arXiv:2601.09577v1 [cs.DS] 14 Jan 2026

Parikh预算下的排列匹配：线性时间检测、装箱与不相交选择

MD Nazmul Alam Shanto、Md. Tanzeem Rahat 和 Md. Manzurul Hasan

孟加拉国达卡孟加拉国美国国际大学计算机科学系，
mdnazmulshanto2001@gmail.com、tanzeem.rahat@aiub.edu、manzurul@aiub.edu

摘要。我们研究一般字母表上的排列（乱序／阿贝尔）模式匹配。给定长度为 m 的模式串 P 和长度为 n的文本 T，经典问题是判断 T 中是否存在一个长度为m 的子串，其 Parikh 向量与 P的相同。尽管这一存在性问题可以通过线性时间滑动窗口解法解决，许多实际应用还需要超越单纯检测的优化与装箱变体。我们提出一个统一的滑动窗口框架，通过维护 P 与 T当前窗口之间的 Parikh向量差，实现时间复杂度为 $O(n+\sigma)$、空间复杂度为 $O(\sigma)$的排列匹配，其中$\sigma = |\Sigma|$。基于此框架，我们引入一种组合优化变体，称为模式供给下的最大可行子串（MFSP）：寻找 T 的最长子串 S，使其各符号的计数均不超过 P中对应符号的计数。我们证明，MFSP 也可以通过双指针可行性维护算法在 $O(n+\sigma)$时间内求解，并将 P 精确地解释为资源预算下的装箱问题。最后，我们将每个排列匹配视为一个等长区间，对不重叠出现项的选择进行建模，并证明最早结束贪心策略能够得到基数最大的互不相交匹配集合；在枚举所有匹配项后，该集合可在线性时间内计算。我们的研究给出了简洁且正确性有保证、复杂度界紧确的算法，并将基于频率的字符串匹配与装箱式优化基元联系起来。

关键词： 乱序（阿贝尔）模式匹配 · Parikh（频率）向量 · 滑动窗口算法 · 区间装箱 / 不重叠选择

1 引言

模式匹配是字符串算法中的一项基础任务，应用涵盖文本处理、生物信息学、网络安全和信息检索。除了经典情形——模式串必须以相同顺序作为子串出现——一种受到广泛研究的变体关注的是模式串是否能以排列后的形式出现。具体而言，给定一个长度为 m 的模式串 P 和一个长度为

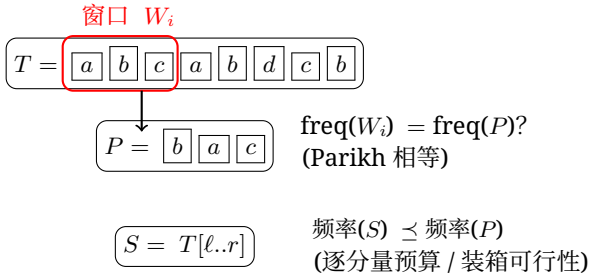


图 1. 本文采用的两种视角。上：排列匹配询问是否存在某个长度为 m 的窗口 W_i，使其 Parikh 向量与 P 相同。下：将 freq(P) 视为供给向量，可得到一种装箱式可行性约束，用于在符号预算下选择较长的子串。

n的文本，排列（乱序/阿贝尔）匹配问题要判断 T 是否包含一个长度为 m 的子串，其符号多重集与 P相同。这个问题也称为乱序模式匹配 [5]，并且与排列 [9,2]的排列模式匹配密切相关。

排列匹配已从多个角度得到研究。在二进制情形中，专门的索引和算法利用窗口 [12,8] 的一维摘要。对于一般字母表，核心对象是字符串的 ***Parikh*** 向量（频率向量），而乱序匹配则转化为判断频率是否相等的问题 [5,3]。大量研究致力于构建索引，以高效回答大量查询，包括针对常数字母表的高效索引 [11], 、一般乱序索引的困难性结果 [1], 、近似变体 [4], 、在线算法 [7], 、SIMD 加速 [6], ，乃至量子模型 [10]。这些成果凸显了基于频率的匹配在理论上的丰富内涵和实际应用价值。

从检测到优化与装箱。 尽管检测排列的出现是基础问题，但许多应用还需要更丰富的优化 和装箱 原语。例如：(i) 当模式串充当资源预算（例如，允许的词元数量）时，我们可能希望找到 T 中不超过该预算的最长 子串；(ii) 当存在多个出现位置时，我们可能希望找到数量最多的一组互不重叠的 匹配项。这些问题自然地将乱序匹配与组合优化联系起来：在给定供给向量下满足可行性，类似于装箱约束；而选择互不相交的出现位置，则构成区间装箱问题。

我们的贡献。 本文提出了一种简洁且具有可证明保证的滑动窗口框架，并进一步构建了适用于组合优化场景的优化与装箱变体。

– 线性时间排列匹配（基线方法）。 我们提出一种 $O(n+\sigma)$ 时间、$O(\sigma)$ 空间的算法，通过维护 Parikh 差向量和非零计数器，判断 T 是否包含 P 的一个长度为m 的排列匹配。

– *MFSP*：模式供给下的最大可行子串。我们提出一个优化变体：寻找 T 的最长子串 S，使得 freq(S) $\preceq$freq(P)。将 freq(P) 解释为资源预算后，该问题便与装箱可行性建立了联系。我们给出一种线性时间双指针算法，并附上完整的正确性证明。

– 通过区间装箱进行不重叠出现位置选择。将所有排列匹配视为等长区间后，选择数量最多的不相交匹配就成为一个区间装箱问题。我们证明，按结束位置最早（等价地，按起始位置最早）的贪心规则是最优的；在枚举出匹配项后，该规则可在线性时间内执行。

组织结构。第2节介绍符号和 Parikh 向量。第3节介绍用于排列匹配的增量式 Parikh 差分算法。第4节提出 MFSP，并证明线性时间可行性维护算法的正确性。第5节将不重叠选择作为区间装箱问题进行研究。第6节报告实验观察，第7节作出总结。

2 预备知识与符号

设 Σ 是一个大小为 $\sigma = |\Sigma|$ 的字母表。Σ 上的字符串（单词）S 的长度为 $|S|$，我们采用从 0 开始的索引，因此 $S[i]$ 表示位置 i 处的字符，其中 $0 \le i < |S|$。对于整数 $\ell \le r$，我们用 $S[\ell..r]$ 表示子串 $S[\ell]S[\ell+1]\cdots S[r]$。对于长度为 n 的文本 T 和整数 $m \le n$，我们记

$$W_i \;=\; T[i..i+m-1] \qquad (0 \le i \le n-m)$$

从索引 i开始的 T 长度为m 的窗口。

2.1 Parikh（频率）向量

对于字符串 S， S 的*Parikh* 向量 （频率向量）是由 freq$_S : \Sigma \to \mathbb{Z}_{\ge 0}$ 定义的函数

$$\mathrm{freq}_S(c) \;=\; |\{\, i : 0 \le i < |S|,\; S[i] = c \,\}| \qquad \text{for each } c \in \Sigma.$$

在上下文明确时，我们用 freq(S) 表示 freq$_S$。如果 $|X| = |Y|$ 且 freq(X) $=$ freq(Y)（逐分量相等），则两个字符串 X 和 Y互为排列 （字母异序词）。

2.2 研究的问题

全文中，我们给定一个模式串 $P \in \Sigma^m$ 和一段文本 $T \in \Sigma^n$。

排列匹配（是否存在）。判断是否存在一个索引 i，使得$0 \le i \le n-m$ 满足

$$\mathrm{freq}(W_i) \;=\; \mathrm{freq}(P).$$

MFSP（模式供给下的最大可行子串）。将 freq(P) 视为供给（预算）向量。如果子串 S 属于 T，且其符号计数不超过供给，则该子串是可行的：

$$\mathrm{freq}(S) \preceq \mathrm{freq}(P) \quad \Longleftrightarrow \quad \forall c \in \Sigma:\ \mathrm{freq}(S)(c) \le \mathrm{freq}(P)(c).$$

MFSP 要求计算 T 的最长可行子串，即，

$$\max\{\, |S| : S = T[\ell..r] \text{ and } \mathrm{freq}(S) \preceq \mathrm{freq}(P) \,\}.$$

不重叠选择。设 $\mathcal{M} = \{\, i : \text{freq}(W_i) = \text{freq}(P) \,\}$ 为所有匹配起始位置的集合。每个 $i \in \mathcal{M}$ 都对应一个区间 $I_i = [i, i+m-1]$。目标是选出一个基数最大的两两不相交区间子集。

2.3 差异向量与非零计数器

本文的一个核心工具是维护 Parikh 向量的差异。对于窗口 W_i 和模式串 P，定义差异向量

$$\Delta_i(c) \;=\; \mathrm{freq}(W_i)(c) - \mathrm{freq}(P)(c) \qquad \text{for } c \in \Sigma.$$

注意，当且仅当对所有 $c \in \Sigma$，$\text{freq}(W_i) = \text{freq}(P)$ 时，$\Delta_i(c) = 0$。为了高效地检验这一点，我们维护

$$nz_i \;=\; |\{\, c \in \Sigma : \Delta_i(c) \neq 0 \,\}|,$$

非零条目的数量。当且仅当$nz_i = 0$时，W_i 是排列匹配。

2.4 实现模型

我们假设以下两种情况之一成立：(i) 显式给出 Σ，且 σ 适中，因此可以使用按符号索引的数组计数；或 (ii) 我们执行字母表压缩：使用哈希映射，仅将 P 中出现的符号以及 T 的相关部分映射为整数 1、.....、σ'，其中 $\sigma' \le \min(\sigma,\ m+n)$。为简单起见，所有渐近界均以 σ 为基准表述，并且在进行压缩后可直接扩展到 σ'。

3 通过 Parikh 差值维护实现线性时间排列匹配

我们首先讨论经典的排列（乱序/阿贝尔）匹配判定问题：给定模式串 $P \in \Sigma^m$ 和文本 $T \in \Sigma^n$，其中 $m \le n$，判断是否存在一个索引 i，使得长度为 m 的窗口 $W_i = T[i..i+m-1]$满足 $\text{freq}(W_i) = \text{freq}(P)$。关键在于维护当前窗口的 Parikh 差向量，以及一个用于证明相等的整数。

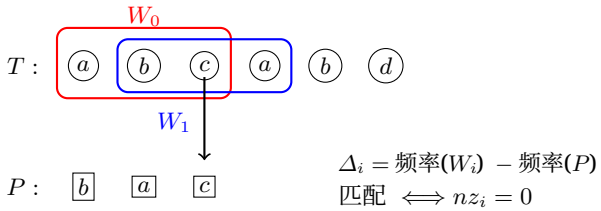


图 2. 符号说明：长度为 m 的窗口 W_i 在 T 上滑动。维护 Parikh 差异向量 Δ_i 和非零计数器 nz_i，即可在每次移动时进行常数时间更新。

3.1 维护的状态

回顾一下差异向量

$$\Delta_i(c) \;=\; \mathrm{freq}(W_i)(c) - \mathrm{freq}(P)(c), \qquad c \in \Sigma,$$

以及 非零计数器

$$nz_i \;=\; |\{\, c \in \Sigma : \Delta_i(c) \neq 0 \,\}|.$$

当且仅当 $nz_i = 0$ 时， W_i 是排列匹配。

要在常数时间内更新 nz_i，我们使用以下原语：当将 $\Delta(c)$ 改变 ± 1 时，只有 $\Delta(c)$ 是否为零可能发生变化。

更新原语。设更新前 $x = \Delta(c)$，更新后 $x' = \Delta(c) + \delta$，其中 $\delta \in \{-1, +1\}$。
– 若 $x = 0$ 且 $x' \neq 0$，则递增 nz。– 若 $x \neq 0$ 且 $x' = 0$，则递减 nz。– 否则，nz 保持不变。

3.2 算法

我们从 –freq(P) 开始初始化 Δ，并加入第一个窗口 W_0中的符号。此后，每次从 W_i 移至 W_{i+1}，都会移除一个离开的符号$T[i]$并加入一个新进入的符号 $T[i+m]$；因此，每一步中 Δ 只有两个坐标发生变化。

3.3 正确性

定理 1. 算法 *1* 当且仅当存在一个索引 i，使得 $0 \le i \le n - m$，满足 频率(W_i) $=$ 频率(P) 时，返回 *True*。

证明。 我们证明如下不变量：初始化后以及每次移动后，对于每个 $c \in \Sigma$，所维护的值 $\Delta(c)$ 等于当前窗口 W_i 中 freq$(W_i)(c)$ 与 freq$(P)(c)$ 的差，而 nz 等于 $\Delta(c)$ 非零的符号数量。

初始化时，将 $\Delta(c)$ 设为 $\mathrm{freq}(P)(c)$（若不存在则设为 0），随后将 W_0 的每个符号各加一次；因此 $\Delta(c)$ 等于 $\mathrm{freq}(W_0)(c)$ 与 $\mathrm{freq}(P)(c)$ 的差。更新规则仅在某个坐标跨越零时修改 nz，因此 $nz = |\{c : \Delta(c) \text{ 成立}$。

从 W_i 移至 W_{i+1}时，恰有一个符号 $out = T[i]$ 离开窗口，另一个符号 $in = T[i+m]$ 进入窗口。因此，对所有 $c \in \Sigma$，$\mathrm{freq}(W_{i+1})(c) = \mathrm{freq}(W_i)(c) - \mathbf{1}[c = out] + \mathbf{1}[c = in]$。算法恰好对 Δ执行这两次± 1 更新，保持差向量不变量，并根据零交叉情况一致地更新 nz。

最后，当且仅当对所有 c，$\mathrm{freq}(W_i) = \mathrm{freq}(P)$ 成立，当且仅当 $\Delta(c) = 0$ 成立，也就是说，当且仅当 $nz = 0$成立。因此，算法当且仅当窗口是 P的排列时返回 True。

3.4 复杂度

定理 2. 算法 *1* 的运行时间为 $O(n + \sigma)$，使用空间为 $O(\sigma)$。

证明。对 Σ 进行初始化需要 $O(\sigma)$ 时间。构建第一个窗口需要执行 m次常数时间更新。每次 $n-m$移位都会执行两次常数时间更新，并对 nz进行一次常数时间检查，总计需要 $O(n)$ 时间。维护的数组/映射在 Σ中为每个符号存储一个整数，因此需要 $O(\sigma)$ 空间。

备注（枚举所有匹配项）。使用相同的维护状态，只需记录每个满足 $nz = 0$ 的 i，即可输出所有匹配索引；这会增加 $O(\#\mathcal{M})$ 的输出时间，但不会改变渐近运行时间。

4 MFSP：模式供给下的最大可行子串

现在，我们从检测转向一种自然适用于装箱解释的优化原语。设 $P \in \Sigma^m$ 为一个供给字符串。其 Parikh 向量 $\mathrm{freq}(P)$ 定义了一个逐分量预算，限定文本 T 中所选子串可以包含哪些符号。

4.1 问题定义

子串 $S = T[\ell..r]$ 是可行的，如果

$$\mathrm{freq}(S) \preceq \mathrm{freq}(P) \quad \Longleftrightarrow \quad \forall c \in \Sigma : \ \mathrm{freq}(S)(c) \le \mathrm{freq}(P)(c).$$

MFSP（模式供给下的最大可行子串）要求找出

$$(\ell^\star, r^\star) \in \arg\max_{0 \le \ell \le r < n} (r - \ell + 1) \quad \text{s.t.} \quad \mathrm{freq}(T[\ell..r]) \preceq \mathrm{freq}(P).$$

我们还输出最大长度 $L^\star = r^\star - \ell^\star + 1$。

装箱视角。将 $\mathrm{freq}(P)$ 视为预算向量，若子串 $T[\ell..r]$ 的符号计数消耗不超过可用供给，则该子串可行。因此，MFSP 要求在 T 中找到最大的可行装箱块（即连续选择）。

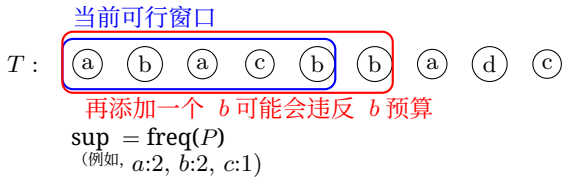


图3. MFSP 在供给向量 sup $=$ freq(P) 下维护一个可行窗口。当窗口变为不可行时，左指针会向前移动，直到恢复可行性。

4.2 双指针可行性维护

可行性在缩小时具有单调性：如果某个窗口不可行，从其左端移除字符只会减少其计数，并最终恢复可行性。这种单调性支持使用线性时间双指针算法（又称滑动窗口），为每个右端点维护一个极大可行窗口。

我们维护一个数组 cnt(c)，用于表示当前窗口 $T[\ell..r]$，以及供给数组 sup(c) $=$ freq(P)(c)。当且仅当对所有 c，cnt(c) $\le$ sup(c) 时，该窗口可行。为了在摊销 $O(1)$ 时间内测试可行性，我们维护一个违规计数器

$$viol \;=\; |\{\, c \in \Sigma : \mathrm{cnt}(c) > \mathrm{sup}(c) \,\}|.$$

与第3节中的 nz 一样，通过 ± 1 更新 cnt(c) 时，只有在阈值跨越时，某个符号是否违规才会发生变化。

4.3 正确性

我们证明算法 2 返回一个最长可行子串。

定理 3. 算法 *2* 输出索引 $(\ell^\star, r^\star)$，使得 $T[\ell^\star..r^\star]$ 可行，且其长度在 T 的所有可行子串中最大。

证明。 我们保持这样一个不变量：对于固定的右端点 r，**while** 循环终止后，当前窗口 $T[\ell..r]$ 是可行的，即对于所有 $c \in \Sigma$，cnt(c) $\le$ sup(c)。

可行性恢复。 加入 $T[r]$ 后，唯一可能出现的新不可行情况是字符 $c = T[r]$ 的 cnt(c) 超过 sup(c)，这可以通过递增 $viol$ 来检测。**while** 循环的每次迭代都会从左侧恰好移除一个字符，使恰好一个计数减 1。对于某个符号 d 的违规，当其计数从 cnt(d) >sup(d) 变为 cnt(d) $\le$sup(d) 时，违规恰好消失，此时我们递减 $viol$。因此，循环恰好在不存在违规符号时停止，也就是说，可行性成立。

每个 r 的最大性。固定 r。令 ℓ_r 为 while 循环结束后 ℓ 的值。此时，$T[\ell_r..r]$ 是可行的。此外，算法会在满足可行性的前提下，尽可能减小 ℓ：如果将 ℓ 减小 1（即尝试纳入 $T[\ell_r - 1]$），那么该字符此前被移除，只是因为

窗口不可行；因此，重新纳入它会再次导致违规。因此，在所有以 r 结尾的可行子串中，窗口 $T[\ell_r..r]$ 的长度最大。

全局最优性。令 $S^\star = T[a..b]$ 为 T中的最长可行子串。考虑迭代 $r = b$。根据此 r的最大性，算法在 b 处结束的可行窗口长度至少为 $|T[a..b]|$。因此，算法在所有右端点记录的最大长度 $L^\star$至少为 $|S^\star|$。由于算法只记录可行窗口的长度，$L^\star$ 不会超过最优值。因此，$L^\star = |S^\star|$，且记录的（$\ell^\star$，$r^\star$）是最优的。

4.4 复杂度

定理 4。 算法 *2* 的运行时间为 $O(n + \sigma)$，使用空间为 $O(\sigma)$。

证明。 每一步将 r 递增一次，因此外层循环运行 n次。指针ℓ 只向前移动；在整个执行过程中，它最多前进 n 次。每次前进都会执行 $O(1)$ 次更新。因此，while 循环中花费的总时间为 $O(n)$，包括对 Σ上的数组进行初始化在内，总运行时间为 $O(n+\sigma)$。算法存储 sup 和 cnt 数组（或压缩映射），需要 $O(\sigma)$ 空间。

备注（恢复子串）。算法已存储 $(\ell^\star, r^\star)$，因此最优子串为 $T[\ell^\star..r^\star]$，并且如有需要，可在 $O(L^\star)$ 时间内输出。

5 将不重叠的排列匹配视为区间装箱

本节讨论如何选取模式串 P 在文本 T 中数量最多且互不重叠的排列匹配。一旦确定所有排列匹配，每个匹配项都对应一个固定长度的区间，而选择任务便转化为经典的区间装箱问题。

5.1 从匹配项到区间

Let

$$\mathcal{M} \;=\; \{\, i \in \{0, \ldots, n-m\} : \mathrm{freq}(W_i) = \mathrm{freq}(P) \,\}$$

设其为所有长度为 m 的排列匹配的起始索引集合。每个 $i \in \mathcal{M}$ 都会诱导一个区间

$$I_i \;=\; [\, i,\; i+m-1 \,].$$

如果对应的区间两两不相交，即对于所有互不相同的 i、$j \in S$, $I_i \cap I_j = \emptyset$，则集合 $S \subseteq \mathcal{M}$ 是非重叠的 。我们的目标是计算 $\mathcal{M}$的一个最大基数非重叠子集。

5.2 贪心选择

由于所有区间的长度相同 m，因此可以通过一个简单的贪心规则得到最优解：从左到右扫描 T，只要可能就选取一个匹配，然后向前跳过 m。

5.3 通过交换论证证明正确性

定理5。 算法 *3* 输出一个由两两不相交的排列匹配区间组成的最大基数集合。

证明。 设 S_g 为算法 3 返回的集合。我们通过标准的交换论证证明，存在一个最优解 $S^\star$，满足 $|S^\star| = |S_g|$。

设 i_1 为 S_g 中的第一个 选定起始索引（即算法选择的最左侧排列匹配）。考虑任意最优解$S^\star$。如果 $S^\star$ 也选择了 i_1，我们将继续处理 $i_1 + m - 1$之后的剩余后缀。

否则，令 j_1 为 $S^\star$中最小的起始索引。由于 i_1 是最小的可用匹配起始位置（由贪心扫描得出），因此有 $i_1 < j_1$。在$S^\star$中用 i_1 替换 j_1。此替换保持可行性（互不相交）：由于区间长度相同，均为 m，将第一个选定区间向左移动不会与 $S^\star$ 中的其余区间重叠（这些区间的起始位置均为 $\geq j_1$，因此结束位置均为 $\geq j_1 + m - 1 > i_1 + m - 1$）。于是，我们得到另一个包含 i_1 且基数相同的最优解。

移除区间I_{i_1}，并将注意范围限制为从索引i_1+m开始的 T 后缀。贪心算法在该后缀上重复相同的选择结构，并且同样的交换论证可以归纳地应用。因此，贪心算法选出的区间数量与最优解相同，即 $|S_g|$ 为最大值。

5.4 复杂度

定理6。 给定匹配集合 $\mathcal{M}$，算法 *3* 的运行时间为 $O(n)$。通过算法 *1* 进行匹配枚举后，总时间为 $O(n+\sigma)$ 加上输出大小，空间复杂度为 $O(\sigma)$。

证明。 贪心扫描使 i 从 0 单调递增至 $n - m$，因此执行 $O(n)$ 次迭代。通过存储长度为 n 的布尔数组，可以在 $O(1)$ 时间内支持成员测试 $i \in \mathcal{M}$（当 n 较大且匹配项稀疏时，也可以使用哈希集合）。使用算法 1 进行匹配枚举需要 $O(n+\sigma)$ 时间和$O(\sigma)$空间。

6 实验观察

本节报告轻量级实验观察，旨在佐证所提出的线性时间过程在实际应用中的效率。我们的重点不是索引构建（如乱序索引），而是单遍扫描和优化原语。

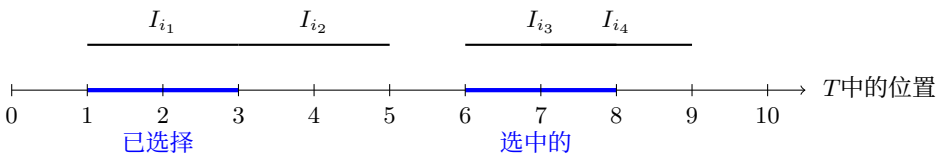


图 4. 枚举所有匹配项后的不重叠选择。每个匹配项都是长度为 m 的区间；贪心策略选择最靠左的可用区间，并向前跳过 m，从而为等长区间得到一个规模最大的互不相交集合。

6.1 实现细节

所有算法均采用标准命令式风格实现，使用 (i) 数组处理中等规模的字母表；(ii) 当 Σ 较大时，使用字母表压缩（将符号映射到 1，...，σ' 的哈希映射）。对于排列匹配（第3节），我们维护 Parikh 差向量 Δ 和非零计数器 nz；每次移位恰好更新两个坐标，并检测 $nz = 0$ 是否成立。对于 MFSP（第4节），我们维护窗口计数 cnt 和违规计数器 $viol$；每次移动指针都会更新一个坐标，并可能在阈值跨越时改变 $viol$ 。

6.2 工作负载与指标

我们在两类输入上评估了这些方法：
– **合成字符串：** 在不同大小的字母表上生成的随机文本（$\sigma \in \{4, 16, 64, 256\}$），模式串长度为 $m \in \{16, 64, 256, 1024\}$，文本长度最长为 $n = 10^7$。– **自然语言语料库：** 纯文本数据集（字母和 ASCII 符号），其中字母表压缩可减少内存占用并改善缓存局部性。

我们报告：(i) 总运行时间随 n 的变化；(ii) 吞吐量（每秒处理的符号数）；(iii) 匹配项数量 $|\mathcal{M}|$；以及 (iv) 对于 MFSP，最优长度 $L^\star$ 及其在文本中的分布。

6.3 关键观察

在所有测试设置中，排列匹配和 MFSP 均随 n 呈现线性扩展趋势，这与定理 2 和 4 中的 $O(n+\sigma)$ 分析一致。常数因子主要受以下因素影响：(a) 计数数组/映射的内存访问模式；(b) 匹配项的密度。后者会影响输出量，但不会影响扫描过程本身。

对于大字母表，字母表压缩显著降低了内存占用；当有效字母表 σ' 远小于 σ时，还提升了性能。对于 MFSP，违规计数器机制消除了进行完整可行性检查的必要：只有在超出预算约束时，才需要移动左指针以恢复可行性；每个字符最多添加或移除一次，这与标准的双指针摊还分析论证一致。

可复现性。 通过固定合成生成所用的随机种子，并对较大的 σ 输入应用相同的字母表压缩映射，即可根据算法1和算法2中的伪代码重现所有实验。

7 结论

我们从组合优化的角度重新审视了排列（乱序/Abelian）模式匹配。借助 Parikh 差值维护，我们得到了一个简洁的 $O(n+\sigma)$ 时间、 $O(\sigma)$ 空间滑动窗口过程，用于排列匹配，并给出了简单的正确性不变量。在此基础上，我们提出了 MFSP，这是一种优化变体，将 $\mathrm{freq}(P)$ 视为供给（预算）向量，并在逐分量约束下寻找 T 的最长可行子串。我们给出了一个线性时间双指针算法，并完整证明了其最优性。最后，我们将互不重叠的排列匹配出现位置建模为等长区间，并证明贪心扫描可得基数最大的互不相交选择，从而将基于频率的字符串匹配与区间装箱联系起来。

未来研究的一个自然方向是探讨 MFSP 和不重叠选择的加权及近似扩展，其中可行性允许有限度的违规，或为出现位置赋予权重，从而有望形成更丰富的优化结构，并开辟新的复杂度前沿。

参考文献

1. Amir, A., Chan, T.M., Lewenstein, M., Lewenstein, N.: 乱序索引的困难性研究。载于：国际自动机、语言与编程研讨会（ICALP）。第 114–125 页。Springer（2014） 2. Bose, P., Buss, J.F., Lubiw, A.: 排列的模式串匹配。《信息处理快报》**65**(5), 277–283 (1998) 3. Burcsi, P., Cicalese, F., Fici, G., Lipt´ak, Z.: 字符串中乱序模式串匹配的算法。arXiv 预印本 arXiv:1102.1746 (2011) 4. Burcsi, P., Cicalese, F., Fici, G., Lipt´ak, Z.: 字符串中的近似乱序模式串匹配。《计算机理论系统》**50**(1), 35–51 (2012) 5. Cicalese, F., Fici, G., Lipt´ak, Z.: 字符串中乱序模式串的搜索。载于：布拉格字符串学会议论文集。第 105–117 页。布拉格捷克理工大学（2009）6. Ghuman, S.S., Tarhio, J.: 基于 SIMD 的乱序匹配。载于：布拉格字符串学会议论文集。第 116–122 页（2016） 7. Ghuman, S.S., Tarhio, J., Chhabra, T.: 改进的乱序匹配在线算法。《离散应用数学》**250**, 122–130 (2018) 8. Giaquinta, E., Grabowski, S.: 二进制乱序模式串匹配的新算法。《信息处理快报》**113**(14–16), 538–542 (2013) 9. Ibarra, L.: 查找排列的模式串匹配。《信息处理快报》**61**(6), 293–295 (1997) 10. Ju´arez-Xochitemol, J., Ch´avez, E.: 量子乱序模式串匹配。arXiv 预印本 arXiv:2203.00164 (2022) 11. Kociumaka, T., Radoszewski, J., Rytter, W.: 面向常数字母表乱序模式串匹配的高效索引。《Algorithmica》**74**(2), 601–619 (2016) 12. Moosa, T.M., Rahman, M.S.: 二进制字符串的排列索引。《信息处理快报》**110**(18-19), 795–798 (2010)

算法 1： 通过 Parikh 差值维持进行排列匹配——维护

输入 ：长度为 n 的文本 T，长度为 m （$m \leq n$）的模式串 P，字母表 Σ
输出： 当且仅当存在某个窗口 W_i 是 P 的排列（可选地全部匹配索引）

初始化： 对于所有 $c \in \Sigma$，设置 $\Delta(c) \leftarrow 0$ 和 $nz \leftarrow 0$。

对每个 $c \in \Sigma$ **执行**
  // 从 $-$频率(P) 开始
  如果 频率$(P)(c) > 0$ **则**
    $\Delta(c) \leftarrow -$频率$(P)(c)$;
    $nz \leftarrow nz + 1$ // 自 $\Delta(c)$ 起 $\neq 0$

for $j \leftarrow 0$ **到** $m - 1$ **do**
  // 添加第一个窗口 W_0
  $x \leftarrow \Delta(T[j])$;
  $\Delta(T[j]) \leftarrow \Delta(T[j]) + 1$;
  如果 $x = 0$ 且 $\Delta(T[j]) \neq 0$ **则**
    $nz \leftarrow nz + 1$
  否则，如果 $x \neq 0$ 且 $\Delta(T[j]) = 0$ **则**
    $nz \leftarrow nz - 1$

如果 $nz = 0$ **则**
  返回 真

对于 $i \leftarrow 0$ **到** $n - m - 1$ **执行**
  // 将移位从 W_i 变更为 W_{i+1}：移除出边，添加入边
  $out \leftarrow T[i]$;
  $in \leftarrow T[i + m]$;
  $x \leftarrow \Delta(out)$;
  $\Delta(out) \leftarrow \Delta(out) - 1$;
  如果 $x = 0$ 且 $\Delta(out) \neq 0$ **则**
    $nz \leftarrow nz + 1$
  否则如果 $x \neq 0$ 且 $\Delta(out) = 0$ **则**
    $nz \leftarrow nz - 1$
  $x \leftarrow \Delta(in)$;
  $\Delta(in) \leftarrow \Delta(in) + 1$;
  如果 $x = 0$ 且 $\Delta(in) \neq 0$ **则**
    $nz \leftarrow nz + 1$
  否则，如果 $x \neq 0$ 且 $\Delta(in) = 0$ **则**
    $nz \leftarrow nz - 1$
  如果 $nz = 0$ **则**
    返回 真

返回 假

算法 2： 基于双指针供给可行性的 MFSP

输入： 长度为 n 的文本 T，供给模式 P，字母表 Σ
输出： 一个最长可行子串 $T[\ell^\star..r^\star]$ 及其长度 $L^\star$

对所有 $c \in \Sigma$ 计算 $\mathrm{sup}(c) \leftarrow \mathrm{freq}(P)(c)$。
初始化 $\ell \leftarrow 0$, $L^\star \leftarrow 0$, $(\ell^\star, r^\star) \leftarrow (0, -1)$.
对于所有 $c \in \Sigma$: $\mathrm{cnt}(c) \leftarrow 0$;
$viol \leftarrow 0$。

对于 $r \leftarrow 0$ **到** $n-1$，**执行**
- $c \leftarrow T[r]$;
- $x \leftarrow \mathrm{cnt}(c)$;
- $\mathrm{cnt}(c) \leftarrow \mathrm{cnt}(c) + 1$;
- **如果** $x \le \mathrm{sup}(c)$ 且$\mathrm{cnt}(c) > \mathrm{sup}(c)$ **那么**
  - $viol \leftarrow viol + 1$ // 越过了违规界限
- **当** $viol > 0$ **执行时**
  - $d \leftarrow T[\ell]$;
  - $y \leftarrow \mathrm{cnt}(d)$;
  - $\mathrm{cnt}(d) \leftarrow \mathrm{cnt}(d) - 1$;
  - **如果** $y > \mathrm{sup}(d)$ 且$\mathrm{cnt}(d) \le \mathrm{sup}(d)$ **则**
    - $viol \leftarrow viol - 1$ // 已解决违规
  - $\ell \leftarrow \ell + 1$;
- // 现在 $T[\ell..r]$ 对于这个 r 是可行的，并且是极大的，满足以下条件维护的 ℓ
- **如果** $r - \ell + 1 > L^\star$ **则**
  - $L^\star \leftarrow r - \ell + 1$;
  - $(\ell^\star,\ r^\star) \leftarrow (\ell,\ r)$;

返回 $(\ell^\star,\ r^\star)$ $L^\star$

算法 3： 最大值 不重叠 排列 匹配项（贪心）

输入： 文本 T 长度为 n，模式串 P 长度为 m
输出： 一个不重叠排列匹配的最大基数集合

使用算法 1 计算所有匹配起始位置 $\mathcal{M}$。
初始化 $S \leftarrow \emptyset$ 和 $i \leftarrow 0$。
当 $i \le n - m$ **时执行**
- **如果** $i \in \mathcal{M}$，**则**
  - 将 i 添加到 S // 选择区间 $[i, i+m-1]$
  - $i \leftarrow i + m$ // 跳过以避免重叠
- **else**
  - $i \leftarrow i + 1$

return S