最大共现非重叠序列规则挖掘

Yan Li、Chang Zhang、Jie Li, Wei Song、Zhenlian Qi、Youxi Wu、*IEEE*会员，以及 Xindong Wu，*IEEE*会士

arXiv:2301.12630v1 [cs.DB] 30 Jan 2023

摘要—**序列模式挖掘（SPM）旨在从给定序列中发现潜在有用的信息。尽管已有多种 SPM 方法得到研究，但其中大多数都着眼于挖掘所有模式。然而，用户有时希望挖掘具有相同特定前缀模式的模式，这类模式称为共现模式。鉴于序列规则挖掘能够更好地利用 SPM 的结果，并获得更好的推荐性能，本文研究最大共现非重叠序列规则（MCoR）挖掘问题，并提出 MCoR-Miner 算法。为提高支持度计算的效率，MCoR-Miner 采用深度优先搜索和回溯策略，并配备索引机制以避免使用顺序搜索。为避免对某些序列进行无用的支持度计算，MCoR-Miner 采用过滤策略，剪枝不包含前缀模式的序列。为减少候选模式的数量，MCoR-Miner 采用频繁项和二项式枚举树策略。为避免通过蛮力法搜索极大规则，MCoR-Miner 使用筛选策略。为验证 MCoR-Miner 的性能，我们在八个序列上与十一种竞争算法进行了比较。实验结果表明，MCoR-Miner 优于其他竞争算法，且其推荐性能优于频繁共现模式挖掘。所有算法和数据集均可从 https://github.com/wuc567/Pattern-Mining/tree/master/MCoR-Miner 下载。**

关键词—**序列模式挖掘、序列规则挖掘、规则前件、共现模式、最大规则挖掘**

I. 引言

AS 作为知识发现的一种重要方法， [1], 序列模式挖掘（SPM） [2] 旨在从序列数据集中挖掘满足特定条件的子序列（模式） [3]。针对不同的挖掘需求，人们开发了多种序列模式挖掘方法，例如面向时间序列的保序序列模式挖掘 [4], 面向大规模数据库的序列模式挖掘 [5, 6],事件模式挖掘 [7], 空间共址模式挖掘 [8, 9], 对比序列模式挖掘 [10, 11], 负序列模式挖掘[12, 13], 高效用序列模式挖掘 [14], 高平均效用序列模式挖掘 [15–17], 离群序列模式挖掘 [18], 三向序列模式挖掘 [19, 20], 共现序列模式挖掘 [21, 22], 以及带间隔约束的序列模式挖掘 [23]。传统序列模式挖掘的一个缺点是，它只考虑给定模式是否出现在某个序列中，而忽略了该模式在序列中的重复出现[24]。例如，根据传统序列模式挖掘，模式 ad 在序列 $adbdad$ 中的支持度（出现次数）为 1，尽管模式 ad 在该序列中出现了多次。从这个例子可以看出，传统序列模式挖掘忽略了重复出现，这意味着一些有趣的模式会被遗漏 [25]。

为解决这一问题，研究者提出了间隙约束序列模式挖掘 [26],，可表示为$\mathbf{p} = p_1p_2 \cdots p_{m-1}p_m$，其中 $gap = [a, b]$ （或$\mathbf{p} = p_1[a, b]p_2...[a, b]p_m$） ， a和 b （$0 \leq a \leq b$）均为整数，分别表示 p_j 和 p_{j+1}之间通配符的最小数量和最大数量 [27, 28]。例如，模式 $a[0, 3]d$ 表示 a 和 d 之间有零到三个通配符。序列中模式的重复出现有助于获取有价值的信息，因此间隙约束序列模式挖掘有诸多应用，例如重症监护病房患者的脓毒性休克预测 [29], 、关键词提取 [30], 、缺失行为分析 [31],以及传销模式挖掘 [32]。然而，间隙约束序列模式挖掘不仅难以求解，而且形式多样，例如周期性间隙序列模式挖掘 [33], 、不相交序列模式挖掘 [34], 、单次序列模式挖掘[35], 和非重叠序列模式挖掘 [23]。以往研究表明，非重叠序列模式挖掘可以避免产生许多冗余模式 [24]；例如，在非重叠条件下，模式 ad 在序列 $adbdad$中出现两次。

遗憾的是，当前基于间隙约束序列模式挖掘的方案只能挖掘出所有频繁模式。与规则挖掘相比，频繁模式挖掘并不适合用于预测或推荐。规则挖掘 [36]侧重于挖掘诸如$\mathbf{p} \rightarrow \mathbf{q}$这样的规则，这表示如果模式$\mathbf{p}$ 出现在序列中，那么模式$\mathbf{q}$ 很可能随后出现，且其概率高于或等于给定的置信度，即 $conf(\mathbf{p} \rightarrow \mathbf{q}) \geq mincf$。然而，当用户只想找出具有相同前件$\mathbf{p}$的规则时，挖掘所有强规则不仅耗时费力，而且毫无意义。例如，在推荐问题中，研究人员希望从历史数据中发现潜在规则，并利用近期事件预测未来事件。显然，发现具有相同近期事件的规则，而不是所有规则，更有价值。这类规则称为共现规则，也更有意义。为便于说明，下面给出一个示例。

稿件收到日期：2022年8月22日。（通讯作者：Y. Wu）。Yan Li、Chang Zhang、Jie Li 就职于河北工业大学经济与管理学院，中国天津，邮编300400（电子邮件：lywuc@163.com）。Wei Song 就职于北方工业大学，中国北京（电子邮件：songwei@ncut.edu.cn）。Zhenlian Qi 就职于广东生态工程职业学院，中国广州510520（电子邮件：qzlhit@foxmail.com）。Youxi Wu 就职于河北工业大学人工智能学院，中国天津，邮编300400（电子邮件：wuc567@163.com）。Xindong Wu 就职于合肥工业大学大数据知识工程重点实验室（中国教育部），中国合肥，邮编230009（电子邮件：xwu@hfut.edu.cn）

示例 1 假设有一个序列s =$adbdadcdccabadcd$、一个间隔约束 gap = *[0,3]*，以及预先设定的支持度阈值 *minsup* = *3*。根据非重叠序列模式挖掘，可得到 *12* 个频繁模式：{a, c,d, ad, cd, dc, dd, adc, add, dcd, ddc, $adcd$}。此外，如果继续将最小置信度阈值 *mincf* =设为 *0.7*，则可得到 *11* 条规则：{$a \rightarrow d$, $a \rightarrow dc$, $a \rightarrow dd$, $a \rightarrow dcd$,$d \rightarrow d$, $d \rightarrow c$, $ad \rightarrow d$, $ad \rightarrow c$, $ad \rightarrow cd$, $dd \rightarrow c$, $adc \rightarrow d$}。显然，用户难以应用如此大量的频繁模式和规则。

然而，如果我们有一个前缀模式（或前件）p =ad，频繁超模式和规则的数量就会减少。详情如下所示。我们知道，p= ad 在序列s中出现了四次，因为子序列s_1s_2 是 ad，而子序列 s_5s_6、 $s_{11}s_{14}$和 $s_{13}s_{16}$ 也都是 ad。因此，p 在s 中的支持度为四。类似地，我们知道子序列 $s_1s_4s_7$、$s_5s_6s_9$和 $s_{11}s_{14}s_{15}$ 都是adc。因此，模式 adc 在s 中的支持度为三，不小于 $minsup$ = *3*。因此，模式 adc 是p = ad的频繁共现模式。此外， $ad \rightarrow c$ 是一条共现规则，其置信度为 *3/4 = 0*。*75* $\geq$ $mincf$= *0.7*。同样，我们知道模式 add 和 $adcd$是频繁模式，且 $ad \rightarrow d$ 和 $ad \rightarrow cd$ 是共现规则。因此，模式p只有三个共现模式： adc、 add 和 $adcd$；以及三条共现规则：$ad \rightarrow c$、 $ad \rightarrow d$和 $ad \rightarrow cd$，这意味着频繁模式和规则的数量大幅减少。

受最大序列模式挖掘的启发，我们提出了最大共现规则（MCoR）的概念，以进一步减少共现规则的数量。例如，如果规则**p**$\rightarrow$ **r** 是一个MCoR，那么我们知道**p** $\rightarrow$ **q** 也是一条共现规则，其中模式**q** 是**r**的前缀模式。此外，如果模式**w** 是**r**的超模式，那么**p** $\rightarrow$ **w**就不是共现规则。从上述示例可以看出，研究MCoR挖掘具有重要意义。本文的主要贡献如下。

1. 为避免挖掘无关模式并获得更好的推荐性能，我们提出了MCoR挖掘，它能够挖掘所有具有相同规则前件的MCoR，并设计了MCoR-Miner算法。2. 在MCoR挖掘中，用户无需设置支持度阈值，因为系统可以根据规则前件的支持度和最小置信度阈值自动计算该阈值。3. 为改进支持度计算，MCoR-Miner由三个部分组成：准备阶段、候选模式生成，以及用于提高效率的筛选策略。4. 为验证MCoR-Miner的性能，我们选取了十一种竞争算法和八个数据集。实验结果证实，MCoR-Miner优于其他竞争算法，且其推荐性能优于频繁序列模式挖掘。

本文结构如下：第二节概述相关工作。第三节定义问题。第四节介绍 MCoR-Miner 算法。第五节报告 MCoR-Miner 的性能。第六节总结全文。

II. 相关工作

序列模式挖掘的目标是在序列数据库中找出满足给定要求 [37] 的子序列（模式）。序列模式挖掘方法常用于数据挖掘，因为其结果直观且易于解释 [38, 39]。传统序列模式挖掘主要关注频繁模式挖掘，即挖掘出的模式具有较高的频率。为减少模式数量，研究人员提出了 top-k 序列模式挖掘 [12], 闭合序列模式挖掘 [40], 和最大序列模式挖掘 [41] ，这些方法都要求数据是静态的，而非动态的。为克服这一缺点，研究人员探索了增量序列模式挖掘 [42] 和窗口序列模式挖掘 [43] 方法。然而，研究界注意到，稀有模式在该领域具有重要意义，因此也提出了稀有模式挖掘 [44]。此外，为了发现缺失事件，研究人员设计了负序列模式挖掘方法，例如 e-NSP [45] 和 e-RNSP [46]。研究人员还设计了一种称为高效用序列模式挖掘的方法 [47, 48]，它能够发现频率较低但效用较高的模式 [49, 50]。

大多数序列模式挖掘方法都可视为经典序列模式挖掘 [51]，因为这些方法只考虑某个模式是否出现在序列中，而重复序列模式挖掘则深入考察模式在序列中的出现次数 [33]。例如，假设有两个序列：$\mathbf{s}_1$=aabbaaba 和$\mathbf{s}_2$=abcbc。根据经典序列模式挖掘，模式 aba 在$\mathbf{s}_1$ 和$\mathbf{s}_2$ 中的支持度为 1，因为模式 aba 出现在序列$\mathbf{s}_1$=aabbaaba 中，但没有出现在序列$\mathbf{s}_2$ 中。因此，经典序列模式挖掘忽略了模式 aba 在序列$\mathbf{s}_1$= aabbaaba 中出现不止一次这一事实，而重复序列模式挖掘则会考虑其出现次数。另一个显著区别是，经典序列模式挖掘侧重于挖掘包含项集的序列中的模式，而重复序列模式挖掘主要旨在挖掘包含项的序列中的模式 [52]。例如，(ab)(bc)(a)(bd)(ad) 是一个包含项集的序列，每个项集都包含多个有序项。如果一个序列中的所有项集都只有一个项，那么该序列就是包含项的序列。因此，包含项的序列可以视为包含项集的序列的一种特殊情况。包含项的序列广泛应用于许多领域，例如 DNA序列、蛋白质序列、点击流和商业数据。

请注意，重复序列模式与情节挖掘 [53] 类似。两者有三点不同。首先，情节挖掘处理一个序列，而重复序列模式处理一个或多个序列。其次，情节挖掘旨在挖掘事件序列中的模式，其中事件序列可表示为 $<(e_1,t_1), (e_2,t_2), \cdot (e_n,t_n)>$，其中 e_i 是一个事件集，而 t_i是 e_i的发生时间。相比之下，在重复序列模式中，序列没有发生时间。第三，在情节挖掘中， e_i 可以是一个集合，而在重复序列模式中，序列由项组成，而非集合。

与经典序列模式挖掘相比，重复序列模式挖掘不仅更具挑战性，而且有多种形式：一般形式（无条件） [33], 不相交形式 [52],单次形式 [54], 和非重叠形式 [55]。请注意，在一些研究中，不相交形式曾被称为非重叠形式 [34, 46], ，这与本文中的非重叠形式有很大不同。为阐明不相交形式与非重叠形式之间的区别，下面给出一个说明性示例。

示例 2 假设我们有一个序列*s=aabbaaba*，以及一个模式 ***p***= *a[0,1]b[0,1]a*。

在不相交形式中 *[34, 46]*, ，某次出现的第一个位置大于其前一次出现的最后一个位置。因此，共有两次出现：*<1,3,5>* 和*<6,7,8>*。请注意， *<1,3,5>* 和 *<2,4,6>* 不满足不相交形式，因为出现 *<2,4,6>2*的第一个位置小于*5*，而*5*是 *<1,3,5>*的最后一个位置。

在非重叠形式中， *[24]*, 每个项不能被同一个 p_j重复使用，但可以被不同的 p_j重复使用。因此，共有三次出现：*<1*、*3*、*5>*、*<2*、*4*、*6>*以及 *<6*、*7*、*8>*。注意， *<2*、*4*、*6>*和 *<6*、*7*、*8>* 满足非重叠形式，因为 p_3 在 *<2*、*4*、*6>*中与 s_6 匹配，而 p_1 在*<6*、*7*、*8>*中与 s_6 匹配。

近年来，人们研究了各种带间隔约束的序列模式挖掘应用。例如，提出了top-k 对比非重叠序列模式挖掘，其中挖掘出的模式可作为序列分类任务的特征 [10]。为了发现频率较低但具有高平均效用的模式，人们开发了高平均效用一次性序列模式挖掘 [49] 和非重叠序列模式挖掘 [56]。为了发现缺失事件，提出了一次性负序列模式挖掘，可用于预测未来的交通流趋势 [13]。受三支决策[19, 57],启发，人们设计了非重叠三支序列模式挖掘 [20] ，以挖掘用户最关注的模式，并能有效挖掘由强兴趣项和中等兴趣项组成的模式。

这些方法大多旨在发现所有满足预定义约束的模式。然而，一般来说，这些方法会发现大量模式。尽管 top-k SPM [10], 非重叠闭合序列模式挖掘 [40],和非重叠最大序列模式挖掘 [41] 可以减少模式数量，但用户可能并不关注所发现的许多模式。然而，这些挖掘方法无法揭示模式之间的关系。规则挖掘是发现模式之间关系的有效方法 [36]。规则挖掘方法有很多，例如关联规则挖掘 [58], 、序列规则挖掘 [36], 、事件规则挖掘[59],和保序规则挖掘 [60]。与模式挖掘类似，也有许多减少冗余规则的方法，例如最大后件和最小前件 [61],、top-k规则挖掘 [58], 、闭合规则挖掘以及最大规则挖掘 [60]。然而，在某些情况下，用户已提前知道一个前缀模式，并希望发现其超模式。这称为共现模式挖掘 [62]。基于共现模式，还可以进一步挖掘共现规则，以揭示前缀模式与其超模式之间的关系。尽管我们可以先挖掘所有模式，然后筛选出具有不同前缀的无用模式，但这种方法会增加运行时间 [63]。

总而言之，尽管使用带间隔约束的序列模式挖掘可以使挖掘结果更有意义，但该方案的挖掘结果缺乏针对性。为了使挖掘结果更适用于推荐，我们根据用户的特殊需求，提出了一种受共现序列模式挖掘 [64] 和最大非重叠顺序模式挖掘[41], 启发的方案，称为 MCoR挖掘。

III. 问题定义

定义 1 （序列）长度为 n 的序列**s** 记作**s** $= s_1 s_2 \cdots s_n$，其中 s_i （$1 \leq i \leq n$） $\in \sum$， $\sum$表示序列**s**中的项集合，$\sum$ 的大小可表示为 $|\sum|$。

定义2 （序列数据库）长度为 k 的序列数据库 D 是一个序列集合，记作 $D = \{s_1, s_2、 \cdots, s_k\}$。

定义 3 （模式）长度为 m 的模式 **p** 记作 **p** $= p_1[a,b]\ p_2 \cdots [a,b]p_m$ （或简写为 **p** $= p_1 p_2 \cdots p_{m-1} p_m$，其中 $gap = [a,b]$），a 和 b （$0 \leq a \leq b$）分别为表示 p_j 与 p_{j+1} 之间通配符数量最小值和最大值的整数。

定义 4 （出现与非重叠出现）假设我们有一个序列 **s** $= s_1\ s_2 \cdots s_n$ 和一个模式 **p**$= p_1[a,b]p_2 \cdots [a,b]p_m$。当且仅当 $p_1 = s_{l_1}$、 $p_2 = s_{l_2}$、 $\cdots$、 $p_m = s_{l_m}$ $0 \leq l_1 \leq l_2 \leq \cdots \leq l_m \leq n$ $a \leq l_i - l_{i-1} - 1$ ( ) 且 时，$\leq b$ $l' = <l'_1\ l'_2$ $l = <l_1$、 l_2、 $\cdots$、 $l_m>$ 是模式 **p** 在序列 **s** 中的一次出现。我们还假设存在另一次出现 、 、 $\cdots$ 、 $l'_m>$。当且仅当对于任意 $(1 \leq j \leq m)$， $l_j \neq l'_j$ 时， l 和 l' 是两次非重叠出现。

示例3 假设有一个序列***s*** =$adbdadcdccabadcd$ 和一个模式 ***p*** $= a[0,3]d$。根据间隔约束 *[0,3],*, ***p*** 在***s*** 中的所有出现为 *<1,2>*,*<1,4>*, *<5,6>*, *<5,8>*, *<11,14>*, *<13,14>* 以及 *<13,16>*。*<1,2>* 和 *<1,4>* 不满足非重叠条件，因为数字*1*在这两次出现中的相同位置上。因此，***s***中***p*** 有四次非重叠出现，分别为 *<1,2>*, *<5,6>*, *<11,14>*以及 *<13, 16>*。

请注意，计算非重叠出现有许多不同的方法，结果也可能有所不同，例如 *NETLAP [65],NETGAP [24], Netback [41], DFOM[56]*。例如，除了两次非重叠出现 *<1,2>* 之外， *<5,6>*, *<1,4>* 和 *<5,8>* 也属于两次非重叠出现。尽管结果可能有所不同，但理论上已证明该问题可在多项式时间内求解 *[65]*。更重要的是，研究表明，寻找非重叠出现有四种不同的方法：按最右侧叶到根的方式寻找极大出现，按最右侧根到叶的方式寻找极大出现，按最左侧叶到根的方式寻找极小出现，以及按最左侧根到叶的方式寻找极小出现 *[66]*。在本例中， *<1,2>* 和*<5,6>* 称为最小非重叠出现 *[56],*, 而 *<1,4>* 和 *<5,8>* 称为极大非重叠出现 *[65]*。本文搜索最小非重叠出现。

定义 5 （支持度）模式 **p** 在序列**s**中的支持度，是非重叠出现的次数，表示为sup(**p**,**s**)。模式 **p** 在序列数据库 D 中的支持度，是各序列中支持度的总和，即 $\sum_{l=1}^{k} sup$ ，i

(**p** D) (**ps**)。

定义 6（频繁模式）如果模式 **p** 在序列 **s** 或 D 中的支持度不低于最小支持度阈值$minsup$，则模式**p** 是频繁模式。

定义 7（前缀模式、子模式和超模式）。给定两个模式 **p** = $p_1p_2 \cdots p_{m-1}p_m$ 和 **q** = $q_1q_2 \cdots q_{n-1}q_n$ $(m < n)$，其中 $gap = [a, b]$，当且仅当 $p_1 = q_1, p_2 = q_2, \cdots$，且 $p_m = q_m$ 时，模式 **p** 是模式 **q** 的前缀模式。此外，模式 **p** 是模式 **q** 的子模式，而模式 **q** 是模式**p** 的超模式。

示例 4 给定两个模式***p*** = ad 和 ***q*** = $adbd$，模式***p*** 是模式 ***q*** 的前缀模式。

定义 8（共现模式、共现规则、规则前件、规则后件和置信度）假设我们有一个模式 **q=p·r** = $p_1p_2 \cdots p_mq_{m+1}q_{m+2} \cdots q_n$，其中**p** = $p_1p_2 \cdots p_m$ 且 **r** = $q_{m+1}q_{m+2} \cdots q_n$。模式**r** 是**p** 的共现模式，而 **p** → **r** 是共现规则，其中**p** 和**r** 分别是规则前件和后件。模式**q** 与**p** 的支持度之比称为序列规则 **p** → **r** 的置信度，记为 *conf*(**p** → **r**) = sup(**p**·**r**,D)/sup(**p**,D)。置信度表示模式**r** 在模式**p** 出现时出现的概率。

定义9（极大共现模式）假设我们有一个模式**q**。如果它的某个超模式**r** 是频繁共现模式，那么模式**q** 就不是极大共现模式；否则，模式**q** 就是极大共现模式。

定义10（强共现规则和 MCoR）如果$conf$(**p** → **r**)大于或等于预定义阈值 $mincf$，即 $conf$(**p** →**r**) ≥ $mincf$，那么**p** →**r** 就是强共现规则。假设**p** → **r** 是强共现规则。对于模式**r**的任意超模式**w**，如果**p**→ **w** 不是强共现规则，那么**p** → **r** 就是 MCoR。

MCoR挖掘： 给定一个序列**s** 或序列数据集 D,$mincf$ 以及前缀模式**p**，MCoR挖掘的目标是发现所有 MCoR。

示例5 从示例*1*可知，$ad \rightarrow c$ 和 $ad \rightarrow cd$ 是两条共现规则。规则 $ad \rightarrow c$ 不是 ***MCoR***，而规则 $ad \rightarrow cd$ 是 ***MCoR***，因为模式 c是 cd 的前缀模式，并且对于 cd的任意超模式（例如 cdc），$ad \rightarrow cdc$都不是强共现规则。

本文中使用的符号如表 I 所示。

表 I：符号说明

符号	描述
s	长度为 n 的序列
D	包含 k 个序列的序列数据库
p	长度为 m的模式
a, b	通配符的最小值和最大值，分别为
sup(**p**，**s**)	**p**在**s**中的出现次数
sup(**p** D)	D 中**p**的出现次数
$\sum$	序列数据库 D 中的项集合
$minsup$	最小支持度阈值
$mincf$	预定义的置信度阈值
p → **r**	一个共现规则
置信度（**p** → **r**）	共现规则**p** →**r**的置信度

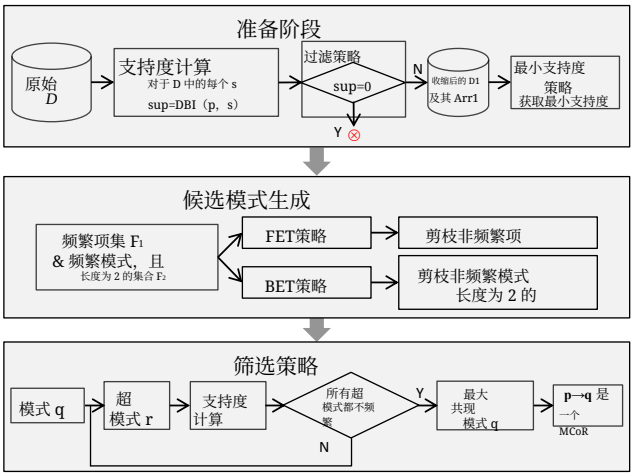


图1：MCoR-Miner 的框架。在准备阶段（见第四节-B），提出一种过滤策略，剪除支持度为零的模式以缩减数据库；随后探索一种最小支持度策略，根据用户定义的参数 $minconf$ 获得 $minsup$。在候选模式生成阶段（见第四节-C），依次提出频繁项枚举树（FET）策略和二项式枚举树（BET）策略，以减少候选模式的数量。最后，提出一种筛选策略（见第四节-D），用于发现极大共现模式，从而避免暴力搜索。

IV. 所提算法

本节提出 MCoR-Miner，这是一种旨在发现所有 MCoR 的算法。MCoR 挖掘基于非重叠序列模式挖掘，其主要问题是支持度计算。

因此，支持度计算是 MCoR 挖掘的关键环节，如第IV-A节所述。除了支持度计算外，MCoR-Miner 还包括三个部分：准备阶段、候选模式生成和筛选策略。MCoR-Miner 的框架如图 1 所示。

A. 支持度计算

给定一个序列和一个带有间隔约束的模式，计算其支持度是一项模式匹配任务 [65]。根据定义4，我们知道，所有非重叠出现都是所有出现的子集，并且这些出现可以用 Nettree 结构表示。Wu等人 [65]首次从理论上证明，计算非重叠出现可以在多项式时间内完成。一些先进算法，例如 NETLAP-Best [65]和 NETGAP[24],，会先创建 Nettree，然后迭代剪枝无用节点，以在 Nettree 上找到所有非重叠出现。NETLAP-Best 和 NETGAP 的时间复杂度均为 $O(m \times m \times n \times w)$，其中 m、n和 w 分别是模式和序列的长度，以及 $b - a + 1$。由于剪除这些无用节点会耗费大量时间，因此提出了 Netback [41]来提升效率。该算法首先创建 Nettree，然后采用回溯策略在 Nettree 上找到所有非重叠出现

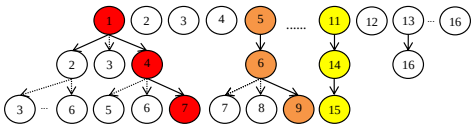


图2：DFOM的出现搜索过程

。NetBack 的时间复杂度降低至$O(m \times n \times w)$。尽管 Nettree 能够直观地表示所有出现，但在表示所有非重叠出现时会包含大量无用信息，例如无用节点和父子关系。为进一步提升效率，研究者提出了 DFOM [56] ；该算法无需创建完整的 Nettree，而是采用深度优先搜索和回溯策略来查找所有非重叠出现。DFOM 的一个缺点是，它采用顺序搜索策略来查找每个当前节点的可行子节点。DFOM 的时间复杂度为 $O(m \times n)$，因为 DFOM 同样采用深度优先搜索和回溯策略，且无需创建完整的 Nettree。为克服这一缺点，我们提出了一种结合索引的深度优先搜索与回溯算法，称为 DBI。该算法采用索引机制来避免顺序搜索。示例6说明了 DFOM 的原理 [56]。

示例*6* 我们使用示例*3*中的同一序列 $s = adbdadcdccabadcd$和模式 $p = a[0,3]d[0,3]c$。图*2*展示了 *DFOM* 的出现搜索过程。我们知道 $p_1 = a$。因此，*DFOM* 按顺序在序列 *s* 中搜索 a 。现在，*DFOM* 创建一个根节点，标记为节点 *1*，因为$s_1 = p_1 = a$。根据深度优先搜索策略，*DFOM* 在节点 *1* 之后，按照间隔约束 $[0,3]$, ，依次在序列 *s* 中搜索 d ，因为 $p_2 = d$。我们知道 $s_2 = d$。因此，*DFOM* 创建节点 *1* 的一个子节点，标记为节点 *2*。由于$p_3 = c$，*DFOM* 按照间隔约束 $[0,3]$，在节点 *2* 之后依次于序列 *s* 中搜索 c 。遗憾的是， s_3 和 s_6之间没有c 。因此，*DFOM* 使用回溯策略回溯到节点 *1*，以寻找新的子节点。我们知道s_3 是 b，它不等于 $p_2 = d$，因此 *DFOM* 继续搜索。我们知道 s_4 是 d ，它等于 p_2= d。因此，*DFOM* 找到了节点 *1* 的一个新子节点，标记为节点 *4*。按照深度优先搜索策略，*DFOM* 在节点 *4* 之后，按照间隔约束 $[0,3]$, ，在序列 *s* 中搜索c ，因为 $p_3 = c$。显然， s_5 和 s_6 不等于 c。由于 $s_7 = c$ 等于 p_3 ，且位置 *7* 和 *4* 之间的间隔为 *2*，因此满足间隔约束 $[0,3]$。于是，*DFOM* 找到节点 *4* 的一个新子节点，标记为节点 *7*。由于 *p* 的长度为 *3*，*DFOM* 找到一个出现 *<1,4,7>*。现在，*DFOM* 继续搜索新的根节点。由于

s_2、 s_3和s_4 不等于 a，这些字符将被忽略。由于 s_5是 a，*DFOM* 找到一个新的根，并将其标记为节点*5*。通过迭代上述过程，*DFOM* 找到一个新的非重叠出现*<5*、*6*、*9>*。最后，找到出现 *<11*、*14*、*15>* 。

从示例6和图2可知，DFOM 的主要缺点是需要顺序搜索。为克服这一不足，DBI 使用索引数组存储每个字符的位置。示例7说明了 DBI 的原理。

a	1	5	11	13		
b	3	12				
c	7	9	10	15		
d	2	4	6	8	14	16

图3：序列**s**的索引数组。

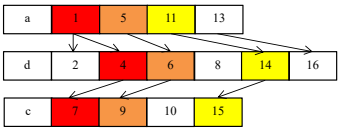


图4：DBI 的出现搜索过程

示例7 我们使用与示例*6*相同的序列$s = adbdadcdccabadcd$和模式$p = a[0,3]d[0,3]c$ 。如图*3*所示，*DBI* 使用索引数组存储每个字符的位置。图*4*展示了 *DBI* 搜索出现的过程。由于 $p_1 = a$，*DBI* 获取数组 a 中的第一个元素，即位置 *1*。因此，*DBI* 创建一个根节点，标记为节点*1*。按照深度优先搜索策略，由于 $p_2 = d$，*DBI* 获取数组 d 中的第一个元素，即*2*；位置*2*和位置*1*之间的间隔为零，满足间隔约束 $[0,3]$。因此，*DBI* 创建节点*1*的一个子节点，标记为节点*2*。按照深度优先搜索策略，由于 $p_3 =c$，*DBI* 获取数组 c 中的第一个元素，即位置*7*。位置*7*和位置*2*之间的间隔为*4*，不满足间隔约束 $[0,3]$。根据回溯策略，*DBI* 回溯到节点*1*以寻找新的子节点。我们知道，数组 d 的第二个元素是位置*4*，满足间隔约束 $[0,3]$。因此，*DBI* 创建节点*1*的一个子节点，标记为节点*4*。现在，*DBI* 选择数组 c的第一个元素，即位置*7*，并且该位置满足间隔约束 $[0,3]$。因此，*DBI* 得到出现 *<1,4,7>*。接着，*DBI* 获取数组 a 的第二个元素，即位置*5*。通过重复上述过程，*DBI* 得到出现*<5,6,9>*。最后，找到 *<11,14,15>* 。

从示例7可知，DBI 首先根据模式**p**的索引数组创建 m 层节点。然后，DBI 基于这些索引数组，采用深度优先搜索和回溯策略，迭代地查找最小非重叠出现。DBI 的伪代码如算法1所示。DBI 的主要步骤如下。

步骤 1：DBI 选择 p_1 的索引数组中的第一个元素作为当前节点，并将当前层设置为第一层（第 1 行至第 3 行）。步骤 2：DBI 获取下一层中第一个未使用的元素作为当前节点的子节点，且当前节点与子节点之间的距离满足间隔约束 gap=$[a,b]$ （第 5 行至第 10 行）。步骤 3：如果 DBI 成功找到子节点，则将该子节点选为当前节点，并将下一层设置为当前层。否则，DBI 回溯到当前节点的父节点，并搜索下一个子节点（第 11 行至第 17 行）。步骤 4：重复步骤 3，直到 DBI 到达第 m 层或

第一层。如果 DBI 到达第 m 层，则 DBI 找到一个出现，即 sup(**p**,**s**)++（第 19 行至第 21 行）。步骤 5：DBI 选择 p_1 的索引数组中的下一个元素作为当前节点，并将当前层设置为第一层。步骤 6：重复步骤 2 至步骤 5，直到检查完 p_1 的索引数组中的所有元素。

算法1DBI

输入： 序列**arr** 的索引数组，模式**p**，以及 gap=$[a,b]$

输出： sup(**p**,**s**)

1: **对于** $root \leftarrow$ 中的每个元素**arr**[**p**[1]] **执行**
2: $CurNode \leftarrow$ **共现**[1] $\leftarrow root$;
3: $CurLevel \leftarrow 1$;
4: **当** $CurLevel > 0$ && $CurLevel < len$(**p**) **执行时**
5: $child \leftarrow$ **arr**[**p**[$CurLevel+1$]]中的下一个元素;
6: $gap \leftarrow child - CurNode - 1$;
7: **当** $gap < a$ **执行时**
8: $child \leftarrow$ **arr**[**p**[$CurLevel+1$]]中的下一个元素;
9: $gap \leftarrow child - CurNode - 1$;
10: **结束循环**
11: **如果** $a \leq gap$ && $gap \leq b$ **则**
12: $CurLevel$++;
13: $CurNode \leftarrow$ **共现**[$CurLevel$] $\leftarrow child$;
14: **else**
15: $CurLevel--$;
16: $CurNode \leftarrow$ **出现次数**[$CurLevel$];
17: **结束 if**
18: **结束 while**
19: **如果** $CurLevel$ = 长度(**p**)**则**
20: sup(**p**，**s**) ++;
21: **结束 if**
22: **结束循环**
23: **返回** sup(**p**,**s**)

定理1 *DBI* 的时间和空间复杂度均为$O(m \times n/h)$，其中 m、 n 和 h 分别是模式和序列的长度，以及 $\sum$的大小。

证明 容易得知，每个项的索引数组大小为 $O(n/h)$，因为序列长度为 n，而项的大小为 h。为了计算支持度，共有m 个索引数组。我们知道，索引数组中的每个元素最多只能使用一次，这意味着 *DBI* 会逐一访问这些元素。因此，*DBI* 的时间和空间复杂度均为 $O(m \times n/h)$。

B. 准备阶段

在本节中，我们提出两种策略：过滤策略和最小支持度策略，以避免一些冗余的支持度计算。我们首先提出过滤策略。

过滤策略。如果模式 **p** 在序列**s** 中的支持度为零，则可以剪枝序列 **s**。

定理 2 过滤策略是正确且完备的。

证明 我们知道，非重叠序列模式挖掘满足反单调性 *[24]*，，这意味着超模式的支持度不大于其子模式的支持度。如果 sup(***p***,***s***)= *0*，那么我们可以断定 sup(***q***,***s***) = *0*，其中模式 ***q*** 是模式 ***p*** 的超模式，因为非重叠序列模式挖掘满足反单调性条件，因此可以剪枝序列 ***s*** 。因此，该过滤策略是正确且完备的。

此外，基于这种反单调性，我们进一步提出了一种最小支持度策略。

最小支持度策略。如果模式**q**在 D 中的支持度小于 $minsup$，则可以剪枝模式 **q** 及其超模式，其中 $minsup$ = sup(**p** D) $\times$ $mincf$。

定理 3 最小支持度策略是正确且完备的。

证明：假设规则 $p \rightarrow r$ 是一条强共现规则，即 $conf$(*p* $\rightarrow$*r*) = sup(*p*·*r* D)/sup(*p* D) $\geq mincf$。那么，sup(*p*·*r* D) $\geq sup$(*p* D) $\times mincf$，这意味着模式 *p*·*r* 的支持度大于或等于 sup(*p* D) $\times mincf$。因此，对于 *MCoR* 挖掘，用户无需设置$minsup$，因为它可以根据$mincf$自动计算得出。因此，最小支持度策略是正确且完备的。

在准备阶段，我们提出了 SDB-Filt 算法，其伪代码如算法 2 所示。SDB-Filt 的主要步骤如下。

步骤 1：在 D 中选择一个序列**s**。步骤 2：SDB-Filt 使用 DBI算法计算sup(**p**, **s**)。步骤 3：如果 sup(**p**,**s**) 为零，则可根据过滤策略收缩序列**s** 。否则，将序列**s**存入收缩后的数据库 $D1$，并更新支持度，即 sup(**p** D) 变为 sup(**p** D) + sup(**p**,**s**) （第 4 至 7 行）。步骤 4：重复步骤 1 至 3，直至计算完所有序列。步骤 5：得到 sup(**p** D) 后，SDB-Filt 根据最小支持度策略计算 $minsup$（第 9 行）。

算法 2SDB-Filt

输入： 序列索引数组 **Arr**、模式 **p**、 gap=$[m,n]$，以及 $mincf$

输出： $minsup$ 以及缩减后的索引数组 $Arr1$

1: sup(**p**,D) $\leftarrow$ 0;
2: **对于** 每个序列 **s** 在 D **执行**
3: $support \leftarrow$ DBI(**数组s,p**, gap);
4: **如果** $support \neq 0$ **则**
5: 将序列索引数组 **s**存入 $Arr1$;
6: sup(**p**,D) $\leftarrow$ sup(**p**,D) $+support$;
7: **结束如果**
8: **结束循环**
9: $minsup \leftarrow sup$（**p** D） $\times mincf$;
10: **返回** $minsup$, $Arr1$

C. 候选模式生成

生成候选模式通常采用两种策略：枚举树策略和模式连接策略。许多非重叠序列模式挖掘方法采用模式连接策略生成候选模式，例如 NOSEP [24] 和 NTP-Miner [20]，因为与枚举树策略相比，该策略能有效减少候选模式。然而，模式连接策略使用所有长度为 m 的频繁模式来生成长度为 $m+1$ 的候选模式，这意味着我们需要挖掘所有频繁模式。显然，要挖掘所有频繁模式，就要使用蛮力算法，然后从中找出模式**q** 及其超模式。因此，我们采用枚举树策略。在经典的枚举树策略中，如果模式**q** 是频繁模式，那么所有模式**q**·a都是候选模式，其中 a 是序列**s**中的任意项。

为减少候选模式的数量，我们首先提出FET策略，然后介绍BET策略。

FET策略： 如果模式**q** 是频繁模式，且 y是序列**s**中的任意频繁项，那么所有模式**q**·y 都是候选模式。

例8展示了FET策略的优势。

例8 假设我们有一个序列$\boldsymbol{s}$ = $s_1s_2s_3s_4s_5s_6$ $s_7s_8s_9s_{10}$ $s_{11}s_{12}s_{13}s_{14}s_{15}s_{16}$ = $adbdadcdccabadcd$、一个模式$\boldsymbol{q}$ = $a[0,3]d[0,3]c$以及 $minsup$ = *3*。根据序列$\boldsymbol{s}$，我们知道所有项都是 $\{a,b,c,d\}$。按照经典的枚举树策略，基于模式$\boldsymbol{q}$有四个候选模式：$a[0,3]d[0,3]c[0,3]a$、$a[0,3]d[0,3]c[0,3]b$、 $a[0,3]d[0,3]c[0,3]c$和 $a[0,3]d[0,3]c[0,3]d$。然而，我们知道模式 a、 b、 c和 d 的支持度分别为四、二、四和六。因此，频繁项为 $\{a,c,d\}$，因为$minsup$ = *3*。于是，根据*FET*，基于模式$\boldsymbol{q}$只有三个候选模式： $a[0,3]d[0,3]c[0,3]a$、$a[0,3]d[0,3]c[0,3]c$和 $a[0,3]d[0,3]c[0,3]d$。这个示例表明，*FET*策略的性能优于经典的枚举树策略。

为进一步减少候选模式的数量，受模式连接策略的启发，我们提出如下 BET 策略。

BET策略： 我们找出所有长度为二的频繁模式，并将其存入集合 F_2。如果 x 是 **q** 的最后一项，且模式 xy 是一个长度为二的频繁模式，即 $xy\in F_2$，那么模式 **r** = **q**·y 是一个候选模式。

例9展示了 BET 策略的优势。

例9 我们使用与例*8*相同的场景。已知频繁项为 $\{a,c,d\}$。然后，我们可以使用模式连接策略生成九个长度为二的候选模式 $\{aa,ac,\ ad,\ ca,\ cc,\ cd,\ da,\ dc,\ dd\}$ 。接着很容易看出，长度为二的频繁候选模式为 <*ad*>, <*ca*>, <*cd*>,<*dc*> 和 <*dd*>。从例*8*可知，基于模式 $\boldsymbol{q}$ 使用*FET*策略只能生成三个候选模式： $a[0,3]d[0,3]c[0,3]a$、$a[0,3]d[0,3]c[0,3]c$ 和$a[0,3]d[0,3]c[0,3]d$。模式 $a[0,3]d[0,3]c[0,3]a$ 无法通过 *BET*策略剪枝，因为模式 $c[0,3]a$ 是一个频繁长度为二的模式，而模式 $a[0,3]d[0,3]c[0,3]c$可以通过*BET*策略剪枝，因为模式 $c[0,3]c$不是长度为二的频繁模式。同样，模式 $a[0,3]d[0,3]c[0,3]d$ 也无法被剪枝。因此，*BET*策略只生成两个候选模式。此示例表明，*BET*策略优于 *FET*策略。图*5*展示了候选模式的比较。

定理4 *FET* 和 *BET* 策略都是正确且完备的。

证明 如上所述，非重叠序列模式挖掘满足反单调性 *[24]*。因此，超模式的支持度不大于其子模式的支持度。因此，如果项 y 是非频繁项，那么模式 $\boldsymbol{q}$·y 就是非频繁模式。因此，*FET*策略是正确且完备的。同理可知，*BET*策略也是正确且完备的。

D. 筛选策略

显然，我们可以根据频繁共现模式轻松获得强共现序列规则。此外，我们可以根据最大共现模式发现极大强共现规则。最简单的方法是挖掘所有频繁共现模式，然后从中找出最大共现模式。这种方法可视为一种蛮力法。为了提高性能，我们提出一种筛选策略来发现最大共现模式。其原理如下。

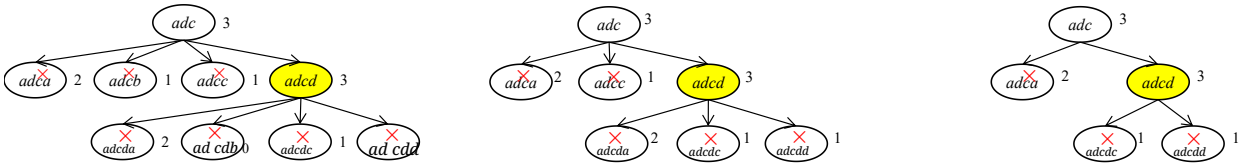
筛选策略： 假设我们有一个模式**q**。如果它的某个超模式**r** 是频繁共现模式，那么模式**q** 就不是最大共现模式；否则，**q**就是最大共现模式。

定理5 筛选策略是正确且完备的。

证明根据极大共现模式的定义，对于模式$\boldsymbol{q}$，如果它的某个超模式$\boldsymbol{r}$ 是频繁共现模式，那么模式$\boldsymbol{q}$ 就不是最大共现模式；否则，模式$\boldsymbol{q}$就是最大共现模式。筛选策略依据极大共现模式的定义来发现最大共现模式。因此，筛选策略是正确且完备的。

我们将所有频繁共现模式存储在一个栈中。假设模式**q** 是栈顶元素。我们使用BET策略基于模式**q** 生成候选模式，然后使用DBI算法计算候选模式的支持度。如果候选模式**r** 是频繁共现模式，则将模式**r** 存入栈中；否则，模式**q** 是极大共现模式，并将其存入集合 M_p。我们检查栈中的所有模式，直到栈为空。示例10说明了筛选策略的原理。

示例10 假设我们有一个序列 $\boldsymbol{s}$ = $s_1s_2s_3s_4$ $s_5s_6s_7s_8s_9s_{10}s_{11}s_{12}s_{13}s_{14}s_{15}s_{16}$ = $adbdadcdccabadcd$, $\boldsymbol{p}$ = $a[0,\ 3]d$, *and* $minsup$ = *3*. 假设栈顶元素是



（a）所有项的枚举树 （b）FET的枚举树 (c) BET的枚举树

图5：不同策略生成的候选模式比较。使用所有项生成了八个候选模式，而使用FET和BET策略分别生成了六个和四个候选模式。

模式 $\boldsymbol{q} = a[0,3]d[0,3]c$。根据 *BET*策略，基于模式 $\boldsymbol{q}$有两个候选模式： $a[0,3]d[0,3]c[0,3]a$ 和 $a[0,3]d[0,3]c[0,3]d$。我们知道，模式 $a[0,3]d[0,3]c[0,3]a$ 的支持度为二。这意味着 $a[0,3]d[0,3]c[0,3]a$ 不能存储在栈中，因为它不是频繁共现模式。此外，我们知道，模式 $a[0,3]d[0,3]c[0,3]d$ 的支持度为三。因此，模式 $a[0,3]d[0,3]c[0,3]d$存储在栈中，因为它是频繁共现模式。同时，模式 $\boldsymbol{q}$ 不是极大共现模式，因为它有一个频繁共现超模式 $a[0,3]d[0,3]c[0,3]d$。现在，栈顶元素是模式 $a[0,3]d[0,3]c[0,3]d$，根据 *BET*策略，它有两个候选模式: $a[0,3]d[0,3]c[0,3]d[0,3]c$ 和 $a[0,3]d[0,3]c[0,3]d[0,3]d$。很容易看出，模式$a[0,3]d[0,3]c[0,3]d[0,3]c$和 $a[0,3]d[0,3]c[0,3]d[0,3]d$ 的支持度都为一。因此，模式 $a[0,3]d[0,3]c[0,3]d$ 是极大共现模式，因为它的所有超模式都不是频繁共现模式。筛选策略的优点在于，无需进行暴力搜索即可发现极大共现模式。

E. MCoR-Miner

我们介绍MCoR-Miner算法，其伪代码如算法3所示。MCoR-Miner主要包含以下步骤。

步骤 1：使用 SDB-Filt 计算模式 **p** 的支持度，即 sup(**p** D)；剪枝支持度为零的序列 **s** ；并计算 $minsup = sup$(**p** D) $\times\ mincf$（第 1 至 2 行）。步骤 2：遍历 $D1$ 以查找所有频繁项，并将其存储在 F_1 中（第 3 行）。步骤 3：使用模式连接策略生成候选模式集 C_2 （第 4 行）。步骤 4：计算集合 C_2 中每个模式的支持度，并将频繁模式存储在 F_2 中（第 5 行）。步骤 5：将模式 **p** 存储在栈 T 中（第 6 行）。步骤 6：获取栈 T 的栈顶元素模式 **q** （第 8 行）。步骤 7：使用频繁项 F_1 生成模式 **q** 的所有候选共现模式，并将其存储在集合C 中（第 10 至 13 行）。步骤 8：使用 BET 对候选模式进行剪枝（第 14 行）。步骤 9：使用 DBI算法计算 C 中每个模式 **r**的支持度（第 15 行）。步骤 10：如果模式 **r** 是频繁共现模式，则将模式 **r** 存储在栈 T 中（第 16 至 19 行）。步骤 11：如果所有模式 **r** 都不是频繁共现模式，则模式 **q** 是极大共现模式，并将其存储在集合 M_p 中（第 22 至 24 行）。步骤 12：重复步骤 6 至 11，直到栈为空。步骤 13：基于 M_p生成极大强规则。

算法 3MCoR-Miner

输入：序列数据库 D，最小置信度 $mincf$, gap =$[a,b]$，前缀模式 p 输出：极大共现序列规则集 M_r1：扫描 D 以获取索引数组 Arr；//使用过滤策略缩减数据集，并使用最小支持度策略进行计算 $minsup$2： $minsup$,$Arr1 \leftarrow$ SDB-Filt(Arr,p,gap, $mincf$)； 3：扫描 $Arr1$ ，在 F_1中查找并存储频繁项； 4：生成长度为二的候选模式，并将其存储在C_2中； 5：发现长度为二的频繁模式，并将其存储在 F_2中； 6：将模式 p 存入栈 T； 7：当 T 不为空时执行 8：q $\leftarrow T$.pop()； 9：flag $\leftarrow$True；//使用FET策略剪枝候选模式 10：对于 F_1 中的每个项 y 11：$C \leftarrow C \cup$ q·y； 12：$x \leftarrow$ q的最后一项； 13：r $\leftarrow$q·y；//使用BET策略剪枝候选模式 14：如果模式 x·y 存在于 F_2 中 15：$support \leftarrow$ DBI($Arr1$,r, gap)； 16：如果 $support \geq minsup$ 17：T.push (r)； 18：flag $\leftarrow$ False； 19：结束如果 20：结束如果 21：结束循环 //使用筛选策略挖掘模式 22：如果 flag= True 23：$M_p \leftarrow M_p \cup$q； 24：结束如果 25：结束循环 26：基于 M_p生成极大强规则； 27：返回 M_r;

定理6 *MCoR-Miner* 的空间复杂度和时间复杂度分别为 $O(N)$和 $O(t \times M \times N/h)$，其中 t、 M、 N 和 h分别是候选模式的数量、候选模式的最大长度、缩减后的序列数据库的总长度以及 $\sum$的大小。

证明 扫描序列数据库 D 以获取其索引数组的时间复杂度和空间复杂度为 $O(N)$。显然，*SDB-Filt* 的空间复杂度为$O(N)$。*SDB-Filt* 的时间复杂度为 $O(m \times N/h)$，因为 *DBI* 的时间复杂度为 $O(m \times n/h)$，且 *SDB-Filt* 会检查所有序列，其中 $N = \sum_{i=1}^{k} n_i$。 F_1 的空间复杂度为 $O(h)$，因为所有频繁项的数量不超过所有项的数量。因此，第*3*行的时间复杂度为 $O(h)$。第*4*行的时间复杂度和空间复杂度为$O(h^2)$。第*5*行的时间复杂度为 $O(h^2 \times 2 \times N/h)$，因为有 $O(h^2)$ 个候选模式，且每个候选模式的长度为*2*。存储长度为*2*的频繁模式的空间复杂度为 $O(h^2 \times 2)$。假设 *MCoR-Miner* 检查 t 个候选模式，且候选模式的最大长度为 M。存储这些候选模式的空间复杂度为 $O(t \times M)$，计算这些候选模式支持度的时间复杂度为$O(t \times M \times N/h)$。因此，*MCoR-Miner* 的空间复杂度为 $O(N + N + h + h^2 + k \times M)$ =$O(N)$，因为通常 h、h^2、 k、 M 和 $k \times M$远小于 N。*MCoR-Miner* 的时间复杂度为 $O(N + m \times N/h + h^2 + h^2 \times 2 \times N/h + t \times M \times N/h)$=$O(t \times M \times N/h)$，因为 m 和$2 \times h^2$ 分别远小于 M和 t。

V. 实验结果与分析

为验证 MCoR-Miner 的性能，我们考虑以下 w八个研 s究问 uestions (RQs) 题。RQ1：在挖掘共现模式方面，MCoR-Miner 是否比其他最先进的算法具有更好的性能？RQ2：在支持度计算方面，DBI 是否比其他最先进的算法具有更好的性能？RQ3：过滤策略能否减少 MCoR-Miner 中支持度计算的次数？RQ4：FET 和 BET 策略能否减少候选模式的数量？BET 在何时能实现更好的性能？RQ5：筛选策略有何影响？RQ6：MCoR-Miner 是否具有更好的可扩展性？RQ7：不同参数如何影响 MCoR-Miner 的性能？RQ8：MCoR-Miner 在推荐任务中的性能如何？

为回答 RQ1，我们选取 NOSEP、DFOM-All 和 DBI-All，以验证 MCoR-Miner 在共现模式挖掘问题上的性能。针对 RQ2，我们提出 NETGAP-MCoR、Netback-MCoR 和 DFOM-MCoR，以探究 DBI 的影响，如第五节 B 部分所述。为解决 RQ3，我们提出 MCoR-NoFilt，以研究过滤策略的影响，如第五节 B 部分所述。为回答 RQ4，我们采用两种算法 MCoR-AET 和 MCoR-FET，以验证 FET 和 BET 策略的影响，如第五节 B 部分所述。为解决 RQ5，我们提出 MCoR-NoScr，以验证第五节 B 部分中筛选策略的影响。为回答 RQ6，我们探究 MCoR-Miner 的可扩展性，如第五节 C 部分所述。为解决 RQ7，我们探究不同间隔约束和最小置信度阈值对第五节 D 部分中 MCoR-Miner 运行时间的影响。为回答 RQ8，我们在第五节 E 部分报告 MCoR-Miner 在置信度、召回率和 F1分数方面的性能。所有实验均在一台配备 Intel(R) Core(TM)i7-10870H 处理器、32 GB 内存并运行 Windows 10 操作系统的计算机上进行，实验环境为 VS2022。所有算法和数据集均可从 https://github.com/wuc567/Pattern-Mining/tree/master/MCoR-Miner 下载。

A. 基准数据集和基线算法

我们使用表II所示的八个数据集作为测试序列。

表II：数据集描述

数据集	Type	∑	序列数量	长度
GAMESALE[1]	商业	12	39	16,446
BABYSALE[2]	商业	6	949	29,971
交易	商业	31	4,661	32,851
电影[4]	商业	20	4,988	741,132
MSNBC[5]	点击流	17	200,000	948,674
SARS[6]	生物序列	4	426	29,751
SARS-CoV-2[7]	生物序列	4	428	29,903
HIV[8]	生物序列	20	1	10,000

[3]

1：GAMESALE 是电子游戏销售数据集，包含评分，可从 http://www.kaggle.com/rush4ratio/video-game-sales-with-ratings 下载。2：BABYSALE 是婴幼儿产品销售数据集，可从 https://tianchi.aliyun.com/dataset/dataDetail?dataId=45 下载。3：TRANSACTION 是销售行为数据集，可从 https://www.kaggle.com/retailrocket/ecommerce-dataset 下载。4：电影包含用户的电影评分，可从 http://grouplens.org/datasets/ 下载。5：MSNBC 是点击流序列数据库，可从 http://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php 下载。6：SARS 包含导致严重急性呼吸综合征的病毒的基因序列（该病毒于 2003年引发疫情），可从 https://www.ncbi.nlm.nih.gov/nuccore/30271926 下载。7：SARS-COV-2包含导致 COVID-19 的病毒基因序列，可从 https://www.ncbi.nlm.nih.gov/nuccore/MN908947 下载。8：HIV 可从 http://archive.ics.uci.edu/ml/machine-learning-databases/00330/ 下载。

为报告MCoR-Miner算法的性能，我们设计了四项实验，并选取了九种具有竞争力的算法。

1. NOSEP [24], DFOM-All [56], 和 DBI-All：为验证是否有必要使用专门的算法来挖掘频繁共现模式，我们选取了两种最先进的算法：NOSEP [24] 和 DFOM [56]，，其中 NOSEP 可以挖掘所有频繁模式。DFOM-All 和 DBI-All 也可以发现所有频繁模式，它们分别采用 DFOM 和 DBI 算法计算支持度。随后，我们可以从所有频繁模式中找出频繁共现模式。根据上述分析，我们知道 NOSEP 、 DFOM-All 和 DBI-All 的时间复杂度分别为 $O(t_a \times M \times M \times N \times w)$、 $O(t_a \times M \times N)$ 和 $O(t_a \times M \times N/h)$，其中 t_a ($t_a \geq t$) 和 w 分别表示所有频繁模式的数量和 $b - a + 1$。

2. NETGAP-MCoR [24], Netback-MCoR [41], 和 DFOM-MCoR [56]: 为探究 DBI 的运行性能，我们提出了三种竞争算法：NETGAP-MCoR、Netback-MCoR 和 DFOM-MCoR。这三种竞争算法分别采用了三种最先进的算法 NETGAP [24], Netback [41], 和 DFOM [56] 来

分别计算支持度。过滤、FET、BET 和筛选等其他策略与 MCoR-Miner 相同。基于上述分析，我们得知 NETGAP-MCoR、Netback-MCoR 和 DFOM-MCoR 的时间复杂度分别为 $O(t \times M \times M \times N \times w)$、$O(t \times M \times N \times w)$ 和 $O(t \times M \times N)$。

3. MCoR-NoFilt：为展示过滤策略的性能，我们提出了 MCoR-NoFilt。它不采用过滤策略，但其他方面与 MCoR-Miner 相同。MCoR-NoFilt 的时间复杂度为 $O(t \times M \times N_o/h)$，而 MCoR-Miner 的时间复杂度为$O(t \times M \times N/h)$，其中 N_o 和 N ($N_o \geq N$)分别是原始序列数据库和缩减后的序列数据库的长度。

4. MCoR-AET 和 MCoR-FET：为研究 BET策略的性能，我们提出了 MCoR-AET 和 MCoR-FET，分别使用所有项和频繁项生成候选模式。MCoR-AET 和 MCoR-FET 的时间复杂度分别为 $O(t_a \times M \times N/h)$ 和 $O(t_f \times M \times N/h)$，其中 t_a ($t_a \geq t$) 和 t_f 分别是候选模式的数量。请注意，t_f可能小于 t，因为 MCoR-Miner 需要计算长度为二的$O(h^2)$个候选模式，而且在某些情况下，它无法使用 BET策略剪枝 $O(h^2)$个候选模式。

5. MCoR-NoScr：为评估筛选策略的性能，我们使用了不包含筛选策略的 MCoR-NoScr。MCoR-NoScr 的时间复杂度与 MCoR-Miner 几乎相同。

6. CoP-Miner：为验证推荐性能，我们提出了 CoP-Miner，它能够挖掘具有预定义前缀模式的所有频繁模式，并采用与 MCoR-Miner 相同的策略。CoP-Miner 的时间复杂度为 $O(t_c \times M \times N/h)$，其中 t_c ($t_c \geq t$) 表示所有频繁模式的数量。

B. 效率

为确定共现模式挖掘的必要性，我们选取 NOSEP、DFOM-All 和 DBI-All 作为竞争算法。此外，为验证 DBI算法以及过滤、BET 和筛选策略的性能，我们采用 NETGAP-MCoR、Netback-MCoR、DFOM-MCoR、MCoR-NoFilt、MCoR-AET、MCoR-FET 和 MCoR-NoScr 作为竞争算法。我们选取了八个数据集。由于这些数据集的特征差异显著，例如字符数和序列数不同，因此我们设置了不同的参数，如表 III 所示。运行时间和内存使用量的对比如表IV和表V所示。挖掘结果的主要指标（例如 CoR 和 MCoR 的数量）如表 VI 所示。

这些结果引出了以下观察。

1. 最大共现规则挖掘可以有效减少规则数量。从表 VI 可以看出，在 BABYSALE 上，共有 98 条共现规则，而极大共现规则有 47 条。所有其他数据集上也都能发现相同的现象。这是因为，如示例

表 III：参数

数据集	前缀	间隔约束	最小置信度
GAMESALE	d	[0,8]	0.3
BABYSALE	a	[0,8]	0.2
交易	z	[0,3]	0.3
电影	d	[0,3]	0.5
MSNBC	e	[0,8]	0.3
SARS	C	[0,3]	0.6
SARS-CoV-2	C	[0,3]	0.6
HIV	C	[0,3]	0.3

5 所示，$ad \rightarrow c$ 和 $ad \rightarrow cd$ 都是规则，而 $ad \rightarrow cd$ 是极大规则，规则 $ad \rightarrow c$则不是极大规则。这意味着 MCoR 挖掘可以有效减少规则数量。

2. 有必要探索共现模式挖掘，因为 MCoR-Miner 的速度明显快于 NOSEP [24]和 DFOM-All [56], , 尤其是快于 DBI-All。例如，在 GAMESALE 上，NOSEP 和 DFOM-All 分别耗时 47.95 秒和 2.47 秒，而 MCoR-Miner 耗时 0.42 秒，这意味着 MCoR-Miner 的速度约为 NOSEP 的 100 倍，是 DFOM-All 的 6 倍。此外，DBI-All 耗时 0.92 秒，慢于 MCoR-Miner。在其他数据集上也能观察到这些现象。原因有两个。首先，与 NOSEP 和 DFOM-All 相比，MCoR-Miner 采用了一种名为 DBI 的高效算法来计算支持度。如第IV-A节所述，NOSEP 必须构建完整的 Nettree 来计算支持度。然而，Nettree 包含许多冗余节点和父子关系。尽管 DFOM 克服了这一缺点，但它采用顺序搜索策略来寻找当前节点的可行子节点，这种方法效率低下。相比之下，DBI 配备了索引机制，可以避免顺序搜索，因此 MCoR-Miner 的速度明显快于 NOSEP 和 DFOM-All。其次，尽管 DBI-All 和 MCoR-Miner 都采用 DBI 来计算支持度，但 DBI-All 的运行速度慢于 MCoR-Miner。这是因为这三种算法都必须发现所有模式，而其中许多并非目标模式，因此三种算法都必须筛除这些无用模式。因此，实验结果表明，有必要探索共现模式挖掘。

3. 在所有数据集上，MCoR-Miner 的运行速度均快于 NETGAP-MCoR、Netback-MCoR 和 DFOM-MCoR，这表明 DBI 的效率较高。例如，从表 IV 可以看出，在 BABYSALE 上，MCoR-Miner 耗时 5.36 秒，而 NETGAP-MCoR、Netback-MCoR 和 DFOM-MCoR 分别耗时 113.00 秒、112.17 秒和 11.34 秒。因此，MCoR-Miner 显然快于 NETGAP-MCoR 和 Netback-MCoR，速度约为 DFOM-MCoR 的两倍。原因如下：这四种算法采用不同的方法计算支持度。NETGAP-MCoR 和 Netback-MCoR 必须构建完整的 Nettree，其中包含许多无用节点和父子关系。因此，它们的时间复杂度高于 DFOM-MCoR 和 MCoR-Miner。尽管 DFOM-MCoR 不会构建

表IV：运行时间（秒）对比

	GAMESALE	BABYSALE	交易	电影	MSNBC	SARS	SARS-CoV-2	HIV
NOSEP	47.95	152.55	10.20	77.66	-	95.47	133.12	211.98
DFOM-All	2.47	12.33	2.11	5.78	-	9.07	10.00	84.28
DBI-All	0.92	8.38	0.45	4.13	15.72	6.56	8.28	1.48
NETGAP-MCoR	16.17	113.00	2.80	50.16	587.91	2.80	2.75	7.74
Netback-MCoR	15.89	112.17	2.70	46.53	582.67	2.58	2.60	7.78
DFOM-MCoR	1.88	11.34	0.70	5.25	97.38	3.60	3.13	5.11
MCoR-NoFilt	0.66	5.53	1.00	4.08	215.83	2.81	2.97	0.13
MCoR-AET	0.70	7.14	3.88	16.22	25.92	1.56	1.42	**0.02**
MCoR-FET	0.59	5.94	0.47	5.25	24.17	**1.55**	**1.41**	**0.02**
MCoR-NoScr	0.63	5.80	0.48	4.11	16.52	3.13	2.97	0.13
MCoR-Miner	**0.42**	**5.36**	**0.38**	**4.00**	**16.08**	2.34	2.66	0.11

表V：内存使用量（Mb）

	GAMESALE	BABYSALE	交易	电影	MSNBC	SARS	SARS-CoV-2	HIV
NOSEP	32.1	203.3	908.1	1092.5	1183.2	106.2	106.5	19.4
DFOM-All	22.0	195.7	903.7	971.2	1188.0	95.1	95.4	20.2
DBI-All	14.4	16.9	21.6	38.5	230.7	17.6	17.6	14.1
NETGAP-MCoR	23.7	25.8	24.8	163.1	404.9	25.2	25.2	19.6
Netback-MCoR	26.7	18.8	35.7	187.2	263.9	27.8	27.4	20.3
DFOM-MCoR	15.9	18.7	22.8	40.0	226.0	15.8	15.9	15.9
MCoR-NoFilt	16.7	19.6	25.0	41.1	445.8	17.6	24.3	23.0
MCoR-AET	16.0	18.8	24.5	43.9	232.4	17.2	17.2	16.0
MCoR-FET	24.0	26.4	30.5	47.8	238.80	**14.9**	**14.8**	**13.4**
MCoR-NoScr	20.8	32.1	30.3	47.4	239.0	24.3	16.9	15.8
MCoR-Miner	**13.6**	**16.5**	**20.8**	**38.0**	**228.8**	17.4	26.6	21.5

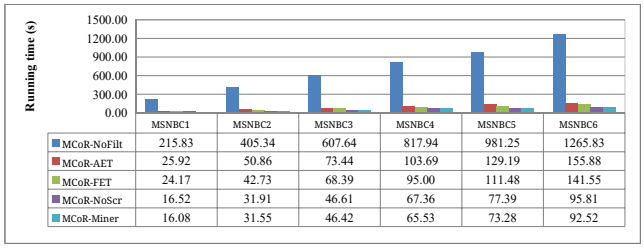
表 VI：挖掘结果的主要指标

	GAMESALE	BABYSALE	交易	电影	MSNBC	SARS	SARS-CoV-2	HIV
CoR 数量	36	98	4	4	12	2	2	3
MCoR 数量	20	47	2	1	1	2	2	3
筛选后的序列数量	642	1,350	104,787	288	49,163,184	28	28	0
未筛选的序列数量	11,877	425,700	21,060	79,520	1,236,816	11,900	11,956	472
所有项的数量	12	6	30	20	17	4	4	20
频繁项的数量	10	5	4	3	15	4	4	20
BET项数量	27	15	4	2	1	13	13	286
未过滤情况下的BET项数量	27	15	6	2	24	13	13	286
被BET剪枝的候选模式数量	149	70	9	8	168	0	0	8
未被BET剪枝的候选模式数量	221	425	11	7	27	12	12	72

完整的 Nettree，但 DFOM 采用顺序搜索策略来查找当前节点的可行子节点，效率较低。因此，MCoR-Miner 的表现优于 NETGAP-MCoR、Netback-MCoR 和 DFOM-MCoR。

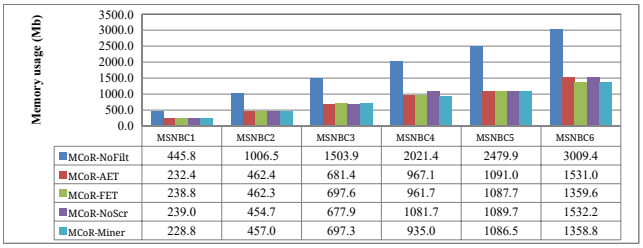
4. 在所有数据集上，MCoR-Miner 的运行速度都快于 MCoR-NoFilt，这表明过滤策略是有效的。根据表IV，在 BABYSALE 上，MCoR-Miner 耗时 5.36 秒，而 MCoR-NoFilt 耗时 5.53 秒。所有其他数据集也呈现出这一现象。其原因在于一些无用序列被过滤掉了。例如，从表 VI 可以看出，在挖掘所有规则时，有 1350 个序列被过滤掉，这意味着无需计算这 1350 个序列的支持度，因此提升了运行性能。过滤策略不仅能减少序列数量，还能减少长度为二的频繁模式数量，这意味着它能够减少长度大于二的候选模式数量。例如，在 TRANSACTION 数据集上，过滤策略下只有四个模式，而非过滤策略下有六个模式，这意味着发现所有规则需要更多时间。总之，过滤策略是有效的，MCoR-Miner 的性能优于 MCoR-NoFilt。

5. MCoR-Miner 的运行速度快于 MCoR-AET 和 MCoR-FET 在 GAMESALE、BABYSALE、交易、电影和 MSNBC 上更快，但在 SARS、SARS-CoV-2 和 HIV 上慢于 MCoR-AET 和 MCoR-FET，这表明 BET策略具有一定的优势和局限性。我们选择两个典型数据集来说明原因。1）以 GAMESALE 为例，我们发现其中有 12 个项和 10 个频繁项。因此，长度为二的候选模式有 $10\times10=100$ 个。检查这 100 个候选模式后，我们得到 27 个长度为二的频繁模式。利用这 27 个模式，BET策略剪枝了 149 个候选模式；根据表 VI，另有 221 个候选模式未被剪枝。因此，在增加 100 次针对长度为二的候选模式的支持度计算后，MCoR-Miner 可以避免对候选模式进行 149 次支持度计算。因此，MCoR-Miner 能够提升其在 GAMESALE 上的运行性能。2）再以 HIV 为例，我们发现其中有 20 个频繁项。因此，长度为二的候选模式有 $20\times20=400$个。检查这 400 个候选模式后，我们得到 286 个长度为二的频繁模式。在这 286 个模式中，BET策略只剪枝了 8 个候选模式；根据表 VI，另有 72 个未被剪枝。因此，在增加 400 次针对长度为二的候选模式的支持度计算后，MCoR-Miner 可以



	MSNBC1	MSNBC2	MSNBC3	MSNBC4	MSNBC5	MSNBC6
MCoR-NoFilt	215.83	405.34	607.64	817.94	981.25	1265.83
MCoR-AET	25.92	50.86	73.44	103.69	129.19	155.88
MCoR-FET	24.17	42.73	68.39	95.00	111.48	141.55
MCoR-NoScr	16.52	31.91	46.61	67.36	77.39	95.81
MCoR-Miner	16.08	31.55	46.42	65.53	73.28	92.52

图6：不同数据集大小下运行时间的比较（秒）



	MSNBC1	MSNBC2	MSNBC3	MSNBC4	MSNBC5	MSNBC6
MCoR-NoFilt	445.8	1006.5	1503.9	2021.4	2479.9	3009.4
MCoR-AET	232.4	462.4	681.4	967.1	1091.0	1531.0
MCoR-FET	238.8	462.3	697.6	961.7	1087.7	1359.6
MCoR-NoScr	239.0	454.7	677.9	1081.7	1089.7	1532.2
MCoR-Miner	228.8	457.0	697.3	935.0	1086.5	1358.8

图7：不同数据集大小下内存使用量的比较（Mb）

仅避免对候选模式进行 8 次支持度计算。因此，运行时间有所增加。由此，实验结果表明，BET策略在生物序列数据集上存在一些局限性。因此，我们的方法 MCoR-Miner 或 MCoR-FET 在所有情况下都能获得更好的性能。

6. 在所有数据集上，MCoR-Miner 的运行速度都快于 MCoR-NoScr，这表明筛选策略是有效的。例如，在 TRANSACTION 数据集上，MCoR-Miner 需要 0.38 秒，而 MCoR-NoScr 需要 0.48 秒。在其他所有数据集上也都能观察到这一现象。原因如下：这两种算法的区别在于，MCoR-Miner 配备了筛选策略，而 MCoR-NoScr 没有。因此，结果表明，筛选策略能够有效缩短运行时间，从而使 MCoR-Miner 的表现优于 MCoR-NoScr。

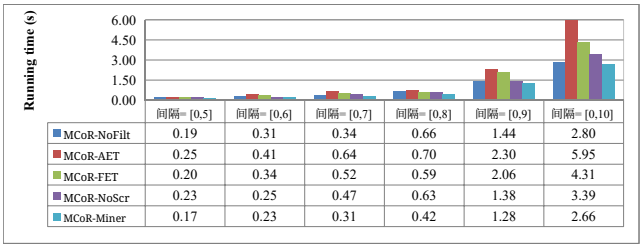
7. 从表V可以看出，MCoR-Miner 在挖掘模式时消耗的内存较少。例如，在 GAME-SALE 数据集上，MCoR-Miner 消耗 13.6Mb 内存，少于其他竞争算法。在大多数数据集上也能观察到这一结果。尤其是，MCoR-Miner 的内存消耗少于 NOSEP 和 DFOM-All 算法。这是因为，为了计算支持度，NOSEP 需要创建一个完整的 Nettree，其中包含大量无用信息。尽管 DFOM 克服了这一缺点，但它使用顺序搜索策略来查找当前节点的可行子节点，效率不高。然而，MCoR-Miner 使用 DBI 算法，该算法配备了索引机制，可避免顺序搜索。因此，MCoR-Miner 消耗的内存较少。

总之，与其他竞争算法相比，MCoR-Miner 的运行性能更好，内存消耗也更低。

C. 可扩展性

为了验证 MCoR-Miner 的可扩展性，我们选择 MSNBC 作为实验数据集，因为它是表II中最大的 数据集。此外，我们创建了 MSNBC1、MSNBC2、MSNBC3、MSNBC4、MSNBC5 和 MSNBC6，其规模分别是 MSNBC 的一、二、三、四、五和六倍。我们设置了 $gap = [0, 8]$, $mincf = 0.3$，前缀模式为 e。运行时间和内存使用量的比较分别如图6和图7所示。

结果引出了以下几点观察。



	间隔= [0,5]	间隔= [0,6]	间隔= [0,7]	间隔= [0,8]	间隔= [0,9]	间隔= [0,10]
MCoR-NoFilt	0.19	0.31	0.34	0.66	1.44	2.80
MCoR-AET	0.25	0.41	0.64	0.70	2.30	5.95
MCoR-FET	0.20	0.34	0.52	0.59	2.06	4.31
MCoR-NoScr	0.23	0.25	0.47	0.63	1.38	3.39
MCoR-Miner	0.17	0.23	0.31	0.42	1.28	2.66

图8：不同 gap(s)下运行时间的比较

从图6和图7可以看出，随着数据集大小增加，MCoR-Miner 的运行时间和内存使用量均呈线性增长。例如，MSNBC6 的大小是 MSNBC1 的六倍。MCoR-Miner 在 MSNBC1 上耗时 16.08 秒，在 MSNBC6 上耗时 92.52 秒，因此 92.52/16.08=5.75。MCoR-Miner 在 MSNBC6 上的内存使用量为 1358.8 Mb，在 MSNBC1 上为 228.8 Mb，因此 1358.8/228.8 = 5.94。由此可见，运行时间和内存使用量的增长率略低于数据集大小的增幅。所有其他数据集也都呈现出这一现象。这些结果表明，MCoR-Miner 的时间和空间复杂度与数据集大小正相关。更重要的是，我们可以看到，MCoR-Miner 的速度比 MCoR-NoFilt 快十倍以上，而 MCoR-NoFilt 的内存使用量是 MCoR-Miner 的两倍以上。因此，我们得出结论：MCoR-Miner 具有很强的可扩展性，因为其挖掘性能不会随着数据集大小的增加而下降。

D. 参数影响

我们评估了不同间隔约束和最小置信度对规则数量及运行时间的影响。

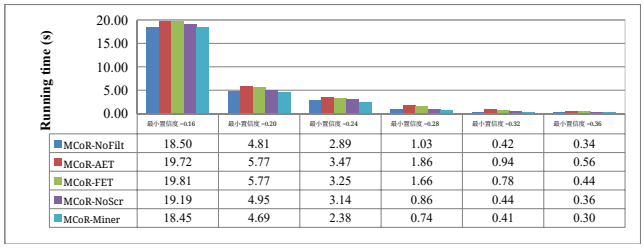
1) 不同间隔约束的影响：为了分析不同间隔约束对 MCoR-Miner 的规则数量和运行时间的影响，我们选取 GAMESALE 作为实验数据集，并将 MCoR-NoFilt、MCoR-AET、MCoR-FET 和 MCoR-NoScr 作为竞争算法。前缀模式为 d，最小置信度为0.3。间隔约束为 [0,5], [0,6], [0,7], [0,8], [0,9],和[0,10]。运行时间比较如图8所示，挖掘出的 CoR 和 MCoR 数量如表 VII 所示。

这些结果引出了以下观察。

随着间隔增大，CoR 和 MCoR 的数量以及运行时间都会增加。例如，当间隔=[0,8]时，

表VII：不同间隔下 CoR 和 MCoR 数量的比较

	gap=[0,5]	gap=[0,6]	gap=[0,7]	gap=[0,8]	gap=[0,9]	gap=[0,10]
CoR 数量	8	14	20	36	63	134
MCoR 数量	6	10	12	20	34	63



	最小置信度 =0.16	最小置信度 =0.20	最小置信度 =0.24	最小置信度 =0.28	最小置信度 =0.32	最小置信度 =0.36
MCoR-NoFilt	18.50	4.81	2.89	1.03	0.42	0.34
MCoR-AET	19.72	5.77	3.47	1.86	0.94	0.56
MCoR-FET	19.81	5.77	3.25	1.66	0.78	0.44
MCoR-NoScr	19.19	4.95	3.14	0.86	0.44	0.36
MCoR-Miner	18.45	4.69	2.38	0.74	0.41	0.30

图 9：不同 $mincf$ (s) 下运行时间的比较

MCoR-Miner 耗时 0.17 秒，挖掘出 8 个 CoR 和 6 个 MCoR；而当间隔=[0,8]时，MCoR-Miner 耗时 0.42 秒，挖掘出 36 个 CoR 和 20 个 MCoR。其他所有算法也都能观察到这一现象。原因如下：我们知道，随着间隔增大，模式的支持度也会提高。因此，可以找到更多模式，而每个共现模式都对应一个 CoR。因此，可以找到更多 CoR 和 MCoR，运行时间也会增加。对于任意间隔约束，MCoR-Miner 的表现都优于其他竞争算法。

2）不同最小置信度的影响：为报告不同最小置信度对规则数量和 MCoR-Miner 运行时间的影响，我们选取 GAME- SALE 作为实验数据集，并将 MCoR-NoFilt、MCoR- AET、MCoR-FET 和 MCoR-NoScr 作为竞争算法。前缀模式为 d，间隔约束为[0,8]。我们将最小置信度设为 0.16、0.20、0.24、0.28、0.32 和 0.36。运行时间的比较如图 9 所示，挖掘得到的 CoR 和 MCoR 数量如表 VIII所示。

结果得出以下观察结论。

随着最小置信度的提高，CoR 和 MCoR 的运行时间及数量都会减少。例如，当 $mincf$ = 0.16 时，MCoR-Miner 耗时 18.45 秒，挖掘出 796 个 CoR 和 431 个 MCoR；而当 $mincf$ = 0.36 时，MCoR-Miner 耗时 0.30 秒，挖掘出 17 个 CoR 和 9 个 MCoR。其他算法也存在这一现象。原因如下：对于某个特定的前缀模式，最小支持度阈值会随着 $mincf$ 的提高而增加。因此，能够成为频繁模式的模式很少，并且每个共现模式都对应一个 CoR。因此，能够发现的 CoR 和 MCoR 更少，

表VIII：不同 $mincf$ 下 CoR 和 MCoR 数量的比较

	最小置信度 =0.16	最小置信度 =0.20	最小置信度 =0.24	最小置信度 =0.28	最小置信度 =0.32	最小置信度 =0.36
CoR 数量	796	246	100	49	27	17
MCoR 数量	431	134	53	25	15	9

表 IX：参数设置

数据集	前缀	Gap 约束	$mincf$ 对于 MCoR-Miner	$minsup$ 对于 CoP-Miner
GAMESALE	d	[0,3]	0.39	400
BABYSALE	fc	[0,3]	0.40	300
交易	xz	[0,3]	0.40	100
电影	dee	[0,3]	0.40	150000

运行时间也会缩短。对于任意最小置信度，MCoR-Miner 的表现都优于其他竞争算法。

E. 用于推荐的 *MCoR-Miner*

推荐任务是根据发现的潜在规则，针对给定的相同前件 **p** 提供推荐方案。为了研究 MCoR-Miner 的推荐性能，我们比较了共现模式挖掘与规则挖掘的推荐性能。因此，我们采用 CoP-Miner 作为竞争算法，并选取了以下商业数据集：GAMESALE、BABYSALE、TRANSACTION 和 MOVIE。前 80% 的序列用作训练集，其余 20% 用作测试集。训练所用的参数见表 IX。

为了评估基于共现规则的推荐性能，除置信度外，还有三个常用指标：精确率 $Pr = TP/(TP+FP)$、召回率 $Re = TP/(TP + FN)$ 和 F1分数 $F1 = 2 \times Pr \times Re/(Pr+Re)$，其中 TP 是正确推荐的数量，FP 是错误推荐的数量，FN 是未包含在推荐列表中的相关项数量。以 GAMESALE 为例，MCoR-Miner 在训练集中有两条极大共现规则：$\{d \rightarrow d, d \rightarrow e\}$，这意味着如果用户购买了项 d，MCoR-Miner 就会向该用户推荐项 d 和 e。在测试集中，前缀模式为 d 的所有长度为二的模式为 da、db、dd、de、dj 和 dk，其支持度分别为 110、108、139、281、126 和 83。因此，在此示例中，TP 为 420，因为模式 dd 和 de 的支持度之和为 139 + 281 = 420。FP 为零，因为 MCoR-Miner 推荐了项 d 和 e，而在测试集中，下一个项实际上可能是 a、b、d、e、j 和 k，这意味着项 d 和 e 都包含在这些项中，即 MCoR-Miner 没有错误推荐。FN 为 427，因为模式 da、db、dj 和 dk 的支持度之和为 110+ 108 + 126+ 83 = 427。因此，MCoR-Miner 在测试集上的精确率、召回率和 F1分数分别为 420/(420+0) = 1、420/(420+427) = 0.4959 和 $2\times1\times0.4959/(1+0.4959)$ = 0.6630。置信度、召回率和 F1分数的比较结果分别如图 10、11 和 12 所示。

结果表明，我们的方法具有以下优势。

1. MCoR-Miner 能有效剪枝置信度较低的共现规则。例如，图10显示，MCoR-Miner 挖掘出的每条极大共现规则的置信度都大于 0.39，而

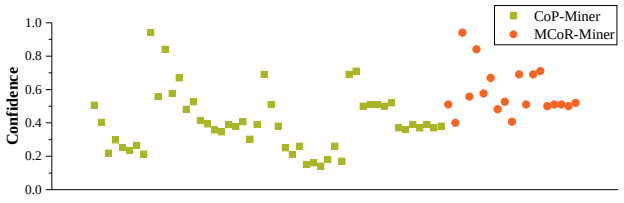


图 10：置信度比较

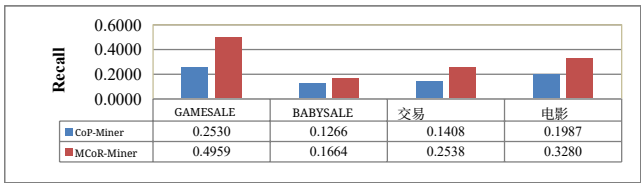


图 11：召回率比较

CoP-Miner 挖掘出的许多共现模式的置信度都小于 0.39，最低仅为 0.14。这是因为 MCoR-Miner 会发现置信度高且频率高的规则，而 CoP-Miner 只会发现频率高的规则。因此，MCoR-Miner 能有效剪枝置信度较低的共现模式，并提高推荐置信度。

2. 更重要的是，与 CoP-Miner 相比，MCoR-Miner 在推荐任务上的性能更好。例如，从图 11 和图 12 可以看出，在 GAMESALE 上，MCoR-Miner 的召回率和 F1分数分别为 0.4959 和 0.6630，高于 CoP-Miner。在其他数据集上也能观察到这一现象，这表明 MCoR-Miner 的推荐性能优于 CoP-Miner。

3. MCoR-Miner 是一种基于序列规则的推荐方法，其可解释性优于基于学习的推荐方法。原因如下：MCoR-Miner 通过计算训练集中的模式支持度，发现置信度较高的规则。以 GAMESALE 为例，模式 d、 dd 和 de 的支持度分别为 1917、970 和 763。因此，MCoR-Miner 生成两条规则： $\{d \rightarrow d, d \rightarrow e\}$，其置信度分别为 0.51 和 0.40。因此，MCoR-Miner 具有更好的可解释性。然而，对于基于学习的推荐方法而言，推荐模型非常复杂，无法用数学公式直观地表达。

总之，MCoR-Miner 的推荐性能优于 CoP-Miner。更重要的是，MCoR-Miner

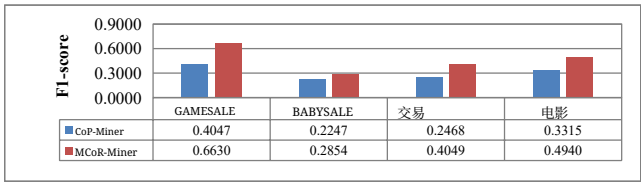


图 12：F1分数比较

能够制定有效的推荐方案，帮助决策者做出正确的决策。

VI. 结论

在非重叠序列模式挖掘或非重叠序列规则挖掘中，如果已知前缀模式或规则前件，就没有必要先发现所有模式或规则，再筛除无关的模式或规则。这个问题称为共现模式挖掘或共现规则挖掘。为了避免发现无关模式并获得更好的推荐性能，受最大模式挖掘概念的启发，我们研究了 MCoR 挖掘。与需要两个参数 —— 最小置信度和最小支持度——的经典规则挖掘不同，MCoR 挖掘不需要最小支持度，因为它可以根据规则前件的支持度和最小置信度自动计算。为了有效发现具有相同规则前件的所有 MCoR，我们提出了 MCoR-Miner算法。除支持度计算外，MCoR-Miner还包括三个部分：准备阶段、候选模式生成和筛选策略。为了有效计算支持度，我们提出了 DBI算法，该算法采用深度优先搜索和回溯策略，并配备索引机制。MCoR-Miner使用过滤策略剪除不含规则前件的序列，以避免无用的支持度计算。它采用 FET 和 BET 策略减少候选模式的数量，并使用筛选策略避免通过蛮力法寻找最大规则。为了评估 MCoR-Miner的性能，我们实现了十一种竞争算法，并选取了八个数据集进行评估。实验结果表明，与其他竞争算法相比，MCoR-Miner具有更好的运行性能和可扩展性。更重要的是，与频繁共现模式挖掘相比，MCoR-Miner具有更好的推荐性能。

本文聚焦于非重叠极大共现规则挖掘。非重叠序列模式挖掘是一种重复序列模式挖掘。我们知道，还有一些类似的挖掘方法，例如一次性序列模式挖掘和不相交序列模式挖掘。不同的挖掘方法会产生不同的挖掘结果。一次性与不相交最大共现规则挖掘值得进一步探索。此外，对于特定的挖掘任务，哪种挖掘方法最佳也值得进一步研究。更重要的是，本文研究的序列是一种特殊序列，即由项构成的序列。然而，一般序列由项集构成，每个项集包含多个有序项。研究一般序列数据库中的频繁重复模式挖掘和重复规则挖掘具有重要价值。

致谢

本研究部分受到了中国国家自然科学基金（61976240、62120106008）、中国国家重点研发计划（2016YFB1000901）以及中国河北省自然科学基金（编号：F2020202013）的资助。

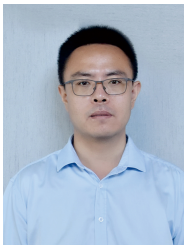
参考文献

[1] X. Wu、X. Zhu 和 M. Wu。“搜索的演变：三种计算范式”，ACM管理信息系统汇刊，第13卷，第2期，第20页，2022年。[2] C. H. Mooney 和 J. F. Roddick，“序列模式挖掘——方法与算法”，ACM计算调查，第45卷，第2期，第1-39页，2013年。[3] W. Gan、J. C. W. Lin、P. Fournier-Viger、H. C. Chao 和 P. S. Yu，“并行序列模式挖掘综述”，ACM知识发现与数据汇刊，第13卷，第3期，第25: 1-34页，2019年。[4] Y. Wu、Q. Hu、Y. Li、L. Guo、X. Zhu 和 X. Wu，“OPP-Miner：时间序列的保序序列模式挖掘”，IEEE控制论汇刊，DOI：10.1109/TCYB.2022.3169327，2022年。[5] J. C. W. Lin、Y. Djenouri 和 G. Srivastava，“大规模数据库中的高效闭合高效用模式融合模型”，信息融合，第76卷，第122-132页，2021年。[6] G. Srivastava、J. C. W. Lin、X. Zhang 和 Y. Li，“物联网中的大规模高效用序列模式分析”，IEEE物联网期刊，第8卷，第16期，第12669-12678页，2021年。[7] X. Ao、H. Shi、J. Wang、L. Zuo、H. Li 和 Q. He，“从具有层次结构的复杂事件序列中进行大规模频繁事件挖掘”，ACM智能系统与技术汇刊，第10卷，第4期，第1-26页，2019年。[8] L. Wang、X. Bao 和 L. Zhou，“流行共位置模式的冗余消减”，《IEEE知识与数据工程汇刊》，第30卷，第1期，第142-155页，2018年。[9] L. Wang、Y. Fang 和 L. Zhou，“基于偏好的空间共位置模式挖掘”，丛书名称：大数据管理，Springer Singapore，DOI：10.1007/978-981-16-7566-9，2022年。[10] Y. Wu、Y. Wang、Y. Li、X. Zhu 和 X. Wu，“Top-k自适应对比序列模式挖掘”，IEEE控制论汇刊，第52卷，第11期，第11819-11833页，2022年。[11] Q. Li、L. Zhao、Y. Lee 和 J. Lin。“受控驾驶行为实验中配对多变量时间序列的对比模式挖掘”，ACM空间算法与系统汇刊，第6卷，第4期，第1-28页，2020年。[12] X. Dong、P. Qiu、J. Lv、L. Cao 和 T. Xu，“通过学习挖掘Top-k有用负序列模式”，IEEE神经网络与学习系统汇刊，第30卷，第9期，第2764-2778页，2019年。[13] Y. Wu、M. Chen、Y. Li、J. Liu、Z. Li、J. Li 和 X. Wu。“ONP-Miner：一次性负序列模式挖掘”，ACM知识发现与数据汇刊，DOI：10.1145/3549940，2022年。[14] W. Gan、J. C. W. Lin、P. Fournier-Viger、H.-C. Chao、V. S. Tseng、P. S. Yu，“效用导向模式挖掘综述”，《IEEE知识与数据工程汇刊》，第33卷，第4期，第1306-1327页，2021年。[15] T. C. Truong、H. V. Duong、B. Le、P. Fournier-Viger，“基于新型上界的高平均效用项集高效垂直挖掘”，《IEEE知识与数据工程汇刊》，第31卷，第2期，第301-314页，2019年。[16] W. Song、L. Liu 和 C. Huang。“挖掘高平均效用项集的广义最大效用”，Knowl. Inf. Syst.，第63卷，2967，2021年 2947pp. – .[17] J. C. W. Lin、T. Li、M. Pirouz、J. Zhang 和 P. Fournier-Viger，“基于不确定数据库的高平均效用序列模式挖掘”，Knowl. Inf. Syst.，第62卷，第3期，第1199-1228页，2020年。[18] T. Wang、L. Duan、G. Dong 和 Z. Bao，“挖掘离群序列模式以分析序列数据的离群性”，ACM知识发现与数据汇刊，第14卷，第5期，第62页，2020年。[19] F. Min、Z. Zhang、W. Zhai 和 R. Shen，“使用三分区字母表发现频繁模式”，《信息科学》，第507卷，第715-732页，2020年。[20] Y. Wu、L. Luo、Y. Li、L. Guo、P. Fournier-Viger、X. Zhu 和 X. Wu，“NTP-Miner：非重叠三向序列模式挖掘”，ACM知识发现与数据汇刊，第16卷，第3期，第51页，2022年。[21] D. Amagata 和 T. Hara，“跨多条数据流挖掘Top-k共现模式”，《IEEE知识与数据工程汇刊》，第29卷，第10期，第2249-2262页，2017年。[22] H. Duong、T. Truong、A. Tran 和 B. Le，“基于简洁表示和项约束快速生成序列模式”，Knowl. Inf. Syst.，第62卷，第6期，第2191-2223页，2020年。[23] Y. Wang、Y. Wu、Y. Li、F. Yao、P. Fournier-Viger 和 X Wu。“自适应非重叠序列模式挖掘”，《应用智能》，第52卷，第6期，第6646-6661页，2021年。[24] Y. Wu、Y. Tong、X. Zhu 和 X. Wu，“NOSEP：带间隔约束的非重叠序列模式挖掘”，IEEE控制论汇刊，第48卷，第10期，第2809-2822页，2018年。[25] Y. H. Ke、J. W. Huang、W. C. Lin 和 B. P. Jaysawal，“利用具有特定数量间隔的序列模式挖掘在DNA序列中寻找可能的启动子结合位点”，IEEE/ACM计算生物学与生物信息学汇刊，第18卷，第6期，第2459-2470页，2020年。[26] M. Zhang、B. Kao、D. Cheung 和 K. Yip，“从序列中挖掘具有间隔要求的周期模式”，ACM知识发现与数据汇刊，第1卷，第2期，第7页，2007年。[27] Q. Shi、J. Shan、W. Yan、Y. Wu 和 X. Wu，“NetNPG：具有通用间隙约束的非重叠模式匹配”，《应用智能》，第50卷，第6期，第1832-1845页，2020年。[28] C. Li、Q. Yang、J. Wang 和 M. Li, M，“高效挖掘间隙约束子序列及其多种应用”，《ACM知识发现与数据汇刊》，第6卷，第1期，第2:1-39页，2012年。[29] S. Ghosh、J. Li、L. Cao 和 K. Ramamohanarao，“通过在序列对比模式上运行的耦合隐马尔可夫模型预测重症监护病房患者的脓毒性休克”，《生物医学信息学杂志》，第66期，第31页，2017年，19页。- .[30] F. Xie、X. Wu 和 X. Zhu，“使用通配符进行高效序列模式挖掘以提取关键词”，《知识型系统》，第115卷，第27-39页，2017年。[31] W. Wang 和 L. Cao，“VM-NSP：具有宽松负元素约束的垂直负序列模式挖掘”，《ACM信息系统汇刊》，第39卷，第2期，第1-27页，2021年。[32] W. Wang、J. Tian、F. Lv、G. Xin、Y. Ma 和 B. Wang，“在自定义约束下从时间序列交易数据中挖掘频繁金字塔模式”，《计算机与安全》，第100卷，第102088页，2021年。[33] Y. Wu、L. Wang、J. Ren、W. Ding 和 X. Wu，“挖掘具有周期性通配符间隙的序列模式”，《应用智能》，第41卷，第1期，第99-116页，2014年。[34] O. Ouarem、F. Nouioua 和 P. Fournier-Viger，“在非重叠频率下从事件序列中挖掘事件规则”，工业、工程及其他应用智能系统国际会议。Springer, Cham，第73-85页，2021年。[35] Y. Wu、X. Wang、Y. Li、L. Guo、Z. Li、J. Zhang 和 X. Wu，“OWSP-Miner：自适应一次性弱间隙强模式挖掘”，《ACM管理信息系统汇刊》，第13卷，第3期，第25页，2022年。[36] P. Fournier-Viger、C.-W. Wu、V.S. Tseng、L. Cao、R. Nkambou，“挖掘多个序列共有的部分有序序列规则”，《IEEE知识与数据工程汇刊》，第27卷，第8期，第2203-2216页，2015年。[37] J. S. Okolica、G. L. Peterson、R. F. Mills 和 M. R. Grimaila，“带变量的序列模式挖掘”，《IEEE知识与数据工程汇刊》，第32卷，第1期，第177-187页，2020年。[38] J. D. Smedt、G. Deeva 和 J. D. Weerdt，“挖掘用于分类的行为序列约束”，《IEEE知识与数据工程汇刊》，第32卷，第6期，第1130-1142页，2020年。[39] P. Ma、Y. Wu、Y. Li、L. Guo、H. Jiang、X. Zhu 和 X. Wu，“HW-Forest：采用哈希筛选和窗口筛选的深度森林”，《ACM知识发现与数据汇刊》，第16卷，第6期，第123页，2022年。[40] Y. Wu、C. Zhu、Y. Li、L. Guo 和 X Wu，“NetNCSP：非重叠闭合序列模式挖掘”，《知识型系统》，第196卷，第105812页，2020年。[41] Y. Li、S. Zhang、L. Guo、J. Liu、Y. Wu 和 X. Wu，“NetNMSP：非重叠最大序列模式挖掘”，《应用智能》，第52卷，第9861–9884页，2022年。[42] J. C. W. Lin、M. Pirouz、Y. Djenouri、C. F. Cheng 和 U. Ahmed，“利用预大型概念增量更新高平均效用模式”，《应用智能》，第50卷，第11期，第3788-3807页，2020年。[43] Y. Baek、U. Yun、H. Kim、H. Nam、H. Kim、J. C. W. Lin、B. Vo 和 W. Pedrycz，“RHUPS：通过基于滑动窗口的到达时间控制，在数据流上挖掘近期高效用模式”，ACM智能系统与技术汇刊，第12卷，第2期，第16:1-16:27页，2021年。[44] Y. S. Koh 和 S. D. Ravana，“无监督稀有模式挖掘：综述”，《ACM知识发现与数据汇刊》，第10卷，第4期，第1-29页，2016年。[45] L. Cao、X. Dong 和 Z. Zheng，“e-NSP：高效负序列模式挖掘”，《人工智能》，第235卷，第156-182页，2016年。[46] X. Dong、Y. Gong 和 L. Cao，“e-RNSP：一种高效的重复负序列模式挖掘方法”，《IEEE控制论汇刊》，第50卷，第5期，第2084-2096页，2020年。[47] J. C. W. Lin、Y. Djenouri、G. Srivastava、Y. Li 和 P. S. Yu，“利用三层MapReduce模型进行可扩展的高效用序列模式挖掘”，《ACM知识发现与数据汇刊》，第16卷，第3期，第1-26页，2022年。[48] W. Gan、J. C. W. Lin、J. Zhang、P. Fournier-Viger、H. C. Chao 和 P. S. Yu，“序列数据上的快速效用挖掘”，《IEEE控制论汇刊》，第51卷，第2期，第487-500页，2021年。[49] Y. Wu、R. Lei、Y. Li、L. Guo 和 X. Wu，“HAOP-Miner：自适应高平均效用一次性序列模式挖掘”，《专家系统应用》，第184卷，第115449页，2021年。[50] M.S. Nawaz、P. Fournier-Viger、U. Yun、Y. Wu 和 W. Song。利用爬山算法和模拟退火挖掘高效用项集。《ACM管理信息系统汇刊》，第13卷，第1期，第4页，2022年。[51] W. Gan、J. C. W. Lin、J. Zhang、H. Yin、P. Fournier-Viger、H. C. Chao、P. S. Yu，“多维序列上的效用挖掘”，《ACM知识发现与数据汇刊》

[52] P. Fournier-Viger、W. Gan、Y. Wu、M. Nouioua、W. Song、T. Truong 和 H. Duong，《模式挖掘：当前挑战与机遇》，*DASFAA 2022*国际研讨会，第 34–49 页，2022年。[53] P. Fournier-Viger、M. S. Nawaz、Y. He、Y. Wu、F. Nouioua 和 U. Yun，《MaxFEM：复杂事件序列中的极大频繁事件挖掘》，人工智能多学科趋势国际会议，第 86–98 页，2022年。[54] Y. Li、L. Yu、J. Liu、L. Guo、Y. Wu 和 Xindong Wu，《NetDPO：一次性条件下带间隔约束的（delta，gamma）近似模式匹配》，《应用智能》，第 52 卷，第 11 期，第 12155-12174 页，2022年。[55] Y. Wu、Z. Yuan、Y. Li、L. Guo、P. Fournier-Viger 和 Xindong Wu，《NWP-Miner：非重叠弱间隔序列模式挖掘》，《信息科学》，第 588 卷，第 124-141 页，2022年。[56] Y. Wu、M. Geng、Y. Li、L. Guo、Z. Li、P. Fournier-Viger、X. Zhu 和 X. Wu，《HANP-Miner：高平均效用非重叠序列模式挖掘》，《知识型系统》，第 229 卷，文章编号 107361，2021年。[57] J. Zhan、J. Ye、W. Ding 和 P. Liu，《基于效用理论的不完备模糊决策系统新型三支决策模型》，《*IEEE*模糊系统汇刊》，第 30 卷，第 7 期，第 2210 - 2226 页，2022 年。[58] X. Liu 、 X. Niu 和 P. Fournier-Viger，《利用规则生成属性剪枝的快速 Top-K 关联规则挖掘》，《应用智能》，第 51 卷，第 4 期，第 2077-2093 页，2021年。[59] X. Ao、P. Luo、J. Wang、F. Zhuang 和 Q. He，《从事件序列中挖掘精确定位事件规则》，《*IEEE*知识与数据工程汇刊》，第 30 卷，第 3 期，第 530-543 页，2017年。[60] Y. Wu、X. Zhao、Y. Li、L. Guo、X. Zhu、P. Fournier-Viger 和 X. Wu，《OPR-Miner：时间序列保序规则挖掘》，《*IEEE*知识与数据工程汇刊》，DOI：10.1109/TKDE.2022.3224963，2022年。[61] R. Wang、W. Ji、M. Liu、X. Wang、J. Weng、S. Deng、S. Gao 和 C. Yuan，《多数据源数据挖掘综述》，《模式识别快报》，第 109 卷，第 120-128 页，2018年。[62] S. Saleti 和 R. B. V. Subramanyam，《一种利用共现信息进行序列模式分布式挖掘的 MapReduce 算法》，《应用智能》，第 49 卷，第 1 期，第 150-171 页，2019年。[63] G. Huang、W. Gan 和 P. S. Yu，《TaSPM：目标序列模式挖掘》，arXiv 预印本 arXiv:2202.13202，2022年。[64] M. Celik、S. Shekhar、J. P. Rogers 和 J. A. Shine，《混合驱动时空共现模式挖掘》，《*IEEE*知识与数据工程汇刊》，第 20 卷，第 10 期，第 1322-1335 页，2008年。[65] Y. Wu、C. Shen、H. Jiang 和 X. Wu，《非重叠条件下的严格模式匹配》，《中国科学：信息科学》，第 60 卷，第 1 期，文章编号 012101，2017年。[66] Y. Wu、X. Liu、W. Yan、L. Guo 和 X. Wu，《解决非重叠条件下严格模式匹配问题的高效算法》，《软件学报》，第 32 卷，第 11 期，第 3331-3350 页，2021年。



Jie Li 于2002年在中国天津的河北工业大学获得电气工程博士学位。她目前是河北工业大学经济与管理学院教授。她的研究兴趣包括智慧医疗、商业和金融领域的大数据分析。



Wei Song 于2008年获得北京科技大学计算机科学博士学位。他现任北方工业大学信息科学与技术学院教授。其研究兴趣包括数据挖掘和知识发现。他已在同行评审期刊和国际会议上发表了40余篇研究论文。



Zhenlian Qi 于2020年获得哈尔滨工业大学环境科学与工程博士学位。2020年至2022年，她在广东工业大学从事博士后研究。她现任中国广州广东生态工程职业学院副教授。她的研究兴趣包括环境工程、生态大数据和数据分析。



Yan Li 获得天津大学管理科学与工程博士学位。她现任河北工业大学副教授。她目前的研究兴趣包括数据挖掘和供应链管理。



Youxi Wu 于中国天津的河北工业大学获得电气工程理论与新技术博士学位。他目前是河北工业大学的博士生导师和教授。他已在IEEE TKDE、IEEE TCYB、ACM TKDD、ACM TMIS、SCIS、INS、JCST、KBS、EWSA、JIS、Neurocomputing和APIN等期刊上发表了30多篇研究论文。他是中国计算机学会高级会员和IEEE会员。目前的研究兴趣包括数据挖掘和机器学习。



Chang Zhang 在中国天津的河北工业大学经济与管理学院获得硕士学位。目前，他的研究兴趣包括数据挖掘和机器学习。



Xindong Wu 于英国爱丁堡的爱丁堡大学获得博士学位。他是中国合肥的合肥工业大学长江学者。目前的研究兴趣包括数据挖掘、大数据分析、知识库系统和网络信息探索。Wu博士曾于2005年至2008年担任IEEE知识与数据工程汇刊（TKDE）的主编，目前担任IEEE国际数据挖掘会议指导委员会主席以及Knowledge and 信息系统。他是美国科学促进会会士，也是 IEEE 会士。