RESEARCH ON FAST ITEMSET MINING ALGORITHMS AND THEIR PERFORMANCE

高性能数据挖掘

——快速项集挖掘算法及性能研究

屈俊峰 著

中国水利水电出版社
www.waterpub.com.cn

高性能数据挖掘——快速项集挖掘算法及性能研究

屈俊峰 著

中国水利水电出版社
www.waterpub.com.cn

内 容 提 要

本书主要探讨数据挖掘中的项集挖掘问题，详细介绍了频繁项集、高可用项集、最大频繁项集、频繁闭项集的定义、挖掘算法、搜索空间剪枝技术、性能优化等方面的内容。本书的重点在于介绍如何提高挖掘速度、提升挖掘时的内存使用效率；本书的特色在于不仅对这些挖掘方法与技术在理论上进行描述，同时作者执行了严格的实验用以佐证结论。

本书可作为高年级本科生、数据挖掘方向的研究生、有兴趣青年学者的参考书。

图书在版编目（CIP）数据

高性能数据挖掘 ：快速项集挖掘算法及性能研究 / 屈俊峰著. -- 北京 ：中国水利水电出版社，2018.8
ISBN 978-7-5170-6691-0

Ⅰ. ①高… Ⅱ. ①屈… Ⅲ. ①数据采集－研究 Ⅳ. ①TP274

中国版本图书馆CIP数据核字(2018)第175240号

责任编辑：杨元泓　　封面设计：李　佳

书　名	高性能数据挖掘——快速项集挖掘算法及性能研究 GAOXINGNENG SHUJU WAJUE——KUAISU XIANGJI WAJUE SUANFA JI XINGNENG YANJIU
作　者	屈俊峰　著
出版发行	中国水利水电出版社 （北京市海淀区玉渊潭南路1号D座　100038） 网址：www.waterpub.com.cn E-mail：mchannel@263.net（万水） sales@waterpub.com.cn 电话：（010）68367658（营销中心）、82562819（万水）
经　售	全国各地新华书店和相关出版物销售网点
排　版	北京万水电子信息有限公司
印　刷	三河市兴国印务有限公司
规　格	170mm×240mm　16开本　10.75印张　200千字
版　次	2018年8月第1版　2018年8月第1次印刷
印　数	0001—2000册
定　价	48.00元

前　言

目前，大部分的计算机应用系统都以数据库作为底层支撑。例如，工业控制系统通过传感网络不断收集工业过程的各种参数，将其存入数据库，这些参数是系统下一步的操作基础，也能为工程师提供重要的状态信息。在网站日志分析系统中，每个页面的访问情况或每个用户的访问历史被存储在数据库里，对这些信息进行分析可以发现用户感兴趣的内容，进而改进网站布局或向用户推介相关信息。在零售商业中，一个顾客一次购物的所有商品形成数据库中的一条记录，对这些记录进行分析可以提示短缺商品或更好地布局终端货架。

在大数据时代，如何从数据库中挖掘有用信息是一个吸引人且具有挑战性的任务。通常，一个数据库包含若干条记录，每条记录由若干项组成。例如，一个顾客的一次购物记录是其所购商品名称的集合。这里，一个商品名称即是一个项。由项组成的集合称为项集，项集是能够从数据库中挖掘出的重要信息。项集挖掘问题通常被定义为：给定一个数据库及特定标准下的一个条件，要求从库中挖掘出能满足这一条件的所有项集。项集挖掘任务通常非常耗时，因为搜索空间巨大。如果一个数据库中出现了 n 个项，那么存在 2^n 个项集在搜索空间中。

因此，如何高效挖掘数据库中的项集是一项有挑战性的任务。本书选定了四类项集挖掘问题。本书的第 1 章为引言，给出概要性的介绍。第 2～6 章介绍项集挖掘领域最经典的频繁项集挖掘问题。第 7～9 章介绍了最近的研究热点高可用项集挖掘问题。第 10、11 章分别介绍了最大频繁项集挖掘问题及频繁闭项集挖掘问题。对每一类问题，在介绍了问题定义之后，作者以挖掘性能提升为导向，分别从挖掘算法、搜索空间剪枝技术、性能优化等方面进行了详细阐述。本书的特色在于，大部分的方法与技术都配有实验验证，实验结论不仅能够佐证方法技术的有效性，而且也给了读者非常直观的认识。

本书图文并茂、以实用为主，力求能够成为高年级本科生、数据挖掘方向的研究生、有兴趣的青年学者在研究相关主题时的参考书。本书的编写得到了湖北文理学院教师科研能力培育基金项目（编号：2017kypy037）的支持。本书在编写内容与特点上均进行了一种新的尝试，缺点和错误在所难免，由于编者水平有限，希望广大读者给予批评和指正，对此作者深表谢意。

屈俊峰

湖北文理学院计算机工程学院

2018 年 5 月

目　录

1 概述

交易数据库是最常见的一类数据库，这种数据库的每一个条目是一个项的集合。当数据库变得越来越大时，人们希望从数据中发现一些规律。对于交易数据库，这些规律可以表示成“项集”。站在不同的角度，人们需要的项集是不一样的。本章将详细描述什么是项集，给出频繁项集及高可用项集的介绍、挖掘方法与历史。在本章的最后，我们给出了本书的组织结构。

- 项集：数据挖掘研究领域的焦点之一
- 频繁项集挖掘问题的研究历史
- 高可用项集挖掘问题的研究历史
- 本书的组织结构

计算机的广泛使用促使现实生活中各领域中的大量数据被数字化，数据库技术的飞速发展则推动了这些数字化数据的有效管理和快捷使用。随着时间的推移和采集渠道的增多，汇入各种数据库中的数据越来越多，于是，一个现实的问题产生了：如何从这些体积日益膨胀的数据库中找出重要的或用户感兴趣的信息？

例如，随着条形码的广泛使用，超市中商品的销售信息得以大量地进入计算机。通常一个顾客的一次购物行为信息能够形成超市交易数据库中的一条记录。这种记录着用户交易行为的数据库不仅仅是一张数字和字符组成的表，其中还蕴含着丰富的信息。再如，店员可能想知道哪些商品的组合被频繁地同时售出，基于这样的信息，他能更好地组织货架布局促进商品销售；销售经理可能想知道哪些商品可以带来较大的利润，基于这样的信息，他能更好地安排营销计划。然而，这些信息并非显而易见而是隐藏在库中大量数据的背后。这就需要合适的计算机技术协助人们从数据的海洋中发现潜在的有用信息。

诸如上面的问题广泛地存在于现实生活中的各个领域，再比如说如何从病例数据库中提取分类规则用于辅助病人的诊疗，如何在天文数据库中寻找宇宙演化的规律然后作出科学预测，等等。这类问题的提出开创了数据库应用的一个新的研究领域，即数据挖掘与知识发现（Data Mining and Knowledge Discovery，DMKD），有些地方也称其为基于数据库的知识发现（Knowledge Discovery from Databases，KDD）。

在该领域中，有一个非常有意思的现象：一个问题被提出之后，解决方案往往层出不穷。比如非常著名的频繁项集挖掘（Frequent Itemset Mining）问题，自 Rakesh Agrawal 等在 1993 年 ACM Special Interest Group on Management of Data（ACMSIGMOD）会议上正式提出后[15]，现在至少有 10 个较著名的算法能够解决此问题。即便是到最近，解决此问题的新算法还在继续被提出[28, 101, 109, 119]。

造成这一现象的本质原因是算法性能提升的无极限性。对于相同的问题，人们总是在不停地探寻着最佳算法（通常以算法的运行速度来衡量），然而现实却是“没有最佳，只有更佳”。以下客观因素加剧了这一现象：

（1）数据库规模的不断膨胀。

（2）硬件体系结构的改进，比如说并行和多核。

总而言之，对于该领域的许多问题，人们不仅关心是否有解决方案，更为关心的是解决方案是否足够地快，同时耗费的其他资源（主要是内存资源）又在可以接受的范围之内。

本书研究数据挖掘领域中的两类项集挖掘问题，一是频繁项集挖掘问题；二是高可用项集挖掘（High Utility Itemset Mining）问题。如上所述，我们不仅致力于探索解决这两个问题的新算法，并且非常重视相关算法（包括先前的算法和我们提出算法）的性能问题。本章下面几节我们将依次阐述为什么要关注项集挖掘，项集挖掘问题目前的研究现状，我们在项集挖掘方面的一些工作，以及本书的组织结构。

1.1 项集：数据挖掘研究领域的焦点之一

从实际中应用最广泛的关系数据库的视角观察，一个数据库能够被看作是一张二维表。这张表是由一条条记录组成，每条记录是由若干个项构成，每一个项又涉及到零个或多个属性值。项集表示若干项的集合。项集的特征可以从许多方面来刻画，例如在数据库中出现的次数是否足够多、价值是否很大，是否满足一定的约束条件，等等。基于不同的特性，从数据库中可以导出不同的项集的集合。对于项集某一（些）方面特性感兴趣的用户或应用系统，关心如何从数据库中挖掘出所有的满足这一（些）方面特性要求的全部项集。

最基本同时也是最重要的一种项集特征是项集在数据库中的出现次数，称为项集的支持度[15]。在许多应用中，用户希望在一个数据库中找出所有支持度超过其指定阈值的项集。此任务即为频繁项集挖掘。频繁项集挖掘在数据挖掘研究领域中扮演着一个非常重要的基础性角色。

首先，频繁项集挖掘问题是数据挖掘领域最早提出的问题之一（1993 年），也是该领域中最受研究者关注的问题之一。根据谷歌学术搜索的统计，正式定义该问题的论文[15]目前已经被引用了上万次。其次，由频繁项集挖掘问题派生出来的数据挖掘问题非常多。派生出的问题可以分为三类，一类是重定义问题内部约束条件而产生的新问题；另一类是扩展问题外部约束条件而产生的新问题；第三类是同时重定义问题内、外部约束条件而产生的新问题。

在第一类问题中，比较著名的有以下几个：最大频繁项集挖掘问题要求挖掘出所有的最大频繁项集[25, 37, 39, 49, 63, 134]，一个最大频繁项集是频繁的且没有任何频繁项集是它的超集；频繁闭项集挖掘问题要求挖掘出所有的频繁闭项集[80, 81, 84, 115, 131, 132]，一个频繁闭项集是频繁的且它的任一频繁超集都和此闭项集有不同的支持度；top-k 频繁项集挖掘问题要求挖掘出支持度最高的 k 个频繁项集[45, 86, 95]，等等。从上面的问题描述中我们能够看出，从最大频繁项集集合或频繁闭项集集合中可以导出完整的频繁项集集合，不同的是从前者中导出的频繁项集没有支持度信息而从后者导出的频繁项集包含着完整的支持度信息。对于 top-k 频繁项集的挖掘则不需要指定最小支持度阈值。

第二类问题是频繁项集挖掘问题在不同软硬件环境下的扩展，例如：从不确定数据中挖掘频繁项集[13,20,26,107]，从数据流中挖掘频繁项集[31,46,69]，从敏感或隐私数据中挖掘频繁项集[21,117]，在多核 CPU 上进行频繁项集挖掘[32,71]，以及在内存受限条件下进行频繁项集挖掘[94, 101]。

第三类问题是前两类问题的叠加，例如在数据流上挖掘 top-k 的频繁项集[53]。解决这些派生问题的方法都直接受原始问题解决方法的影响，即只要有新的高效

的频繁项集挖掘算法被提出，那么就有可能通过改进新算法来解决这些派生问题。因此，频繁项集挖掘问题是以上所有问题的基础和核心。

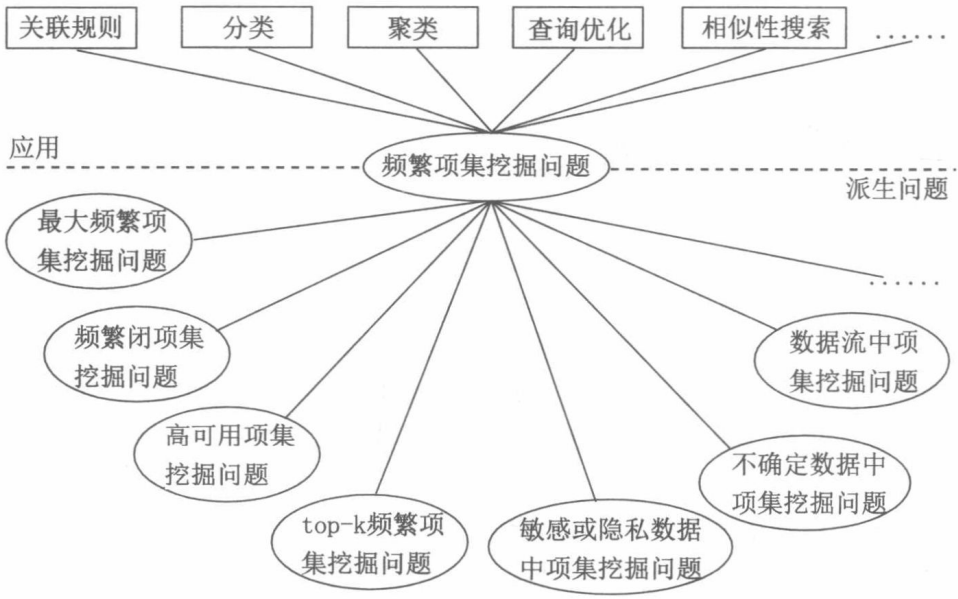

图 1-1　项集挖掘相关问题及应用

再者，挖掘出来的频繁项集有着广泛的应用。例如，频繁项集最开始是作为关联规则挖掘的中间结果而开始得到研究者的关注[15, 16]，后来发现也可以基于频繁项集做分类及聚类处理[14, 19, 30, 54, 116, 126]。频繁项集也能用在演绎数据库和查询处理的优化上[75, 124]。例如，对查询处理的一个非常简单的优化可以是先找出用户频繁执行的一些查询操作，然后在查询结果尺寸不大的情况下，对结果做直接保存。这样做的好处有两点，一是可以实现查询提示，即用户在输入部分关键字后，系统对剩下的可能关键字作出提示，既方便用户输入又简化系统处理流程，目前像著名的搜索引擎谷歌及百度都实现了这样的功能；二是当这样的查询提交到系统之后，系统可以快速直接返回查询结果。最近的一些文献表明频繁项集还能在下面一些应用中发挥作用：复杂结构的索引及相似性搜索[121-123]，时空数据挖掘[27, 55, 120, 135]，多媒体数据挖掘[127]，Web 数据挖掘[29, 34, 51, 85, 90, 106]，甚至软件缺陷的定位[60, 61, 62, 65]。

图 1-1 总结了本节的内容，可以看出，项集挖掘的研究不仅具有重要的理论意义，而且也有广泛的应用价值。如何挖掘具有某些特征的项集以及如何高效地挖掘出这些项集毫无疑问已经成为数据挖掘研究领域的焦点之一。本书主要研究频繁项集及高可用项集的挖掘算法及其性能问题，下面我们将详细介绍这两个问题的研究历史。

1.2 频繁项集挖掘问题的研究历史

1993年，频繁项集挖掘作为关联规则挖掘的一个子问题由Rakesh Agrawal在ACMSIGMOD会议上首先提出来[15]。当发现这个子题是关联规则挖掘中最困难的步骤时，人们将这个子问题独立出来专门展开研究。解决这个问题的第一个算法是AIS[15]。

早期的Apriori算法已经被许多教科书收录并奉为经典[16, 42, 64, 76]。Apriori有两个经常被诟病的问题：生成了大量的候选项集及多遍的数据库扫描。针对这两个问题一些Apriori算法的改进版本相继被提出[100, 108]。例如，Partition算法首先把一个数据库水平切分成可以导入内存的一个个子库，接着依次挖掘每个子库上的频繁项集，最后通过一次数据库遍历来验证从子库中生成的频繁项集在整个数据库上是否仍是频繁的[100]。另一些Apriori的优化则采用哈希表来实现项集的快速定位用以加速计数过程[79]。

在1997年的ACM Special Interest Group on Knowledge Discovery and Data Mining（ACM SIGKDD）会议上由Mohammed J. Zaki提出的Eclat算法非常新颖地采用了垂直的数据库视角来挖掘频繁项集[133]，之后Mohammed J.Zaki连续提出了扩展性非常好的Eclat版本和集成了diffset技术的Eclat算法[129, 130]，其中后者称为dEclat算法。dEclat显著地减少了Eclat算法核心结构Tid-list的长度，从而获得了明显的性能提升。

三年之后，Jiawei Han等在2000年的ACM SIGMOD会议上提出了著名的FP-growth算法[43, 44]，在这个算法中，他们首次将一个数据库压缩成一种前缀树结构，并在此结构上采用递归构造条件数据库的方式来挖掘频繁项集。FP-growth算法在数据结构上与之前的算法大相径庭，此后，许多基于FP-growth的新算法被陆续开发出来。例如，Jiawei Han针对FP-growth算法在稀疏数据库上性能较差的问题，2001年提出了能更有效地处理这种数据库的H-struct结构，以及基于此结构的H-mine算法[82, 83]。

Junqiang Liu等人在2002年提出了OP算法[70]，此算法是FP-growth算法和H-mine算法的混合体，能根据数据库的特性自动地在FP-growth算法和H-mine算法之间切换。在这一时期其他较有影响的算法包括，采用位图表示数据库的VIPER算法[11, 103]，采用树投影策略的TreeProject算法[12]，以及采用直接计数的DCI算法[77, 78, 89]。

由于频繁项集的重要理论意义与广泛应用价值逐步地凸现出来（见上节），在认识到实际解决此问题需要大量的计算时间后，为了对大量出现的算法作性能上的对比，找出快速的算法，著名国际会议International Conference on Data Mining

（ICDM）在 2003 年及 2004 年连续举办了两次 Frequent Itemset Mining Implementation（FIMI）工作组。会议的组织者鼓励全世界所有对此问题感兴趣的研究者提交自己的算法和相关论文，然后由主委会独立地对所有提交的算法进行性能上的评测。这次会议不仅接受对已有算法的高性能实施，而且也接受新提出的算法。于是，大量性能令人激动的算法出现在这两次工作组上。例如，Ferenc Bodon 和 Walter A. Kosters 等了分别提出了两种 Apriori 算法的快速实施方案[22,52]，Christian Borgelt 提出了基于前缀树的 Apriori 及 Eclat 算法[23]，Lars Schmidt-Thieme 详细地研究了 Eclat 算法的特性，并提出了自己的 Eclat 版本[102]。一系列 FP-growth 算法的改进和优化也在这次会议上被提出。例如，Liu Guimei 详细地研究了 FP-growth 的特点及流程，识别出了其中几个制约其性能的,关键因素，提出了 AFOPT 算法[66,67,68]；Osmar R. Zaiane 提出 COFI-tree 结构可以减少候选生成的花费[128]；Andrea Pietracaprina 提出了采用 Patricia Tries 结构来快速地实现 FP-growth 流程的 PatriciaMine 算法[50, 87]；Balazs Racz 采用了线性投影的方式提出的 nonordfp 算法能够快速地构造前缀树[96]。

在 2003 年的工作组中，经过主委会独立的性能测试后，由 Gosta Grahne 和 Jianfei Zhu 提出的 FPgrowth*算法脱颖而出[38,40]，FPgrowth*是集成 FP-array 技术的 FP-growth 算法的一个高效变种。在 2004 年的工作组中，由 Takeaki Uno，Masashi Kiyomi 和 Hiroki Arimura 提出的 LCMv2 算法拔得头筹[112, 113]，LCMv2 算法是集多种优化策略于一体的杂交算法。

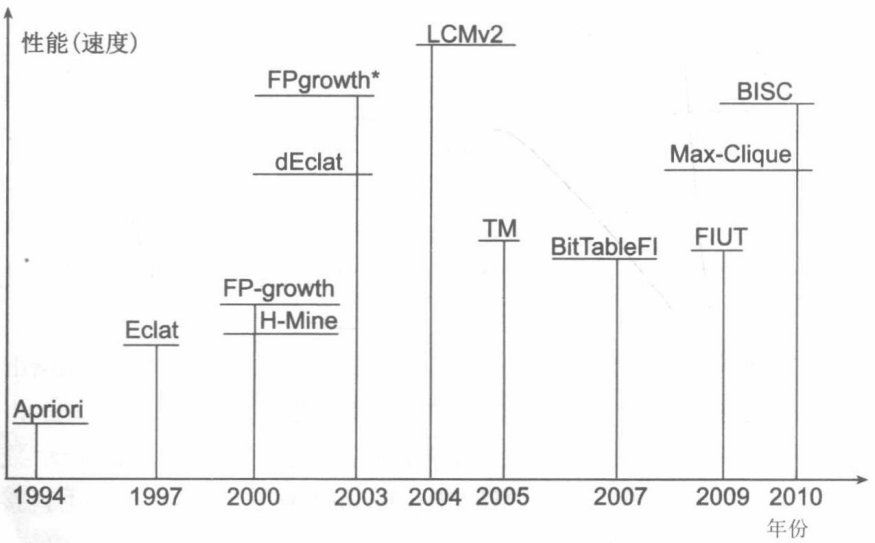

图 1-2　频繁项集挖掘算法发展历史

2003、2004 年 ICDM 的 FIMI 工作组把频繁项集挖掘算法的研究推到了一个顶峰。由于 FPgrowth*算法及 LCMv2 算法优异的性能表现（体现在极快的运行速

度上），此后，在性能上能够超越他们的算法可谓凤毛麟角。

2004年之后提出的一些较有影响的算法如下。2005年Mingjun Song提出了TM算法[104, 105]，TM算法采用了交易树结构，这是一种前缀树和Tid-list的混合结构。同样是2005年由Amol Ghoting等提出了Tiling算法[35, 36]，此算法采用了Cache敏感的策略来提高性能。2007年，Dong Jie等提出了BitTableFI算法[33]，这个算法是先前基于位图算法的扩展。

2009年，Yuh-Jiuan Tsay等提出了FIUT算法[109]，采用从最大频繁项集往下分解的方式，逐层生成长度较小的频繁项集。2010年Xie Yan等提出了Max-Clique算法[119]，采用图论中的相关理论来挖掘频繁项集。同年，Chen Jinlin也提出了性能优异的杂交算法BISC[28]。2011年Benjamin Schlegel等提出了CFP-growth算法能够极大地减少FP-growth的内存消耗[101]。

然而，这些算法在性能上，主要指运行时间上，很难胜过FPgrowth*及LCMv2算法。例如，TM算法在文献中实测的性能只是接近FPgrowth*；BitTableFI和FIUT算法仅仅是优于基本的FP-growth算法（不是FPgrowth*）；Tiling算法只是在特定机器上性能表现优异，而Max-Clique算法则是在特定的数据库上能够和LCMv2匹敌。这些算法中最有希望的是BISC算法，按照文献中的测试它的性能非常好，但问题是对于杂交算法，很难确定到底是其中的什么策略导致了较好的性能，况且此算法的过程过于复杂不易实现。因此，可以认为FPgrowth*及LCMv2算法是当前最快的算法或属于最快算法的集合。图1-2给出了频繁项集挖掘算的法发展历史。

1.3 高可用项集挖掘问题的研究历史

目前，大部分的计算机应用系统都以数据库作为底层支撑。例如，工业控制系统通过传感网络不断收集工业过程的各种参数，将其存入数据库，这些参数是系统下一步的操作基础也能为工程师提供重要的状态信息。在网站日志分析系统中，每个页面的访问情况或每个用户的访问历史被存储在数据库里，对这些信息进行分析可以发现用户感兴趣的内容，进而改进网站布局或向用户推介相关信息。在零售商业中，一个顾客一次购物的所有商品形成数据库中的一条记录，对这些记录进行分析可以提示短缺商品或更好地布局终端货架。

在大数据时代，如何从数据库中挖掘有用信息是一个吸引人且具有挑战性的任务。通常，一个数据库包含若干条记录，每条记录由若干项组成。例如，一个顾客的一次购物记录是其所购商品名称的集合。这里，一个商品名称即是一个项。由项组成的集合称为项集，项集是能够从数据库中挖掘出的重要信息。项集挖掘问题通常被定义为：给定一个数据库及特定标准下的一个条件，要求从库中挖掘

出能满足这一条件的所有项集。项集挖掘任务通常非常耗时，因为搜索空间巨大。如果一个数据库中出现了 n 个项，那么存在 2^n 个项集在搜索空间中。最常见的一种项集重要性衡量标准是支持度。如果一个项集中的每个项都出现在一条记录中，那么称这条记录包含这个项集或者这个项集出现在这条记录中。给定一个数据库，一个项集的支持度是库中包含此项集记录的数量。按照支持度标准，项集的支持度越大就越重要。经典的频繁项集挖掘问题，即是给定一个数据库和一个最小支持度阈值，要求找出支持度大于等于阈值的所有项集，这些项集称为频繁项集。此问题是数据挖掘领域最基础的问题之一，我们已在上节中作了讨论。

在频繁项集挖掘问题中，数据库中出现的所有项被赋予了同等价值，支持度反映了项集的重要性。然而，在一些场景下，项集的重要性并不能简单地由支持度衡量。例如，需要从零售数据库中挖掘出利润最大的商品组合。通常,项集{牙膏、香皂}的支持度要高于项集{手机、内存卡}的支持度，但是后者产生的利润或许要远高于前者。于是，研究者提出了项集重要性衡量的效用标准。在此标准下,每一个项被赋予了一个权重值（外部效用），一条记录中的每一个项关联着一个计数（内部效用）。一个项集的效用由包含此项集的所有记录中相关项的内、外部效用确定。同支持度标准一样，多数情况下人们感兴趣的是效用高的项集，故提出了高效用项集挖掘问题：给定一个数据库及效用阈值，要求挖掘出效用大于等于此阈值的所有项集（称为高效用项集）。

高效用项集有各种重要的应用，例如，在上面的例子中，从交易数据库中挖掘出高效用项集，即利润大的商品组合，有助于策划更加赢利的商业活动。

高可用项集挖掘问题是频繁项集挖掘问题的扩展，但因问题定义的不同，频繁项集挖掘算法不能直接用于高可用项集的挖掘。高可用项集挖掘问题在 2005 年由 YaoHong 在 Society for Industrial and Applied Mathematics on Data Mining（SIAMDM）国际会议上正式提出来之前[125]，已经有一些算法，可以挖掘份额频繁项集，例如 ZP 算法[18]。份额频繁项集是高可用项集的一种特殊形式，适用于挖掘份额频繁项集的算法经改造后也可用于高可用项集的挖掘。

后来出现的高可用频繁项集挖掘算法按照基本思路的不同可以分为两大类。一类是基于 Apriori 的算法，他们逐层生成候选项集再扫描数据库计算项集的可用性，例如：Brock Barber 等在 2003 年提出的 ZP 及 ZSP 算法[18]；Yu-Chiang Li 等在 2005、2008 年连续提出的 FSH 算法[58]、ShFSH 算法[57]、DCG 算法[56]、FUM 算法[59]、以及 DCG+算法[59]。另一类是基于前缀树生成候选项集然后再进行测试的算法，例如 Chowdhury Farhan Ahmed 等在 2009 年提出了 IHUPTWU 算法[17]，Vincent S. Tseng 等 2010 提出的 UP-Growth 算法[111]及 2012 年提出的 UPGrowth+算法[110]。