# k-子序列通用词的排名与反排名

Duncan Adamson

Leverhulme 功能材料设计研究中心，利物浦大学，英国*
d.a.adamson@liverpool.ac.uk

arXiv:2304.04583v1 [cs.DS] 10 Apr 2023

摘要

词 $w$的一个子序列是一个词 $u$，满足 $u = w[i_1]w[i_2],\dots w[i_{|u|}]$，其中索引集合为 $1 \leq i_1 < i_2 < \cdots < i_k \leq |w|$。如果字母表 $\Sigma$ 上的每个词 $\Sigma^k$ 都作为子序列出现在 $w$ 中，则词 $w$ 是 $k$-子序列通用的。本文针对字母表 $\Sigma = \{1, 2, \dots, \sigma\}$上的固定长度 $n$ 的 $k$-子序列通用词，提出了新的算法。令 $\mathcal{U}(n, k, \sigma)$表示字母表 $\Sigma$上的 $n$长度 $k$-子序列通用词集合，我们提出：

- 一种 $O(nk\sigma)$ 时间算法，用于计算 $\mathcal{U}(n, k, \sigma)$ 的大小；
- 一种 $O(nk\sigma)$ 时间算法，用于对集合 $\mathcal{U}(n, k, \sigma)$ 中的词进行排名；
- 一个 $O(nk\sigma)$ 时间算法，用于从集合 $\mathcal{U}(n, k, \sigma)$ 中对词进行反排名；

一种用于枚举集合 $\mathcal{U}(n, k, \sigma)$ 的算法，在经过 $O(nk\sigma)$ 预处理后具有 $O(n\sigma)$ 延迟。

## 1 引言

词和子序列是两种基本的组合对象。非正式地说，词 $w$ 的子序列是一个词 $u$，它可以通过删除符号 $w$的某个子集而得到。子序列是计算机科学 [Bar+20; Day+21; Fle+22; HSZ17; Kos+21;Lot97; MSY04;Sim03;Tro03; Zet16] 领域内以及更广泛范围内受到大量研究的对象，其应用领域十分广泛，包括生物信息学 [HWG20;Shi+19]、数据库理论 [Art+17]以及并发建模 [Sha78]。Kosche等人 [Kos+22]最近提供了一份关于子序列算法的综述，重点介绍了词中子序列查找问题的主要成果。

本文研究 $k$-子序列通用词。词 $w$ 是一个在字母表 $\Sigma$ 上关于 $k$的子序列通用词，当且仅当 $w$ 包含字母表 $\Sigma$ 上的每个长度为 $k$ 的词作为子序列。这些词最初由 Karandikar 和 Schnoebelen [KS16; SK19] 定义为 $k$-丰富词，不过近期研究使用了术语 $k$-子序列通用性 [Bar+20; Day+21; Kos+21]，本文也将采用这一术语。对这些词的研究源于关于Simon 同余 [Sim75]的工作。非正式地说，若两个词 $w$、$v$ 满足 $k$-同余，则 $w$ 和 $v$ 具有相同的长度为 $k$的子序列集合。这种关系已得到广泛研究 [FK18;Sim03; Tro03; Zet16], ，近期还提出了一种渐近最优算法，用于检验两个词是否 $k$-同余 [Gaw+21]。

与这项工作最相关的是 Barker等人的论文 [Bar+20]和Day等人的论文 [Day+21]，它们直接研究了 $k$-子序列通用词。在 [Bar+20]中，作者证明了可以在线性时间内确定一个词是否为 $k$-子序列通用词，以及给定词的最短 $k$-子序列通用前缀。此外，他们还给出了相关结果，表明一个词的最小 $\ell$-因子集合 $w, w_1w_2\dots, w_\ell$ 使得 $w_1w_2\dots w_\ell$是 $k$-子序列通用词，以及使得 $w^i$ 是 $k$-子序列通用词的索引 $i$，都可以被高效地确定。

*本项工作在哥¨廷根大学完成，并获得了部分资助。4

[Day+21]在此基础上，作者给出了一系列算法结果，用于最小化将一个词转换为 $k$-子序列通用词所需的编辑操作数，并给出了关于插入、删除和替换的结果。他们证明，假设 $k < n$，将词 $w$ 转换为 $k$-子序列通用词 $w'$ 所需的最少插入和替换次数可以在 $O(nk)$时间内计算出来。此外，他们还证明，可以在 $O(nk)$时间内确定将一个词的通用性指数（使该词成为 $k$-子序列通用词的最大 $k$ ）降低至 $k$ 所需的删除次数。

本文旨在为 $k$-子序列通用词类的一些基本操作提供算法，包括计数、排名、反排名和枚举。在提供这些算法时，我们旨在拓展对固定长度 $n$ 的$k$-子序列通用词空间的理解。我们使用 $\mathcal{U}(n, k, \sigma)$ 表示字母表大小为 $\sigma$ （假定该字母表为$\{1, 2, \ldots, \sigma\}$）时长度为 $n$ 的 $k$-子序列通用词集合。计数问题要求给定类别中词的数量。排名问题以一个词 $w$ 为输入，确定集合中按字典序小于 $w$ 的词的数量。反排名问题是排名问题的逆问题，它以排名 $i$ 为输入，要求找出集合中排名为 $i$ 的词。最后，枚举问题要求按照某种固定顺序显式输出集合中的每一个词。这些问题中的每一个都已在其他词类中得到深入研究，包括循环词[Ada22；Ada+21；FM78；GR61；KRR14；SW17] 以及格雷码 [FM78；KRR14；Sav97]。

我们的成果。本文基于关于 $k$-子序列通用词的现有研究成果，旨在进一步深入理解固定长度的 $k$-子序列通用词空间。我们为固定长度 $n$的 $k$-子序列通用词提供了一系列算法结果。我们将字母表为1、2、...、$\sigma$ 且长度为 $n$ 的所有 $k$-子序列通用词的集合记为 $\mathcal{U}(n, k, \sigma)$。在第 3 节中，我们提供了一个 $O(nk\sigma)$ 时间算法，用于计算 $\mathcal{U}(n, k, \sigma)$ 的大小。在第 4 节中，我们利用该计数算法中的观察结果，为集合 $\mathcal{U}(n, k, \sigma)$ 中的词提供了一个 $O(nk\sigma)$ 时间的排名算法。最后，在第 5 节中，我们为集合 $\mathcal{U}(n, k, \sigma)$ 中的反排名提供了一个 $O(n\sigma)$ 时间算法，其预处理时间为 $O(nk\sigma)$。我们注意到，该反排名算法直接为集合 $\mathcal{U}(n, k, \sigma)$ 提供了一个延迟为 $O(n\sigma)$ 的枚举算法。

计算模型。在本文中，我们假设采用单位成本 RAM 计算模型，在这种情况下，它等价于字长为 $O(\log(N) \log(\sigma))$ 的单位成本字 RAM，其中 $N$ 是输入或输出中较大的一个。我们注意到，相对于长度为 $n$ 的 $k$-子序列通用词的数量，这仍然是对数级的，因此也相对于输出此类词数量的整数表示所需的位数是对数级的。通过对所述界乘以因子 $O(n/\log(n))$，即可将我们所有的复杂度调整为字长为 $O(\log(n) \log(\sigma))$ 的单位成本 RAM 计算模型，其中 $n$ 是输入的大小。我们通过注意到，这些算法仅执行至少一个整数的大小不超过 $\sigma$ 的乘法，或在大小不超过 $\sigma^n$ 的整数之间进行加法，从而避免因子 $O(n^2/\log^2(n))$。

## 2 预备知识

我们使用以下记号。给定一对自然数 $m, n \in \mathbb{N}$，记号 $[m, n]$表示有序集合 $\{m, m+1, \ldots, n\}$，或者在 $m > n$时表示空集。词 $w$是某个字母表 $\Sigma$上的符号的有序序列。字母表 $\Sigma$上长度为 $n$ 的词的集合记为 $\Sigma^n$，字母表 $\Sigma$ 上的所有词的集合记为 $\Sigma^*$。词 $w$ 的词长记为 $|w|$。记号 $w[i]$ 用于表示词 $w$中的第 $i^{th}$ 个符号，而 $w[i, j]$ 用于表示 $w$ 中与词 $w[i]w[i+1] \ldots w[j]$对应的连续子序列（或者在 $i > j$时表示空词 $\varepsilon$ ）。如果存在某对索引 $i, j$ 使得 $v = w[i, j]$，则词 $v$ 是词 $w$ 的因子。

$w = 11234,\ 4321,\ 22314,\ 33214,$
$4v = 12234,\ 323134,\ 11234,\ 4412$

图 1：两个词 $w, v \in \Sigma^{20}$ 的拱分解示例，其中 $\Sigma = \{1, 2, 3, 4\}$。每个弓形（或末尾后缀）都用逗号分隔，并以不同颜色突出显示。请注意，$w$ 是 4-子序列通用的，而 $v$ 仅是 3-子序列通用的，尽管它们具有相同的 Parikh 向量 (5,5,5,5)。

我们假设字母表为 $\Sigma = [1, \sigma]$，其中 $\sigma \geq 1$是某个自然数。给定两个词 $w$、$v \in \Sigma^n$，$w$ 按字典序小于 $v$，当且仅当存在某个索引 $i \in [1, n]$，使得$w[1, i-1] = v[1, i-1]$ 且 $w[i] < v[i]$。给定两个词 $w$、$v \in \Sigma^*$，$v$是$w$ 的子序列，当且仅当存在某个索引序列 $1 \leq i_1 < i_2 < \cdots < i_{|v|} \leq |w|$，使得 $v = w[i_1]w[i_2] \dots w[i_{|v|}]$。

定义 1（$k$-子序列通用性）。词 $w \in \Sigma^n$ 是 $k$-子序列通用的，当且仅当每个词 $v \in \Sigma^k$ 都是 $w$ 的子序列。字母表大小为 $\sigma$ 时，长度为 $n$ 且具有 $k$-子序列通用性的词的集合记为 $\mathcal{U}(n, k, \sigma)$。

词 $w$ 的子序列通用性指数是满足 $w$ 为 $k$-子序列通用的最大值 $k$。为了判断一个词是否为 $k$-子序列通用，我们使用由 Hebard [Heb91] 首次提出的弓形因子分解。简而言之，词的一个弓形是包含字母表 $\Sigma$ 中每个符号至少一次的最短因子。本文余下部分采用以下正式定义：

定义 2（拱）。字母表 $\Sigma = \{1, 2, \dots, \sigma\}$ 上的一个拱是包含 $\Sigma$ 中每个符号至少一次的词。拱 $v$ 的通用子序列是满足以下条件的索引集合（$i_1, i_2, \dots, i_\sigma$）：

- $v[i_1], v[i_2], \dots, v[i_\sigma]$ 恰好包含 $\Sigma$ 中的每个符号一次。
- 索引 $i_j$ 是 $v$ 中符号 $v[i_j]$ 首次出现的位置。
- 索引 $i_\sigma$ 是 $v$中的最后一个位置。

任何不属于通用子序列的符号都称为自由符号。

我们注意到，这个弓形的定义对应于一个 1-子序列普适词，其中最后一个符号是唯一的，也就是说，它不会出现在词的其他位置。

定义 3（[Heb91], 拱分解）。具有普适性指数 $k$的词 $w \in \Sigma^n$ 的拱分解是一个因子集合 $\{w_1, w_2, \dots, w_k, v\}$，记作 $Arch(w)$，满足：$w_i$ 是 $w$ 的一个因子，并且对于每个 $i \in [1, k]$，它都是一个弓形；$v$ 是 $w$ 的一个后缀，且不包含任何作为因子的弓形；并且 $w_1 w_2 \dots w_k v = w$。

图 1 给出了这种分解的一个示例。Day 等人 [Day+21] 扩展了定义 3，证明一个词是 $k$-子序列普适的，当且仅当存在一个 $w$ 的拱分解，其中至少包含 $k$个弓形；此外，这种分解可以在线性于该词长度的时间内计算出来。

定理 1（[Day+21]）。当且仅当$Arch$（$w$）至少包含 $k$ 个弓形时，词 $w \in \Sigma^n$ 在 $\Sigma$ 上是 $k$-子序列普适的。此外，$Arch$（$w$）可以在 $O$（$n$）时间内计算出来。

本文使用来自 [Day+21] 的以下技术性引理。

引理 1（[Day+21]）。设 $\Delta(w,i,j)$ 表示 $w[i,j]$ for some$w \in \Sigma^n$ 中不同符号的数量。我们可以在 $O(n)$ 时间内，计算出每个 $j \in [1,n]$ 对应的 $\Delta(w,1,j)$ 的值。

我们结合定理 1 和引理 1，得出以下观察结果。

观察 1. 设 $w \in \Sigma^n$ 是一个具有弓形分解 $w_1, w_2, \ldots, w_k, v$ 的 $k$-子序列通用词，并进一步令 $\mathcal{A} = \{A_1, A_2, \ldots, A_k\}$ 表示满足 $A_\ell = 1 + \sum_{i \in [1,\ell-1]} |w_i|$ 的索引集合，即 $w$ 中对应于 $Arch(w)$ 中一个弓形首个位置的索引集合。那么，对于每个 $\ell \in [k]$ 和 $i_\ell \in [A_\ell, A_{\ell+1} - 1]$，$\Delta(w, A_\ell, i_\ell)$ 的值可在 $O(n)$ 时间内计算，其中 $\Delta(w, A_\ell, i_\ell)$ 表示 $w[A_\ell, i_\ell]$ 中不同符号的数量。

使用这一记号，我们给出本文所考虑的排名和反排名问题的形式化定义。词 $w$ 在词的有序集合 $\mathcal{S}$ 中的排名，是指在该集合的排序下，小于 $w$ 的 $\mathcal{S}$ 中的词的数量。在本文中，我们假设 $k$-子序列通用词的集合按字典序排列，因此，词 $w$在集合 $\mathcal{U}(n,k,\sigma)$中的排名，是 $\mathcal{U}(n,k,\sigma)$ 中按字典序小于 $w$的词的数量。排名问题的输入是一个词 $w$ 整数三元组 $n, k, \sigma \in \mathbb{N}$，满足 $n \geq k\sigma$，并返回 $\mathcal{U}(n,k,\sigma)$中按字典序小于 $w$的词的数量。反排名问题在概念上是排名问题的逆问题。给定一个整数$i \in [1, |\mathcal{S}|]$，反排名问题要求给出 $\mathcal{S}$ 中排名为 $i$的词。在本文中，反排名问题的输入是一个排名 $i$ 和整数三元组 $n, k, \sigma \in \mathbb{N}$，满足 $n \geq k\sigma$，并返回 $\mathcal{U}(n,k,\sigma)$中排名为 $i$的词。

# 3 弓形计数与 $k$-子序列通用词

首先，我们介绍一种用于计数 $\mathcal{U}(n,k,\sigma)$大小的工具，即在字母表 $\Sigma = \{1, 2, \ldots, \sigma\}$上长度为 $n$ 的 $k$-子序列通用词的数量。除了其本身是一项有趣的成果之外，这还为我们的排名和反排名工具奠定了基础。

本节分为两个部分。首先，我们给出计数长度为 $n$ 、字母表大小为 $\sigma$ 的拱、0-子序列普适词和 1-普适词数量的公式。其次，我们提供一种递归方法，用于计数长度为 $n$ 、字母表大小为 $\sigma$ 的 $k$-子序列普适词数量。

## 3.1 拱、0-子序列普适和1-子序列普适词 s

我们首先考虑如何计算长度为 $n$ 的拱、0-子序列普适词和 1-子序列普适词的数量。我们注意到，这三个特殊情况彼此密切相关。首先，注意任何不是 0-子序列普适的词都必须至少是 1-子序列普适的。因此，1-子序列普适词的数量等于词的数量减去 0-子序列普适词的数量。类似地，长度为 $n$ 的拱的数量等于字母表大小为 $\sigma - 1$ 的、长度为 $(n-1)$ 的 1-子序列普适词的数量乘以 $\sigma$。我们从 0-子序列普适词开始。

引理2。字母表 $\Sigma = \{1, 2, \ldots, \sigma\}$ 上长度为 $n$ 的 0-子序列普适词的数量为：

$$\sum_{i \in [1,\sigma]} (-1)^{i+1} \binom{\sigma}{i} (\sigma - i)^n.$$

证明。设 $\varsigma \subseteq \Sigma$ 是一个长度为 $i$ 的字母表。首先注意，在 $\varsigma$ 上长度为 $n$ 的词的数量为 $i^n$；此外，由于这样的字母表有 $\binom{\sigma}{i}$ 个，因此在任意长度为 $i$ 的字母表上的词总数为 $\binom{\sigma}{i} i^n$。注意，$\varsigma^n$ 中的任意字符串也属于 $(\varsigma \cup \{x\})^n$，其中 $x \in \Sigma \setminus \varsigma$；此外，有 $\sigma - i$ 个这样的长度为 $i+1$ 的字母表包含 $\varsigma$ 中的每个符号。更一般地，有 $\binom{\sigma-i}{j}$ 个大小为 $i+j$ 的字母表包含

长度为 $i$ 的字母表 $\varsigma$ 中的每个符号。因此，对所有长度为 $i$ 的字母表中的长度为 $n$ 的词求和，结果为 $\binom{\sigma}{i}(i)^n$，也会使集合 $\{\Sigma \setminus \{x\} \mid x \in \Sigma\}$ 中长度为 $j < i$ 的语言里的每个词都被计数 $\binom{\sigma-j}{i-j}$ 次。结合众所周知的二项式系数恒等式，可得集合 $\{\Sigma \setminus \{x\} \mid x \in \Sigma\}$ 中任意字母表里的不同词总数的公式:

$$\begin{aligned}&\binom{\sigma}{1}(\sigma-1)^{n-1} - \binom{\sigma}{2}(\sigma-2)^{n-1} + \binom{\sigma}{3}(\sigma-3)^{n-1} \ldots (-1)^{\sigma+1} \\ &= \sum_{i\in[1,\sigma]} (-1)^{i+1}\binom{\sigma}{i}(\sigma-i)^n\end{aligned}$$

□

根据引理2，可以直接对长度为 $n$的弓形和1-子序列普适词进行计数。

推论1. 字母表为 $\Sigma = \{1, 2, \ldots, \sigma\}$ 时，长度为 $n$的1-普适词的数量为:

$$\sigma^n - \sum_{i\in[1,\sigma]} (-1)^{i+1}\binom{\sigma}{i}(\sigma-i)^n$$

推论 2. 字母表为 $\Sigma = \{1, 2, \ldots, \sigma\}$ 时，长度为 $n$的拱的数量为:

$$\sigma(\sigma-1)^{n-1} - \sum_{i\in[2,\sigma]} (-1)^i i\binom{\sigma}{i}(\sigma-i)^{n-1}.$$

### 3.2 $k$-子序列普适词的计数

为了对 $k$取任意值时的 $k$-子序列普适词进行计数，我们采用递归方法。其基本思路是，计算具有给定前缀 $v$的 $k$-子序列普适词的后缀数量。令 $\mathcal{S}(v)$表示长度为 $n-|v|$ 的词的集合，其中对于每个词 $u \in \mathcal{S}(v)$，词 $vu$ 都是一个 $k$-子序列普适词。令 $Arch(v) = v_1, v_2, \ldots, v_\ell, v'$为 $v$的拱分解，或 $v'$的前 $k$ 个拱。为了计算$\mathcal{S}(v)$的大小，我们注意到，每个词 $u \in \mathcal{S}(v)$都必须包含一个前缀 $u'$，使得 $v'u'$ 是一个弓形，而后缀 $u[|u'|+1, |u|]$ 必须包含 $k-\ell-1$ 个拱。我们的递归方法基于这样的观察： $\mathcal{S}(v)$的大小等于 $\bigcup_{x\in\Sigma} \mathcal{S}(vx)$的大小。这带来了两个主要问题：确定集合 $\mathcal{S}(vx)$是否为空，以及确保无需对每个后缀 $w \in \Sigma^{n-|v|}$显式检查 $\mathcal{S}(vw)$，便能计算出 $\mathcal{S}(v)$的总大小。

我们通过引入一个新函数 $\mathsf{CS}(q, m, c)$（计算后缀）来解决这些问题，使得 $|\mathcal{S}(v)| = \mathsf{CS}(q, m, c)$，其中:

- $q$ 是 $v'$中不同符号的数量。
- $m$ 是每个词 $u \in \mathcal{S}(v)$ 中的自由符号数，即 $u$中不属于 $u[|u'|+1, |u|]$前 $k-\ell-1$个拱中的任何通用子序列，或词 $v'u'$中 $u'$ 的通用子序列的符号数。
- $c$ 是 $v'u$中拱的最小数量，等于 $k-\ell$。

$\mathsf{CS}(q, m, c)$的值以递归方式确定。我们首先给出基本情形。如果 $m = 0$，那么每个剩余符号都必须位于剩余 $c$ 拱中的某一个通用子序列中，从而得到 $\mathsf{CS}(q, 0, c) = (\sigma-q)!(\sigma!)^{c-1}$。另一方面，如果 $c = 0$，那么剩余符号可以任意从 $\Sigma$ 中选取，从而得到 $\mathsf{CS}(q, m, 0) = \sigma^m$。假设$c$和 $m$都大于0，那么 $\mathsf{CS}(q, m, k)$ 的值可以递归确定。如果 $q = \sigma$，那么下一个符号必须是该词的 $(k-c)^{th}$ 弓形的第一个符号。由于

$\sigma$ 此类可能的符号，后接一个 CS$(1, m, c-1)$ 后缀，CS$(\sigma, m, c)$ 的值为$\sigma$ CS$(1, m, c-1)$。否则，下一个符号可以是当前弓形的通用子序列中已有的 $q$ 个符号之一，也可以是通用子序列中没有的 $\sigma - q$ 个符号之一，从而得到 CS$(q, m, c)$ $=(\sigma-q)$ CS$(q+1, m, c)+q$ CS$(q, m-1, c)$。综上，函数 CS$(q, m, c)$ 可定义为：

$$\mathrm{CS}(q,m,c) = \begin{cases} (\sigma-q)!(\sigma!)^{c-1} & m=0 \\ \sigma^m & c=0 \\ \sigma\,\mathrm{CS}(1,m,c-1) & q=\sigma, m>0, c>0 \\ (\sigma-q)\,\mathrm{CS}(q+1,m,c)+q\,\mathrm{CS}(q,m-1,c) & q>0, m>0, c>0 \end{cases}$$

引理 3。令 $\mathcal{S}(v)$ 表示满足以下条件的后缀集合：对于每个 $u \in \mathcal{S}(v)$，词 $vu$ 是一个$n$-长度的 $k$-子序列词。此外，令 $\ell$ 为 $v$ 中完整拱的数量，并令 $v'$为 $v$ 的后缀，使得 $Arch(v)$ $= v_1 v_2 \ldots v_{\ell-1} v_\ell v' = v$，其中 $v_i$是针对 $v$的拱分解中的第 $i^{th}$ 个弓形。然后， $\mathcal{S}(v)$ 的大小等于 CS$(q, m, k-\ell)$，其中 $q$是 $v'$ 中不同符号的数量，且 $m = n - (|v| + \sigma(k-\ell-1))$。

证明。为简化记号，我们假设如果 $v' = \varepsilon$，那么 $q = \sigma$。我们从基本情形开始。如果 $m = 0$，则 $\mathcal{S}(v)$的后缀中的每个符号都必须位于剩余拱之一的通用子序列中。由于 $v'$中有 $q$ 个符号，因此将 $v'$ 延伸为一个拱有 $(\sigma - q)!$ 种可能方式，而长度为 $\sigma$的拱有 $\sigma!$ 个，故$\mathcal{S}(v)$中后缀的总数为 $(\sigma-q)!(\sigma!)^{k-\ell-1} = (\sigma-q)(\sigma!)^{c-1}$。或者，如果 $\ell \geq k$ （这意味着 $v$ 已经是一个 $k$-子序列通用词），那么后缀中的每个剩余符号都是一个自由符号，因此对后缀的内容没有任何限制。因此在这种情况下， $\mathcal{S}(v)$ 中有$\sigma^m$ 个后缀。注意，在 $m = 0$ 且 $\ell \geq k$ 的情况下，这两个公式都返回 0，对应于空词。

在一般情况下，假设 $\mathcal{S}(vx)$的大小等于 CS$(q', m', c')$，其中 $q'$ 是 $v'x$中的不同符号数量，$m'$ 等于 $n - (|v| + 1) - (q + \sigma(k-\ell))$，而 $c'$ 是 $k - \ell$。注意到， $\mathcal{S}(v)$中的后缀总数等于 $\sum_{x\in\Sigma} |\mathcal{S}(vx)|$。如果 $q = \sigma$，那么下一个符号必须属于下一个弓形的通用子序列，其值等于 CS$(1, m, k-\ell-1)$。如果 $x$ 是已经出现在 $v'$中的 $q$ 个符号之一，那么 $\mathcal{S}(vx)$的大小为 CS$(q, m-1, c)$。否则， $\mathcal{S}(vx)$的大小为CS$(q+1, m, c)$。由于 $v'$中有 $q$个不同符号，后缀数量由总和 $q$ CS$(q, m-1, c)+(\sigma-q)$CS$(q+1, m, c)$给出。因此， $\mathcal{S}(v)$的大小为 CS$(q, m, k-\ell)$。

□

引理 4。对于每个 $q \in [1, \sigma], m \in [0, n], c \in [0, k]$， CS$(q, m, c)$ 的值都可以在 $O(nk\sigma)$ 时间内计算。

证明。正确性可由上述论证得出。我们假设，对于每个 $q \in [1, \sigma], c \in [0, k]$， $(\sigma - q)!(\sigma!)^c$ 的值已预先计算好，这需要 $O(k\sigma)$ 时间；对于每个 $m \in [0, n]$ ， $\sigma^m$ 的值也已预先计算好，这需要 $O(n)$ 时间。为确定时间复杂度，注意到在 CS$(q+1, m, c)$ 、 CS$(1, m, c-1)$ 和 CS$(q, m-1, c)$ 的值已计算好的前提下，CS$(q, m, c)$ 的值可以在常数时间内计算。由于 $c = 0$ 和 $m = 0$ 的基本情形可以在常数时间内计算，且对于其他所有情形， $m$ 和 $(\sigma - q) + \sigma c$ 的值都单调递减，因此可以采用动态方式计算每个 $q \in [1, \sigma], m \in [0, n], c \in [0, k]$ 对应的 CS$(q, m, c)$ 的值：从基本情形开始，按照 $m$、 $\sigma - q$ 和 $c$ 的值递增的顺序进行计算。需要指出的是，递增 $m$、 $q$ 和 $c$ 的顺序无关紧要，只要在计算 CS$(q, m, c)$ 之前，先计算好 CS$(q+1, m, c)$、CS$(1, m, c-1)$ 和 CS$(q, m-1, c)$ 即可。因此，对每个 $q \in [1, \sigma], m \in [0, n], c \in [0, k]$ 计算 CS$(q, m, c)$ 的值所需的总时间复杂度为 $O(nk\sigma)$。

□

注意， $k$-子序列通用词的数量等于集合 $\mathcal{S}(\varepsilon)$ 中词的数量，即具有空前缀的 $n$-长度词的数量。因此，由引理 4 可知， $\mathcal{U}(n, k, \sigma)$ 的大小可以通过计数 $\mathcal{S}(\varepsilon)$ 的大小来计算，这等价于计算 $\sigma$ CS$(1, n - (k\sigma)\ k)$。定理 2 由此观察得出。

定理2。 $\mathcal{U}(n,k,\sigma)$ 的大小可以在 $O(nk\sigma)$ 时间内计算。

# 4 排名

利用第 3 节中介绍的计数技术，我们现在可以对给定的词 $w \in \Sigma^n$在所有长度为 $n$ 的 $k$-子序列通用词中的排名进行计算。这个过程以迭代方式进行。对于每个 $i \in [1,n]$，我们计算前缀为 $w[1,i-1]x$ 的长度为 $n$ 的 $k$-子序列通用词的数量，其中 $x$ 是按字典序小于 $w[i]$ 的某个符号。对每个 $i \in [1,n]$ 对应的此类词的数量求和，即可得到按字典序小于 $w$ 的长度为 $n$ 的 $k$-子序列通用词的总数。对每个前缀对应的此类词的数量求和，就能计算出按字典序小于 $w$ 的长度为 $n$ 的 $k$-子序列通用词的总数。

设 $Arch(w) = w_1, w_2, \ldots, w_m v$。第一个关键观察是，给定任意词 $u = w_1 w_2 \ldots w_i u'$，对于某个 $i \le k$，$u$是 $k$-子序列通用词，当且仅当 $u'$是 $(k-i)$-子序列通用词。其次，给定一个词 $s = w_1 w_2 \ldots w_i[1:j]s'$，$s$是 $k$-子序列通用词，当且仅当 $w_i[1:j]s'$是 $(k-i+1)$-子序列通用词。

预处理。为了提高我们的排名算法的效率，我们首先概述在主要排名算法之前执行的预处理。令$Arch(w) = w_1, w_2, \ldots, w_k, v$。使用观察 1 中的记号，令 $A_1, A_2, \ldots, A_k$为满足以下条件的索引：$A_\ell$ 对应于 $w$ 中弓形 $w_\ell$ 首次出现在 $w$中的位置，即 $w_\ell = w[A_\ell, A_{\ell+1}-1]$。此外，令 $\Delta(w, A_\ell, i)$表示 $w_\ell$的长度为$i$的前缀中不同符号的数量。我们假设，对于每个$i \in [A_\ell, A_{\ell+1}-1]$，$\ell \in [0,k]$，$\Delta(w, A_\ell, i_\ell)$的值均已计算得出。

为了计算 $w$的每个后缀中自由符号的数量，令 $m$ 为一个长度为 $n$的数组，其中 $m[i]$ 包含 $w$的长度为 $i$的后缀中的自由符号数量。通过从 $i=n$开始，并按 $i$递减的顺序处理，可以计算出 $m[i]$ 的值。注意，$m[n]$的值等于 0。一般情况下，$m[i]$的值为 $m[i+1]$（如果 $w[i]$属于拱分解中某个弓形的通用子序列），否则为 $m[i+1]+1$。令 $\ell$ 为满足 $A_\ell \le i < A_{\ell+1}$的索引。注意，如果 $w[i]$ 位于 $w_\ell$的通用子序列中，则 $\Delta(w, A_\ell, i) = \Delta(w, A_\ell, i-1) + 1$；否则，$\Delta(w, A_\ell, i) = \Delta(w, A_\ell, i-1)$。因此，利用前述计算，对于每个$i \in [1,n]$，可以在 $O(n)$时间内确定 $m[i]$ 的值。

为了确定小于 $w[i]$ 的符号数量，需要额外构造一个长度为 $n$ 的数组$l$，使得 $l[i]$ 包含在 $w$ 中出现在 $A_\ell$ 和 $i$ 之间的符号集合，其中 $\ell$ 是满足 $A_\ell \le i < A_{\ell+1}$ 的索引。这补充了 $m$，确保能够快速检查给定符号 $x$ 是否已被 $w_\ell[1, i+1-A_\ell]$ 使用。数组 $l$ 可按如下方式在 $O(n\sigma)$ 时间内计算。对于每个 $i \in [1,n]$，注意到 $l[i]$ 的值要么为 $l[i-1] \cup \{w[i]\}$（如果对每个 $\ell \in [0,k]$ 都有 $i \ne A_\ell$），要么为 $\{w[i]\}$。将每个数组存储为长度为 $\sigma$ 的二进制向量，初始化最多需要 $O(n\sigma)$ 时间；如此一来，可在 $O(n)$ 时间内为每个$i \in [1,n]$ 计算出 $l[i]$ 的值。最后，我们假设 $\mathsf{CS}(q,m,c)$ 的值已针对每个 $q \in [1,\sigma]$、$m \in [n]$ 和 $c \in [k]$ 预先计算。

排名。现在我们已经掌握了对输入词 $w \in \Sigma^n$ 进行排名所需的工具。我们注意到，$w$不必是一个 $k$-子序列通用词，因此该工具可以用于更一般的情形。总体而言，我们的方法是依次取 $w$ 的每个前缀 $w[1,i]$，并计算 $\mathcal{U}(k,n,\sigma)$ 中按字典序小于以 $w[1,i]x$ 为前缀的 $w$ 的词的数量，其中 $x < w[i+1]$。将 $w$ 的每个前缀对应的此类词数相加，即可确定小于 $w$ 的词的总数。

设 $\mathcal{R}(i) = \{u \in \mathcal{U}(k,n,\sigma) \mid u < w, u[1,i] = w[1,i], u[i+1] < w[i+1]\}$，并令 $\ell$ 为满足 $A_\ell \le i < A_{\ell+1}$的索引。注意，符号 $u[i+1]$的可能取值数量等于 $w[i+1]-1$。此外，$\mathcal{R}(i)$ 中以 $w[1,i]x$ 为前缀的词的数量为某个

固定 $x < w[i+1]$ 等于CS($\Delta(w, A_\ell, i) + 1, m[i], k + 1 - \ell$)，如果 $x$ 位于 $w_\ell$的通用子序列中；否则等于 CS($\Delta(w, A_\ell, i)$ $m[i]-1, k+1-\ell$)。回想一下，数组 $l$ 在位置 $i$ 包含 $w$ $w[A_\ell, i]$的因子中的不同符号集合。因此， $\mathcal{R}(i)$ 的大小可以通过以下求和计算得出：

$$|\mathcal{R}(i)| = \sum_{x \in [1, w[i+1]-1]} \begin{cases} \mathrm{CS}(\Delta(w, A_\ell, i) + 1, m[i], k + 1 - \ell) & x \in l[i] \\ \mathrm{CS}(\Delta(w, A_\ell, i), m[i] - 1, k + 1 - \ell) & x \notin l[i] \end{cases}$$

使用 $\mathcal{R}(i)$， $w$ 在集合 $\mathcal{U}(n, k, \sigma)$ 中的排名，记为 $rank(w)$，其定义如下：

$$rank(w) = \sum_{i \in [0, n-1]} |\mathcal{R}(i)|$$

定理 3. 给定词 $w \in \Sigma^n$ 的排名可在 $O(nk\sigma)$ tim 内确定。 $e.$

证明。注意，对于任何形式为 $w[i]xv$ to be $k$-子序列通用的词，后缀 $v$必须属于 $\mathcal{S}(w[i]x)$。设 $\ell$ 为满足 $A_\ell \le i < A_{\ell+1}$的索引， $q$ 为 $w[A_\ell, i]$ 中的不同符号数量， $m$ 为 $w[1, i]x$之后的自由符号数量。注意， $v$ 的可能取值数量为CS($q + 1, m, k - \ell$)（如果 $x$ 不在$w[A_\ell, i]$的通用子序列中），或者为 CS($q, m - 1, k - \ell$)（如果 $x$ 已经出现在 $w[A_\ell, i]$中）。利用列表 $l$，可以在常数时间内确定符号 $x$ 是否出现在 $w[A_\ell, i]$中。进一步地，假设已预先计算出 CS($q, m, k - \ell$) 的值，则带有前缀 $w[i]x$ 的所有长度为 $n$的 $k$-子序列通用词的总数可以在常数时间内计算出来，因此 $\mathcal{R}(i)$ 的值可以在 $O(\sigma)$ 时间内计算出来。由于 $w$有 $n$ 个可能的前缀，因此在经过 $O(n\sigma k)$预处理后，$w$在 $\mathcal{U}(n, k, \sigma)$ 中的总排名可以在 $O(n\sigma)$ 时间内计算出来。 □

## 5 反排名

我们通过展示如何对长度为 $n$ 的 $k$-子序列通用词进行反排名，补充了计数和排名技术。请注意，高效的反排名技术可作为枚举所有 $k$-子序列通用词集合的有效工具。我们假设 CS($q, m, k$) 的值已针对每个 $q \in [1, \sigma]$、 $m \in [0, n]$ 和 $c \in [0, k]$ 预先计算。

我们的反排名过程以迭代方式运行。设 $w$ 为正在进行反排名的、排名为 $i$ 的词。从 $j = 1$ 开始，通过计数前缀为 $w[1, j-1]x$、且以 $x = 1$ 开头的长度为 $n$ 的 $k$-子序列通用词的数量，计算 $w[j]$ 的值，其中 $x \in \Sigma$。不断增大 $x$ 的值，直到前缀小于或等于 $w[1, j-1]x$ 的词的数量大于 $i$。计算出 $x$ 的值后，将 $w[j]$ 设为 $x - 1$，然后算法继续计算 $w[j+1]$ 的值。

定理4. 长度为 $n$ 、排名为 $i$ 的 $k$-子序列通用词 $w$ 可在 $O(n\sigma + nk\sigma)$ 时间内确定。

证明。从 $w[1]$ 开始，注意到，对于任意 $x \in \Sigma$， 以前缀 $x$ 开头的词的数量由 CS(1, $n - (k\sigma)$ $k$) 给出。此外，首个符号为 $x$ 的任何词，其排名范围为 $(x-1)$ CS(1, $n - (k\sigma)$ $k$) $+1$ 到 $x$ CS(1, $n - (k\sigma)$ $k$)。因此， $w[1]$ 的值就是满足 $(x-1)$ CS(1, $n - (k\sigma)$ $k$) $< i \le x$ CS(1, $n - (k\sigma)$ $k$) 的 $x$ 的值。

更一般地，令 $t(j)$ 表示以前缀 $w[1, j]$ 开头的词的最小排名，由以下总和确定：

$$t(j) = \sum_{\ell \in [1, j]} \sum_{x \in [1, w[\ell]-1]} |\mathcal{S}(w[1, \ell-1]x)| = t(j-1) + \sum_{x \in [1, w[j]-1]} |\mathcal{S}(w[1, j-1]x)|.$$

注意，因此可以使用 $t(j-1)$ 和 CS($q, m, c$) 的值，在 $O(\sigma)$ 时间内计算 $t(j)$ 的值。因而，$w[j+1]$ 的值是满足 $t(j) + \sum_{y \in [1, x-1]} |\mathcal{S}(w[1, j]y)| < i \le t(j) + \sum_{y \in [1, x]} |\mathcal{S}(w[1, j]y)|$ 的符号 $x$ ，并且还可以在 8 સમયમાં计算得出

$O(\sigma)$ 时间，使得 $w$ 的反排名总时间复杂度为 $O(n\sigma)$，经过 $O(n\sigma k)$ 预处理。

□

推论3. 长度为 $n$ 的 $k$-子序列通用词集合可以在经过 $O(nk\sigma)$ 预处理后，以 $O(n\sigma)$ 的延迟显式输出。

证明。根据定理4，每个索引 $i \in [1, |\mathcal{U}(n,k,\sigma)|]$ 均可在至多 $O(n\sigma k)$ 的预处理后，于 $O(n\sigma)$ 时间内反排名。因此，集合 $\mathcal{U}(n,\sigma,k)$ 可在经过 $O(nk\sigma)$ 预处理后，以 $O(n \cdot \sigma)$ 的延迟进行枚举。

□

## 6 结论

本文为理解 $k$-子序列通用词的空间提供了新的工具。值得注意的是，我们展示了如何使用高效算法对固定长度的词进行计数、排名、反排名和枚举。我们指出，所有这些算法都可以扩展到长度至多为 $n$ 的词的情形。本文提出了两个关键的开放问题。首先，是否存在一个通用公式，用于计算 $n$-长度的 $k$-子序列通用词的数量。事实上，如果能将这样的公式扩展到高效计算 $\mathcal{S}(v)$ 的大小，那么它或许可以加速排名、反排名和枚举算法的预处理。其次，是否存在一种枚举算法，能够在多项式时间预处理后，以至多 O($n$) 的延迟输出 $\mathcal{U}(n,k,\sigma)$ 中的每个词。

作者感谢 Leverhulme Trust 通过 Leverhulme 功能材料设计研究中心为本研究提供资助。此外，作者还感谢审稿人提出的宝贵意见，这些意见提升了本文的可读性。

## 参考文献

[Ada+21] Duncan Adamson、Argyrios Deligkas、Vladimir V. Gusev 和 Igor Potapov。《多项式时间内的手镯排名》。载于：第32届组合模式匹配年度研讨会（2021），第4–17页。[Ada 22] Duncan Adamson。《多项式时间内二进制无标号项链的排名》。载于：形式系统描述复杂性：第24届IFIP WG 1.02国际会议，DCFS 2022，匈牙利德布勒森，2022年8月29–31日，会议论文集。Springe

r.

2022，第15–29页。[Art+17] Alexander Artikis 等。“复杂事件识别语言：教程”。载于：第11届ACM分布式与事件系统国际会议论文集。2017，第7–10页。[Bar+20] LauraBarker 等。“词的散布因子通用性”。载于：语言理论进展：第24届国际会议，DLT2020，美国佛罗里达州坦帕，2020年5月11–15日，论文集。施普林格。2020，第14–28页。[Day+21] Joel D. Day等。“达到k-子序列通用性的编辑距离”。载于：第38届理论计算机科学国际研讨会（STACS 2021）。MarkusBl¨aser 和BenjaminMonmege 编。第187卷。信息学领域莱布尼茨国际论文集（LIPIcs）。德国达格斯图尔：Schloss Dagstuhl– 莱布尼茨信息学中心，2021，25:1–25:19。[FK18] Lukas Fleischer 和 ManfredKufleitner。“检验Simon 同余”。载于：第43届数学计算机科学基础国际研讨会（MFCS 2018）。Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik。2018。

[Fle+22] Pamela Fleischmann 等人。“近乎 k-通用词——研究 Simon 同余的一部分”。载于：形式系统描述复杂性：第24届IFIP WG 1.02国际会议，DCFS 2022，匈牙利德布勒森，2022年8月29日至31日，论文集。施普林格。2022年，第57–71页。[FM78] Harold Fredricksen 和 James Maiorana。“k种颜色的珠子项链与k元 de Bruijn序列”。载于：《离散数学》23.3（1978），第207–210页。[Gaw+21] Pawe l Gawrychowski 等人。“高效测试 Simon 同余”。载于：第38届理论计算机科学专题国际研讨会（STACS 2021）。Markus Bl¨aser 和 Benjamin Monmege 编。第187卷。莱布尼茨信息学论文集（LIPIcs）。德国达格斯图尔：Schloss Dagstuhl – 莱布尼茨信息学中心，2021年，34:1–34:18。[GR61] Edgar N. Gilbert 和 John Riordan。“周期序列的对称类型”。载于：《伊利诺伊数学杂志》5.4（1961），第657–665页。[Heb91] Jean-Jacques Hebrard。“一种通过子序列高效区分位字符串的算法”。载于：《理论计算机科学》82.1（1991），第35–49页。[HSZ17] Simon Halfon、Philippe Schnoebelen 和 Georg Zetzsche。“子词序上的一阶逻辑的可判定性、复杂性与表达能力”。载于：2017年第32届逻辑与计算机科学年度研讨会（LICS）。IEEE。2017年，第1–12页。[HWG20] Renmin Han、Sheng Wang 和 Xin Gao。“用于纳米孔原始信号中高效子序列搜索与映射、以实现靶向测序的新型算法”。载于：《生物信息学》36.5（2020），第1333–1343页。[Kos+21] Maria Kosche、Tore Koß、Florin Manea 和 Stefan Siemer。“词中的缺失子序列”。载于：《可达性问题》。Paul C. Bell、Patrick Totzke 和 Igor Potapov 编。Cham：施普林格国际出版，2021年，第115–131页。[Kos+22] Maria Kosche、Tore Koß、Florin Manea 和 Stefan Siemer。“子序列匹配的组合算法：综述”。载于：第12届非经典自动机模型及应用国际研讨会论文集，匈牙利德布勒森，2022年8月26–27日。Henning Bordihn、G´eza Horv´ath 和 Gy¨orgy Vaszil 编。第367卷。《理论计算机科学电子论文集》。开放出版协会，2022年，第11–27页。[KRR14] Tomasz Kociumaka、Jakub Radoszewski 和 Wojciech Rytter。“计算第 $k$个 Lyndon 词并解码按字典序最小的 de Bruijn序列”。载于：《组合模式匹配》。CPM 2014。《计算机科学讲义》，第8486卷。施普林格，2014年，第202–211页。[KS16] Prateek Karandikar 和 Philippe Schnoebelen。“分段可测试语言的高度及其在逻辑复杂性中的应用”。载于：第25届EACSL计算机科学逻辑年度会议（CSL 2016）。Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik。2016年。[Lot97] Monsieur Lothaire。《词组合学》。第17卷。剑桥大学出版社，1997年。[MSY04] Alexandru Mateescu、Arto Salomaa 和 Sheng Yu。“子词历史与 Parikh 矩阵”。载于：《计算机与系统科学杂志》68.1（2004），第1–21页。[Sav97] Carla Savage。“组合格雷码综述”。载于：《SIAM评论》39.4（1997），第605–629页。[Sha78] Alan C. Shaw。“带流表达式的软件描述”。载于：《IEEE软件工程汇刊》3（1978），第242–254页。

[Shi+19] Rayhan Shikder、ParimalaThulasiraman、PourangIrani和Pingzhao Hu。“一种基于OpenMP的生物信息学最长公共子序列查找工具”。载于：BMC研究札记12（2019），第1–6页。[Sim03] Imre Simon。“由其子词区分的词”。载于：WORDS2003 27（2003），第6–13页。[Sim75] ImreSimon。“分段可测试事件”。载于：《自动机理论与形式语言：第2届凯泽斯劳滕GI会议，1975年5月20日至23日》。施普林格。1975年，第214–222页。[SK19] Philippe Schnoebelen和Prateek Karandikar。“分段可测试语言的高度与子词逻辑的复杂性”。载于：计算机科学逻辑方法15（2019）。[SW17] Joe Sawada和AaronWilliams。“对项链、Lyndon词和de Bruijn序列进行排名的实用算法”。载于：离散算法杂志43（2017），第95–110页。[Tro03] Zdenˆek Troniˆcek。“公共子序列自动机”。载于：《自动机的实现与应用：第7届国际会议，CIAA2002，法国图尔，2002年7月3日至5日，修订论文集》。施普林格。2003年，第270–275页。[Zet16] Georg Zetzsche。“向下闭包比较的复杂性”。载于：《第43届国际自动机、语言与程序设计学术研讨会（ICALP 2016）》。Ioannis Chatzigiannakis、Michael Mitzenmacher、YuvalRabani和Davide Sangiorgi编。第55卷。莱布尼茨国际信息学会议录（LIPIcs）。德国达格斯图尔：达格斯图尔城堡—莱布尼茨信息学中心，2016年，123:1–123:14。