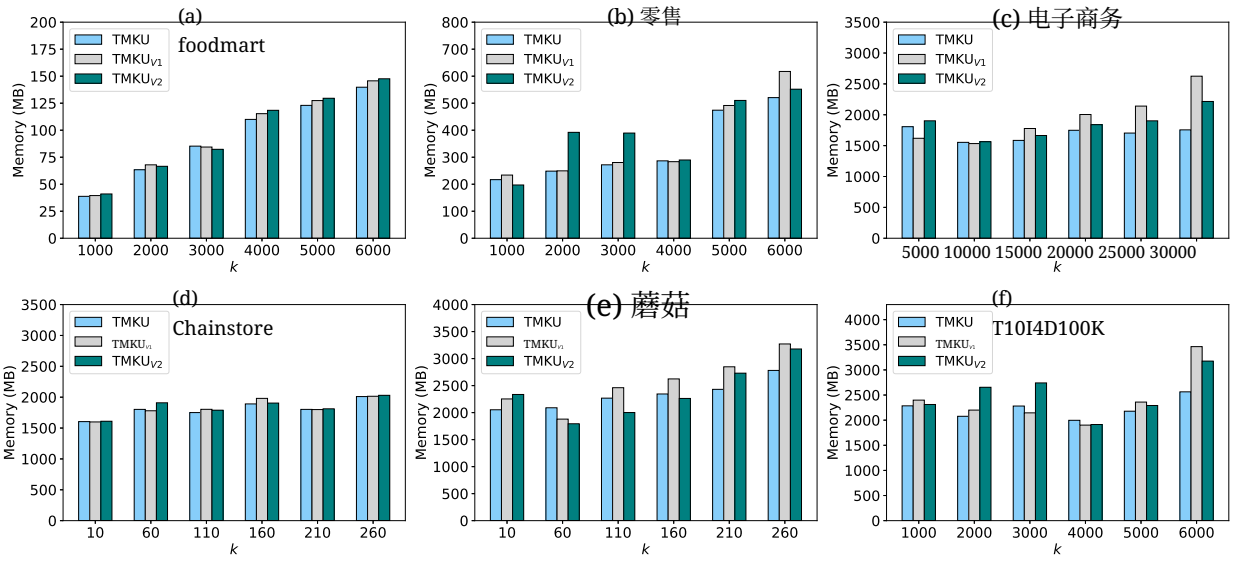
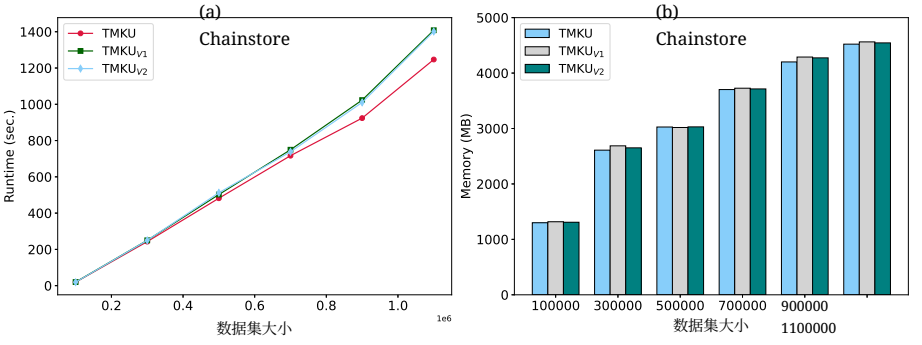


图4： 不同 k 值下的内存。(a) foodmart ($T' = \{1340\}$). (b) 零售 ($T' = \{1370\}$). (c) 电子商务 ($T' = \{150561222\}$). (d) Chainstore ($T' = \{16967\}$). (e) 蘑菇 ($T' = \{110\}$). (f) T10I4D100K ($T' = \{71\}$)。



Figu5： Chainstore ($T' = \{16967\}$, $k = 5$ 上各变体的运行时间和内存可扩展性 0)

.

6. 结论

截至目前，尚未提出用于挖掘目标 top-k 高效用项集的算法。在本研究中，针对这一任务提出了 TMKU 算法。该算法利用多种数据结构，即效用列表、TP 树、TopKMap，以及两种阈值提升策略（SUR 和 RIU）来完成该任务。此外，TMKU 算法还应用了多种剪枝策略，以降低执行时间。在此任务中，用户可以根据自身需求确定 k 和T' 。在六个数据集上测试 TMKU 算法后发现，TMKU 能够完整地构建 TP 树、快速提升阈值，并高效地获得 top-k 目标高效用项集。此外，TMKU 具有较短的处理时间、稳定的内存使用率和出色的可扩展性。TMKU 算法为数据挖掘领域中的一个新问题提供了解决方案；未来，我们将致力于设计更先进的算法，以探索这一富有前景的主题。此外，为了探索 TMKU 的更多应用并将其扩展到更多场景，例如在分布式环境中使用 TMKU，未来的研究还可以考虑挖掘结果可视化，以提升可解释性。

致谢

本研究部分得到国家自然科学基金委员会（编号：62002136、62272196、61472049 和 61572225）、广东省自然科学基金（编号：2022A1515011861）、广州市基础与应用基础研究基金（编号：202102020277）、琶洲实验室青年学者计划（编号：PZL2021KF0023）、教育部可信人工智能工程研究中心（暨南大学）以及广东省数据安全与隐私保护重点实验室的资助。

参考文献

[1] Aggarwal, C.C., Bhuiyan, M.A., Al Hasan, M., 2014. 频繁模式挖掘算法：综述，载于：频繁模式挖掘。Springer，第19–64页。[2] Agrawal, R., Imieliński, T., Swami, A., 1993. 大型数据库中项集之间关联规则的挖掘，载于：ACM SIGMOD 数据管理国际会议论文集，第207–216页。[3] Agrawal, R., Srikant, R., 等，1994. 关联规则挖掘的快速算法，载于：第20届超大型数据库国际会议，Citeseer，第487–499页。[4] Chai, W., 2009. 运用 Web 数据挖掘技术分析并解决企业网络数据安全的主要问题，载于：第一届数据库技术与应用国际研讨会，IEEE，第312–314页。[5] Chan, R., Yang, Q., Shen, Y.D., 2003. 高效用项集挖掘，载于：第三届 IEEE 数据挖掘国际会议，IEEE 计算机学会，第19–19页。[6] Chand, C., Thakkar, A., Ganatra, A., 2012. 利用新近性和货币约束进行面向目标的序列模式挖掘。《国际计算机应用期刊》45。[7] Cheng, H., Han, M., Zhang, N., Wang, L., Li, X., 2021. ETKDS：在滑动窗口模型下对数据流中的前k 高效用项集进行挖掘的高效算法。《智能与模糊系统期刊》，1–22。[8] Chueh, H.E., 2010. 利用时间间隔挖掘面向目标的序列模式。arXiv 预印本 arXiv:1009.0929。[9] Duong, Q.H., Liao, B., Fournier-Viger, P., Dam, T.L., 2016. 使用新颖的阈值提升和剪枝策略挖掘前k 高效用项集的高效算法。《基于知识的系统》104，106–122。[10] Fournier-Viger, P., Gan, W., Wu, Y., Nouioua, M., Song, W., Truong, T., Duong, H., 2022. 模式挖掘：当前挑战与机遇，载于：第27届高级数据库系统应用国际会议研讨会论文集，Springer，第34–49页。[11] Fournier-Viger, P., Mwamikazi, E., Gueniche, T., Faghihi, U., 2013. MEIT：用于面向目标关联规则挖掘的内存高效项集树，载于：高级数据挖掘与应用国际会议，Springer，第95–106页。[12] Fournier-Viger, P., Wu, C.W., Zida, S., Tseng, V.S., 2014. FHM：利用估计效用共现剪枝实现更快速的高效用项集挖掘，载于：智能系统方法论国际研讨会，Springer，第83–92页。[13] Gan, W., Du, Z., Ding, W., Zhang, C., Chao, H.C., 2021a. 序列上的可解释模糊效用挖掘。《IEEE 模糊系统汇刊》29，3620–3634。[14] Gan, W., Lin, J.C.W., Chao, H.C., Fujita, H., Yu, P.S., 2019a. 基于相关效用的模式挖掘。《信息科学》504，470–486。[15] Gan, W., Lin, J.C.W., Chao, H.C., Yu, P.S., 2019b. 高效用情节的效用驱动挖掘，载于：IEEE 大数据国际会议，IEEE，第2644–2653页。[16] Gan, W., Lin, J.C.W., Fournier-Viger, P., Chao, H.C., Hong, T.P., Fujita, H., 2018. 增量式高效用项集挖掘综述。《Wiley 跨学科评论：数据挖掘与知识发现》8，e1242。[17] Gan, W., Lin, J.C.W., Fournier-Viger, P., Chao, H.C., Tseng, V.S., Yu, P.S., 2021b. 效用导向模式挖掘综述。《IEEE 知识与数据工程汇刊》33，1306–1327。[18] Gan, W., Lin, J.C.W., Fournier-Viger, P., Chao, H.C., Yu, P.S., 2019c. 并行序列模式挖掘综述。《ACM 数据知识发现汇刊》13，1–34。[19] Gan, W., Lin, J.C.W., Fournier-Viger, P., Chao, H.C., Yu, P.S., 2020. HUOPM：高效用占用模式挖掘。《IEEE 控制论汇刊》50，1195–1208。[20] Han, J., Pei, J., Yin, Y., 2000. 无需候选生成的频繁模式挖掘。《ACM SIGMOD 纪要》29，1–12。[21] Hemalatha, M., 2012. 购物篮分析——印度零售业中的一项数据挖掘应用。《国际商业信息系统期刊》10，109–129。[22] Krishnamoorthy, S., 2015. 高效用项集挖掘剪枝策略。《应用专家系统》42，2371–2381。[23] Krishnamoorthy, S., 2019. 使用有效的阈值提升策略挖掘前k 高效用项集。《应用专家系统》117，148–165。[24] Kubat, M., Hafez, A., Raghavan, V.V., Lekkala, J.R., Chen, W.K., 2003. 用于定向关联查询的项集树。《IEEE 知识与数据工程汇刊》15，1522–1534。[25] Kuriakose, S., Nedunchezhian, R., 2017. 用于市场分析的序列与并行系统高效自适应频繁模式挖掘技术。《阿拉伯国际信息技术期刊》14，175–185。[26] Li, Y.C., Yeh, J.S., Chang, C.C., 2008. 用于发现高效用项集的孤立项丢弃策略。《数据与知识工程》64，198–217。[27] Lin, J.C.W., Gan, W., Fournier-Viger, P., Hong, T.P., Tseng, V.S., 2016. 不确定数据库中高效用项集挖掘的高效算法。《基于知识的系统》96，171–187。

[28] Liu, M., Qu, J., 2012. 无需候选生成的高效用项集挖掘，载于：第21届 ACM 信息与知识管理国际会议论文集，第55–64页。[29] Liu, Y., Liao, W.k., Choudhary, A., 2005. 快速发现高效用项集的两阶段算法，载于：亚太知识发现与数据挖掘会议，Springer，第689–695页。[30] Miao, J., Wan, S., Gan, W., Sun, J., Chen, J., 2021. 定向高效用项集查询，载于：IEEE 大数据国际会议，IEEE，第5534–5543页。[31] Peek, N., Holmes, J.H., Sun, J., 2014. 生物医学与健康领域大数据的技术挑战：数据源、基础设施与分析。《医学信息学年鉴》23，42–47。[32] Ryang, H., Yun, U.,
2015 k 76, . 使
109, 126
用有效的阈值提升策略挖掘前 高效用模式。《基于知识的系统》–。[33] Shabtay, L., Yaari, R., Dattner, I., 2018. 面向大数据多目标定向挖掘的引导式 FP-growth 算法。arXiv 预印本 arXiv:1803.06632。[34] Song, W., Liu, Y., Li, J., 2014. 通过动态剪枝树结构挖掘高效用项集。《应用智能》40，29–43。[35] Song, W., Zhang, Z., Li, J., 2016 49, 315 340 . 基于包含索引的高效用项集挖掘算法。《知识与信息系统》–。[36] Sun, R., Han, M., Zhang, C., Shen, M., Du, S., 2021. 含负效用的前k 高效用项集挖掘。《智能与模糊系统期刊》，1–16。[37] Tseng, V.S., Shie, B.E., Wu, C.W., Yu, P.S., 2012. 从事务数据库中挖掘高效用项集的高效算法。《IEEE 知识与数据工程汇刊》25，1772–1786。[38] Tseng, V.S., Wu, C.W., Fournier-Viger, P., Yu, P.S., 2015. 挖掘前k 高效用项集的高效算法。《IEEE 知识与数据工程汇刊》28，54–67。[39] Tseng, V.S., Wu, C.W., Shie, B.E., Yu, P.S., 2010. UP-Growth：高效的高效用项集挖掘算法，载于：第16届 ACM SIGKDD 知识发现与数据挖掘国际会议论文集，第253–262页。[40] Wu, C.W., Shie, B.E., Tseng, V.S., Yu, P.S., 2012. 前k 高效用项集挖掘，载于：第18届 ACM SIGKDD 知识发现与数据挖掘国际会议论文集，第78–86页。[41] Wu, Y., Lei, R., Li, Y., Guo, L., Wu, X., 2021. HAOP-Miner：自适应高平均效用一次性序列模式挖掘。《应用专家系统》184，115449。[42] Zhang, C., Du, Z., Dai, Q., Gan, W., Weng, J., Yu, P.S., 2021a. TUSQ：定向高效用序列查询。arXiv 预印本 arXiv:2103.16615。[43] Zhang, C., Du, Z., Gan, W., Yu, P.S., 2021b. TKUS：前k 高效用序列模式挖掘。《信息科学》570，342–359。[44] Zhang, C., Du, Z., Yang, Y., Gan, W., Yu, P.S., 2021c. 序列数据的在架效用挖掘。《ACM 数据知识发现汇刊》16，1–31。[45] Zida, S., Fournier-Viger, P., Lin, J.C.W., Wu, C.W., Tseng, V.S., 2015. EFIM：高效的高效用项集挖掘算法，载于：墨西哥人工智能国际会议，Springer，第530–546页。