通过前缀无关解析加速 FM 索引查询

AaronHong[1] ！D

计算机与信息科学及工程系，赫伯特·韦特海姆工程学院
E工程系、佛罗里达大学、盖恩斯维尔、佛罗里达州、美国
MarcoOliva ！

计算机与信息科学及工程系、赫伯特·韦特海姆工程学院、佛罗里达大学、盖恩斯维尔、佛罗里达州、美国

Dominik Köppl
明斯特大学计算机科学研究所，48149 明斯特，德国
HideoBannai ！

M&D数据科学中心，东京医科齿科大学，日本东京，日本

Christina Boucher
佛罗里达大学赫伯特·韦特海姆工程学院计算机与信息科学及工程系，盖恩斯维尔，佛罗里达州，美国

Travis Gagie
加拿大新斯科舍省哈利法克斯市达尔豪斯大学计算机科学学院

arXiv:2305.05893v1 [cs.DS] 10 May 2023

摘要

例如，FM 索引是 DNA 比对中的一种关键数据结构，但使用它们进行搜索时，通常每个查询模式字符至少需要进行一次随机访问。Ferragina 和 Fischer [5]于 2007 年指出，基于词的索引通常比基于字符的索引需要更少的随机访问，因此能够支持更快的搜索。然而，由于 DNA 缺乏天然的词边界，在应用基于词的 FM 索引之前，必须以某种方式对其进行解析。去年，Deng 等人 [3]提出了一种通过诱导后缀排序来解析基因组数据的方法，并证明，当模式长度达到数千个字符或更长时，由此生成的基于词的 FM 索引所支持的计数查询速度快于标准 FM 索引。本文表明，与诱导后缀排序相比，采用前缀无关解析（其参数允许我们调整短语的平均长度）能够显著提高仅有几百个字符的模式的处理速度。我们实现了该方法，并证明，在针对 GRCh38 的查询中，其速度比竞争方法快 3 至 18 倍。此外，在针对 25,000、50,000 和 100,000 个 SARS-CoV-2基因组的查询中，该方法始终表现出更快的速度。因此，非常明确的是，与所有最先进的方法相比，我们的方法能够加快计数操作的性能，而内存占用仅略有增加。 `PFP-FM` 的源代码可在 `https://github.com/marco-oliva/afm`获取。

2012 年 ACM 学科分类 计算理论 → 模式匹配

关键词和短语 FM 索引、泛基因组学、可扩展性、基于词的索引、随机访问

数字对象唯一标识符 10.4230/LIPIcs.CVIT.2016.23

资助*Aaron Hong*：NIH/NHGRI 向 Ben Langmead 提供的 R01HG011392 项目资助，NSF/BIO 向 Christina Boucher 提供的 DBI- 2029552 项目资助
Marco Oliva：NIH/NHGRI 向 Ben Langmead 提供的 R01HG011392 资助，NSF/BIO 向 Christina Boucher 提供的 DBI-2029552 资助
Dominik Köppl：JSPS KAKENHI 资助编号 JP21K17701、JP22H03551 和 JP23H04378*Hideo Bannai*：JSPS KAKENHI 资助编号 JP20H04141

[1] 通讯作者

© Aaron Hong、Marco Oliva、Dominik Köppl、Hideo Bannai、Christiana Boucher 和 Travis Gagie；依据知识共享署名 4.0 国际许可协议授权。第 42 届重要主题会议（CVIT 2016）。编辑：John Q. Open 和 Joan R. Access；文章编号 23；页码 23:1–23:19。莱布尼茨国际信息学会议论文集，施洛斯·达格施图尔—莱布尼茨信息学中心，达格施图尔出版社，德国

LIPICS

Christina Boucher：NIH/NHGRI 资助 R01HG011392（资助对象为 Ben Langmead），NSF/BIO 资助 DBI- 2029552（资助对象为 Christina Boucher）
Travis Gagie：NIH/NHGRI 向 Ben Langmead 提供的 R01HG011392 资助、NSERC 向 Travis Gagie 提供的 RGPIN-07185-2020 资助、NSF/BIO 向 Christina Boucher 提供的 DBI-2029552 资助

1 引言

FM 索引 [4] 是生物信息学领域最著名的数据结构之一，因为它已应用于生物数据分析中的无数任务。鉴于这一数据结构的长期影响，Burrows、Ferragina 和 Manzini 荣获了2022年ACM Paris Kanellakis理论与实践奖[2]。它是重要读取比对器背后的数据结构——例如 Bowtie [9] 和 BWA [10] ——这些读取比对器接收一个或多个参考基因组，为这些基因组构建 FM 索引，并利用所得索引，在一组测序读段与参考基因组之间寻找短精确比对，随后可将其扩展为近似匹配 [9, 10]。简而言之，FM 索引由后缀数组（记为 SA）的一个采样以及 Burrows–Wheeler变换（BWT）数组组成。给定一个输入字符串 S和一个查询模式 Q，`count`可以利用 BWT 高效地支持查询，以回答 Q在 S中出现的最长匹配次数。要定位所有这些出现位置，则需要 SA 采样。因此，FM 索引能够高效地同时支持 `count` 和 `locate` 查询。下一节将对 SA 和 BWT 进行数学定义。

关于缩小 FM 索引规模的研究论文已经层出不穷（参见例如 [12, 8, 6]），关于加速查询的研究也同样如此。基本查询 `count` 返回模式 Q 在索引文本 S 中出现的次数，但通常至少需要对 S 的 BWT 进行 $|Q|$ 次随机访问，而这些访问通常比我们利用其返回的信息所执行的后续计算慢得多。更具体地说，`count` 查询 Q 使用 BWT 中 $|Q|$ 个位置的秩查询；如果我们使用一棵覆盖整个 BWT 的小波树来回答这些查询，那么每向下遍历小波树的一个层级，都可能需要进行一次随机访问，因此总共需要 $\Omega(|Q| \log \sigma)$ 次随机访问，其中 σ 是字母表的大小；如果我们将 BWT 划分为多个块，并为每个块 [8]，使用一棵独立的小波树，那么每次秩查询可能只需要一次或几次随机访问，但随机访问的总次数仍可能达到 $\Omega(|Q|)$。早在 2007 年，Ferragina 和 Fischer [5] 就研究了压缩索引对随机访问的依赖问题，并证明基于词的索引比基于字符的索引执行的随机访问更少：“最终基于词的后缀数组的空间缩减也影响了其查询时间（即二分搜索步骤中的随机访问次数更少！），速度最高可提升 3 倍。”

因此，加速基因组数据的随机访问（这一应用场景十分广泛）的一种可能方法，是将序列拆分为单词或短语。基于这一思路，Deng 等人在 2022年 [3] 使用最长匹配后缀（LMS）分解 [2] 将输入序列 S 拆分为短语。不幸的是，经过一轮 LMS 解析后得到的短语通常过短，因此只有当 Q 达到数千个字符时，才能获得加速效果。尚未解决的问题是，如何根据输入序列控制短语的长度，从而获得更长的短语，进一步显著提升随机访问的加速效果。

在这里，我们将前缀无关解析的概念应用于加速 FM 索引中的 `count` 这一问题。前缀无关解析使用滚动哈希，先选择触发字符串，再用这些触发字符串定义输入字符串 S的解析；也就是说，前缀无关解析将 S解析为若干短语，这些短语以触发字符串为起点和终点，且不包含其他触发字符串。所有

[2] https://awards.acm.org/kanellakis

唯一的短语按字典序排序，并存储在前缀无关解析的字典中，我们将其记为 D。前缀无关解析可以存储为 D 中各短语秩构成的有序列表。因此，前缀无关解析会将输入序列拆分为若干短语，其大小可通过选择触发字符串进行更灵活的控制。与 Deng 等人 [3] 所实现的方法相比，这带来了更加灵活的加速效果。

现在，假设我们为 S 构建一个 FM 索引，为解析 P 构建一个 FM 索引，并构建一个长度为 n 的位向量 B，其中的 1 标记 S 的 BWT 中紧邻短语边界之前的字符，也就是说，紧邻触发字符串之前的字符。我们注意到，所有的 1 最多聚集成与 S 中不同触发字符串数量相同的若干段。此外，只要短语的秩与短语本身的字典序一致，我们就可以利用该位向量，将 S 的 BWT 中以触发字符串开头的任意模式对应的区间映射到 P 的 BWT 中的相应区间，反之亦然。这意味着，给定查询模式 Q，我们可以在 S 的 FM 索引中逐字符对 Q 进行反向搜索，直到到达 Q 中最右侧触发字符串的左端；然后映射到 P 的 BWT 中，逐短语对 Q 进行反向搜索，直到到达 Q 中最左侧触发字符串的左端；接着映射回 S 的 BWT 中，最后继续逐字符进行反向搜索。

我们实现了这一方法，将其称为 `PFP-FM`，并使用从 NCBI 网站获取的 SARS-CoV-2基因组集合，以及具有不同查询字符串长度的基因组参考联盟人类基因组版本 38，与 `sdsl` [7]、`RLCSA` [15]、`RLFM` [12, 11]和 `FIGISS` [3] 中的 FM 索引实现进行了广泛比较。当我们使用 100,000 个 SARS-CoV-2基因组，将 `PFP-FM`与 `sdsl` 中的 FM 索引进行比较时，发现 `PFP-FM` 能够完成多出 2.1 至 2.8 倍的查询。此外，在 100,000 个 SARS-CoV-2基因组上，`PFP-FM` 分别比 `FIGISS`、`RLCSA`和`RLFM`快 64.38% 至 74.12%、59.22% 至 78.23% 以及 49.10% 至 90.70%。我们评估了`PFP-FM` 在基因组参考联盟人类基因组版本 38 上的性能，发现其速度分别是 `RLCSA`、`RLFM`和 `FIGISS`的 3.86 至 7.07 倍、2.92 至 18.07 倍以及 10.14 至 25.46 倍。就构建时间而言，在所有 SARS-CoV-2 数据集上，`PFP-FM` 的构建时间效率最高；在基因组参考联盟人类基因组版本 38 上，其构建速度位居第二。对于 SARS-CoV-2 数据集，所有方法在构建时使用的内存均少于 60 GB，因此可以在任何入门级商用服务器上完成构建，甚至包括 100,000 个 SARS-CoV-2 数据集的构建。对于基因组参考联盟人类基因组版本 38，所有方法的构建需要 26 GB 至 71 GB 内存，其中我们的方法使用的内存最多。总而言之，我们开发并实现了一种加速 FM 索引的方法，实现了 2 至 25 倍的加速，其中较长模式的加速效果最为显著。因此，加速的 FM 索引方法——例如本文开发的方法——非常适合在查询序列与参考数据库之间查找长度很长的匹配（长度为 125 至 1,000）。随着测序读段变得更长且更准确（即 Nanopore 数据），我们很快就能以超越传统基于 FM 索引的比对方法的效率，将长读段比对到参考数据库。源代码公开于 `https://github.com/marco-oliva/afm`。

2 预备知识

2.1 基本定义

长度为 n 的字符串 S 是字母表 $\Sigma = \{c_1, \ldots, c_\sigma\}$ 上由符号 $S = S[0..n-1] = S[0] \cdots S[n-1]$ 构成的有限序列。我们假设这些符号可以明确排序。我们用 ε 表示空字符串，并将 S 的长度记为 $|S|$。给定字符串 S，我们记为

S的逆序形式为 $rev(S)$，即 $rev(S) = S[n-1]\cdots S[0]$。

我们用 $S[i..j]$ 表示从位置 i开始、到位置 j结束的 S的子串 $S[i]\cdots S[j]$，其中，如果 $i > j$，则 $S[i..j] = \varepsilon$。对于字符串 S 和 $0 \le i < n$，$S[0..i]$ 称为 S的第 i个前缀，$S[i..n-1]$ 称为 S的第 i个后缀。如果 $0 \le i < n-1$，则称 S的前缀 $S[0..i]$ 为真前缀。类似地，如果 $0 < i < n$，则称 S的后缀 $S[i..n-1]$ 为真后缀。

给定字符串 S、符号 $c \in \Sigma$和整数 i，我们将 $S.\mathsf{rank}_c(i)$（若上下文明确，也简写为 `rank`）定义为 $S[0..i-1]$中 c的出现次数。我们还将 $S.\mathsf{select}_c(i)$ 定义为 $\min(\{j-1 \mid S.\mathsf{rank}_c(j) = i\} \cup \{n\})$，即如果 c在S中存在第 i次出现，则其在 S中的位置；否则为 n。对于位向量 $B[0..n-1]$（即由 $\Sigma = \{0,1\}$ 构成的字符串），为简化记号，我们分别将 $B.\mathsf{rank}_1(i)$ 和 $B.\mathsf{select}_1(i)$ 记为 $B.\mathsf{rank}(i)$ 和 $B.\mathsf{select}(i)$。

2.2 后缀数组、BWT 和反向搜索

我们将给定字符串 $S[0..n-1]$ 的后缀数组 [13] 记为 SA_S，并将其定义为集合$\{0,\ldots,n-1\}$的一个排列，使得 $S[\mathsf{SA}_S[i]..n-1]$ 是 S中字典序第 i 小的后缀。当上下文明确时，我们将 SA_S 简称为 SA。出于技术原因，我们假设输入字符串的最后一个符号为 $S[n-1] = \$$，该符号不会在字符串的其他位置出现，且小于任何其他符号。

我们考虑矩阵 W，它包含 S 的所有排序后的循环移位，称为 S 的 BWT 矩阵，并令 F 和 L 分别为矩阵的第一列和最后一列。最后一列定义了 BWT 数组，即 BWT =L。现在令 $\mathsf{C}[c]$ 为以小于 c 的字符开头的后缀数量。我们将 LF 映射定义为 $\mathsf{LF}(i, c) = \mathsf{C}[c] + \mathsf{rank}_c(\mathsf{BWT}, i)$，以及 $\mathsf{LF}(i) = \mathsf{LF}(i, \mathsf{BWT}[i])$。利用 LF 映射，可以从 BWT 重建字符串$S$。实际上，只需设置一个迭代器 $s = 0$ 和 $S[n-1] = \$$，并对每个 $i = n-2, \ldots, 0$ 执行 $S[i] = \mathsf{BWT}[s]$ 和 $s = \mathsf{LF}(s)$ 即可。LF 映射还可用于支持 `count`，方法是执行反向搜索，下面将对此进行介绍。

给定长度为 m 的查询模式 Q，*反向搜索* 算法包含 m 个步骤，并保持以下不变量：在第 i 步，p 存储 W 中以 $Q[i,m]$ 为前缀的第一行的位置，而 q存储 W 中以 $Q[i,m]$ 为前缀的最后一行的位置。要从 i 推进到 $i-1$，我们对 p 和 q 使用 LF 映射，$p = \mathsf{C}[c] + \mathsf{BWT}$。$\mathsf{rank}_c(p)$ 和$q = \mathsf{C}[c] + \mathsf{BWT}$。$\mathsf{rank}_c(q+1) - 1$。

2.3 FM 索引和计数查询

给定查询字符串 $Q[0..m-1]$ 和输入字符串 $S[0..n-1]$，有两个基本查询：(1) `count`，用于统计 Q 在 S 中出现的次数；(2) `locate`，用于查找这些匹配在 S中的位置。Ferragina 和 Manzini [4] 指出，将后缀数组与 BWT 结合后，可以高效地支持 `count` 和 `locate`。简而言之，在 BWT 上进行反向搜索，以找到 Q 在 S 中出现位置的字典序范围；该范围的大小等于 `count`。此范围内的后缀数组位置就是这些出现位置在 S 中的位置。

2.4 前缀无关解析

如前所述，前缀无关解析 (PFP)以字符串 $S[0..n-1]$以及正整数 w 和 p作为输入，并生成 S 的解析（记为 P），以及该解析中所有唯一子串（或短语）组成的字典（记为 D）。下面简要介绍生成该字典和解析的算法。首先，我们令 T 为一个任意的

w长度的、由 Σ 上的字符构成的字符串集合，并称其为触发字符串集合。如前所述，我们假设 $S[n-1] = \$$，并认为 S 是循环的，也就是说，对于所有 i，$S[i] = S[i \bmod n]$。此外，我们假设 $\$S[0..w-2] = S[n-1..n+w-2] \in T$，即以 $ 开始、长度为 w 的子串是一个触发字符串。

我们将字典 D $= \{d_1, .., d_{|D|}\}$ 定义为 S 的一个（按字典序排序的）最大子串集合，并满足以下条件：对于每个 d_i：i）恰有一个真前缀d_i包含在 T中；ii）恰有一个真后缀 d_i包含在 T中；iii）且 d_i 的其他任何子串都不在 T中。该定义的一个重要推论是，D 是前缀无关的，即对于任意 $i \neq j$，d_i 都不能是 d_j的前缀。

由于我们假设 $S[n-1..n+w-2] \in T$，因此可以通过扫描 $S' = \$S[0..n-2]S[n-1..n+w-2]$ 来查找 T的所有出现位置，并将S' 中每个以触发字符串为起点和终点的子串加入 D，其中包括起始和结束触发字符串。我们还可以构造 D 在 S'中的出现位置列表，该列表定义了解析 P。

我们通过长度为 w的字符串的 Karp-Rabin 指纹 f 来选择 T。我们在 S'上滑动一个长度为 w 的窗口，对于 S'中每个长度为 w 的子串 r，当且仅当 $f(r) \equiv 0 \pmod{p}$ 或者 $r = S[n-1..n+w-2]$时，将 r 纳入 T。令 $0 = s_0 < \cdots < s_{k-1}$ 为 S' 中的位置，使得对于任意 $0 \leq i < k$，都有 $S'[s_i..s_i+w-1] \in T$。字典为 D $= \{S'[s_i..s_{i+1}+w-1] \mid i = 0, \ldots, k-1\}$，解析定义为子串 $S'[s_0..s_1+w-1], \ldots, S'[s_{k-2}..s_{k-1}+w-1]$在 D 中的字典序排名序列。

例如，假设我们有 $S' = \$\texttt{AGACGACT\#AGATACT\#AGATTCGAGACGAC\$A}$，其中触发字符串以红色、蓝色或绿色突出显示。那么，假设 $ < # < A < C < G < T，我们有 D $= \{\texttt{\$AGAC, AC\$A, ACGAC, ACT\#AGATAC, ACT\#AGATTC, TCGAGAC}\}$ 和 P $= 0, 2, 3, 4, 5, 2, 1$。

3 方法

如前所述，我们将使用前缀无关解析来构建基于词的 FM 索引，并通过参数 w 和 p控制短语的长度。为了说明我们的数据结构，我们将先介绍其各个组成部分，然后再说明如何以比标准 FM 索引更高效的方式支持 `count` 查询。

3.1 数据结构设计

通过示例最容易解释我们的两级设计，因此请考虑一个文本

$$S[0..n-1] = \texttt{TCCAGAAGAGTATCTCCTCGACATGTTGAAGACATATGAT\$}$$

长度为 $n = 41$，并以一个特殊的字符串结束字符 `$` 结尾，该字符按字典序小于字母表中的其余字符。假设我们使用 $w = 2$ 和 Karp-Rabin 哈希函数对 S 进行解析，使得在 S 中出现的普通触发字符串为 `AA`、`CG` 和 `TA`。我们将 S 视为循环的，并且有 $\$S[0..w-2] = \$\texttt{T}$ 作为特殊触发字符串，因此字典 D 为

$$\mathsf{D}[0..5] = \{\texttt{\$TCCAGAA, AAGACATA, AAGAGTA, CGACATGTTGAA, TATCTCCTCG, TATGAT\$T}\},$$

其中短语按字典序排序。（回想一下，S 中相邻的短语有 $w = 2$ 个字符重叠。）如果我们从 `$` 开始解析，那么 S 的前缀无关解析为

$$\mathsf{P}[0..5] = (0, 2, 4, 3, 1, 5),$$

其中，P 中的每个元素（或短语 ID）都是 D 中该短语的字典序排名。

0	2	4	3	1	5	$TCCAGAAGAGTATCTCCTCGACATGTTGAAGACATATGAT
1	5	0	2	4	3	AAGACATATGAT$TCCAGAAGAGTATCTCCTCGACATGTTG
2	4	3	1	5	0	AAGAGTATCTCCTCGACATGTTGAAGACATATGAT$TCCAG
3	1	5	0	2	4	CGACATGTTGAAGACATATGAT$TCCAGAAGAGTATCTCCT
4	3	1	5	0	2	TATCTCCTCGACATGTTGAAGACATATGAT$TCCAGAAGAG
5	0	2	4	3	1	TATGAT$TCCAGAAGAGTATCTCCTCGACATGTTGAAGACA

图 1 我们前缀无关解析的 BWT 矩阵 P（**左**），以及以触发字符串开头的 S 的循环移位（**右**），按字典序排列。

接下来，我们考虑 P 的 BWT 矩阵。图 1 展示了我们示例中 P 的 BWT 矩阵。注意，由于 S 中只有一个 `$`，因此 P 中也只有一个 0；我们可以将这个 0 视为与 S 中的 `$` 对应的 P（某个适当旋转形式）的字符串结束字符。若取该矩阵的第 i 行，并将短语 ID 替换为短语，同时合并重叠部分，那么便可得到 S 中字典序第 i 小、以触发字符串开头的循环移位，如图右侧所示。这是我们稍后将用到的关键见解之一。

▶**引理 1**。 P 的循环旋转的字典序与其对应的 S 的循环旋转的字典序一致。

证明。 P 中的字符是充当元字符的短语 ID。由于元字符继承了其底层字符的字典序排名，并且短语具有前缀无关性，P 的后缀数组会以与P 的后缀数组相同的方式排列 S 中短语的 S。这意味着BWT 中 S 的短语顺序与它们在P中的对应短语 ID 顺序相同。◀

接下来，我们令 $B[0..n-1]$ 为一个位向量，用于标记这些循环移位在 S 的所有循环移位中的字典序排名，即它们在 S 的 BWT 矩阵各行中的位置。图 2 展示了 S 的后缀数组、BWT 矩阵和 BWT，以及 B；我们用红色标出 BWT（矩阵的最后一列）以及图 1 中的循环移位。请注意，对于 S 中每个不同的触发字符串，B 至多包含一段连续的 1，因此在实践中通常高度适合游程编码压缩。

除了位向量，我们还存储一个作用于短语的哈希函数 h，以及一个映射 M，将 D 中短语的哈希值映射到这些短语的字典序排名，也就是它们的短语 ID；M 在接收到任何其他键时都会返回 NULL。因此，我们总共构建了：S 的 FM 索引、P 的 FM 索引、标记循环旋转的位向量 B、作用于短语的哈希函数 h，以及映射 M。对于我们的示例，假设

$$
\begin{aligned}
h(\texttt{\$TCCAGAA}) &= 91785 & M(91785) &= 0\\
h(\texttt{AAGACATA}) &= 34865 & M(34865) &= 1\\
h(\texttt{AAGAGTA}) &= 49428 & M(49428) &= 2\\
h(\texttt{CGACATGTTGAA}) &= 98759 & M(98759) &= 3\\
h(\texttt{TATCTCCTCG}) &= 37298 & M(37298) &= 4\\
h(\texttt{TATGAT\$T}) &= 68764 & M(68764) &= 5
\end{aligned}
$$

且对于任何其他 x 值，$M(x) = \text{NULL}$。

如果我们选择的 h 范围足够大，那么仍然可以用与 D 中短语数量成正比且系数相对恒定的空间存储 M，并以较高概率在常数时间内计算 $M(h(\cdot))$，但该概率可以忽略不计

i	后缀数组$[i]$	$B[i]$	T[后缀数组$[i]$..(后缀数组$[i]-1$) 模 n]	BWT$[i]$
0	40	1	$TCCAGAAGAGTATCTCCTCGACATGTTGAAGACATATGAT	T
1	28	1	AAGACATATGAT$TCCAGAAGAGTATCTCCTCGACATGTTG	G
2	5	1	AAGAGTATCTCCTCGACATGTTGAAGACATATGAT$TCCAG	G
3	31	0	ACATATGAT$TCCAGAAGAGTATCTCCTCGACATGTTGAAG	G
4	20	0	ACATGTTGAAGACATATGAT$TCCAGAAGAGTATCTCCTCG	G
5	3	0	AGAAGAGTATCTCCTCGACATGTTGAAGACATATGAT$TCC	C
6	29	0	AGACATATGAT$TCCAGAAGAGTATCTCCTCGACATGTTGA	A
7	6	0	AGAGTATCTCCTCGACATGTTGAAGACATATGAT$TCCAGA	A
8	8	0	AGTATCTCCTCGACATGTTGAAGACATATGAT$TCCAGAAG	G
9	38	0	AT$TCCAGAAGAGTATCTCCTCGACATGTTGAAGACATATG	G
10	33	0	ATATGAT$TCCAGAAGAGTATCTCCTCGACATGTTGAAGAC	C
11	11	0	ATCTCCTCGACATGTTGAAGACATATGAT$TCCAGAAGAGT	T
12	35	0	ATGAT$TCCAGAAGAGTATCTCCTCGACATGTTGAAGACAT	T
13	22	0	ATGTTGAAGACATATGAT$TCCAGAAGAGTATCTCCTCGAC	C
14	2	0	CAGAAGAGTATCTCCTCGACATGTTGAAGACATATGAT$TC	C
15	32	0	CATATGAT$TCCAGAAGAGTATCTCCTCGACATGTTGAAGA	A
16	21	0	CATGTTGAAGACATATGAT$TCCAGAAGAGTATCTCCTCGA	A
17	1	0	CCAGAAGAGTATCTCCTCGACATGTTGAAGACATATGAT$T	T
18	15	0	CCTCGACATGTTGAAGACATATGAT$TCCAGAAGAGTATCT	T
19	18	1	CGACATGTTGAAGACATATGAT$TCCAGAAGAGTATCTCCT	T
20	13	0	CTCCTCGACATGTTGAAGACATATGAT$TCCAGAAGAGTAT	T
21	16	0	CTCGACATGTTGAAGACATATGAT$TCCAGAAGAGTATCTC	C
22	27	0	GAAGACATATGAT$TCCAGAAGAGTATCTCCTCGACATGTT	T
23	4	0	GAAGAGTATCTCCTCGACATGTTGAAGACATATGAT$TCCA	A
24	30	0	GACATATGAT$TCCAGAAGAGTATCTCCTCGACATGTTGAA	A
25	19	0	GACATGTTGAAGACATATGAT$TCCAGAAGAGTATCTCCTC	C
26	7	0	GAGTATCTCCTCGACATGTTGAAGACATATGAT$TCCAGAA	A
27	37	0	GAT$TCCAGAAGAGTATCTCCTCGACATGTTGAAGACATAT	T
28	9	0	GTATCTCCTCGACATGTTGAAGACATATGAT$TCCAGAAGA	A
29	24	0	GTTGAAGACATATGAT$TCCAGAAGAGTATCTCCTCGACAT	T
30	39	0	T$TCCAGAAGAGTATCTCCTCGACATGTTGAAGACATATGA	A
31	10	1	TATCTCCTCGACATGTTGAAGACATATGAT$TCCAGAAGAG	G
32	34	1	TATGAT$TCCAGAAGAGTATCTCCTCGACATGTTGAAGACA	A
33	0	0	TCCAGAAGAGTATCTCCTCGACATGTTGAAGACATATGAT$	$
34	14	0	TCCTCGACATGTTGAAGACATATGAT$TCCAGAAGAGTATC	C
35	17	0	TCGACATGTTGAAGACATATGAT$TCCAGAAGAGTATCTCC	C
36	12	0	TCTCCTCGACATGTTGAAGACATATGAT$TCCAGAAGAGTA	A
37	26	0	TGAAGACATATGAT$TCCAGAAGAGTATCTCCTCGACATGT	T
38	36	0	TGAT$TCCAGAAGAGTATCTCCTCGACATGTTGAAGACATA	A
39	23	0	TGTTGAAGACATATGAT$TCCAGAAGAGTATCTCCTCGACA	A
40	25	0	TTGAAGACATATGAT$TCCAGAAGAGTATCTCCTCGACATG	G

图 2 T 的后缀数组、BWT 矩阵和 BWT，以及位向量 B，其中 1 表示矩阵中以触发字符串开头的行。BWT 和标记为 1 的列均以红色突出显示。

即对于任何不在 D 中的特定字符串 γ， $M(h(\gamma)) \neq$ NULL。这意味着在实践中，我们可以使用 $M(h(\cdot))$ 作为 D 的成员字典，而无需存储 D 本身。

3.2 查询支持

接下来，给定我们上面定义的数据结构，我们描述如何支持给定模式 Q 的 `count` 查询。我们首先使用解析 S 时所用的相同 Karp-Rabin 哈希来解析 Q，这意味着我们将拥有与之前相同的所有触发字符串，但可能还会有一些未在 S 中出现的额外触发字符串。然而，我们不会将 Q 视为循环的，也不会假设存在一个字符串末尾符号来确保 Q 以触发字符串开头和结尾。

如果 Q 是 S 的子串，那么，由于 Q 包含其在 S 中对应出现位置的相同触发字符串，由 Q 中触发字符串诱导的短语序列必定是 S 中短语序列的子串。再结合 Q 的前缀和后缀——它们分别是 S 中共享短语左侧和右侧短语的后缀和前缀——我们将其称为 Q 的部分编码，正式定义如下。

▶ **定义 2**（部分编码）。给定 S 的一个子串 $S[i..j]$，$S[i..j]$ 的部分编码 定义如下：如果 $S[i..j]$ 中没有出现触发字符串，那么 $S[i..j]$ 的部分编码就是 $S[i..j]$ 本身。否则，$S[i..j]$ 的部分编码由以下部分连接而成：*(1)* $S[i..j]$ 中最短的前缀 α，该前缀不以触发字符串开头，且以触发字符串结尾；*(2)* 完全包含在 $S[i..j]$ 中的短语所对应的短语 *ID* 序列；*(3)* $S[i..j]$ 中最短的后缀 β，该后缀以触发字符串开头，且不以触发字符串结尾。

因此，部分编码将 $S[i..j]$ 划分为一个前缀 α、一列短语 ID 和一个后缀 β。如果 $S[i..j]$ 以触发字符串开头（分别以触发字符串结尾），那么 α（分别是 β）就是空字符串。

解析 Q 的时间复杂度与 Q 的长度呈线性关系。

▶**引理 3.** 我们可以使用一种数据结构来表示 M，其所需空间（以字为单位）与 D 中不同短语的数量成正比。给定一个查询模式 Q，如果 Q 未出现在 S 中，该数据结构将以高概率返回 *NULL*。否则（Q 出现在 S 中），它将返回 Q 的部分编码。在这两种情况下，该查询所需的时间均为 $O(|Q|)$。

证明。 我们保存 D中短语的 Karp-Rabin（KR）哈希值，其中 KR 哈希函数的映射范围为 $[1..n^3]$ 因此每个哈希值都只需 $O(\log n)$ 位。我们还保存一个常数时间映射（通过哈希表实现，该哈希表使用的哈希函数对 D中的短语而言是完美的），将 D 中短语的 KR 哈希值映射到其 ID；对于任何不是 D 中短语 KR 哈希值的输入，该映射都会返回 NULL。我们令 M 为该映射与 KR 哈希函数的复合。

给定 Q，我们对其进行扫描以查找其中的触发字符串，并将其转换为由以下部分组成的子串序列：(a) α，即 Q 的前缀，在 Q 中第一个触发字符串的右端结束；(b) 一系列 PFP 短语，每个短语都以一个触发字符串开头和结尾，且中间没有触发字符串；以及 (c) β，即 Q 的后缀，从 Q 中最后一个触发字符串的左端开始。

我们对 (b) 中的每个完整短语应用 M。如果 M 对 (b) 中的任何完整短语返回 NULL，那么 Q 就不是 S 的子串，因此我们返回 NULL；否则，我们返回 α、M，即为 (b) 中各短语返回的短语 ID 序列，以及 β。

请注意，如果 Q 中的某个短语也存在于 S 中，那么 M 会将其映射为 D 中的字典序排名；否则，Q 中存在但不在 D 中的任意特定短语的 KR 哈希发生冲突的概率】【。

D 中某个短语的 KR 哈希，其值至多为 $n/n^3 = 1/n^2$。因此，如果 Q 是 S的子串，则我们返回 Q 的部分编码；否则，我们以高概率返回 NULL。◀

▶**推论 4**。 如果我们允许以高概率达到 $O(|Q|)$ 查询时间，那么我们可以修改 M，使其对于 Q 不在 S中时始终报告 *NULL*。

证明。 我们将哈希表中存储的每个 Karp-Rabin（KR）哈希值与其短语的实际字符一并存储，这样我们便能逐个字符检查匹配到的短语 Q 是否确实属于 D。如果发生冲突，我们就重新计算 D 的 KR 哈希值并重建哈希表。这是可行的，因为我们可以为D中的短语选择不同的 Karp-Rabin 指纹。◀

接着上面的示例，假设触发字符串为 `AA`、 `CG` 和 `TA`，并且给定一个查询模式 Q，

$Q[0..34] =$ `CAGAAGAGTATCTCCTCGACATGTTGAAGACATAT`

我们可以计算解析 Q ，从而得到以下内容

`CAGAA, AAGAGTA, TATCTCCTCG, CGACATGTTGAA, AAGACATA, TAT.`

接下来，我们使用 $M(h(\cdot))$ 将对 Q 的解析所得的完整短语映射到它们的短语 ID——也就是它们在 D 中的秩。如果任何完整短语映射到 NULL，那么我们就知道 Q 不会出现在 T 中。以我们的示例为例，得到部分编码

`CAGAA`,2,4, 3, 1, `TAT`.

接下来，我们考虑所有可能的情况。首先，考虑解析 Q 时最后一个子串β 以触发字符串结尾的情况，这意味着它是一个完整短语。此时，我们可以立即在 P 的 FM 索引中开始对 Q 的解析进行反向搜索。接下来，如果 β 不是一个完整短语，我们就在 S 的 FM 索引中对 β 进行反向搜索。如果对 β 的这次反向搜索没有返回任何结果，那么我们就知道 Q 不会出现在 S 中。如果对 β 的反向搜索返回了 BWT 中的一个区间，且该区间不包含在某个触发字符串的 BWT 区间内，那么 β 就不是以触发字符串开头，因此 $Q = \beta$，并且我们已经完成对 Q 的反向搜索。

最后，我们考虑这样的情况： β 是一个短语的真前缀，而对 β 进行反向搜索会返回一个 BWT_S 区间，该区间包含在某个触发字符串对应的 BWT_S 区间内。在我们的例子中，$\beta =$ `TAT` ，我们在 S 的 FM 索引中对 β 进行反向搜索，得到区间 $\text{BWT}_S[31..32]$，它是触发字符串 `TA`对应的区间。接下来，我们使用 B 将 BWT_S 中 β 对应的区间映射到 BWTP 中的一个区间，该区间对应于以 β开头的 S 的循环移位。

▶**引理 5**。 我们可以使用数据结构 B ，其空间（以字为单位）与 S 中不同触发字符串的数量成正比，并且利用该数据结构，

- 给定 S 中以字符串 β 开头的后缀的字典序范围，其中 β 以一个触发字符串开头且不包含其他触发字符串，我们可以在 $O(\log\log n)$ 时间内找到以从 β 开始的短语开头的 P 的后缀的字典序范围；给定 P 的一个后缀字典序范围，使得对应的 S 后缀都以相同的触发字符串开头，我们可以在 $O(\log\log n)$ 时间内找到这些对应的 S 后缀的字典序范围。

证明。 设 $B[0..n-1]$ 为一个位向量，其中 1 标记以触发字符串开头的 S 后缀的字典序排名。 B 中的 1 的连续段数量至多等于 S中不同触发字符串的数量，因此我们可以用与该数量成正比的空间存储它，并在 $O(\log\log n)$ 时间内支持 rank 和 select 操作。

如果 $\mathsf{BWT}_S[i..j]$ 包含字符串 β 在 S 中紧邻其出现位置之前的字符，并且该字符串以触发字符串开头且不包含其他触发字符串，那么 $\mathsf{BWTP}^{[B.\text{秩}_1(i)..B.\text{秩}_1(j)]}$ 包含 P 中紧邻以 β 开头的短语 ID 之前的短语 ID。

如果 $\mathsf{BWTP}^{[i..j]}$ 包含 P 中紧邻以下后缀的短语 ID，且这些后缀在 S 中对应的后缀均以相同的触发字符串开头，那么 $\mathsf{BWT}_S[B.\text{选择}_1(i+1)..B.\text{选择}_1(j+1)]$ 就包含 S 中对应后缀紧邻前面的字符。

正确性由引理 1 得出。 ◀

继续我们的示例，映射 $\mathsf{BWT}_S[31..32]$ 得到以下区间：

$$\mathsf{BWTP}^{[B.\text{秩}_1(31)\ B.\text{秩}_1(32)]} = \mathsf{BWTP}^{[4..5]}$$

如图 1 所示。从 BWTP 中的这个区间出发，我们现在在 FM 索引中针对 P 进行反向搜索，搜索对象是 Q 的解析结果中由完整短语 ID 组成的序列。在我们的示例中，区间为 $\mathsf{BWTP}^{[4..5]}$，由此得到以下短语 ID：2 4 3 1。

如果在 FM 索引中对 P 进行反向搜索后没有返回结果，那么我们就知道 Q 不会出现在 S 中。否则，它会返回 BWTP 中的一个区间，该区间对应于以 Q 的第一个完整短语开头的 Q 的后缀所起始的 S 的循环移位。在我们的示例中，如果从 $\mathsf{BWTP}^{[4..5]}$ 开始，对 2 4 3 1 进行反向搜索，那么我们会得到 $\mathsf{BWTP}^{[2],}$，它对应于该循环移位。

```
AAGAGTATCTCCTCGACATGTTGAAGACATATGAT$TCCAG
```

以 S 开头的后缀

```
AAGAGTATCTCCTCGACATGTTGAAGACATAT
```

Q 的解析结果为 $2, 4, 3, 1,$ `TAT`.

为了完成对 Q 的搜索，我们使用 B 将 BWTP 中的区间映射到 BWT_S 中对应的区间。该区间是 BWT 矩阵中以我们迄今为止搜索的 S Q 后缀开头的各行所对应的区间。在我们的示例中，$\mathsf{BWTP}^{[2]}$ 映射到

$$\mathsf{BWT}_S[B.\mathsf{select}_1(2+1)] = \mathsf{BWT}_S[2].$$

请注意，我们的示例中，BWT 区间仅包含一个条目，这是因为示例规模很小；但一般情况下，这些区间会更长。如果解析 Q 时得到的解析结果中的第一个子串 α 是一个完整短语，那么对 Q 的反向搜索就已完成。否则，我们从 BWT_S 中的这个区间开始，在 S 的 FM 索引中反向搜索 α，但忽略 α 最后的 w 个字符（我们已经搜索过这些字符，因为它们也包含在解析 Q 时得到的下一个短语中）。

在我们的示例中， $\alpha =$ `CAGAA` 因此，从 $\mathsf{BWT}_S[2]$ 开始，我们对 `CAG` 进行反向搜索，得到区间 $\mathsf{BWT}_S[14]$。如图 2 所示，

$S[\text{后缀数组}[4]..n] = S[2..n] =$ `CAGAAGAGTATCTCCTCGACATGTTGAAGACATATGAT$`

确实以.....开头

$Q =$ `CAGAAGAGTATCTCCTCGACATGTTGAAGACATAT`.

至此，我们对 count的说明结束。

最后，我们说明一些直观原因，以解释为什么我们预计两级 FM 索引在实践中会比标准反向搜索更快。首先，我们注意到，标准反向搜索的时间与 Q 的长度呈线性关系，并且通常对 Q 中的每个字符至少使用一次随机访问。而前缀无关解析 Q 的时间虽也呈线性关系，但不使用随机访问；对 S 的 FM 索引进行反向搜索与标准反向搜索相同，但我们仅将其用于 Q 的解析中的第一个和最后一个子串。对 P 的 FM 索引进行反向搜索时，解析中的每个完整短语大约需要 lg |D| 次随机访问：P 的 BWT 基于一个有效字母表，其大小等于 D 中短语的数量。因此，用于支持该 BWT 上秩查询的平衡小波树的深度应约为 lg |D| ，并且每一层树最多大约需要进行一次随机访问。

总之，如果我们能够找到前缀无关解析参数 w 和 p 的设置，使得

- 大多数查询模式会跨越多个短语，而这些模式中的大多数短语都相当长，lg |D| 明显小于这些短语的平均长度，因此，通过减少随机访问次数所节省的开销，应该足以抵消解析 Q 所增加的额外成本。

4 结果

我们实现了该算法，并测量了其相对于所有已知竞争方法的性能。所有实验均在一台配备 AMD EPYC 75F3 中央处理器、运行 Red Hat Enterprise Linux 7.7（64 位，内核 3.10.0）的服务器上进行。编译器为 g++ version 12.2.0。运行时间和内存使用量由 Snake Make 基准测试工具 [14]记录。我们将内存限制设为 128 GB，时间限制设为 24 小时。

数据集。 我们使用了以下数据集。首先，我们考虑了从 NCBI 网站获取的 SARS-CoV-2 基因组集合。其次，我们使用了来自 EMBL-EBI 的 COVID-19 数据门户的三个集合，分别包含 25,000、50,000 和 100,000 个 SARS-CoV-2 基因组。每个集合都是前一个集合的超集。我们将它们分别记为 `SARS-25k`、 `SARS-50k` 和 `SARS-100k`。接下来，我们考虑了一个人类参考基因组，记为 `GRCh38`，该基因组从 NCBI 下载。我们在表1中以每个数据集包含的字符数报告其大小。我们用 n 表示字符数。

实现。 我们使用 C++ 11 和 `sdsl-lite` 库 [7]实现了我们的方法，并扩展了 Oliva 的前缀无关解析方法，其源代码可在此处公开获取 `https://github.com/marco-oliva/pfp`。 `PFP-FM` 的源代码可在`https://github.com/marco-oliva/afm`获取。

竞争方法。 我们将 `PFP-FM` 与以下方法进行了比较： `sdsl-lite` 库中的标准 FM 索引、[7]、 `RLCSA` [15]、 `RLFM` [12, 11] 和 `FIGISS` [3]。值得注意的是， `RLCSA` 和 `FIGISS` 提供了公开的源代码，而 `RLFM` 仅提供可执行文件。我们通过从输入文件中随机选取 1,000 个指定长度的字符串，对每个查询模式执行 `count` 操作，并测量所有待比较方法的耗时。值得注意的是， `FIGISS` 和 `RLCSA` 仅支持 `count` 查询，且字符串必须在输入文本文件中提供。更具体地说，原始的 `FIGISS` 实现支持将整个文件内容视为单个模式进行计数。为克服这一限制，我们修改了

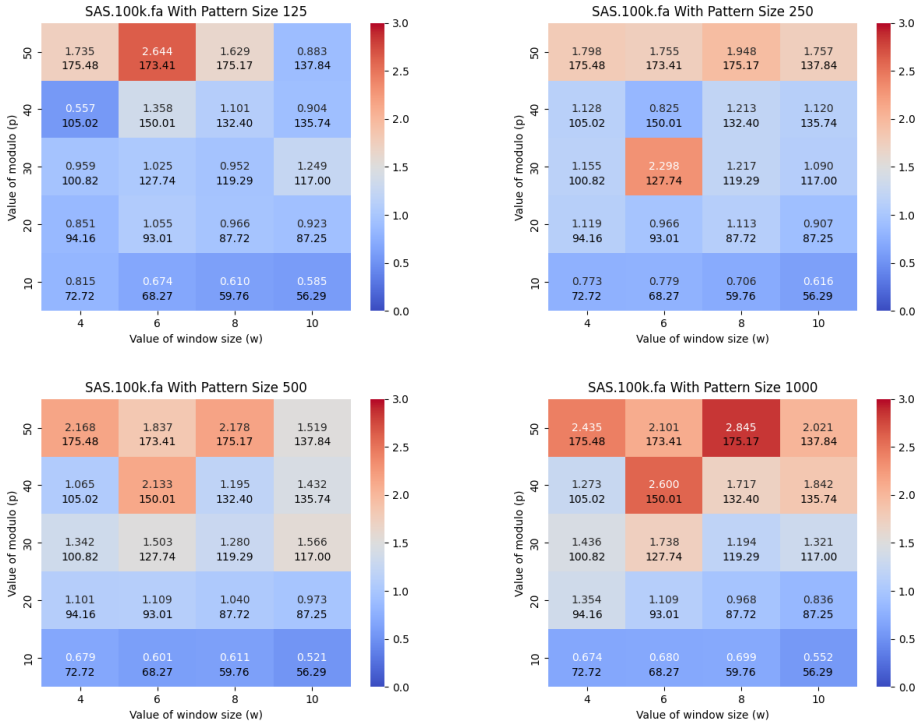


图 3 展示了 w、 p 以及查询模式长度对 FM 索引加速效果的影响。在这里，我们使用了 SARS-100K 数据集，并将查询模式的长度分别设置为 125、250、500 和 1000。y 轴对应于 p ， x 轴对应于 w。热力图展示了在加速版本与来自 sdsl 的标准 FM 索引之间，每 CPU 秒内能够执行的查询数量之比，即 PFP-FM / sdsl。

源代码，以便在单个文件中处理多个查询模式。除了考虑 count 的时间外，我们还测量了构建数据结构所需的时间和内存。

4.1 加速效果与基线方法的对比

在本小节中，我们比较了 PFP-FM 与标准 FM 索引在 sdsl 中的性能，其中窗口大小（w）和模数值（p）取不同值，同时改变查询模式的长度。我们计算了使用 PFP-FM 与标准 FM 索引时，每 CPU 秒能够执行的 count 查询数量。我们生成了热力图，以展示不同查询模式长度（即 125、250、500 和 1,000）下， PFP-FM 相对于 sdsl 的 count 查询数量。我们针对每个 SARS-CoV-2 基因组集合均进行了这项实验。图 3 展示了 SARS-100K对应的热力图结果。如图所示，在 w 和 p取最优值时， PFP-FM的速度比标准 FM 索引快 2.178 至 2.845 倍。具体而言，当模式长度分别为 125、250、500 和 1,000 时，观察到的最优性能提升倍数分别为 2.6、2.3、2.2 和 2.9。与这些结果对应的（w, p）参数对分别为 (6,50)、(6,30)、(8,50) 和 (8, 50)。

对于 SARS-25K 和 SARS-50K，也观察到了类似的结果。对于 SARS-25K，当模式长度分别为 125、250 和 500 时，最优加速比分别为 1.508、2.261、4.213 和 5.467（见附录中的图 6）。对于 SARS-50K，最优加速比为

模式长度为125、250和500时，分别为1.288、2.984、3.619和4.835（见附录中的图7）。对于 SARS-50K，模式长度为125、250和500时，最优加速比分别为1.288、2.984、3.619和4.835（见附录中的图7）。对于 GRCh38，模式长度为125、250和500时，最优加速比分别为1.963、1.805、2.492和2.843（见附录中的图8）。这些结果将指导我们在后续实验中对 w 和 p 的选择。

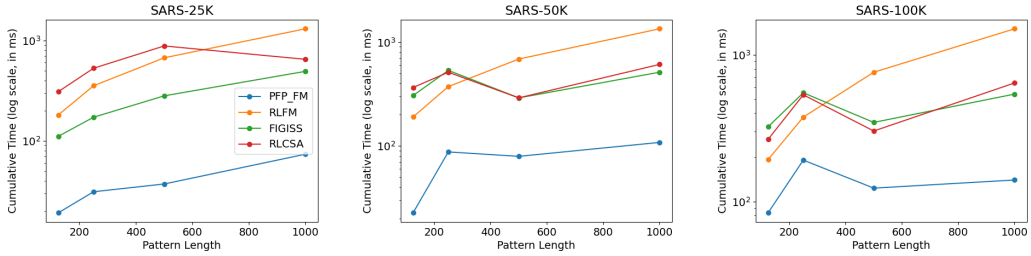


图4 数据集大小和查询模式长度对回答 count 的查询时间的影响示意图。我们将查询模式的长度分别设为 125、250、500 和 1000，并报告 SARS-25K、 SARS-50K 和 SARS-100K 的耗时。我们展示执行 1,000 count 次查询所需的累计时间。y 轴采用对数刻度。

4.2 SARS-CoV-2基因组上的结果

本节采用了前一项实验中获得的最优参数。我们使用这些参数为每个 SARS-CoV-2 数据集构建索引，并评估了使用所有竞争方法和 PFP-FM 执行 1,000 count 次查询所需的时间。我们在图4中展示了该实验的结果。从这一结果可以清楚地看出， PFP-FM 始终表现出最低的时间消耗，且呈现出逐渐、稳定的趋势。对于 SARS-25K 数据集， FIGISS 的时间消耗比我们的方法高出 451% 至 568%。而 RLCSA 和 RLFM 的时间消耗分别比 PFP-FM高出 780% 至 1598% 和 842% 至 1705%。在使用 SARS-25K 数据集时， FIGISS 的性能优于 RLFM 和 RLCSA ；然而，对于规模更大的数据集， FIGISS 和 RLCSA 的性能趋于一致，两种方法均未明显优于另一种方法。此外，在较大的数据集上，当查询模式长度为 125 和 250 时， RLFM 的性能优于 RLCSA 和 FIGISS ，但在其他查询模式长度下则较慢。因此，很明显， PFP-FM 在所有最先进的方法中提升了 count 的性能。

随着数据集大小的增加， PFP-FM 与竞争方法之间的性能差距也随之扩大。对于 SARS-50K, FIGISS、 RLCSA 和 RLFM 分别慢了3.65至13.44倍、3.65至16.08倍以及4.25至12.39倍。对于 SARS-100K， FIGISS、 RLCSA 和RLFM 分别慢了2.81至3.86倍、2.45至4.59倍以及1.96至10.75倍。

接下来，我们考察构建所需的时间和内存；相关数据见表1。我们的实验表明，在所有 SARS-CoV-2 数据集上，各方法的内存使用量均低于 60 GB； PFP-FM 的内存使用量最高，在 SARS-100K 数据集上峰值达到 54 GB。不过，在所有数据集上， PFP-FM 生成 FM 索引的构建时间效率最高，而且随着数据集规模增大，这一时间差距也随之扩大。具体而言，对于 SARS-100K 数据集，与其他方法相比， PFP-FM 分别减少了 71.04%、65.81% 和 73.41% 的耗时。总之， PFP-FM 显著缩短了 count 构建时间，且构建速度最快。所有方法的内存使用量均低于 60 GB,

这一内存容量在大多数商用服务器上都可提供 s.

4.3 人类参考基因组的结果

在使用 GRCh38 数据集测量所有方法构建数据结构所需的时间和内存占用后，我们发现 PFP-FM 的构建时间效率位居第二，但其构建空间占用最大（71 GB，而其他方法为 26 GB 至 45 GB）。具体而言， PFP-FM 构建索引的速度比 FIGISS 和 RLFM快 1.25 至 1.6 倍。

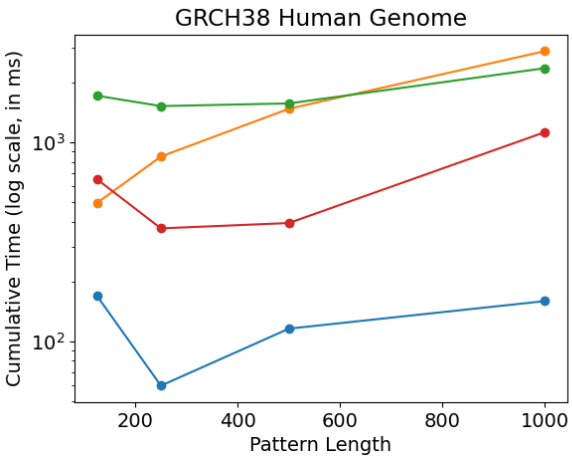


图 5 比较所述解决方案在查询模式长度变化时对 count 的查询时间。对于 125、250、500 和 1000 的每种模式长度，我们报告 GRCH38 数据集的时间。图中绘制了执行 1,000 count 次查询所需的累计时间。y 轴采用对数刻度。 PFP-FM以蓝色显示， RLFM 以橙色显示， RLFM 以红色显示，FIGISS 以绿色显示。

接下来，我们通过执行 1,000 次count 查询来比较 PFP-FM 与其他方法的性能，并在图 5 中展示结果。我们的研究结果表明， PFP-FM 始终优于所有其他方法。尽管当模式长度超过 125 时， RLCSA 的性能优于RLFM 和 FIGISS ，但它仍比 PFP-FM慢 3.9、6.2、3.4 和 7.1 倍。与此同时， RLFM 方法的耗时稳步增加，比 PFP-FM慢 2.9、14.2、12.8 和 18.07 倍。值得注意的是，正如 Akagi 等人的研究所示，FIGISS 语法在非重复数据集上的效率较低，这解释了其在 GRCh38 上的性能为何（更差于） SARS-100K数据集。因此， FIGISS 比 PFP-FM 慢 10.1 、 25.5 、 13.6 和 14.8 倍。这些结果与我们此前的结果一致，表明 PFP-FM 兼具竞争力的构建内存和构建速度，并实现了显著加速。

5 结论

在这项工作中，我们提出了 PFP-FM ，其速度显著快于现有最先进的方法。因此，这项工作开始着手解决数据结构领域一个长期存在的问题：如何解析没有自然词边界的输入，从而实现 FM 索引的加速。我们指出，类似的方法也可以用于增强 locate 查询，因为我们只需在最后一步匹配 α 时（或者在 $Q = \beta$的情况下匹配 β ）使用后缀数组采样，而这可以通过 FM 索引通常采用的后缀数组采样来实现。如果 α 为空，那么我们可以改为使用 FM 索引在 S 上匹配模式的第一个块，而不是在 P 上匹配。我们将此留待未来研究。就实际应用而言，随着测序读段变得越来越长、越来越准确，我们很快就能加速查找长度在 125 到

数据集	n	方法	构建内存（GB）	构建时间（秒）
SARS-25k	751,526,774	RLCSA	9.90	322.85
		RLFM	3.47	363.74
		FIGISS	4.89	378.49
		PFP-FM	12.99	117.29
SARS-50k	1,503,252,577	RLCSA	19.88	679.89
		RLFM	6.94	701.36
		FIGISS	12.44	795.70
		PFP-FM	26.12	233.04
SARS-100k	3,004,588,730	RLCSA	39.47	1690.22
		RLFM	25.01	1432.16
		FIGISS	25.57	1840.80
		PFP-FM	53.90	489.45
GRCh38	3,189,750,467	RLCSA	45.45	924.60
		RLFM	26.31	1839.25
		FIGISS	34.65	1440.19
		PFP-FM	71.13	1154.12

表1 所有数据集的构建性能比较，包括构建时间和内存占用。第二列给出了每个数据集中的字符数（记为 n）。时间以秒（s）为单位，内存以吉字节（GB）为单位。

1,000 之间的模式。因此，未来值得考虑的一个更广泛方向是采用 PFP-FM 等方法加速反向搜索，以便将 Nanopore 测序读段比对到数据库。我们的最后一项实验表明，对于长度为 1,000、针对完整人类参考基因组的查询模式，速度有了显著提升，这证明研究界已具备开展此类工作的条件。

参考文献

1 Tooru Akagi、Dominik Köppl、Yuto Nakashima、Shunsuke Inenaga、Hideo Bannai 和 Masayuki Takeda。基于诱导后缀排序的语法索引。载于 第 *28* 届国际字符串处理与信息检索研讨会（*SPIRE*）论文集，第 85–99 页，2021年。2 David E Daykin、Jacqueline W Daykin 和 William F Smyth。使用 v-order 的混合 Lyndon 字符串线性划分算法。载于 理论计算机科学，483:149–161，2013 年。3 Jin-Jie Deng、Wing-Kai Hon、Dominik Köppl 和 Kunihiko Sadakane。针对长模式的、通过后缀排序诱导的语法 FM 索引。在 *IEEE* 数据压缩会议（*DCC*）论文集中，第 63–72 页，2022年。4 Paola Ferragina 和 Giovanni Manzini。压缩文本索引。*ACM* 学报，52:552–581，2005 年。5 Paolo Ferragina 和 Johannes Fischer。词上的后缀数组。在 第 *18* 届年度组合模式匹配研讨会（*CPM*）论文集中，第 328–339 页，2007 年。6 Travis Gagie、Gonzalo Navarro 和 Nicola Prezza。在 BWT 运行受限空间中实现完全功能的后缀树和最优文本搜索。 *ACM* 学报，67(1):1–54，2020 年。7 S Gog、T Beller、A Moffat 和 M Petri。从理论到实践：使用简洁数据结构即插即用。在 第 *13* 届实验算法研讨会（*SEA*）论文集中，第 326–337 页，2014 年。

8 Simon Gog、Juha Kärkkäinen、Dominik Kempa、Matthias Petri 和 Simon J Puglisi。FM 索引中的固定块压缩增强：理论与实践。 *Algorithmica*，81:1370–1391，2019 年。**9** Ben Langmead、Cole Trapnell、Mihai Pop 和 Steven L. Salzberg。对人类基因组中的短 DNA 序列进行超快速且节省内存的比对。 基因组生物学，10(3):R25–R25，2009 年。**10** Heng Li。使用 BWA-MEM 比对序列测序读段、克隆序列和组装重叠群。2013 年。URL: http://arxiv.org/abs/1303.3997。**11** Veli Mäkinen 和 Gonzalo Navarro。运行长度 FM 索引。在 *DIMACS* 研讨会论文集：*“Burrows–Wheeler* 变换：十年之后”中，第 17–19 页，2004 年。**12** Veli Mäkinen 和 Gonzalo Navarro。基于运行长度编码的简洁后缀数组。在年度组合模式匹配研讨会论文集中，第 45–56 页，2005 年。**13** Udi Manber 和 Gene W. Myers。后缀数组：一种在线字符串搜索的新方法。*SIAM* 计算学报，22(5):935–948，1993 年。**14** Felix Mölder、Kim Philipp Jablonski、Brice Letcher、Michael B Hall、Christopher H Tomkins-Tinch、Vanessa Sochat、Jan Forster、Soohyun Lee、Sven O Twardziok、Alexander Kanitz 等。使用 Snakemake 进行可持续数据分析。 *F1000Research*，10，2021年。**15** Jouni Siren。面向海量数据的压缩后缀数组。在第 *16* 届国际字符串处理与信息检索研讨会（*SPIRE*）论文集中，第 63–74 页，2009 年。

A 附录

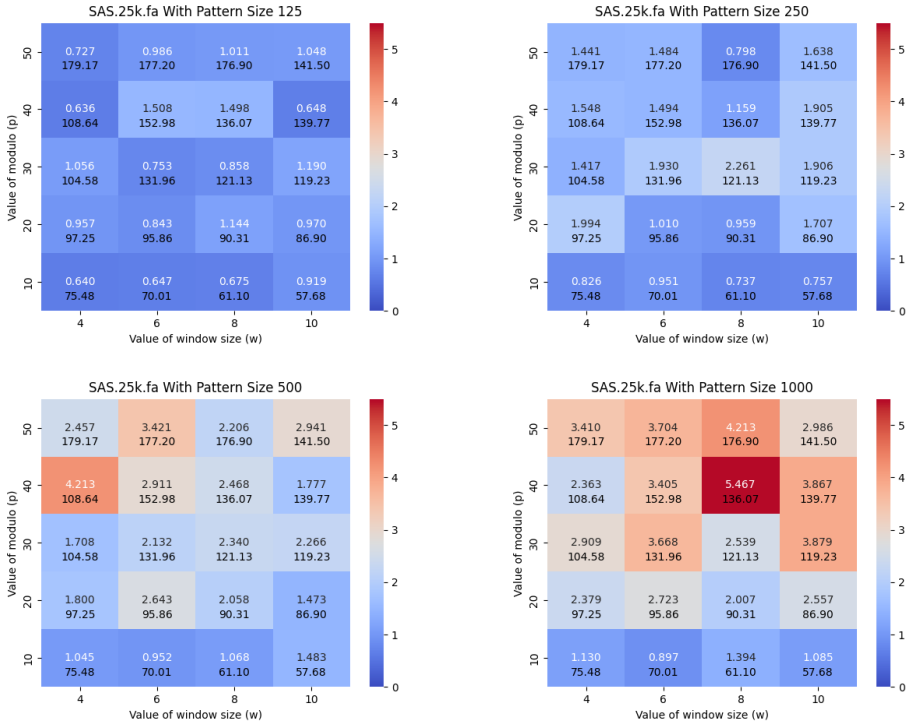


图 6 w、p 及查询模式长度对 FM 索引加速效果的影响示意图。此处，我们使用了 SARS-25K数据集，并将查询模式长度设为 125、250、500 和 1000。y 轴对应于 p，x 轴对应于 w。热力图展示了相较于来自 `sdsl` 的标准 FM 索引，加速后的 FM 索引每 CPU 秒可执行的查询数量，即 `PFP-FM / sdsl`。

23:18 通过前缀无关解析加速 FM 索引查询

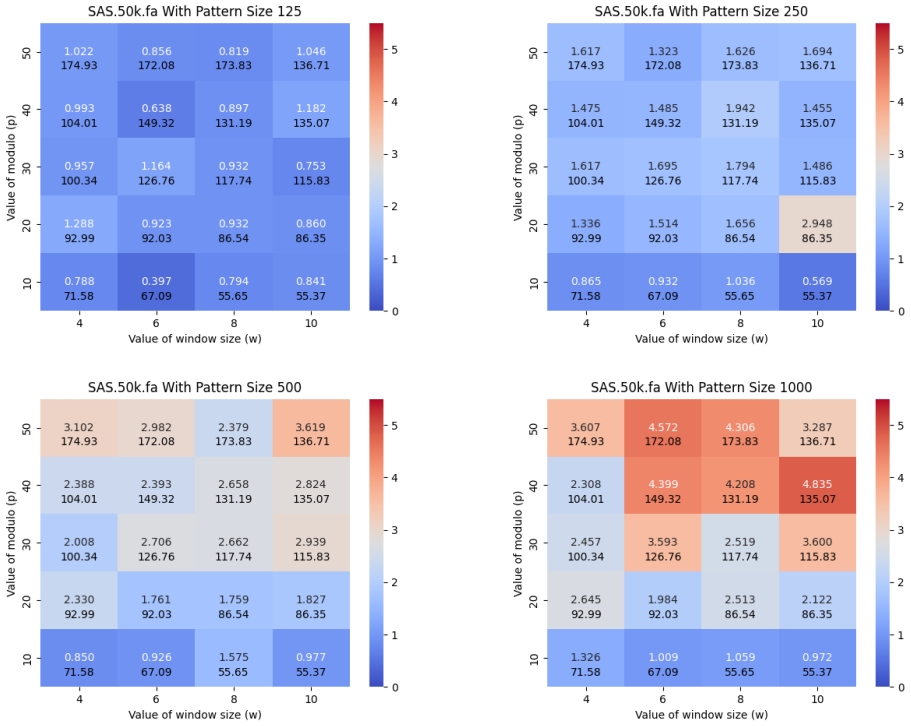


图7 展示了 w、 p 和查询模式长度对 FM 索引加速效果的影响。这里，我们使用了 `SARS-50K` 数据集，并将查询模式长度分别设为 125、250、500 和 1000。y 轴对应 p，x 轴对应 w。热力图展示了在一个 CPU 秒内，使用加速方法与标准 FM 索引执行的查询数量之比（ `sdsl`），即 `PFP-FM` / `sdsl`。

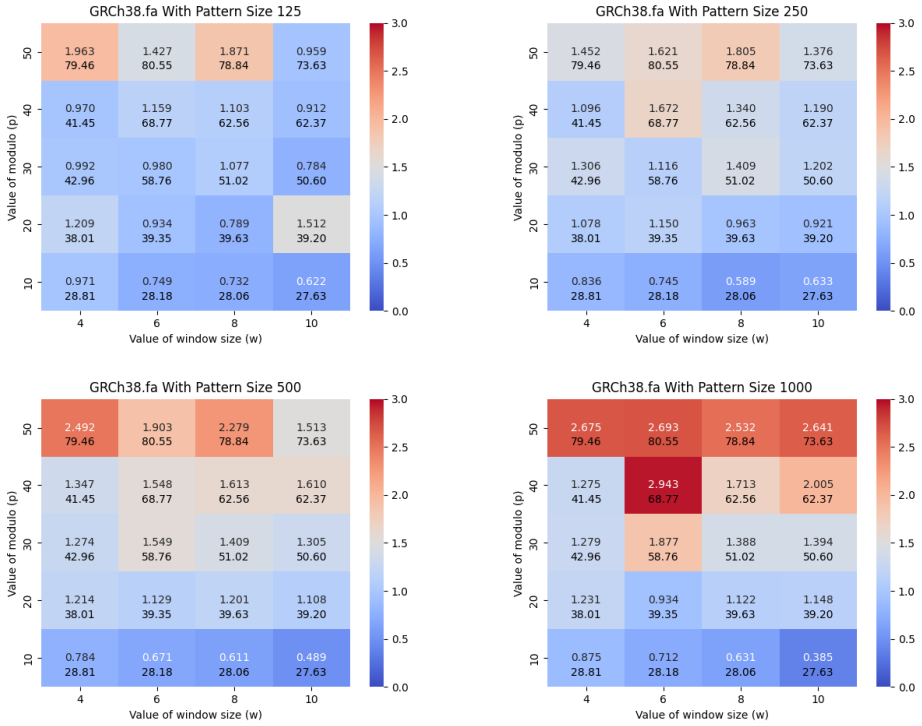


图8 w、 p 以及查询模式长度对 FM 索引加速效果影响的示意图。这里，我们使用了 `GRCh38` 数据集，并将查询模式长度分别设为 125、250、500 和 1000。纵轴对应 p ，横轴对应 w。热力图展示了使用加速方法与来自 `sdsl`的标准 FM 索引相比，每 CPU 秒可执行的查询数量，即 `PFP-FM / sdsl`。