面向 Top-K 非重叠序列模式

Zefeng Chen[a], WenshengGan[a, b], GengsenHuang[a], Yan Li[c] 和Zhenlian Qi[d, *]

[a]暨南大学网络空间安全学院，中国广州 *510632*[b]暨南大学智能科学与工程学院，中国珠海 *519070*[c]河北工业大学，中国天津 *300131*[d]广东生态工程职业学院，中国广州 *510520*

arXiv:2304.11947v1 [cs.DB] 24 Apr 2023

文章信息

关键词：模式挖掘 top-k 非重叠序列模式 间隔约束

摘 要

序列模式挖掘（SPM）前景广阔、应用空间广泛，已在不同领域得到广泛应用。非重叠 SPM 作为一种数据挖掘技术，已用于发现某些特定挖掘任务中具有间隔约束要求的模式，例如生物数据挖掘。针对带有间隔约束的非重叠序列模式，研究者提出了 Nettree结构，以高效计算模式的支持度。在模式挖掘中，用户通常需要设定最小支持度阈值（最小支持度）。在大型数据库中，这尤其困难。尽管已有一些算法能够挖掘 top-k 模式，但它们都是针对固定长度模式的近似算法。本文提出一种用于挖掘 Top-k 非重叠序列模式（TNOSP）的精确算法。SPM 的 top-k 解决方案能够有效发现频率最高的非重叠序列模式，而无需设定最小支持度。作为一种新颖的模式挖掘算法，TNOSP 能够精确搜索具有不同间隔约束的非重叠序列中的 top-k 模式。我们进一步提出一种名为队列元集剪枝（QMSP）的剪枝策略，以提升 TNOSP 的性能。TNOSP 能够减少非重叠序列挖掘中的冗余，并在精确挖掘非重叠序列模式方面表现更佳。在多个数据集上的实验结果与比较表明，在精确率、效率和可扩展性方面，TNOSP 均优于现有算法。

1. 引言

近年来，丰富的数据已成为每个人生活中不可或缺的一部分。利用序列模式挖掘（SPM）从各种序列数据中提取有价值的信息，已成为一个重要课题 [25, 29]。通过 SPM，用户可以从序列项或事件序列中发现并分析具有统计显著性的模式 [20]。总而言之，SPM 在各个方向具有广泛的应用，例如 Web 挖掘 [6], 蛋白质检测 [10],室内轨迹 [8], 等。尤其是当用户需要发现某些生物数据（如基因组和蛋白质序列）时，带间隔约束的 SPM 比一般的 SPM 更有意义 [23]。然而，在这些领域中，间隔约束需要预先定义。因此，为了应对这些挖掘场景并处理各种有趣的数据挖掘任务，研究人员提出了带间隔约束的序列模式挖掘算法，以尽量减少用户不相关且无用的模式数量 [26, 45]。例如，Antunes等人 [3] 提出了带间隔约束的序列模式挖掘问题，Li等人 [22] 提出了一种高效挖掘带间隔约束的闭合子序列的方法。Pei等人 [28]研究了基于约束的序列模式挖掘中的模式增长方法，并提出了一个扩展框架。这些算法最重要的优势在于，间隔约束允许用户通过设置间隔灵活性来满足自身需求，而带间隔约束的序列模式挖掘问题则较难解决。对于类 Apriori 的挖掘算法，非频繁模式的所有超模式都是非频繁的，而频繁模式的所有子模式都是频繁的。这为挖掘提供了一种强大的剪枝策略，即在序列模式挖掘任务中，如果某个挖掘模式的支持度低于最小支持度阈值，则可以从候选集中删除该模式的所有超模式。在这种情况下，需要应用非重叠条件。与无条件相比，它对模式挖掘的约束更强；与单次条件相比，其限制性更弱。因此，Wu等人 [38] 提出了一种名为 NOSEP 的带间隔约束的非重叠 SPM 方法，其中可以利用 Apriori性质发现非重叠序列模式。

*通讯作者 czf1027@gmail.com (Z. Chen)；wsgan001@gmail.com (W. Gan)；hgengsen@gmail.com (G. Huang)；lywuc@163.com (Y. Li)；qzlhit@foxmail.com (Z. Qi)

需要注意的是，类 Apriori 序列模式挖掘算法可以按层级计算具有较高重要性的模式的支持度。因此，如果某个模式的子模式不是频繁模式，大量模式将被过滤掉，不再纳入考虑。

在大型数据库中找出所有频繁序列是一项艰巨的任务 [43]。此外，尽管上述算法可用于搜索模式，但在缺乏先验知识的情况下设定最小支持度阈值（最小支持度）仍然是个棘手的问题 [7, 44]。最小支持度越高，能够发现的频繁模式就越少。如果用户将最小支持度设得过高，就可能遗漏一些潜在的目标模式。随着最小支持度降低，更多模式会被挖掘为频繁模式。这可能需要更长的运行时间，并导致一定的资源浪费。因此，如何有效地发现频繁模式并找出有价值的模式，已成为一个重要课题 [7]。为应对这一挑战，基于 Top-k的算法提供了一种变通方案，将挖掘任务转变为发现 Top-k 频繁模式。符号 k 表示已处理数据集中模式的数量 [31, 33, 35]。Top-k从字面上看，指的是 k 最频繁的模式。一般而言，使用频繁模式挖掘算法时，Top-k 模式挖掘可以大幅减少冗余模式的数量，并有效保留高频模式 [7]。如 NOSEP[38], 所述，采用非重叠条件的序列模式挖掘可以弥补单次条件和无条件的不足，在序列模式挖掘领域具有坚实的基础和广阔的前景。NOSEP 会挖掘出所有支持度大于所设定的最小支持度的序列模式。

利用基于前k的算法，用户可以直接设置模式数量 k ，而无需考虑最小支持度阈值最小支持度。对于非重叠序列模式挖掘任务，在某些应用场景中，使用 k 来发现模式比使用最小支持度更高效。例如，CpG岛的分布[1]（位于人类基因启动子中的二核苷酸）具有很强的异质性。在基因组的某些片段中，CpG岛仍保持在其正常概率 [34]或更高水平。间隔约束和长度约束（其定义将在第3 节的定义4 中介绍）3需要进行设置。在挖掘频繁序列模式时，间隔约束意味着可挖掘模式中的两个项之间存在若干其他项。因此，可以避免序列模式中项不连续所产生的一些显著间隔。已有相关研究用于发现人类基因组中的CpG岛二核苷酸，并对其在基因组不同片段中的情况进行比较。这些研究有助于更好地理解CpG岛甲基化在人类发育、衰老和疾病过程中的变化 [5]。通常，我们希望方便地挖掘某一长度下的前几个模式。显然，在DNA序列中找到数量为 k 的模式对应的CpG岛的有价值规则更加直观。这使CpG岛的序列检测更加便捷。因此，不仅带有间隔约束的序列模式挖掘可以得到有效应用，而且还可以直接获得所需支持度排名前k的模式。类似地，该挖掘算法也可以应用于其他领域。Chai等 [7] 提出了NOSTOPK，用于在非重叠条件下挖掘具有间隔约束的前k个序列模式。然而，需要注意的是，NOSTOPK中的挖掘任务是在每个长度下发现前k 个模式。这与传统的前k序列模式发现不同。

在本研究中，我们设计了一种基于 Nettree 的 top-k SPM 挖掘算法，命名为 Top-kNon-Overlapping Sequential Pattern mining（TNOSP）。统一指标后，TNOSP 算法比基于启发式的 NOSTOPK 算法更加精确，并具有更优的性能。当前仅有一种算法用于发现非重叠 SPM 中的 top-k 模式，且其结果是近似的。然而，该算法仅适用于固定长度模式，在某些情况下无法发现最有用的模式。本文的主要贡献如下：

- 我们定义了 top-k 非重叠序列模式的相关概念，并形式化了 top-k非重叠 SPM 问题。与数据挖掘中传统的最小支持度阈值设置相比，采用 top-k 策略更加实用。

- 本文提出了 TNOSP，这是一种有效且高效的算法，能够在无需设置最小支持度阈值的情况下挖掘有用模式，并减少过多模式的数量。通过采用名为 QMSP 的队列元集剪枝策略，TNOSP 算法可以显著减少运行时间和计算成本。

- 为验证 TNOSP算法 相较于用于挖掘前-k 个非重叠序列模式的算法所具有的高效率和高精确率，我们在选定的真实 DNA 数据集上开展了一系列实验。

[1] https://en.wikipedia.org/wiki/CpG_岛_高甲基化

本文其他章节安排如下。第 2 节简要回顾了关于序列模式挖掘、非重叠 SPM 和前-k 模式挖掘的相关研究。第 3 节介绍前-k 非重叠 SPM 的定义。第 4 节详细提出高效算法 TNOSP。第 5 节介绍实验评估。第 6 节总结本文并讨论未来工作。

2. 相关工作

本节将分别讨论序列模式挖掘（SPM）、非重叠序列模式挖掘和 Top-k模式挖掘。

2.1. 序列模式挖掘

数据挖掘中最经典的任务之一，是在序列中发现有用的模式 [11]。通过序列模式挖掘算法，用户输入序列数据库后，所有支持度不低于最小支持度阈值的序列都可以被输出。序列模式挖掘之所以如此流行，是因为我们能够在大型数据库中发现模式，并通过挖掘任务解释这些数据。序列模式挖掘所挖掘出的这些模式有助于理解数据并进行决策。因此，序列模式挖掘在许多领域都有大量现实应用。这是一个非常活跃的研究主题，近年来大量论文提出了新的算法和应用，包括针对不同领域各种需求的序列模式挖掘扩展。Agrawal 和 Srikant [1] 于 1995 年提出了序列模式挖掘，用于挖掘序列模式。此后，序列模式挖掘这一主题受到了更多关注。第一个简单的序列模式挖掘算法是 AprioriAll，它基于 Apriori性质 [2]。上一节对此进行了介绍。随后，许多序列模式挖掘算法被提出 [11, 13, 24]。例如，Zaki 等人 [43] 提出了 SPADE，通过垂直数据库表示方法减少扫描数据库的时间。Ayres 等人 [4] 提出了 SPAM，通过深度优先搜索策略降低输入和输出成本；Han 等人 [15] 提出了 PrefixSpan，通过使用两种投影策略使模式快速增长。然而，算法在处理由组合爆炸导致的稠密数据集时，会生成大量候选模式。为此，Yang 等人 [42] 开发了 LAPIN，以直接解决这一问题。换言之，某个项的最后一次出现决定是否继续为该项连接候选模式，从而生成该项的超模式。此外，随着序列模式挖掘应用的不断增加，研究人员还研究了更多有趣且有用的序列模式挖掘发现，以提高挖掘效率，例如并行序列模式挖掘 [13], 高效用序列模式挖掘 [14, 12], 基于显著性的判别式序列模式挖掘 [17], 基于约束的序列模式挖掘 [32, 18], 等。一些应用要求算法允许两个项集之间存在一定的间隔。然而，这些算法存在一个共同缺陷：未考虑间隔约束，因此无法满足某些特定挖掘任务的需求。

2.2. 非重叠序列模式挖掘

上述算法的缺点此前已有讨论。Antunes 等人 [3] 首次提出了间隔约束序列模式挖掘的概念。Ji 等人 [19] 提出了带间隔约束的最小子序列挖掘。非重叠 SPM 是间隔约束序列模式挖掘的一个分支，这意味着同一个序列字母不能在同一位置出现两次 [41]。本节将讨论非重叠序列模式挖掘。一个字母可能匹配一个或多个模式字母，而在多字母匹配的情况下，不同的匹配可能导致不同的频率计数结果。为应对这些挑战，现有算法通常采用三种条件来计算模式的频率，分别是无条件 [46], 、单次条件 [21]，和非重叠条件 [9], 。采用无条件的挖掘任务不满足 Apriori性质，例如，在$AAGGGCC$ 中， $A[0,3]G$ 和 $A[0,3]G[0,3]C$ 的支持度分别为 6 和 12。然而，采用单次条件和非重叠条件的挖掘任务满足 Apriori性质。此外，非重叠条件比无条件更具体，但比单次条件的限制更少。因此，在 SPM 中，采用非重叠条件的序列模式挖掘更有可能从序列或数据集中发现更多有意义的模式 [38]。近年来，关于非重叠 SPM 的研究不断增多。例如，研究人员对 NTP-Miner [37] 采用深度优先和回溯策略进行了研究，以提高时间效率和空间效率。NetNPG [30] 解决了具有一般间隔的非重叠 SPM 问题，表明间隔不仅可以是自然数，也可以是负值。同时，HANP-Miner [36] 使用了简化的 Nettree，显著降低了算法的空间和时间复杂度。本研究旨在设计一种能够更高效地挖掘非重叠序列模式的算法。NOSEP [38] 通过使用 Nettree 来发现序列模式，其效果优于其他算法 [39]。因此，本文将介绍 NOSEP 的优化与改进。

表 1序列数据库

SID	序列
S_1	$<AG\ G\ AT>$
S_2	$<AT\ G\ G>$
S_3	$<C\ C\ T\ AT\ A>$
S_4	$<G\ C\ G\ C\ G\ T>$

2.3. Top-k模式挖掘

尽管 NOSEP 能够高效地搜索模式并计算支持度值，但最小支持度阈值的设置较为棘手。较低的阈值可能产生大量模式，导致计算成本高昂，而较高的阈值则可能导致一些模式无法满足用户的需求。为应对这一挑战，基于 top-k的算法 [16, 31] 被用于为用户提供充分的机会，使其能够直接决定所需的模式数量，而不是设置最小支持度阈值 [7]。Top-k 模式挖掘是一种能够大幅减少过量模式数量的经典策略，可以有效保留高频模式[7]。Han 等人 [16] 提出了使用 k 频繁模式进行频繁模式挖掘，其中 k 表示排名前列的频繁模式数量。此后，这一策略在模式挖掘中逐渐普及。Tzvetkov 等人 [31] 提出了 TSP，利用长度约束和 top-k 闭合序列模式的性质来执行动态支持度提升和投影数据库剪枝。Wang 等人 [33] 开发了一种基于 FP-tree 的高效算法，该算法能够在 FP-tree 的构建过程中及构建完成后进行动态剪枝。因此，本文提出了用于 top-k 非重叠条件下 SPM 的 TNOSP算法。用户可以利用 Nettree 挖掘带间隔约束的非重叠 SPM；与 SPM 算法中的其他结构相比，Nettree 具有显著更高的效率。TNOSP 还可以利用 top-k 模式挖掘来减少过量模式的数量。

3. 初步知识与问题陈述

本节将介绍与非重叠序列模式挖掘相关的基本概念和符号。为更清楚地阐述所要解决的问题，我们采用并说明了先前研究中的一些定义。此外，还将形式化定义 TNOSP 问题。下面给出与序列相关的定义。

定义 1（序列数据库）。在数据库中，设项的集合为 $I = \{i_1, i_2, i_3, ..., i_N\}$。作为一个项集， X 包含一个或多个来自 I 的项，即 $X \subseteq I$，且该项集非空。记号 $|X|$ 称为集合基数，表示项集 X 中项的数量。当项集 X 包含 k 个项（数学上表示为 $|X| = k$ ）时，称其为长度为 k 的 k-项集。然而，在非重叠序列模式挖掘任务中，一个项集中通常只有一个项。换言之， $k = 1$，且所有项集都是 1-项集。在形如 $S =<X_1\ X_2 ... X_n>$ 的序列中， X_i 位于 X_{i+1} 之前，其中 $i < n$，且它们的顺序不能改变。总之， S 中的所有项集都是有序的，而 n 表示 S 的大小。此外， L-序列是长度为 L 的序列 S，其长度由 $L = \sum_{k=1}^{n} |X_k|$ 计算。在我们的任务中， $L = n$。另外， $\mathcal{D}$ 表示一个序列数据库，记作 $\mathcal{D} = \{S_1, S_2, S_3, ..., S_n\}$。每个 $\mathcal{D}$ 中都有一组元组 $\langle \text{SID, DS} \rangle$ ，其中 DS 是数据序列，SID 是 DS 的唯一标识符。

定义 2（模式支持度）。项集 X 的支持度表示项集 X 在数据库中的出现频率，记作$sup(X)$。例如，给定一组事务 $\mathcal{D}$，序列模式挖掘的目标是找出所有支持度不低于最小支持度阈值的序列模式，记作$minsup$。在传统序列模式挖掘算法中，$minsup$ 由用户设定。支持度值低于$minsup$ 的模式不会在结果中频繁出现。在序列数据库 $\mathcal{D}$中，模式 P的支持度是其在每个序列 S中的支持度之和，计算方式为$sup(P, \mathcal{D}) = \sum_{i=1}^{n} |sup(P, S_i)|$。

例如，在表 1 中，$\text{sup(A, } S_1) = 2$，$\text{sup(A, } S_2) = 1$，$\text{sup(A, } S_3) = 2$，$\text{sup(A, } S_4) = 0$。因此，在 $\mathcal{D}$ 中，$\text{sup(A)} = 2+ 1 + 2 + 0 = 5$，$\text{sup(A)} = 5$，$\text{sup(C)} = 4$，$\text{sup(G)} = 7$，$\text{sup(T)} = 5$。如果最小支持度 $= 5$，则根据 Apriori性质，C 及其超模式将被剪枝 [2]。

定义 3（间隔约束）。带有间隔约束的挖掘模式 P 表示为 $p_1\ [min_1, max_1]\ p_2 ...[min_{m-1}, max_{m-1}]\ p_m$，其中 $p_j \subseteq S$、$j = 1$、2、3、...、min_j 和 max_j 是两个分别表示最小和最大间隔约束的非负整数，记为 间隔 $=$ [最小间隔，最大间隔]。间隔约束 间隔 $=$[最小间隔，最大间隔] 表示在 P 中，$min_1 = min_2 = min_3 = ... = min_{n-1}$ = 最小间隔 以及 $max_1 = max_2 = max_3 = ... = max_{n-1}$= 最大间隔 。在本研究中，用户可以根据自身需求设置间隔约束。

例如，如表 1 所示，如果间隔 $=$ [0,2]，sup(AT, S_1) $=$ 1，sup(AT, S_2) $=$ 1，sup(AT, S_3) $=$ 1，sup(AT,S_4) $=$ 0，因此在 D 中，sup(AT) $= 1 + 1 + 1 + 0 = 3$。考虑 $S_4 = s_1s_2s_3s_4s_5s_6$ = GCGCGT 和模式 P =G[0,2]C[0,2]G，P 在 S_4 <1、2、3>、<1、2、5>、<1、4、5> 以及 <3、4、5> 中的出现位置都满足条件。相反，如果间隔 $=$ [0,1]，则只有 <1、2、3> 和 <3、4、5> 满足条件。

定义 4（长度约束）。长度约束可表示为 长度 $=$ [最小长度、最大长度]，，表示长度约束介于最小长度约束和最大长度约束之间。如果模式中序列的长度 L 满足 最小长度 $\leq L \leq$最大长度，则认为其满足长度约束。

在表 1 中，序列为 $S_4 = s_1s_2s_3s_4s_5s_6$ = GCGCGT，间隔为 $=$ [0,2]。如果长度 $=$ [0,5]，则 P 在S_4 <1中的出现位置 2、3> <1、2、5> <1、4、5>以及 <3、4、5> 均满足条件。出现位置 <1、2、5> 的长度随后计算为长度 $= 5 - 1 + 1 = 5$。如果长度 $=$ [1,4]，则只有 <1、2、3> 和 <3、4、5> 满足条件。

定义 5（非重叠条件）。如果存在两个出现位置 $L_1 = <l_1, l_2, l_3, ..., l_\mu>$ 和 $L_2 = <l'_1, l'_2, l'_3, ..., l'_\mu>$，并且在 $\forall\ 1 \leq i \leq \mu$的情况下，其满足 $l_i \neq l'_i$ 的条件，则将 L_1 和 L_2 称为两个非重叠出现。当一个出现集中的所有出现位置均为非重叠出现时，这些出现集被定义为非重叠出现集。因此，对于 P中的 S，支持度是最大非重叠出现集的大小。

例如，在表 1 中，当间隔 $=$ [0, 2] 且长度 $=$ [0, 5] 时，P 在 S_4 <1、2、3>、<1、2、5>、<1、4、5> 以及 <3、4、5> 中的出现位置都满足条件。然而，两组出现位置 <1、2、3> 和 <1、2、5>、<1、4、5> 以及 <3、4、5> 存在重叠。因此，P 在 S_4 中的支持度为 2，而不是 4；4 是最大不重叠出现位置集的大小。

定义 6（前k 非重叠序列模式挖掘）。在前k 非重叠 SPM 中，频繁模式是指在满足间隔和长度约束的条件下，对应于挖掘序列 S 或挖掘序列数据库 D 中前k 个支持度的模式。本文的挖掘任务是在非重叠条件下搜索所有这些频繁模式。

4. TNOSP算法

基于上述讨论，本文提出了一种名为 TNOSP 的新算法，用于精确且高效地挖掘前k模式（满足非重叠条件）。该算法可分为三个小节进行讨论。在第 4.1节中，将介绍基于 Nettree 的非重叠序列模式挖掘算法。在第 4.2节中，我们设计了名为 TNOSP 的前k 算法。在第 4.3节中，提出了队列元集剪枝（QMSP）策略。

4.1. 基于Nettree的算法

在计算模式支持度时，需要解决的一个主要问题是：如果不采用回溯策略，就很难找到出现位置。为了解决这一问题，Wu等人提出了 NETGAP 算法 [38]。在表 1 中，序列为 $S_1 = \{s_1s_2s_3s_4s_5\}$ = AGGAT；如果用户需要检查带有间隔 $=$ [0, 1] 的模式 P_1 = AG，由于非重叠条件，将无法找到第一个非重叠出现位置 <1, 2>和 <1, 3>。搜索模式 P_2 = AGT 的用户将找不到出现位置 <1, 2, 5>，因为 5 - 2 - 1 = 2 不满足条件：间隔 $=$ [0, 1]。出现位置 <1, 3, 5> 被视为模式 $P_2 = A[0, 1]G[0, 1]T$的非重叠出现，该模式是 P_1的超模式。在某些算法（如 INSgrow）中 [27]，，如果根据其子模式的子出现位置来计算该模式的出现位置，那么在不使用回溯策略的情况下，可能会遗漏一些可行的出现位置，例如上述提到的 <1, 3, 5> [9]。为高效解决这一问题，提出了无需使用回溯策略的 NETGAP 算法。

定义 7（Nettree结构）。Nettree [40] 是一种基于树的结构，包括父节点、子节点、层级、根节点、叶节点等元素。Nettree 可以有多个根节点，这与树结构不同，树结构只能有一个根节点。相应地，为了便于描述节点，我们定义： j^{th} 层级中的节点 i 记作 n_j^i。Nettree 与树结构的另一个区别是，每个非根节点都可能有多个父节点，但其所有父节点必须处于同一层级。从一个节点到根节点之间可能存在多条路径。相应地，Nettree 最后一层中的叶节点被定义为绝对叶节点。此外，在 Nettree 中，完整路径是指从根节点到绝对叶节点的路径。

给定一个序列 $S = \{s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13}\} = GACTACGGCTCAT$、一个模式 $P = \{p_1 p_2 p_3 p_4\} = GCTA$、间隔约束间隔 = [0, 2]以及长度约束长度 = [1, 6]。根据 P和 S，算法可以构建其 Nettree，如图1（a）所示。

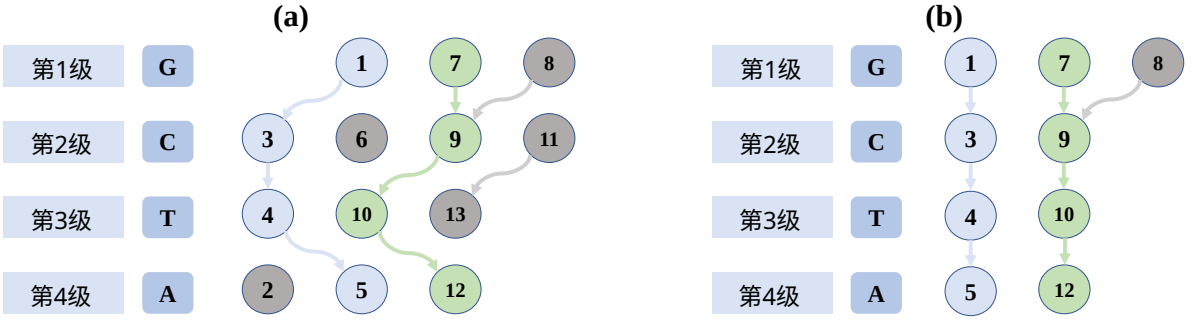

图1：序列中某个模式的 Nettree。

本文的任务是发现序列中的模式，该任务可分为三个部分。我们提出的算法伪代码如算法1所示。

- 首先，如果高层级的出现位置高于低层级的出现位置，则在满足间隔约束和长度约束的条件下，需要连接两个相邻层级中出现位置的叶节点。例如，在图1 (a) 中， n_1^1 应与 n_2^3 相连， n_2^3 应与 n_3^4 相连，并且 n_3^4 也应与n_4^5 相连。

- 其次，对于没有路径到达前一层级（除非它是第一个节点）或后一层级（除非它是最后一个节点）中叶节点的叶节点，应进行剪枝，这种节点称为孤立节点。例如，节点n_4^2 被剪枝，因为它没有通往前一层级的路径，且它不是第一个节点。 n_1^8 、 n_2^6 、 $n_2^1$1和 $n_3$13 在相同情况下也被剪枝。

- 第三，显然，剪枝后，在更新后的 Nettree 中得到第一个出现位置 <1、3、4、 5> ，如图1中浅蓝色高亮所示。遍历完成后，对已获得的出现位置进行剪枝。重复此过程，找到 <7、9、10、 12> 的出现位置。找到模式后，还会剪枝对应的叶节点，直至没有更多模式可供发现，因此依次剪枝出现位置 <1、3、4、 5> 和 <7、9、10、 12> 。此时，挖掘过程结束，得到两个非重叠出现位置，分别为 <1、3、4、5>和 <7、9、10、 12>，对于 P在 S中。$sup(P, S) = 2$ 由此得到。

4.2. TNOSP算法

在介绍 top-k 算法之前，首先介绍 *generateCandidate* 过程。此过程[38] 用于生成候选模式。例如，挖掘的目标是在序列 $S = s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 = AGTTAGCAC$ 中找出所有满足以下条件的频繁模式： 最小支持度 = 2、间隔约束间隔 = [0, 3]以及长度约束 长度 = [1, 8]。分析后可得，长度为 1 的频繁模式有四种：A、C、G、T；长度为 2 的频繁模式有五种：AA、AC、AG、GA、TA。因此，长度为 3 的候选模式数量为 $5 \times 4 = 20$，因为每种长度为 2 的频繁模式都能生成四种候选模式。然而，分析后发现，候选集具有这样的特性：如果某个超模式的后缀不是频繁

算法 1： NETGAP 算法

输入： 序列 S；挖掘模式 P；最小支持度阈值 $minsup$；间隔约束 间隔 = [最小间隔，最大间隔] 和长度约束长度 = [最小长度，最大长度]。

输出： 支持度(P, S)。

创建一个由 P 到 S的Nettree;
在适用条件下连接两个相邻层级的出现位置的叶节点;
剪除孤立节点处的模式;
用于 $n_1^j \in Nettree$ **执行**
    节点[1] $\leftarrow n_1^j$; // 出现位置存储在节点中
    对于 j = 1到 *Nettree.层级 -1* **执行**
        节点[j+1] = 的最左侧子节点 节点[j]满足长度约束;
    end
    支持度(P, S) $\leftarrow$ 支持度(P, S) + 1;
    剪枝节点[j];
结束
返回 支持度(P, S)

模式，那么该超模式也不会是频繁模式。尽管 AC 是频繁模式，但模式 ACG 并不频繁，因为模式 CG 并不频繁。采用模式增长方法，可以得到 13 种不同的长度为 3 的候选模式，分别是 AAA、AAC、AAG、ACA、ACG、AGA、ATA、GAA、GAC、GAG、TAA、TAC 和 TAG，它们将被放入长度为 $L+1$ 的候选集中。generateCandidate 的过程如算法2所示。假设存在一个长度为 L 的候选模式，则用它来生成长度为 $L+1$的候选模式。这种模式增长方法效率更高，计算开销也显著更低，因此本文继续采用此方法。

采用基于 Nettree 的方法和 *generateCandidate* 算法，研究人员开发了用于提取具有非重叠条件的 top-k 序列模式的算法 [7]，并将其命名为 NOSTOPK。然而，该算法是一种启发式算法；换言之，它并不精确，可能会遗漏一些关键模式。该算法也用于查找给定长度 k 的 top- L模式。因此，如果用户需要在所有模式中查找 top-k 模式，且模式长度限制为小于 L，NOSTOPK 就必须指定挖掘长度 L。只有达到指定的 L后，挖掘过程才会停止。用户很难汇总所有长度的模式并计算 k。用户也很难在不指定 L 的情况下设置最小支持度。为应对这两项挑战，TNOSP 能够比近似计算更准确地计算非重叠序列模式的支持度，从而提高非重叠序列模式挖掘的可信度。此外，用户在 TNOSP 中无需设置 L。

为应对上述挑战，设计了一些基于 Nettree 的改进方法。已知 NETGAP 算法需要进行多次计算。完成 NETGAP 步骤后，计算相应 Pin S 的支持度。基于此，这些步骤主要分为以下三个部分：

- 创建一个大小为 k 、名为 最小堆 的空 *minheap* ，按照模式支持度进行排序，并构建一个名为 候选模式 的候选集，用于存储候选模式。然后，将最小支持度 初始化为 0，并将 候选模式 初始化为长度为 1、满足长度约束和间隔约束的模式。
- 随后遍历所有现有候选模式。如果Q 未满，则直接将该候选模式放入 Q。如果Q 已满，则需要将该候选模式的支持度与最小支持度进行比较。如果该候选模式的支持度大于最小支持度，则从堆中弹出Q 的顶部元素，并将该候选模式作为Q的新元素添加进去。此外，还会更新最小支持度的值。
- 下一步是根据上一长度的频繁模式生成下一长度的候选集。然后重复上述过程，直到不再生成新的候选模式。最后，输出Q中的模式。

4.3. QMSP策略

基于上述陈述，用于挖掘非重叠序列模式的算法的精确率和可信度已大幅提高。此外，对效率的追求也是算法优化的需求之一，

算法2： TNOSP算法

输入： 序列 S；前K个模式的数量k；间隔约束 间隔 =[最小间隔，最大间隔] 和长度约束长度 =[最小长度，最大长度]。

输出： 前k个非重叠序列模式。

```
初始化Q ← ∅、最小支持度 ← 0、 i ← 1 并初始化FreArr ← 1个模式;
初始化候选模式 ← generateCandidate(FreArr、间隔、长度);
当候选模式 ≠ 为空 时执行
    FreArr = ∅;
    对于 p ∈ 候选模式 执行
        初始化 支持度 ← NETGAP(S、p、最小支持度、间隔、长度);
        如果 支持度 > 最小支持度 || (支持度 == 最小支持度 && Q.size() < k) 则
            FreArr。add(p);
            如果 Q.size() < k则
                Q.添加(p);      //将模式 P 加入最小堆 Q
            end
            否则，如果 Q.size() == k则
                更新 最小支持度;
                如果 sup > 最小支持度 则
                    Q。更新(p);      // 从最小堆中移除小于 最小支持度的模式 Q 并添加
                    p到 Q
                end
            end
        end
    end
    从 FreArr 中移除支持度小于 最小支持度的模式;                    //QMSP策略
    候选模式 = generateCandidate(FreArr，间隔，长度);
结束
返回 Q
```

因此，提出了一种名为队列元集剪枝（QMSP）的剪枝策略，以提高效率。根据上一小节，算法 2 中名为 freArr 的长度为 L+1 的队列元是由长度为L 的队列元生成的，用于生成超模式。若与长度为 L 的 *freArr* +1 所对应模式的支持度小于 *freArr* 的长度 L，则生成长度为 L 的 freArr 会造成一些资源浪费，因为尽管不满足 *top-k* 的模式在更新 最小堆 后不再位于其中，但这些模式的超模式仍会被生成。

因此，提出QMSP策略，以减少生成超模式的内存开销并提高计算效率。其主要思想如下：如上一小节所述，长度为 L+1 的候选集由前一长度 L的频繁模式生成。在原始NOSEP中，最小支持度 是固定的，因此，只要候选模式进入Q，就会进入队列元*freArr*。因此，在每轮 L模式遍历之后，支持度小于最新最小支持度 的候选模式都会被移除。这将大幅减少长度为 $L+1$的候选集数量。

为了评估QMSP算法的效率，我们在第5.3 节和第5.4节比较使用QMSP策略前后的访问节点差异。例如，对于序列 $S=s_1s_2s_3s_4s_5s_6s_7s_8s_9$ =*AGTCAGCAC*，其中长度和间隔分别为 [1,9] 和 [0,3]，。此外，参数k 设置为3。下面我们通过示例说明采用QMSP策略的TNOSP算法。

- 首先，它创建一个最小堆，用于存放支持度最大的三个模式。算法开始时，堆为空，并将最小支持度设置为 0。
- 其次，它生成长度为 1 且满足长度约束和间隔约束的模式，即：A、C、G、T。值得一提的是，这些模式按照字典序生成，其支持度通过 NATGAP算法计算。它们的支持度计算结果如表2所示。因此，第一个

表2 S中长度为1和2的所有模式的支持度

模式	A				C				G				T			
支持度	3				3				2				1			
模式	AA	AC	AG	AT	CA	CC	CG	CT	GA	GC	GG	GT	TA	TC	TG	TT
支持度	2	3	2	1	2	2	1	0	2	2	1	1	1	1	1	0

生成的模式是A，其支持度为 3。它是第一个被放入堆中的模式。由于堆未满，接下来进入堆的模式是C 和G，其支持度分别为 3 和 2。此时，堆已满，且最小支持度 更新为 2。最后一个长度为 1 的模式是T。由于其支持度小于最小支持度 2，因此被拒绝放入堆中。

• 随后，基于前一步生成超模式。根据Apriori性质和QMSP策略，将按如下方式进行剪枝：未能进入堆的模式，也就是支持度低于当前最小支持度的模式，将被剔除。这里，T 不再用于生成超模式。按字典序只会生成9个长度为2的模式，分别是AA、AC、AG、CA、CC、CG、GA、GG。依次将这9个长度为2的模式与最小支持度进行比较，从表2 可以明显看出，模式AC 进入了堆，而模式G 被弹出堆。由于不存在支持度高于最小支持度的模式，因此迭代结束。

5. 实验

在进行比较时，需要判断挖掘结果的精确率。精确率定义为发现的准确模式占所有已挖掘模式的比例。在Chai等人 [7], 提出的方法中，通过带权重的比率计算公式来衡量前k 项挖掘结果的精确率。在以下实验中，为便于计算和统计，并重新确立一致的判断标准，我们对该方法进行了修订，公式可表示为精确率 $= \frac{\sum_{i=1}^{L} c_i}{\sum_{i=1}^{L} t_i}$。在该公式中， c_i 和 t_i 分别表示长度为k 的正确前k 项频繁模式数和前 i项频繁模式总数。长度 L 的定义见第3节。一般来说，$\sum_{i=1}^{L} c_i = \sum_{i=1}^{L} t_i = k$ 只要 k 不小于频繁模式的数量即可。

为了评估我们提出的 TNOSP 算法的精确率、性能和可扩展性，实验在一台配备 Intel (R) CoreTM i7-8750H CPU 和 8 GB 内存的 64 位 Windows 10 个人计算机上进行。此外，TNOSP 使用 Java 编程和实现，并以 Eclipse集成开发环境作为开发环境。我们将算法 TNOSP 与当前最先进的 NOSTOPK 算法进行了比较 [7]。除 k外，还有一些参数会影响所提出算法的性能，例如长度约束和间隔约束。为确保实验中其他变量的一致性，在以下实验中，将两个约束分别设置为长度 $= [1, 20]$和 间隔 $= [0, 5]$ 。为便于比较，将 NOSTOPK 的输出改为长度为 1 至 L的所有符合条件的模式。换言之，长度短于 L 的输出在每次迭代前不再被清空，因此前k 模式包含所有长度不超过 L的模式。我们在六个真实数据集上开展了一些实验。这些实验的详细信息如下。

5.1. 实验数据集

我们使用六个数据集开展了这些实验，以评估TNOSP算法的效率和精确率。其中三个数据集是由病毒引起的特定疾病患者的DNA序列片段。这些数据集可从美国国家生物技术信息中心（NCBI）获取[2]。这三个数据集均包含来自不同患者的多个DNA序列。这三个数据集分别称为EBHF、MERS和COVID-19。顾名思义，这三个数据集分别表示多名埃博拉出血热、中东呼吸综合征和2019冠状病毒病患者的DNA片段。每个序列代表一名患者的DNA序列。

• ***EBHF*** 是由埃博拉病毒引起的埃博拉出血热（EBHF）患者的DNA序列片段集合。

[2] https://www.ncbi.nlm.nih.gov/labs/virus/vssi/

表 3实验数据集详情

序列数据集	总长度	序列数量	每个序列的平均长度
EBHF	118,779	6	19,796.5
MERS	181,126	6	30,187.7
COVID-19	536,097	18	29,783.2
dog	1,664,669	820	2,030.1
黑猩猩	1,711,934	873	1961.0
人类	2,231,011	1,691	1,319.3

Tabl 4
eL_{max} TNOSP 在六个数据集上的各种 k

数据集	EBHF	MERS	COVID-19	dog	黑猩猩	人类
$k = 10$	5	5	5	3	3	3
$k = 20$	6	6	6	5	4	4
$k = 30$	7	7	6	5	5	5
$k = 40$	7	7	7	5	5	5
$k = 50$	7	7	7	6	6	5
$k = 60$	7	7	7	6	6	6

• ***MERS*** 是一个数据集，包含由中东呼吸综合征冠状病毒（MERS-CoV）引起的中东呼吸综合征（MERS）患者的DNA序列片段。

• ***COVID-19*** 仍然是一个包含多名感染广为人知的2019冠状病毒病（COVID-19）患者的DNA序列片段的数据集，该疾病由2019年发现的新型冠状病毒（2019-nCoV）引起。

另外三个数据集同样是DNA序列片段，但取自不同的动物。其中三个数据集分别来自狗、黑猩猩和人类的DNA序列。这些数据集可在Kaggle上找到[3]。此外，我们还对黑猩猩和人类数据集中的序列进行了截短，因为实验不需要如此大量的数据。

•狗是一个包含狗的DNA序列片段的数据集。

• 黑猩猩是一个包含黑猩猩DNA序列片段的数据集；由于实验不需要如此大量的数据，这些序列经过了截短。

• 人类仍然是一个包含人类DNA序列片段的数据集；由于实验不需要如此大量的数据，这些序列经过了截短。

5.2. 精确率评估

本小节评估TNOSP和NOSTOPK在上述六个数据集上的实验精确率。NOSTOPK是一种启发式算法，而TNOSP是一种精确算法。因此，我们开展了一系列实验来比较这两种算法的精确率。在实验中，我们首先测试TNOSP，并获取输出模式集合中的最大长度，记为 L_{max}。测试后 L_{max} 的取值如表4所示。然后将 L_{max} 赋给NOSTOPK中的 L。实验结果如图5所示。结果表明，尽管NOSTOPK的精确率很高，但仍无法以完全精确的方式完成所有挖掘任务，这意味着可能会遗漏一些重要模式。然而，TNOSP能够精确地发现所有模式，在所有挖掘任务中均达到100%的精确率。因此，TNOSP不会出现上述模式遗漏的情况，这一点至关重要。

一般而言，NOSTOPK 的主要思路如下：如果一个模式 P_{sub} 是长度为 k 的 top- L 模式，那么它的超模式 P_{super} 很可能是长度为（k ）的 top-L + 1模式，但这并不完全准确。需要注意的是，这里所说的 top-k 是针对每种长度而言的，但在实验中，所有长度的输出统一为 top-k 模式。从宏观角度来看，显然，数据集的特征对精确率有很大影响。如表 5所示，MERS 和 COVID-19 数据集的特征相当相似，其精确率也相当高。尤其是 MERS 数据集，精确率达到 100%。此外，在 top-10

[3]https://www.kaggle.com/nageshsingh/dna-sequence-dataset

表5不同数据集中各种 k 的精确率

数据集	算法	$k=10$	$k=20$	$k=30$	$k=40$	$k=50$	$k=60$
EBHF	TNOSP	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
	NOSTOPK	100.0%	95.0%	93.3.0%	90.0%	92.0%	95.0%
MERS	TNOSP	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
	NOSTOPK	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
COVID-19	TNOSP	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
	NOSTOPK	100.0%	100.0%	100.0%	100.0%	100.0%	95.0%
dog	TNOSP	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
	NOSTOPK	100.0%	95.0%	93.3%	95.0%	92.0%	95.0%
黑猩猩	TNOSP	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
	NOSTOPK	100.0%	100.0%	93.3%	97.5%	94%	96.7%
人类	TNOSP	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
	NOSTOPK	100.0%	100.0%	97.5%	95.0%	96.0%	96.7%

结果中，精确率也能达到 100%。然而，图 5表明，NOSTOPK 的精确率并不稳定。随着 k 增大，精确率并非单调递减或递增，而是没有规律。因此，如上所述，算法的精确率在一定程度上取决于数据集。尽管 NOSTOPK 的精确率已经很高，超过了 90%，但对于算法 TNOSP，用户可以使用该算法精确挖掘所有符合条件的非重叠序列模式，而且其精确率表现相当稳定。因此，我们可以得出结论：就精确率而言，TNOSP 是一种更先进的算法。

5.3 效率评估

在本小节中，我们进行了一系列实验，以研究TNOSP的执行效率。需要注意的是，运行时间由算法运行时间和文件 I/O 处理时间组成。因此，在实验中，算法的运行时间可以衡量其效率，其中“运行时间”定义为从程序刚开始运行到找到所有符合条件的频繁序列模式所经过的时间。在这一指标下，我们使用上述六个数据集对TNOSP和NOSTOPK进行了实验。DNA 序列上 k 取10、20、30、40、50和60时，两种算法运行时间的差异和比较如图2所示。此外，为了评估上文提到的QMSP策略的效率，我们还将不采用QMSP策略的TNOSP算法加入了比较。

对于这两种算法，随着 k 的增加，最小堆的大小也会增加。进入堆的判断次数和计算出的模式数量也会增加，TNOSP中的 L_{max} 以及NOSTOPK中的 L也会随着运行时间增加，这是因为根据NETGAP算法，长度为 $L+1$ 的模式是由长度为 L 的模式得到的。基于以上分析，我们对TNOSP和NOSTOPK进行了比较。图2显示，对于所有 k取值和数据集，TNOSP的运行时间始终短于NOSTOPK，尤其是在MERS和COVID-19数据集上。这证明TNOSP表现出更好的性能，更适用于每个序列的平均长度较大的情况。此外，与NOSTOPK相比，TNOSP运行时间的增加更加均衡，尤其是在狗、黑猩猩和人类数据集上。对于NOSTOPK，图2 直观地显示，随着 k 的增加，一些运行时间仅略有增加，而另一些则大幅增加。在一定程度上，这也表明TNOSP不仅在运行时间方面效率更高，而且具有更高的稳定性。

此外，TNOSP 的剪枝策略将分析如下：总体而言，不采用 QMSP策略的 TNOSP 的运行时间基本上比 NOSTOPK 更短（见图 2）。然而，仍有少数情况下不采用 QMSP策略的 TNOSP 效率较低，这发生在最后三个数据集（$k=10$）以及数据集狗（$k=50$）中。然而，可以清楚地看出，采用 QMSP 后 TNOSP 的效率高于 NOSTOPK，与不采用 QMSP 的情况相比也有显著提升。这足以说明 QMSP策略具有很强的剪枝效果，在提高 TNOSP 的效率方面发挥了重要作用。

根据实验结果，在执行传统的 top-k 挖掘任务时，TNOSP算法在所有数据集上的执行效率都更高，在大型数据集上进行测试时可节省大量计算时间。接下来，我们将评估该算法的性能。在 NOSTOPK 中，使用

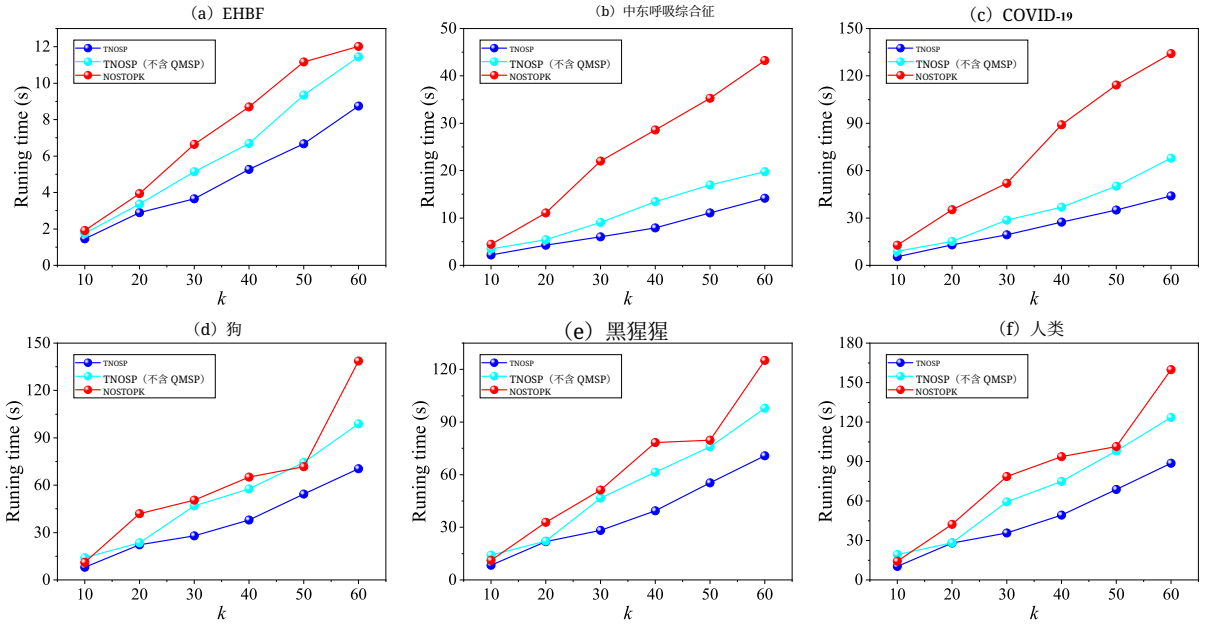

图2： 不同 k下的运行时间

启发式算法的目的是显著提高效率，同时使结果的偏差较小。这相当于牺牲一定程度的精确率，以获得更高的执行效率。这证明，在使用 Nettree 的情况下，使用 最小堆 调整 最小支持度 以处理 top-k 问题，以及使用 QMSP 减少计算量，都具有显著优势。在实现精确挖掘的前提下，TNOSP 也能达到更高的效率，这表明该算法的性能已得到大幅提升。

5.4. 访问节点的评估

由于在基于Nettree的算法中，需要访问节点才能获取每个符合条件的模式及其超模式。因此，访问节点数可以作为评估算法性能的重要指标。我们继续开展了实验，实验结果如图4所示。同样，随着 k 的增加，访问节点数也会增加。在所有k 值和数据集中，TNOSP的访问节点数均少于NOSTOPK。此外，TNOSP中访问节点数的增加也更加均衡。总之，访问节点数与上一小节中的运行时间相对应，这证明访问节点数是影响计算时间和执行效率的重要因素。

此外，为了评估QMSP策略，我们开展了以下实验来证明该策略的效率。我们分析了TNOSP以及使用QMSP策略前的算法，并记录和比较使用QMSP策略前后的访问节点数。在上述比较中可以看出，除数据集human中 $k = 20$ 的情况外，在不同的 k 值和不同数据集下，使用该策略后的访问节点数均少于未使用QMSP时的访问节点数。这进一步说明了使用QMSP策略的必要性。

5.5. 可扩展性

为了更好地衡量 TNOSP 的可扩展性，我们在八个数据集上开展了实验。所有这些数据集都是上述human数据集的子数据集，并首先选取 human数据集的前 200 个序列用于以下实验。随后，数据集规模以 200 个为单位递增，直到序列数量达到 1,600。因此，它们在本实验中的特征与上述 human数据集一致。

根据运行时间和访问节点数，实验结果如图5所示。图中显示，TNOSP的运行时间增长速度比NOSTOPK慢。此外，可以注意到，当序列大小为1,000时，NOSTOPK的运行时间与序列大小为800时几乎相同。原因是

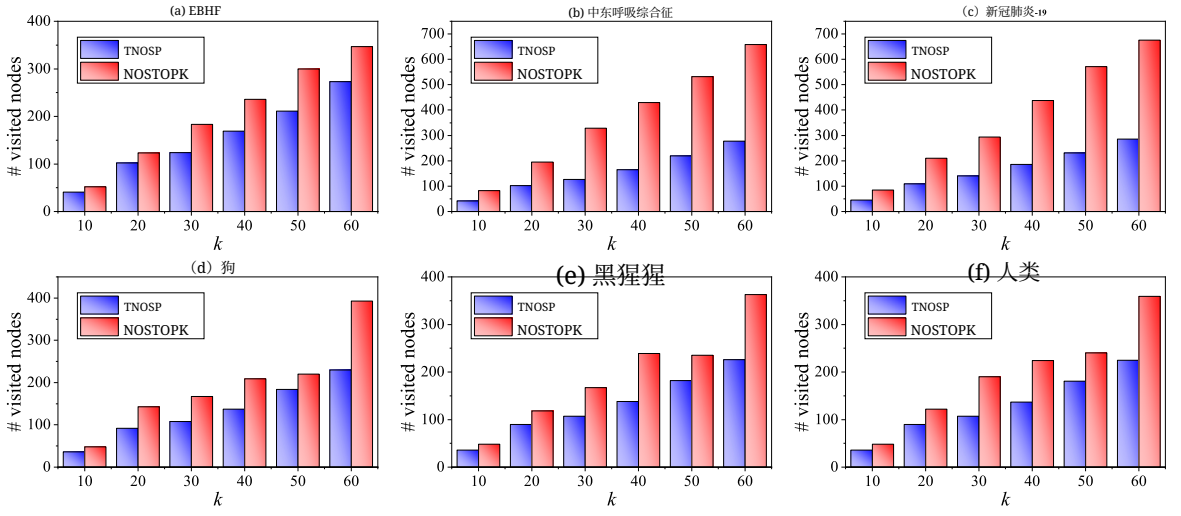

图3： TNOSP和NPSTOPK之间具有不同 k 的访问节点

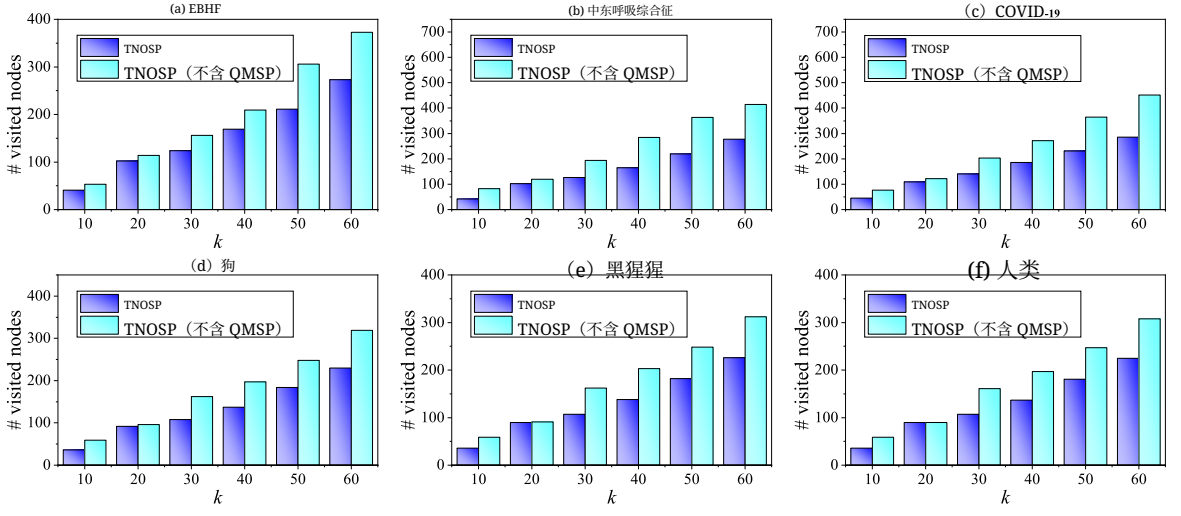

图4： 使用QMSP策略前后具有不同 k 的访问节点

当数据集大小为1000时，其 L_{max} 为4，而当大小为其他值时， L_{max} 为5。显然，随着数据集大小的增加，TNOSP的运行时间呈现出近似线性的增长关系。此外，随着数据集大小的增加，TNOSP中的访问节点数几乎保持不变，而NOSTOPK中的访问节点数会发生变化。这意味着TNOSP在访问节点方面具有稳定的可扩展性。如图5所示，随着数据集大小的增加，程序的运行时间呈线性关系，不会增长过快，而访问节点数则保持稳定。因此，我们可以得出结论：TNOSP算法在长序列和大型数据集上具有更强的可扩展性。

我们进一步分析该算法的优势。上述实验表明，在执行传统的top-k 非重叠序列模式挖掘时，TNOSP展现出更为先进的优势。这些优势体现在多个方面，并通过精确率、运行时间、访问节点数和可扩展性这四个可量化指标得到直观体现。上述实验结果表明，与NOSTOPK相比，TNOSP具有更高的稳定性，并且在精确率、效率和可扩展性方面表现更佳。因此，可以得出结论：TNOSP是用于挖掘传统top-k非重叠序列模式的一种更先进的算法。

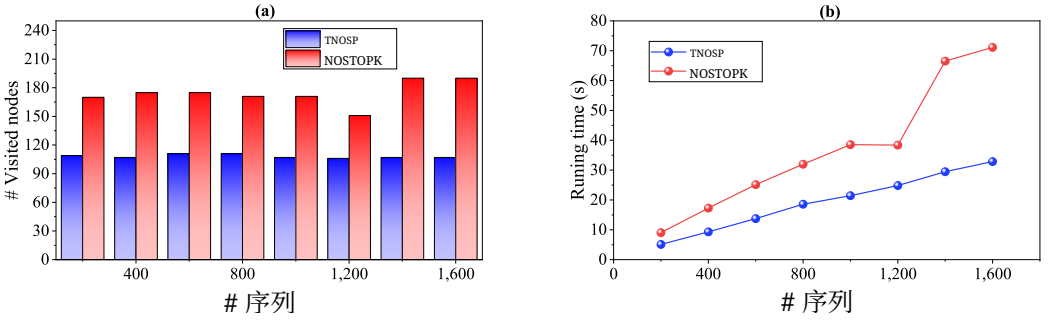

图5： 比较方法在数据集 $k = 30$ 时的可扩展性

6. 结论与未来工作

本文针对现有非重叠序列模式挖掘算法无法聚焦于最有价值且最精确的模式这一问题，提出了一种名为TNOSP的新算法，用于在不设置最小支持度阈值的情况下发现top-k非重叠序列模式。TNOSP更加灵活，能够更有效地搜索所有符合条件的模式。此外，TNOSP无需为目标模式指定固定长度，因此比传统的top-k挖掘算法更为适用，并具有更多现实应用场景。最后，我们在六个不同的数据集上，针对 k的不同取值开展了多项实验。实验结果表明，TNOSP始终具有更高的精确率、效率和可扩展性。在未来的工作中，我们计划将TNOSP与并行处理、流数据处理等技术相结合，以提高运行速度并减少数据或文件输入/输出（I/O）的处理时间。我们还可能研究并提出一种更优的方法，用于挖掘具有非重叠序列的top-k模式。

致谢

本研究部分得到国家自然科学基金委员会（编号：62272196和62002136）、广东省自然科学基金（编号：2022A1515011861）以及琶洲实验室青年学者计划（编号：PZL2021KF0023）的资助。

参考文献

[1] Agrawal, R., Srikant, R., 1995。序列模式挖掘，载于：第十一届国际数据工程会议，IEEE。第3–14页。[2] Agrawal, R., Srikant, R.等，1994。关联规则挖掘的快速算法，载于：第20届超大规模数据库国际会议，Citeseer。第487–499页。[3] Antunes, C., Oliveira, A.L., 2003。带间隔约束的序列模式挖掘中模式增长方法的泛化，载于：模式识别中的机器学习与数据挖掘国际研讨会，Springer。第239–251页。[4] Ayres, J., Flannick, J., Gehrke, J., Yiu, T., 2002。使用位图表示进行序列模式挖掘，载于：第八届ACM SIGKDD知识发现与数据挖掘国际会议。第429–435页。[5] Bock, C., Paulsen, M., Tierling, S., Mikeska, T., Lengauer, T., Walter, J., 2006。人类淋巴细胞中的CpG岛甲基化与DNA序列、重复序列和预测的DNA结构高度相关。《PLoS Genetics》2，243–252。[6] Boggan, S., Pressel, D.M., 2007。图形处理器：通用计算的新兴平台。技术报告。美国陆军研究实验室阿伯丁试验场计算与信息科学局。[7] Chai, X., Yang, D., Liu, J., Li, Y., Wu, Y., 2018。带非重叠条件的前k 序列模式挖掘。《Filomat》32，1703–1710。[8] Chen, Y., Yuan, P., Qiu, M., Pi, D., 2019。基于模糊网格序列的室内轨迹频繁模式挖掘算法。《专家系统与应用》118，614–624。[9] Ding, B., Lo, D., Han, J., Khoo, S.C., 2009。高效地从序列数据库中挖掘闭合重复间隔子序列，载于：IEEE第25届国际数据工程会议，IEEE。第1024–1035页。[10] Ferreira, P.G., Azevedo, P.J., 2005。带约束的蛋白质序列模式挖掘，载于：欧洲数据挖掘与知识发现原理会议，Springer。第96–107页。[11] Fournier-Viger, P., Lin, J.C.W., Kiran, R.U., Koh, Y.S., Thomas, R., 2017。序列模式挖掘综述。《Data Science and Pattern Recognition》1，54–77。[12] Gan, W., Du, Z., Ding, W., Zhang, C., Chao, H.C., 2021。序列上的可解释模糊效用挖掘。《IEEE模糊系统汇刊》29，3620–3634。

[13] Gan, W., Lin, J.C.W., Fournier-Viger, P., Chao, H.C., Yu, P.S., 2019a。并行序列模式挖掘综述。《ACM数据发现汇刊》13，1–34。[14] Gan, W., Lin, J.C.W., Zhang, J., Chao, H.C., Fujita, H., Yu, P.S., 2019b。ProUM：高效用序列模式挖掘，载于：IEEE系统、人类与控制论国际会议，IEEE。第767–773页。[15] Han, J., Pei, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., Hsu, M., 2001。PrefixSpan：通过前缀投影模式增长高效挖掘序列模式，载于：第17届国际数据工程会议，Citeseer。第215–224页。[16] Han, J., Wang, J., Lu, Y., Tzvetkov, P., 2002。在没有最小支持度的情况下挖掘前k频繁闭合模式，载于：IEEE国际数据挖掘会议，IEEE。第211–218页。[17] He, Z., Zhang, S., Wu, J., 2019。基于显著性的判别序列模式挖掘。《专家系统与应用》122，54–64。[18] Huynh, H.M., Nguyen, L.T., Pham, N.N., Oplatková, Z.K., Yun, U., Vo, B., 2022。一种带索引序列模式挖掘的高效方法。《知识型系统》239，107946。[19] Ji, X., Bailey, J., Dong, G., 2007。带间隔约束的最小判别子序列模式挖掘。《知识与信息系统》11，259–286。[20] Kumar, S., Mohbey, K.K., 2019。基于大数据的模式挖掘并行与分布式方法综述。《沙特国王大学学报：计算机与信息科学》。[21] Lam, H.T., Mörchen, F., Fradkin, D., Calders, T., 2014。压缩序列模式挖掘。《统计分析与数据挖掘：美国统计学会数据科学期刊》7，34–52。[22] Li, C., Wang, J., 2008。高效挖掘带间隔约束的闭合子序列，载于：SIAM国际数据挖掘会议，SIAM。第313–322页。[23] Li, C., Yang, Q., Wang, J., Li, M., 2012。带间隔约束的子序列高效挖掘及其各种应用。《ACM数据发现汇刊》6，1–39。[24] Mabroukeh, N.R., Ezeife, C.I., 2010。序列模式挖掘算法分类。《ACM计算调查》43，1–41。[25] Masseglia, F., Teisseire, M., Poncelet, P., 2005。序列模式挖掘，载于：《数据仓储与挖掘百科全书》。IGI Global，第1028–1032页。[26] Orlando, S., Perego, R., Silvestri, C., 2004。一种新的间隔约束序列挖掘算法，载于：ACM应用计算研讨会。第540–547页。[27] Pan, H., 2009。一种高效的子序列挖掘算法，载于：第三届生物信息学与生物医学工程国际会议，IEEE。第1–4页。[28] Pei, J., Han, J., Wang, W., 2007。基于约束的序列模式挖掘：模式增长方法。《智能信息系统期刊》28，133–160。[29] Pinto, H., Han, J., Pei, J., Wang, K., Chen, Q., Dayal, U., 2001。多维序列模式挖掘，载于：第10届信息与知识管理国际会议。第81–88页。[30] Shi, Q., Shan, J., Yan, W., Wu, Y., Wu, X., 2020。NetNPG：具有一般间隔约束的非重叠模式匹配。《应用智能》50，1832–1845。[31] Tzvetkov, P., Yan, X., Han, J., 2005。TSP：前k闭合序列模式挖掘。《知识与信息系统》7，438–457。[32] Van, T., Vo, B., Le, B., 2018。带项集约束的序列模式挖掘。《知识与信息系统》57，311–330。[33] Wang, J., Han, J., Lu, Y., Tzvetkov, P., 2005。TFP：挖掘前k频繁闭合项集的高效算法。《IEEE知识与数据工程汇刊》17，652–663。[34] Wang, X., Duan, L., Dong, G., Yu, Z., Tang, C., 2014。带间隔约束的密度感知判别序列模式高效挖掘，载于：高级应用数据库系统国际会议，Springer。第372–387页。[35] 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。第78–86页。[36] Wu, Y., Geng, M., Li, Y., Guo, L., Li, Z., Fournier-Viger, P., Zhu, X., Wu, X., 2021a。HANP-Miner：高平均效用非重叠序列模式挖掘。《知识型系统》229，107361。[37] Wu, Y., Luo, L., Li, Y., Guo, L., Fournier-Viger, P., Zhu, X., Wu, X., 2021b。NTP-Miner：非重叠三向序列模式挖掘。《ACM数据发现汇刊》16，1–21。[38] Wu, Y., Tong, Y., Zhu, X., Wu, X., 2017。NOSEP：带间隔约束的非重叠序列模式挖掘。《IEEE控制论汇刊》48，2809–2822。[39] Wu, Y., Wang, L., Ren, J., Ding, W., Wu, X., 2014。带周期性通配符间隔的序列模式挖掘。《应用智能》41，99–116。[40] Wu, Y., Wu, X., Min, F., Li, Y., 2010。用于带灵活通配符约束模式匹配的网树，载于：IEEE信息重用与集成国际会议，IEEE。第109–114页。[41] Wu, Y., Zhu, C., Li, Y., Guo, L., Wu, X., 2020 196, 105812。NetNCSP：非重叠闭合序列模式挖掘。《知识型系统》。[42] Yang, Z., Wang, Y., Kitsuregawa, M., 2007。LAPIN：通过末位置归纳实现密集数据库中的高效序列模式挖掘算法，载于：高级应用数据库系统国际会议，Springer。第1020–1023页。[43] Zaki, M.J., 2001。SPADE：挖掘频繁序列的高效算法。《机器学习》42，31–60。[44] Zhang, C., Du, Z., Gan, W., Yu, P.S., 2021。TKUS：前k高效用序列模式挖掘。《信息科学》570，342–359。[45] Zhang, M., Kao, B., Cheung, D.W., Yip, K.Y., 2007。从序列中挖掘具有间隔要求的周期模式。《ACM数据发现汇刊》1，623–633。[46] Zhu, X., Wu, X., 2007。带间隔要求的跨序列复杂模式挖掘，载于：第20届人工智能国际联合会议。第2934–2940页。