arXiv:2305.00615v1 [cs.DS] 1 May 2023

基于字符串分解的流式 k-编辑距离近似模式匹配

Sudatta Bhattacharya*[1] 和 Michal Koucký†[1]

[1]查理大学计算机科学研究所，Malostranské nám˘estí 25，118 00 捷克共和国布拉格 1

2023年5月2日

摘要

本文给出了一种用于流式 k-编辑距离近似模式匹配的算法，其所需空间为 $\widetilde{O}(k^2)$，每个到达符号所需时间为 $\widetilde{O}(k^2)$。与 Kociumaka、Porat 和 Starikovskaya [近期提出的算法 ~ ~相比，该算法有了显著改进；后者所需空间为 $O(k^5)$，每个到达符号所需时间为 $O(k^8)$。在 k-编辑距离近似模式匹配问题中，给定一个模式 P 和文本 T，我们希望找出文本 T 中与 P的编辑距离至多为 k 的所有子串。在该问题的流式版本中，模式和文本都以流式方式逐个符号到达；文本的每个符号到达后，我们都需要报告文本当前是否存在一个后缀，其与 P的编辑距离至多为 k 。我们衡量算法所需的总空间，以及每个到达符号所需的时间。

1 引言

模式匹配是一项经典问题，旨在文本 T中查找给定模式 P的出现位置。该问题可以在与模式和文本长度成线性的时间内解决 [KJP77, BM77, KR87]。经典算法所需的空间与模式长度成正比。在一项令人惊讶的工作中 [PP09], , Porat 和 Porat 首次设计出了一种占用空间更少的模式匹配算法。他们设计了一种在线算法，预先将模式 P 处理为一个小型数据结构，然后逐个接收文本符号。每接收一个文本符号，该算法便能够报告模式是否与文本当前后缀匹配。该算法使用多对数级内存来存储数据结构并处理文本。这是模式匹配算法设计领域的一项重大成就。

Porat 和 Porat 还提出了一种占用空间较小的在线算法，用于解决汉明距离不超过 k, k-失配的近似模式匹配问题。在这个问题中，给定模式P 和参数 k，我们需要找出文本 T 中所有与 P的汉明距离至多为k 的子串。他们的算法使用 $\widetilde{O}(k^3)$ 空间，每个到达的文本符号需要 $\widetilde{O}(k^2)$ 时间。

*电子邮箱：sudatta@iuuk.mff.cuni.cz。部分经费由捷克共和国资助机构根据资助协议编号 19-27871X 提供。

†电子邮箱：koucky@iuuk.mff.cuni.cz。部分经费由捷克共和国资助机构根据资助协议第 19-27871X 号提供。

随后，这一结果被改进为占用 $\widetilde{O}(k)$ 空间、耗时 $\widetilde{O}(\sqrt{k})$ [CFP+16]。关于在线与流式模式匹配，已有一系列研究成果 [BG14, CFP+15, CFP+16, GP17, GKP18, GS19, Sta17, RS20, GKKP20]；这一研究方向最终发展到 Clifford、Kociumaka 和 Porat 的工作 [CKP19]，他们提出了一种完全流式算法，其参数与 [CFP+16]类似。

在流式场景中，模式也会逐个符号到达，而且我们没有足够的空间一次性存储整个模式。Clifford、Kociumaka 和 Porat 的算法有一个重要特点：它不仅报告模式的 k-失配出现位置，还能针对 P 的每个 k-失配出现位置，输出 P 与文本当前后缀之间差异的完整信息，这称为失配信息。

除了研究基于汉明距离的近似模式匹配，研究人员也考虑基于其他相似性度量的近似模式匹配，例如编辑距离。两个字符串 x 和 y 的编辑距离 ED(x, y)，是将 x 转换为 y 所需的插入、删除和替换操作的最少次数。k-编辑近似模式匹配问题是近似模式匹配的一种变体，我们需要找出 T 中与 P 的编辑距离至多为 k 的所有子串。由于这样的子串可能有二次方个，我们通常只要求针对 T 中的每个位置，判断是否存在一个以该位置结尾、且与 P 的编辑距离至多为 k 的 T 子串。在该问题的流式版本中，每接收到 T 的一个符号，我们都希望输出 P 与文本当前后缀之间的最小距离。同样，我们假设文本和模式都逐个符号到达，并关注算法使用多少空间，以及处理每个符号需要多长时间。

Starikovskaya [Sta17] 提出了一种针对 k-编辑距离模式匹配问题的流式算法，该算法使用 $\widetilde{O}(k^8\sqrt{m})$ 空间，每个到达符号的处理时间为 $\widetilde{O}(k^2\sqrt{m}+k^{13})$。这里，我们记 $m=|P|$和 $n=|T|$。最近，Kociumaka、Porat 和 Starikovskaya [KPS22] 采用一种截然不同的技术构造了一种流式算法，该算法使用 $\widetilde{O}(k^5)$ 空间，并以 $\widetilde{O}(k^8)$ 的摊销时间处理文本的每个到达符号。

在这项工作中，我们显著改进了 Kociumaka、Porat 和 Starikovskaya 的结果。我们给出了一种用于 k-编辑距离近似模式匹配的流式算法，该算法使用 $\widetilde{O}(k^2)$ 空间，每个到达符号的处理时间为 $\widetilde{O}(k^2)$。

定理 1.1。给定整数 $k \geq 0$，存在一种用于 k-编辑距离近似模式匹配问题的随机化流式算法，该算法使用 $\widetilde{O}(k^2)$ 位空间，并以 $\widetilde{O}(k^2)$ 的时间处理文本的每个到达符号。

我们推测，一些摊销技术或许能进一步降低我们的 k-编辑近似模式匹配算法的时间复杂度。不过，似乎不太可能将每个到达符号的复杂度降至 $\sim O(k)$ 以下，因为这样就能以次二次时间解决普通编辑距离问题，从而与强指数时间假设（SETH） [BI15] 相矛盾。能否实现低于 $\widetilde{O}(k^2)$ 的空间复杂度，是一个值得研究的开放问题。目前，人们用于 k-编辑近似模式匹配的所有已知编辑距离草图技术，其草图大小都为 $\Omega(k^2)$。

Kociumaka、Porat 和 Starikovskaya [KPS22] 的编辑距离模式匹配技术，在很大程度上模仿了汉明近似模式匹配算法的内部工作机制。为此，Kociumaka、Porat 和 Starikovskaya 必须为编辑距离设计一种滚动 草图，使多个草图能够“同态地”合并为一个。这需要复杂的机制。这里我们采用一种略有不同的方法。我们使用 Bhattacharya 和 Koucký [BK23]最近提出的一种保持编辑距离的字符串局部一致分解。该分解本质上将编辑距离转换为汉明距离。因此，我们将 Clifford 的 k-不匹配近似模式匹配算法，

Kociumaka 和 Porat [CKP19] 的算法作为黑盒，应用于由该分解产生的符号流。Bhattacharya 和 Koucký [BK23] 还构造了一种更新能力有限的滚动草图，即可以添加和删除符号。我们这里不使用该草图。

1.1 相关工作

Landau 和 Vishkin [LV86] 提出了首个用于 k-失配近似模式匹配问题的算法，其运行时间为 $O(k(m\log\ m+n))$，空间占用为 $O(k(m+n))$。随后，Galil 和 Giancarlo [GG86]将其改进为时间复杂度 $O(m\log\ m+kn)$、空间复杂度 $O(m)$。之后，Amir、Lewenstein 和 Porat [ALP04] 提出了两种算法，运行时间分别为 $O(n\sqrt{k}\log\ k)$ 和 $\widetilde{O}(n+k^3(n/m))$。Clifford、Fontaine、Porat、Sach 和 Starikovskaya [CFP+16] 进一步改进了后一种算法，给出了一种运行时间为 $\widetilde{O}(n+k^2(n/m))$ 的算法。Charalampopoulos、Kociumaka 和 Wellnitz 在其 FOCS'20 论文中 [CKW20], 也提出了一种运行时间为 $\widetilde{O}(n+k^2(n/m))$ 的算法，其 $polylog$ 因子略优。Gawrychowski 和 Uzna´nski [GU18], 给出了一种运行时间为 $\widetilde{O}(n+kn/\sqrt{m})$ 的算法，展示了$O(n\sqrt{k}\log k)$ 与 $\widetilde{O}(n+k^2(n/m))$ 两种运行时间之间的良好权衡。此外，他们还通过证明一个匹配的条件下界，表明其算法在 $polylog$ 因子范围内基本达到了最优。随后，Chan、Golan、Kociu- maka、Kopelowitz 和 Porat [CGK+20]通过一种随机化算法进一步改进了运行时间中的 $polylog$因子，该算法的运行时间为 $O(n+kn(\sqrt{\log m/m}))$。因此，这个问题已得到相当充分的研究。

然而，对于该问题的编辑距离对应问题，最佳上界与已知条件下界之间仍存在显著差距。Landau 和 Vishkin [LV89] 提出了一种运行时间为 $O(nk)$ 的算法。对于较大的 k 值，该算法至今仍是最先进的。Cole 和 Hariharan [CH02] 提出了一种运行时间为 $O(n+m+k^4(n/m))$ 的算法（当 $m\geq k^3$ 时运行速度更快）。在他们关于统一方法的论文中， [CKW20], Charalampopoulos、Kociumaka 和 Wellnitz 也提出了一种运行时间为 $O(n+m+k^4(n/m))$ 的算法。同一作者团队在其 FOCS'22 论文 [CKW22] 中提出了一种运行时间为 $O(n+k^{3.5}\sqrt{\log\ m\log\ kn/m})$ 的算法，终于在 20 年后改进了这一上界。对于下界，Backurs 和 Indyk [BI 15] 证明，若存在一个能够计算编辑距离的真正次二次时间算法，则 SETH 将不成立。这意味着，对于 k-编辑距离近似模式匹配，运行速度显著快于 $O(n+k^2(n/m))$ 的算法极不可能存在。

在线 k-不匹配近似模式匹配问题最早由 Benny Porat 和 Ely Porat 在 2009 [PP09] 中解决。他们提出了一种在线算法，对文本的每个到达符号的处理时间为 $\widetilde{O}(k^2)$，空间占用为 $\widetilde{O}(k^3)$。Clifford、Fontaine、Porat、Sach 和 Starikovskaya 在其 SODA'16 论文 [CFP+16] 中，将其改进为 $\widetilde{O}(k^2)$ 空间，以及对文本的每个到达符号的处理时间为 $O(\sqrt{k}\log k+poly(\log(n)))$ 。Clifford、Kociumaka 和 Porat [CKP19] 提出了一种随机化流式算法，其空间占用为 $O(k\log(m/k))$，每个到达符号的处理时间为 $O(\log(m/k)(\sqrt{k}\log k+\log^3 m))$。该空间上界在对数因子范围内是最优的，与通信复杂度下界相匹配。所有这些算法都使用了某种形式的滚动草图。

在流式模型中，Starikovskaya 提出了一种随机化算法 [Sta17] ，用于解决 k-编辑近似模式匹配问题；该算法占用 $O(k^8\sqrt{m}\log^6 m)$ 的空间，并且每个到达符号的处理时间为 $O((k^2\sqrt{m}+k^{13})\log^4 m)$。Kociumaka、Porat 和 Starikovskaya [KPS22] 提出了一种改进的随机化流式算法，该算法占用 $\widetilde{O}(k^5)$ 的空间，并对文本中的每个到达符号使用 $\widetilde{O}(k^8)$ 的摊销时间。

2 记号与预备知识

我们采用标准记号。对于任意字符串 $x = x_1x_2x_2 \dots x_n$ 和整数 p, q， $x[p]$ 表示 x_p， $x[p, q]$表示 x 的子串 $x' = x_p \dots x_q$，而 $x[p, q) = x[p, q-1]$。如果 $q < p$，则 $x[p, q]$ 是空字符串 ε。 $x[p, \dots]$ 表示 $x[p, |x|]$，其中 $|x|$ 是 x 的长度。"·"运算符用于表示拼接，例如， $x \cdot y$ 表示两个字符串 x 和 y的拼接。对于字符串 x 和 y，ED(x, y) 是将 x 修改为 y所需的最少修改次数（编辑操作），其中一次修改可以是在 x中添加、删除或替换一个字符。除非另有说明，所有对数均以 2 为底。对于整数 $p > q$，根据定义，无论 a_i的情况如何， $\sum_{i=p}^{q} a_i = 0$ 。

2.1 文法

我们将使用 [BK23] 中的以下定义。这些定义基本上是逐字引用的。设 $\Sigma \subseteq \Gamma$ 为两个字母表，且 $\# \notin \Gamma$。文法 G 是一组形如 $c \to ab$ 或 $c \to a^r$ 的规则，其中 $c \in (\Gamma \cup \{\#\}) \setminus \Sigma$、$a$、$b \in \Gamma$ 和 $r \in \mathbb{N}$。 c 是规则的左部， ab 或 a^r 是规则的右部。# 是开始符号。文法的大小 $|G|$ 是 G 中规则的数量。我们只考虑这样的文法：每个 $a \in \Gamma \cup \{\#\}$ 在 G 的左部中至多出现于一条规则中；我们称这类文法为确定性文法。eval(G) 是通过反复应用 G 中的规则重写中间结果，从 # 得到的 Σ^* 中的字符串。如果重写过程永不停止，或停止时得到的字符串不属于 Σ^*，则 eval(G) 未定义。我们用 eval(G_1, G_2, . . ., G_t) 表示 eval(G_1) · eval(G_2) $\cdots$ eval(G_t) 的拼接。通过对确定性文法 G 进行深度优先遍历，我们可以在 O（$|G|$）时间内计算其求值结果长度 |eval(G)| 。给定一个确定性文法 G 和一个不大于其求值结果长度的整数 m，我们可以在 O（$|G|$）时间内构造另一个大小为 O（$|G|$）的文法 G'，使得 eval(G') = eval(G)$[m, \dots]$。 G' 将使用一些新的辅助符号。

We 将使用 Ganesh、Kociumaka、Lincoln 和 Saha 的以下观察 [GKLS22] ：

命题 2.1 ([GKLS22])。存在一种算法，对于大小至多为 m 的两个文法 G_x 和 G_y，可在时间 $O((m + k^2) \cdot \mathrm{poly}(\log(m + n)))$ 内计算 $\mathrm{eval}(G_x)$ 与 $\mathrm{eval}(G_y)$ 的编辑距离 k，其中 $n = |\mathrm{eval}(G_x)| + |\mathrm{eval}(G_y)|$。

我们指出，上述算法还可以输出将 eval(G_x) 转换为 eval(G_y) 的编辑操作的完整信息。我们还将使用以下命题，该命题可由 Landau-Vishkin 算法 [LV86] 得出；例如，可结合 [CKW20] 中的引理 6.2 和定理 7.13 得到：

推论 2.2。对于任意一对分别表示字符串 x 和 y 的文法 G_x 和 G_y，给定参数 k，我们可以在时间 $O((m+k^2)\cdot\mathrm{poly}(\log(m+n)))$ 内找到 $n = |x|+|y|$ 和 $m = |G_x|+|G_y|$，其中 $m = |G_x|+|G_y|$ 是 x 的某个后缀；在 x 的所有后缀中，该后缀与 y 的编辑距离最小，且该后缀与 y 的编辑距离至多为 k。如果 x 的所有后缀与 y 的编辑距离都大于 k，算法将在给定时间内停止，并报告未找到后缀。

3 分解算法

Bhattacharya 和 Koucký [BK23] 提出了一种字符串分解算法（*BK* 分解算法），将输入字符串拆分为多个块，并用小型文法表示每个块。以高概率（针对算法的随机性选择而言），对于长度至多为 n 且编辑距离至多为 k 的两个字符串，分解后块数相同，且对应块中不同的块对至多有 k 对。这两个字符串的编辑距离等于不同块对的编辑距离之和。

具体而言，BK 分解算法接收两个参数 n 和 k、 $k \le n$ 以及一个输入 x。它从特定的哈希函数族中随机选取成对独立函数 C_1、.....、C_L 以及 S-路独立函数H_0、.....、H_L，并利用这些哈希函数将 x 分解为多个块，再为每个块输出一个文法。我们将生成的文法序列称为 x 的*BK* 分解。这里，参数 $L = \lceil \log_{3/2} n \rceil + 3$ 和 $S = O(k \log^3 n \log^* n)$。如 [BK23], 所示，该算法满足以下性质。

命题 3.1 (定理 3.1 [BK23])。 设 x 是一个长度至多为 n的字符串。*BK* 分解算法输出一系列文法 G_1，...，G_s，使得当 n 足够大时：

1. 以至少 $1 - 2/n$ 的概率， $x = \mathrm{eval}(G_1, \ldots, G_s)$。

2. 以至少 $1 - 2/\sqrt{n}$ 的概率，对所有$i \in \{1, \ldots, s\}$， $|G_i| \le S$ 均成立。

算法的随机性 来自函数 $C_1, \ldots, C_L$ 和 $H_0, \ldots, H_L$ 的随机选择 .

函数 $C_1, \ldots, C_L$ 总共可用 $O(\log^2 n)$ 位描述，而 S-路独立函数 $H_0, \ldots, H_L$ 总共可用 $O(S \log^2 n)$ 位描述。我们还需要定理 3.12 [BK23] 的以下特殊情形。

命题 3.2 （定理 3.12 [BK23]）。 设 u、 x、 $y \in \Gamma^*$ 是满足 $|ux|$、 $|y| \le n$ 和 $\mathrm{ED}(x, y) \le k$ 的字符串。设 G_1^x、.....、G_s^x 和 G_1^y、.....、$G_{s'}^y$ 分别是对输入 ux 和 y 使用相同的随机函数选择 C_1、.....、C_L 和 H_0、.....、H_L 后，由 *BK* 分解算法输出的文法序列。以至少 $1 - 1/5$ 的概率，以下结论成立：存在一个整数 $r \ge 1$，使得

$$x = \mathrm{eval}(G_{s-s'+1}^x)[r, \ldots] \cdot \mathrm{eval}(G_{s-s'+2}^x, \ldots, G_s^x) \quad \& \quad y = \mathrm{eval}(G_1^y, \ldots, G_{s'}^y),$$

and

$$\mathrm{ED}(x, y) = \mathrm{ED}(\mathrm{eval}(G_{s-s'+1}^x)[r, \ldots], \mathrm{eval}(G_1^y)) + \sum_{i=2}^{s'} \mathrm{ED}(\mathrm{eval}(G_{s-s'+i}^x), \mathrm{eval}(G_i^y)).$$

可以逐步构建 x 的文法。对于固定的函数选择 C_i， H_i 和字符串 x，如果对于任意字符串 z 及使用相同函数 C_i， H_i、 $G_1^x = G_1^{xz}$、.....、$\sim$ 得到的 xz 的 BK 分解 G_1^{xz}，....., $G_{s'}^{xz}$，文法 G_1^x、.....、G_t^x 在其 BK 分解 G_1^x、.....、G_s^x 中具有 $G_t^x = G_t^{xz}$，则称这些文法是确定的。事实证明，除了最后 $O(1)$ 个文法外， x 的 BK 分解中的所有文法始终都是确定的。以下命题见于 [BK23]:

命题 3.3 (引理 4.2 [BK23])。 设给定 n 和 k，并且 $R = O(\log\, n \log^* n)$ 是适当选取的参数。设 x, $z \in \Gamma^*$, $|xz| \le n$。设给定 $H_0, \ldots, H_L, C_1, \ldots, C_L$。设 $G_1^x, G_2^x, \ldots, G_s^x$ 是 *BK* 分解算法在输入 x时的输出，而 $G_1^{xz}, G_2^{xz}, \ldots, G_{s'}^{xz}$是使用给定哈希函数时分解算法在输入 xz 上的输出。

1. $G_i^x = G_i^{xz}$ *for all* $i = 1 \ldots, s - R$.

2. $|x| \le \sum_{i=1}^{\min(s+R, s')} |\mathrm{eval}(G_i^{xz})|$.

以下命题界定了在末尾追加符号 a 时，更新 x 的 BK 分解所需的资源。

命题 *3.4*（定理 *5.1* [*BK*23]）。给定 $k \le n$，并设 $R = O(\log\ n \log^*\ n)$为经过适当选择的参数。给定函数 C_1、...、C_L 以及 H_0、...、H_L。设 $a \in \Sigma$ 和 $x \in \Sigma^*$ 的长度至多为 n，并设 G_1^x、...、G_s^x为 *BK* 分解算法以 x 为输入、使用函数 C_1、...、C_L、H_0、...、H_L时输出的文法。*UpdateActiveGrammars*($G^x_{s-\min(s,R+1)+1}$、...、G_s^x、a)输出一列文法 G'_1、...、$G'_{t'}$，使得 G_1^x、...、$G^x_{s-\min(s,R+1)}$、G'_1、...、$G'_{t'}$ 正是 *BK* 分解算法以 $x \cdot a$ 为输入、使用相同函数 C_1、...、C_L、H_0、...、H_L时会输出的序列。~更新算法的运行时间为 $O(k)$，并输出 $t' \le 4RL$个文法。

3.1 文法编码

令 S和 $M = O(S\log\ n) = O(k\log^4 n\log^* n)$ 为 BK 分解算法确定的参数。 [BK23] 表明，每个大小至多为 S 的文法都可以编码为一个字符串，其大小为 M，且其字母表 $\{1,\ldots,2\alpha\}$的大小为多项式级，其中整数 α可以选取为满足 $2M/\alpha \le 1/n$。编码 Enc 满足：若两个文法不同，则它们的编码在每个坐标上都不同。该编码是随机化的，选择编码函数需要 $O(\log n)$ 个随机比特。编码可在关于 M 的线性时间内计算；给定 Enc(G)，我们可以在 $O(M)$ 时间内解码 G。该编码满足：

命题 3.5. 设 G、G' 是由 *BK* 分解算法输出的两个大小至多为 S 的文法。随机选取编码 Enc。

1. Enc(G) $\in \{1,\ldots,2\alpha\}^M$。*2.* 若 $G = G'$ 则Enc(G) $=$ Enc(G')。

3. 若 $G \ne G'$，则以至少 $1-(2M/\alpha)$ 的概率，$\mathrm{Ham}(\mathrm{Enc}(G), \mathrm{Enc}(G')) = M$，也就是说，它们在每个符号上都不同。

3.2 k-失配近似模式匹配

Clifford、Kociumaka 和 Porat [CKP19] 设计了一种用于 k-失配近似模式匹配 的流式算法，具有以下性质。算法首先逐个符号读取模式 P，然后逐个符号读取文本 T。每读取文本的一个符号，算法都会报告由文本最后接收的 $|P|$ 个符号组成的字符串与模式之间的汉明距离是否至多为 k。如果它们之间的汉明距离至多为 k，我们可以要求算法报告文本当前后缀与模式之间的失配信息。参数 k 和 n 在算法开始时给定，其中 n 是模式和文本总长度的上界。对于两个长度相同的字符串 x 和 y，我们将它们之间的失配信息 定义为 MIS(x,y) $=$ $\{(i, x[i], y[i]); i \in \{1,\ldots,|x|\}$ 且 $x[i] \ne y[i]\}$。因此， x 和 y 的汉明距离为 Ham(x,y) $=$ |MIS(x,y)|。Clifford、Kociumaka 和 Porat [CKP19] 给出了以下主定理。

命题 3.6 ([CKP19])。 存在一种流式 k-失配近似模式匹配 算法，使用 $O(k\log\ n\log(n/k))$ 位空间，并且每处理一个到达符号需要 $O((\sqrt{k}\log\ k + \log^3 n)\log(n/k))$ 时间。该算法采用随机化，其答案以高概率正确，也就是说，它出错的概率关于 n 为多项式倒数。对于每个报告的出现位置，可在需要时用 $O(k)$ 时间报告失配信息。

4 算法概述

现在，我们概述一下我们的处理方法。我们将对模式 P 应用 BK 分解算法。这样便可得到文法 $G_1^P, G_2^P, \ldots, G_r^P$，用于编码该模式。这需要随着 P 中的符号到达而增量完成。然后，我们将对文本 T 增量应用 BK 分解算法。

我们不会将所有文法都存储在内存中，而是使用 Clifford、Kociumaka 和 Porat 提出的 K-不匹配近似模式匹配算法 [CKP19]（*CKP* 匹配算法）处理这些文法。~此处 $K = k \cdot M$，其中 M 是每个文法的编码大小。对于合适的参数 $R = O(1)$，我们将文法 $G_1^P, \ldots, G_{r-R}^P$ 作为模式输入 CKP 匹配算法。具体而言，我们将使用第 3.1 节中的编码函数 Enc 对每个文法进行编码，并将编码逐个符号地输入 CKP 匹配算法。

随后，随着文本 T 中的符号不断到达，我们将逐步构建 T 的文法，同时只维护一小组活动文法。变为确定的文法将作为输入文本送入 CKP 匹配算法。（同样，每个文法都由 Enc 编码。）CKP 匹配算法将报告模式在文本中的 K-不匹配出现位置。每个 K-不匹配出现位置都对应模式文法与文本文法的一次匹配，其中至多有 k 对文法不同。我们将找出这些不同的文法对，并计算它们的总编辑距离。然后，将此编辑距离与模式的最后 R 个文法和文本的最后 R 个文法之间的编辑距离相加。（文本的最后 R 个文法包含尚未输入 CKP 匹配算法的活动文法。）如果此次匹配的总编辑距离不超过阈值 k，我们就将其报告为 P 在 T中的 k-编辑出现位置。如有需要，我们还可以输出将模式转换为 T的某个后缀所需的编辑操作。（在 T 当前的各个后缀中，我们选择与 P编辑距离最小的那个。）

我们的方案报告 T 中 P 的某个出现位置的成功概率为某个常数 $\geq 1/2$。因此，我们使用独立选择的随机性并行运行这些过程 $O(\log n)$ 次，以实现较低的错误概率。

接下来，我们将更详细地介绍我们的算法。

5 算法描述

现在我们介绍算法的一次运行。算法接收参数 n 和 k，并据此设置参数 $L = O(\log\ n)$、$R = O(\log\ n \log^* n)$、$S = O(k \log^3 n \log^* n)$、$M = O(k \log^4 n \log^* n)$、$K = k \cdot M = O(k^2 \log^4 n \log^* n)$。随后，它随机选择 BK 分解算法所需的成对独立函数 C_1、……、C_L 和 S-路独立函数 H_0、……、H_L。它还为编码函数 Enc 选择所需的随机性。它初始化 CKP 匹配算法，用于在长度至多为 $n \cdot M$ 的字符串上进行 K- 不匹配 近似模式匹配。

该算法分为两个阶段。在第一阶段，算法逐个接收模式 P 符号，并逐步构建一系列文法 G_1^P、……、G_r^P来表示模式 P。除最后一个 R 文法外，其余文法均使用 Enc 编码，并作为模式发送给我们的 CKP 匹配算法实例（逐个发送每个编码中的符号）。在第二阶段，我们的算法逐个接收输入文本 T 符号，并逐步构建一系列文法 G_1^T、G_2^T……来表示接收到的文本。每当某个文法变为确定的 时，就使用 Enc 对其进行编码，并将其作为输入文本的下一部分发送给我们的 CKP 匹配算法实例（逐个发送符号）。

在第一阶段，我们的算法使用命题 3.4 给出的过程来构造文法

G_1^P，……，G_r^P通过逐步添加 P 的符号来完成。算法维护一个包含 $2R$ 活动 文法的缓冲区；每添加一个符号，这些文法都会更新。每当活动文法的数量超过$2R$ 时，我们就对其中确定的最旧的（最靠左的）文法进行编码，并将它们作为其模式的后续部分传递给 CKP 匹配算法实例。更新模式文法的具体细节与更新文本文法的方式类似，我们稍后将进一步说明。输入模式结束后，我们只保留 R 个文法 G_{r-R+1}^P，……，G_r^P，并将其余所有文法发送给 CKP 匹配算法。然后，我们告知 CKP 匹配算法其输入模式已经结束。因此，CKP 匹配算法按此顺序接收了文法 G_1^P，……，G_{r-R}^P 的编码。（如果最终表示 P（$r \leq R$）的文法少于 $R+1$ 个，我们就采用朴素的模式匹配算法，无需使用 CKP 匹配算法。这个简单的情况留作读者练习。）在接下来的描述中，我们假设 $r > R$。

在第二阶段，算法将逐个接收输入文本 T 符号。它将使用命题 3.4 中的算法，逐步构建表示该文本的一系列文法。我们最多保留 R 活动文法 G_1^a，……，G_t^a，并对它们应用命题 3.4 中的算法。活动文法表示 T的一个当前后缀。T 中截至该后缀之前的前缀由文法G_1^T，……，G_s^T表示，这些文法是确定的。在这些确定的文法中，我们只会将最后一个 R 显式存储在缓冲区中，其余文法不会显式存储。（它们将用于计算当前编辑距离，并运行命题 3.4 中的更新算法。）每当检测到某个文法是确定的，我们就会将所有确定文法的编码作为输入文本馈入 CKP 匹配算法。

算法逐步处理文本时，会计算一个整数序列 m_1，m_2，……，m_s，其中只将最后 R 个整数存储在缓冲区中。每个值 m_i 表示 eval(G_1^P，……，G_{r-R}^P)（模式的一个前缀）与 eval(G_1^T，……，G_i^T)（文本某个前缀的一个后缀）的任意后缀之间的最小编辑距离，前提是该编辑距离小于 k。否则，m_i 视为无穷大。（值 m_1，……，m_{r-R-1} 均视为无穷大。）在 G_i^T 变为确定的并且我们将该文法发送给 CKP 匹配算法之后，才会计算值 m_i。（CKP 匹配算法将协助完成该计算。）值 m_i 将用于计算算法接收到的输入文本当前后缀的编辑距离。示意图见图 1。

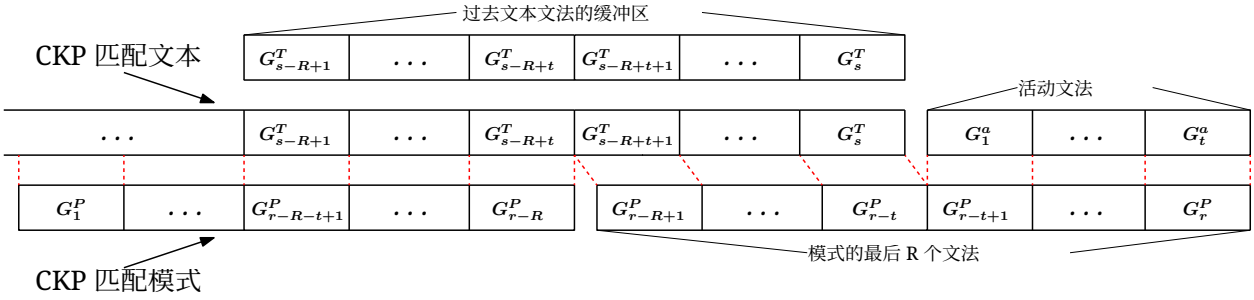

图 1：某个文本符号到达后，文本文法与模式文法的对齐。模式P由文法 G_1^P、……、G_r^P表示。文法 G_1^P、……、G_{r-R}^P 由 Enc 编码，并作为模式发送给 CKP 匹配算法。当前文本 T 由文法序列 G_1^T、……、G_s^T、G_1^a、……、G_t^a表示。文法 G_1^T、……、G_s^T经过编码后，作为文本提交给 CKP 匹配算法。文法 G_1^a、……、G_t^a是文本的活动文法，随着更多符号添加到文本中，这些文法可能会发生变化。

现在我们可以描述算法执行的基本过程。符号到达。 收到输入文本的下一个符号 a 后，我们的算法会调用——

命题 3.4 中的算法，将符号 a追加到文法 $R+1$、……、G_s^T、……、G_t^a中。算法返回文法 G_{s-R+t}^T、……、G_s^T、$G_1'^a$、……、$G_{t'}'^a$，其中 $t' < 4RL$。（这里，eval($G_1'^a$，……，$G_{t'}'^a$) = eval(G_1^a，……，G_t^a) $\cdot\, a$。从算法返回的文法 G_{s-R+t}^T、……、G_s^T将被丢弃，因为它们是确定的，不应发生改变。（更新算法需要这些文法，以便在压缩时拥有正确的上下文。）如果 $t' > R$，那么文法 $G_1'^a$、……、$G_{t'-R}'^a$就会变为确定的，我们将按照后文的说明，将它们逐一提交给 CKP 匹配算法。我们将按 $G_1'^a$、……、$G_{t'-R}'^a$的顺序提交。其余文法 $G_{t'-R+1}'^a$、……、$G_{t'}'^a$将重新标记为 G_1^a、……、G_t^a，并成为追加下一个符号时的活动文法。

此时，我们的算法可以输出模式与截至目前收到的文本的任意后缀之间可能达到的最小编辑距离。下面说明如何计算此类查询。

提交文法。 当文法 G 变为确定的时，算法按如下方式提交该文法。到目前为止，已提交文法 $G_1, \ldots, G_s$，并计算出值序列 $m_1, \ldots, m_s$。我们令 $G_{s+1} = G$，计算编码 Enc(G_{s+1})，并将编码逐个符号发送给 CKP 匹配算法。此时，我们可以利用 CKP 匹配算法提供的失配信息计算 m_{s+1}。如果 $s+1 < r-R$，那么我们令 m_{s+1} 为 ∞；否则，继续按如下方式计算 m_{s+1}。

我们向 CKP 匹配算法查询 $G_1^P, \ldots, G_{r-R}^P$（CKP 匹配算法的模式）的编码与 $G_{s-r+R+2}^T$, $G_{s-r+R+3}^T, \ldots, G_{s+1}^T$（CKP 匹配算法当前文本后缀）的编码之间的汉明距离。如果汉明距离小于 $K = k \cdot M$，则让 CKP 匹配算法恢复失配信息。根据编码函数的设计，如果两个文法不同，那么它们的编码在全部 M 个位置上都不同（除非编码函数 Enc 失效；这种情况仅以可忽略概率发生）。因此，失配信息包含至多 k 对文法的编码，以及它们相对于模式的索引。由此，我们可以从失配信息中恢复文法对 $(G_1, G_1'), \ldots, (G_{k'}, G_{k'}')$，其中 $k' \le k$，G_i 来自文本，而G_i' 来自模式。

如果 (G_1，G_1') 不是第一个文法对 ($G_{s-r+R+2}^T$，G_1^P)（我们通过它们在失配信息中的索引来识别这一点），那么我们计算每一对字符串 eval(G_i) 和 eval(G_i')、$i = 1$、……、k' 的编辑距离。我们将 m_{s+1} 设为这些距离之和。

如果 (G_1，G_1') 是文法对 (G_{s-R+2}^T，G_1^P)，那么我们应用推论 2.2 中的算法，计算 eval(G_1) 的任意后缀与字符串 eval(G_1') 之间的最小编辑距离。对于 $i = 2$、……、k'，我们计算 eval(G_i) 与 eval(G_i') 的编辑距离。我们将 m_{s+1} 设为所计算的 k' 个值之和。

但是，如果 CKP 匹配算法判定其模式与当前后缀之间的汉明距离大于 K，我们就将 $m_{s+1} = \infty$设为。

最后，我们从最近已提交的 R 个文法的缓冲区中丢弃 G_{s-r+R}，并从值为 m_i的缓冲区中丢弃 m_{s-R+2}。我们将 s设为 $s+1$。至此，单个文法G的提交过程完成，接下来可能会提交另一个文法。

模式编辑距离查询。 处理新符号到达后，按照上述说明更新活动文法，并在必要时提交文法，算法即可在当前文本后缀 T 和模式 P 上回答编辑距离查询。此时，文法 $G_1^T, \ldots, G_s^T$ 已经提交给 CKP 匹配算法。当前存在尚未提交给 CKP 匹配算法的活动文法 $G_1^a, \ldots, G_t^a$，此外，输入模式中还有 R 个文法 $G_{r-R+1}^P, \ldots, G_r^P$ 尚未作为其模式的一部分提交给 CKP 匹配算法。为回答编辑距离查询，我们将比较模式的最后 R 个文法 P 与文本的最后几个文法之间的编辑距离，并将其与某个特定值 m_i（即 m_{s-R+t}）结合起来。

设 $d = R - t$。如果 $d > 0$，对于 $i = 1, \ldots, d$，计算每一对 eval(G^T_{s-d+i}) 和 eval(G^P_{r-R+i}) 的编辑距离。（每个文法 G^T_{s-d+i} 都可在最后 R 个已提交文法的缓冲区中获得。）对于 $i = d+1, \ldots, R$，计算每一对 eval(G^a_{i-d}) 和 eval(G^P_{r-R+i}) 的编辑距离。将这些 R 个值与 m_{s-d} 相加。如果总和小于 k，则输出该总和；否则输出 ∞。

由于我们运行了 $O(\log n)$ 个相互独立的算法副本，每个副本都会对编辑距离给出一个估计值，而我们输出最小的估计值。以高概率而言，这就是正确的值。

6 算法的正确性

在本节中，我们论证该算法能够产生正确的输出。首先，我们分析算法失败时某些坏事件发生的概率，然后在假设这些坏事件均未发生的情况下，论证输出的正确性。我们的算法有多个失败来源。

1. BK 分解算法可能会对模式或文本的某个后缀进行分解，所得文法过大，或所得文法无法表示预期的字符串。（命题 3.1 失效。） 2. BK 分解算法能够正确地对模式和文本的所有后缀进行分解，但文本的某个后缀 T 与模式 P 的文法无法良好对齐。（命题 3.2 失效。） 3. 对于 BK 分解算法生成且 CKP 匹配算法需要进行比较的某一对文法，编码函数 Enc 可能会失效。（命题 3.5 失效。） 4. BK 分解算法未失效，但 CKP 匹配算法未能识别其模式的某个 K-失配出现位置，或未能生成正确的失配信息。（命题 3.6 失效。）

事件 1)、3) 和 4) 的失败概率分别受 n 的多项式倒数上界约束，其中 n 是作为已处理字符串长度上界传递给这些算法的参数。因此，如果我们预计算法处理的文本和模式大小至多为 N，则可以将 BK 分解算法的参数 n 设为 N^4，将 CKP 算法的参数设为 $N^4 \cdot M = \widetilde{O}(N^5)$，其中 M 根据 BK 分解算法的 $n = N^4$ 和 k 计算得出。（BK 分解算法的参数 k 设为 k，CKP 算法的参数设为 $K = k \cdot M = \widetilde{O}(k^2)$。）我们将在同一文本和模式上独立运行算法的 $2 \log N$ 个副本。接下来，我们计算算法某个副本在情况 1)、3) 和 4) 下失败的概率。

事件 1. 长度不超过 N 的模式 P 只有一个，命题 3.1 中两个条件之一在 P 上不成立的概率至多为 $4/\sqrt{n} = 4/N^2$。命题 3.1 在文本 T 的至多 N 个前缀上不成立的概率至多为 $N \cdot 4/\sqrt{n} = 4/N$。因此，坏事件 1) 发生的概率至多为 $4/N + 4/N^2$。

事件 3. 由 Enc 编码的模式文法至多有 N 个，由 Enc 编码并已提交的文本文法至多有 N 个。因此，至多有 N^2 对文法可能导致命题 3.5 不成立：将两个不同的文法编码为汉明距离小于 M 的字符串（命题 3.5 第三部分失效）。根据我们的参数设置，坏事件 3) 发生的概率至多为 $N^2/n = 1/N^2$。

事件 4。CKP 匹配算法在执行过程中失败的概率至多为 $1/n = 1/N^4$。

因此，当 N 足够大时，事件 1)、3) 或 4) 导致失败的概率至多为 $5/N$。我们运行 $2\log N$ 份算法副本，因此任何副本因事件 1)、3) 或 4) 而失败的概率至多为 $10\log N/N$。

如果算法执行过程中事件 1)、3) 和 4) 均未发生，那么 BK 分解会将模式和文本正确地分解为文法，Enc 会对这些文法进行正确编码，而 CKP 匹配算法会正确识别已提交的文本文法中所有模式文法的出现位置，并且对于每个出现位置，我们都能正确恢复模式文法与文本文法之间不同的那一对。假设这一情况发生，我们希望说明，我们的算法将以高概率正确识别文本 T 中模式 P 的 k-编辑距离出现位置。

收到文本 $T[1,\ell]$, $\ell \le N$ 的一个前缀后，我们希望确定 $T[1,\ell]$ 的某个后缀与模式 P 的编辑距离是否至多为 k。令 a 满足 $T[a,\ell]$ 与 P 的距离最小。显然，如果 $T[a,\ell]$ 与 P 之间的编辑距离至多为 k，那么 $a \in \{\ell-|P|-k+1,\ldots,\ell-|P|+k+1\}$。将命题 3.2 应用于 $u = T[1,a-1]$、$x = T[a,\ell]$ 和 $y = P$，我们的算法的 $2\log N$ 个副本中，每个副本都有至少 $4/5$ 的概率使 T 的文法与 P 的文法良好对齐。良好对齐意味着 $T[a,\ell]$ 是 $\mathrm{eval}(G^T_{s-r+t+1},\ldots,G^T_s,G^a_1,\ldots,G^a_t)$ 的一个后缀，并且

$$\begin{aligned}\mathrm{ED}(T[a,\ell],P) &= \mathrm{ED}(\mathrm{eval}(G^T_{s-r+t+1})[b,\ldots],\mathrm{eval}(G^P_1))\\ &+ \sum_{i=2}^{r-t}\mathrm{ED}(\mathrm{eval}(G^T_{s-r+t+i}),\mathrm{eval}(G^P_i))\\ &+ \sum_{i=r-t+1}^{r}\mathrm{ED}(\mathrm{eval}(G^a_{i-r+t}),\mathrm{eval}(G^P_i)),\end{aligned}$$

对于合适的 b。此外，a 的最小性意味着

$$\begin{aligned}\mathrm{ED}(T[a,\ell],P) &= \min_b \mathrm{ED}(\mathrm{eval}(G^T_{s-r+t+1})[b,\ldots],\mathrm{eval}(G^P_1))\\ &+ \sum_{i=2}^{r-t}\mathrm{ED}(\mathrm{eval}(G^T_{s-r+t+i}),\mathrm{eval}(G^P_i))\\ &+ \sum_{i=r-t+1}^{r}\mathrm{ED}(\mathrm{eval}(G^a_{i-r+t}),\mathrm{eval}(G^P_i)).\end{aligned}$$

请注意，无论命题 3.2 是否成立，最后一个等式右侧的值始终至少为 $\mathrm{ED}(T[a,\ell],\ P)$，因为它是 P 与 T 的某个后缀之间真实编辑距离的上界。我们将说明，如果该值不超过 k，算法的每个副本都会输出该等式右侧的值；否则输出 ∞。此外，如果我们的算法至少有一个副本使 $T[a,\ell]$ 和 P 对齐良好，那么不同副本输出的值中的最小值就是 $\mathrm{ED}(T[a,\ell],\ P)$。

由于我们有 $2\log N$ 个算法副本，因此没有任何分解能够良好对齐$T[a,\ell]$和 P的概率至多为 $(1/5)^{2\log N} < 1/N^4$。这为接收到 ℓ 个文本符号后输出错误的 $\min_b \mathrm{ED}(T[b,\ell],\ P)$ 值的错误概率提供了上界。由于 ℓ至多有 N个不同的值，在未发生坏事件 1)、3) 或 4) 的条件下，输出 P与T的某个后缀之间编辑距离的错误估计值的概率至多为 $N\cdot 1/N^4 = 1/N^3$。总的来说，当 N足够大时，我们的算法失败的概率至多为 $O(\log N/N) \le 1/\sqrt{N}$；通过采用不同的参数（$n$ 与 N），还可以将其降至 N 的任意小多项式。

接下来需要论证：那个能够良好对齐 $T[a,\ell]$ 和 P的算法副本会输出它们的编辑距离。考虑能够良好对齐 $T[a,\ell]$ 和 P的文法的算法副本。在符号 $T[\ell]$ 到达并更新文法后，存在活动文法 G_1^a，……，G_t^a、已提交文法G_1^T，……，G_s^T以及模式文法 G_1^P，……，G_r^P。如果 ED($T[a,\ell]$，P) 至多为 k，那么 P 与 T 最后 r 个文法不同的文法数量至多为 k。因此，计算设为 m_{s-R+t} 的值时，CKP 匹配算法可以识别出不同的文法，该值设为

$$\begin{aligned} m_{s-R+t} &= \min_b \mathrm{ED}(\mathrm{eval}(G_{s-r+t+1}^T)[b,\ldots],\mathrm{eval}(G_1^P)) \\ &+ \sum_{i=2}^{r-R} \mathrm{ED}(\mathrm{eval}(G_{s-r+t+i}^T),\mathrm{eval}(G_i^P)). \end{aligned}$$

由于 $m_{s-R+t} \leq$ ED($T[a,\ell],P$) $\leq k$，我们得到了 m_{s-R+t} 的真实值。因此，

$$\begin{aligned} \mathrm{ED}(T[a,\ell],P) &= m_{s-R+t} \\ &+ \sum_{i=r-R+1}^{r-t} \mathrm{ED}(\mathrm{eval}(G_{s-r+t+i}^T),\mathrm{eval}(G_i^P)) \\ &+ \sum_{i=r-t+1}^{r} \mathrm{ED}(\mathrm{eval}(G_{i-r+t}^a),\mathrm{eval}(G_i^P)). \end{aligned}$$

这正是我们使用算法评估编辑距离查询的方式。

如果 ED($T[a,\ell],P$) $> k$，那么我们将输出一个值 $> k$，因为我们输出的是编辑距离的某个上界。任何值 $> k$ 都视为无穷大。

7 算法的时间复杂度

在第一阶段，我们使用命题 3.4 中的 BK 分解算法，逐个处理 P 的符号，从而逐步构造模式 P 的文法。为每个~~新符号更新活动文法需要 $O(k)$ 时间，将每个可能的 $O(1)$ 个确定的文法提交给 CKP 匹配算法需要 $\widetilde{O}(M\cdot\sqrt{K}) = \widetilde{O}(k^2)$。因此，模式中每个到达符号所需的时间为 $\widetilde{O}(k^2)$。

在算法的第二阶段，每当文本中有符号到达时，我们都需要更新文本的活动文法、更新 m_s，并计算模式与文本当前后缀之间的编辑距离。这包括算法的 符号到达、提交文法和模式编辑距离查询部分。

符号到达。使用命题 3.4 中的 BK 分解算法追加一个符号需要~$O(k)$ 时间。

提交文法。使用命题 3.5 中的算法对文法进行编码需要 $O(M)$ 时间，而将其提交给 CKP 匹配算法需要 $\widetilde{O}(k^2)$ 时间，与模式情形相同。

查询 CKP 匹配算法最多可以加复 k 对不同的文法（G_i, G_i'），$1 \leq i \leq \widetilde{k}$。使用命题 2.1 计算每对字符串 eval($G_i$) 和 eval($G_i'$) 的编辑距离 k_i，需要 $\widetilde{O}(S+k_i^2) = \widetilde{O}(k+k_i^2)$ 时间。如果 $\sum_i k_i \leq k$，则编辑距离计算的总时间上限为 $\widetilde{O}(k^2)$。如果计算耗时更长，我们可以停止，因为我们知道 m_s 大于 k。对第一对不同的文法运行推论 2.2 中的算法，计算 eval(G_1) 的任意后缀与字符串 eval(G_1') 之间的最小编辑距离

需要 $\widetilde{O}(S + k^2)$ 时间。因此，提交一个文法最多需要 $\widetilde{O}(k^2)$ 时间，其中耗时最长的是对第一对文法运行最小化算法。

模式编辑距离查询。 此步骤需要将模式最后的 R 个文法与文本中相应的文法进行对齐，并计算它们的编辑距离。根据命题 2.1，计算 R 对文法的编辑距离需要 $R \times \widetilde{O}(k^2) = \widetilde{O}(k^2)$ 时间。

由于处理每个新符号后，最多有 $O(1)$ 个已提交文法，因此此步骤对每个到达符号所需的总时间为 $\widetilde{O}(k^2)$。

8 算法的空间复杂度

在算法的任一阶段，我们都会存储 $O(RL)$ 个活跃文法和已更新文法，并将最后的 $O(R)$ 个已提交文法存入缓冲区。这需要 $O(k)$ 的空间。此外，CKP 匹配算法需要 $\widetilde{O}(K) = \widetilde{O}(k^2)$ 的空间。命题 2.1 中的编辑距离算法所需空间不会超过其运行时间，因此每次调用最多使用 $\widetilde{O}(k^2)$ 的空间。同样，推论 2.2 使用 $\widetilde{O}(k^2)$ 的空间。因此，在计算过程中的任意时刻，我们的算法最多使用 $\widetilde{O}(k^2)$ 的空间。

致谢

我们感谢 Tomasz Kociumaka 为推论 2.2 指出了相关参考文献。我们感谢匿名审稿人提出的宝贵意见。

参考文献

[ALP04] Amihood Amir、Moshe Lewenstein 和 Ely Porat。针对 k 次失配的字符串匹配的更快算法。《算法杂志》，50(2):257–275，2004年。[BG14] Dany Breslauer 和 Zvi Galil。实时流式字符串匹配。《*ACM* 算法汇刊》（*TALG*），10(4):1–12，2014年。[BI15] Arturs Backurs 和 Piotr Indyk。除非 SETH 为假，否则无法在强次二次时间内计算编辑距离。载于第四十七届 *ACM* 计算理论年会论文集（*STOC*），第 51–58 页，2015年。[BK23] Sudatta Bhattacharya 和 Michal Koucký。字符串的局部一致分解及其在编辑距离草图化中的应用。载于第*55*届 *ACM SIGACT* 计算理论研讨会论文集（*STOC*，即将出版），2023年。[BM77] Robert S Boyer 和 J Strother Moore。快速字符串搜索算法。《*ACM* 通讯》，20(10):762–772，1977年。[CFP+15] Raphaël Clifford、Allyx Fontaine、Ely Porat、Benjamin Sach 和 Tatiana Starikovskaya。数据流中的字典匹配。载于*Algorithms-ESA 2015*：第*23*届欧洲算法年会，希腊帕特雷，*2015*年*9*月*14–16*日，论文集，第 361–372 页。Springer，2015年。[CFP+16] Raphaël Clifford、Allyx Fontaine、Ely Porat、Benjamin Sach 和 Tatiana Starikovskaya。重新审视 k-失配问题。载于第二十七届 *ACM-SIAM* 离散算法研讨会论文集（*SODA*），第 2039–2052 页。SIAM，2016年。

[CGK+20] Timothy M Chan、Shay Golan、Tomasz Kociumaka、Tsvi Kopelowitz 和 Ely Porat。文本到模式汉明距离的近似计算。载于第*52*届 *ACM SIGACT* 计算理论研讨会论文集，第 643–656 页，2020年。[CH02] Richard Cole 和 Ramesh Hariharan。近似字符串匹配：一种更简单、更快速的算法。*SIAM*计算杂志，31(6):1761–1782，2002年。[CKP19] Raphaël Clifford、Tomasz Kociumaka 和 Ely Porat。流式 k-失配问题。载于第三十届 *ACM-SIAM* 离散算法研讨会论文集（*SODA 2019*），第 1106–1125 页。SIAM，2019年。[CKW20] Panagiotis Charalampopoulos、Tomasz Kociumaka 和 Philip Wellnitz。更快的近似模式匹配：一种统一方法。载于*2020*年*IEEE*第*61*届计算机科学基础年会（*FOCS*），第 978–989 页。IEEE，2020年。[CKW22] Panagiotis Charalampopoulos、Tomasz Kociumaka 和 Philip Wellnitz。编辑距离下更快的模式匹配：将问题归约为动态谜题匹配和置换矩阵的海草幺半群。载于*2022*年*IEEE*第*63*届计算机科学基础年会（*FOCS*），第 698–707 页。IEEE，2022年。[GG86] Zvi Galil 和 Raffaele Giancarlo。改进的 k 次失配字符串匹配。*ACM SIGACT* 新闻，17(4):52–54，1986 年。[GKKP20] Shay Golan、Tomasz Kociumaka、Tsvi Kopelowitz 和 Ely Porat。流式 k-失配问题：空间与总时间之间的权衡。载于 Inge Li Gørtz 和 Oren Weimann 编，第*31*届组合模式匹配年会（*CPM 2020*），第161卷莱布尼茨国际信息学会议论文集（*LIPIcs*），第 15:1–15:15 页，德国达格施图尔，2020年。Schloss Dagstuhl–Leibniz-Zentrum für Informatik。[GKLS22] Arun Ganesh、Tomasz Kociumaka、Andrea Lincoln 和 Barna Saha。压缩与近似如何影响字符串距离度量的效率。载于*2022*年 *ACM-SIAM* 离散算法研讨会论文集（*SODA*），第 2867–2919 页，2022年。[GKP18] Shay Golan、Tsvi Kopelowitz 和 Ely Porat。通过在多个数据流中匹配多个模式，实现最优近似流式模式匹配。载于第*45*届自动机、语言与编程国际研讨会（*ICALP 2018*）。Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik，2018年。[GP17] Shay Golan 和 Ely Porat。常数字母表下的实时流式多模式搜索。载于第*25*届欧洲算法研讨会（*ESA 2017*）。Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik，2017年。[GS19] Paweł Gawrychowski 和 Tatiana Starikovskaya。带失配的流式字典匹配。*Algorithmica*，第 1–21 页，2019年。[GU18] Pawel Gawrychowski 和 Przemyslaw Uznanski。迈向统一的汉明距离与编辑距离近似模式匹配。载于第*45*届自动机、语言与编程国际研讨会（*ICALP 2018*）。Schloss Dagstuhl-Leibniz-Zentrum für Informatik，2018年。

[KJP77] Donald E. Knuth、James H. Morris Jr. 和 Vaughan R. Pratt。字符串快速模式匹配。《*SIAM*计算杂志》，6(2)：323–350，1977年。[KPS22] Tomasz Kociumaka、Ely Porat 和 Tatiana Starikovskaya。使用 k 次编辑的小空间与流式模式匹配。载于*2021* 年 *IEEE* 第 *62* 届计算机科学基础年会（*FOCS*），第 885–896 页。IEEE，2022年。[KR87] Richard M. Karp 和 Michael O. Rabin。高效随机化模式匹配算法。《*IBM*研究与开发杂志》，31(2)：249–260，1987年。[LV86] Gad M Landau 和 Uzi Vishkin。允许 k 次失配的高效字符串匹配。《理论计算机科学》，43：239–249，1986年。[LV89] Gad M Landau 和 Uzi Vishkin。快速并行与串行近似字符串匹配。《算法杂志》，10(2)：157–169，1989年。[PP09] Benny Porat 和 Ely Porat。流式模型中的精确与近似模式匹配。载于*2009* 年 *IEEE* 第 *50* 届计算机科学基础年会，第 315–323 页。IEEE，2009年。[RS20] Jakub Radoszewski 和 Tatiana Starikovskaya。带纠错的流式 k 失配及其应用。《信息与计算》，271：104513，2020 年。[Sta17] Tatiana Starikovskaya。近似模式匹配的通信复杂度与流式复杂度。载于第 *28* 届组合模式匹配年会（*CPM 2017*）。Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik，2017年。