arXiv:2304.05270v2 [cs.DS] 2 Jun 2023

带间隙约束的最长公共子序列

Duncan Adamson、Maria Kosche、Tore Koß、Florin Manea 和 Stefan Siemer

德国哥廷根哥廷根大学计算机科学系

摘要。 我们研究具有间隙约束的最长公共子序列问题。具体而言，遵循 Day 等人于 2022年的研究，我们考虑这样一种情形：子序列中两个连续符号之间的距离（即间隔）必须处于某个下界和上界之间（该上下界可能取决于这些符号在子序列中的位置，也可能取决于间隔两侧的符号）；此外，我们还考虑整个子序列位于一个有界范围内的情形（由单个上界定义），这一情形由Kosche 等人于 2022 年研究。在所有这些情形中，我们都给出了用于确定给定两个字符串之间最长公共约束子序列长度的高效算法。

1 引言

字符串 $w = w[1]w[2]\dots w[n]$ 的子序列 是一个字符串 $v = w[i_1]w[i_2]\dots w[i_k]$，其中 $w[i]$ 是有限字母表 Σ 中的一个符号，且满足 $k \leq n$ 和 $1 \leq i_1 < i_2 \leq \ldots < i_k \leq n$。符号 v 在 w 中出现的位置 i_1、i_2、.....、i_k 定义了 v 在 w 中的嵌入。

一般而言，子序列这一概念出现在理论计算机科学的许多不同领域中，并发挥着重要作用，例如：形式语言与逻辑（如与分段可测试语言 [62,63,38,39,40], 或子词序关系和向下闭包 [32,47,46,66] 相关的研究）；词组合学（如与二项等价、二项复杂性或子词历史 [58,23,50,49,60,55,59] 相关的研究）；算法的设计与复杂性。为此，我们提及一些经典的算法问题，例如计算最长公共子序列 或最短公共超序列 [17,34,35,53,54,56,6,11], 、检验字符串的 Simon 同余，以及计算词的弧因式分解和普遍性 [33,29,64,65,19,7,20,22,30,44]; 另请参见 [45]，其中综述了一些与子序列匹配相关的组合算法问题。此外，近年来，细粒度复杂性领域重新关注了这些问题及其他一些密切相关的问题（见 [13,14,1,3]）。不过，子序列也出现在更具应用性的场景中：用于并发建模 [57,61,15], 、数据库理论（尤其是事件流处理 [5,31,67]）、数据挖掘 [51,52], , 以及与生物信息学相关的问题 [12]。

大多数与子序列相关的问题通常是在子序列任意嵌入词中的情形下进行研究的。然而，在 [21], 中，研究者考虑了一种新颖的情形，其依据是这样一种直觉：在实际场景中，

可以推断嵌入所产生的间隙 具有某些性质。因此， [21] 引入了带间隙约束的子序列这一概念：这类字符串 v 可以通过某个映射 e 嵌入词 w 中，并且嵌入的间隙 （即映射像之间的因子）满足特定性质。在 [21] 中引入并研究这一子序列模型，主要是出于数据-库理论 [41,42], 的动机；间隙必须满足的性质则以长度约束（即对间隙长度的界限）或正则语言约束的形式给出。关于带间隙约束的子序列及其动机的详细介绍，以及对各种相关模型的讨论，请参阅 [21] 。[21] 的主要结果涉及匹配问题的复杂度：给定两个字符串 w 和 v，判断是否存在一个将 v 嵌入 w 作为子序列的嵌入 e，使得 e 所产生的间隙满足给定的长度约束和正则间隙约束。研究者还获得了一系列其他复杂度结果，涉及对满足给定间隙约束集合的词的子序列集合进行分析的问题。 [21] 的结果在 [43], 中得到进一步扩展，作者在那里研究了有界范围内的子序列：这类字符串 v 可以通过某个映射 e 嵌入词 w 中，并且嵌入 v 中所有符号的范围长度至多为 B，其中 B 是给定整数。这项研究源于滑动窗口算法 [24,25,26,27,28], 的应用背景，所得结果同样涉及匹配和分析问题的复杂度。

关于子序列，研究最为广泛的算法问题之一是：求两个字符串的最长公共子序列的长度问题（简称最长公共子序列），参见例如 [17,34,35,53,54,56,6] 或综述 [11]。在这个问题中，给定两个字符串 v 和 w，它们的长度分别为 m 和 n, 且来自大小为 σ的字母表；我们的目标是求最大的 k ，使得存在一个长度为k 的字符串，可以作为子序列嵌入到 v 和 w中。关于最长公共子序列的研究成果包括高效算法（在上述引用的大多数论文中），以及条件复杂度下界 [3,1,2]。特别地，有一种经典算法可以在 $O(N)$ 时间内求解最长公共子序列问题，其中 $N = mn$；有趣的是，若存在一种复杂度为 $O(N^{1-c})$的算法，其中 $c > 0$，则将推翻强指数时间假设（SETH），参见 [1]。

我们的贡献。在本文中，我们研究具有间隔长度约束的子序列背景下的最长公共子序列问题，这类问题似乎具有很强的动机和许多应用（见 [21,43] 及其中的参考文献）。显然，在 [21], 所考虑的模型中，间隙约束取决于子序列的长度，而在最长公共子序列问题中，该长度是未知的，实际上我们需要计算这个长度。因此，需要将 [21] 的模型调整到最长公共子序列的设定中。一种实现方式是认为，我们所寻找的公共子序列的所有间隔都受到同一对上下界的限制；在这种情况下，公共子序列的长度不再发挥作用。我们显著扩展了这一初始想法。一方面，我们考虑存在常数个不同长度约束的情形，这些约束限制间隔（并且它们与词一起给出）。进一步的扩展是考虑以下情形：当我们

除了输入字符串之外，还给定一个长度任意的间隔长度约束元组：该元组中的第 i^{th} 个间隔约束指的是我们尝试寻找的公共子序列中第 i^{th} 个和第 $(i+1)^{th}$ 个符号之间的间隔（并且只有当该子序列的长度至少为 $i+1$ 时才起作用）；显然，最长公共子序列的长度不会超过输入字符串的长度，因此其长度至多为 $\min\{m,n\}$。我们还考虑一种情况：间隔约束由界定间隔的实际字母给出。LCS 的所有这些扩展都涉及受约束子序列模型，其中约束是局部的，即取决于界定间隔的符号的嵌入。最后，我们还考虑有界范围内的子序列的 LCS 问题，其中作为输入给出我们搜索子序列的范围大小上界 B；在这种情况下，我们对子序列的嵌入施加全局约束。

在定义了这些我们认为有趣且有充分动机的最长公共子序列变体之后，我们为每个变体提出了高效算法。在大多数情况下，这些算法都是求解最长公共子序列的标准动态规划算法的非平凡扩展。当间隙是局部的时，我们对最长公共子序列变体的结果作如下简要概述：如果所有间隙约束都相同，或者我们有常数个不同的间隙约束（且间隙约束序列满足某些额外的同步条件），则可以得到运行时间为 $O(N)$的算法；如果我们有任意多个不同的约束，则可以得到运行时间为 $O(Nk)$的算法，其中 k 是输入字符串的最长公共受约束子序列的长度；此外，如果约束序列是递增的，则该问题可以在 $O(N\,\mathrm{polylog}\,N)$ 时间内求解；如果间隔上的约束是根据界定它们的两个字母定义的，则可以得到运行时间为 $O(\min\{N\sigma\log N, N\sigma^2\})$ 的算法；如果间隔上的约束仅根据其后的字母（分别为其前的字母）定义，则可以得到运行时间为 $O(\min\{N\log N, N\sigma\})$ 的算法。对于有界范围内的子序列，我们展示了一种算法，该算法对于最长公共子序列的相应扩展可在 $O(NB^{o(1)})$ 时间内运行（即，对于某个$0<d<1$，其运行时间为 $O(NB^d)$），以及一种运行时间为 $O(N)$ 的 $\frac{1}{3}$–近似算法。

相关工作。关于算法， [37] 的结果涵盖了所有间隙约束均相同的情况。具体而言， [37] 研究了 LCS 的一种变体：两个输入字符串中公共子序列的嵌入所产生的间隙长度均受相同的上下界约束；此外，对于所有 i，还要求 w 中由 i^{th} 间隙所产生的长度与 v中由 i^{th}间隙所产生的长度之差的绝对值不超过某个上界。该论文的作者为此问题提出了一种平方时间算法，并在某些特定情况下推导出更高效的算法。据我们所知，此前文献尚未研究多个间隙长度约束的情况。在 [1], 中，作者将有界范围内的子序列 LCS（称为 Local-2-LCS）作为证明 LCS 复杂度下界的中间步骤；他们只提及了求解该问题的平凡 $O(NB^2)$ 算法。 [16] 的结果给出了该问题的 $O(N^{1+o(1)})$ 解；我们的解建立在这一方法之上。

关于下界，LCS 是本文所考虑所有问题的一个特例，因为只需将所有长度约束设为平凡约束即可：对于间隙约束，取 $(0, n)$；对于有界范围内的子序列，取 $B = n$。因此，对于我们研究的每个问题，若存在一个最坏情况下复杂度为 $O(N^{1-c})$、且满足 $c > 0$ 的算法，就会推翻 SETH。因此，若 $\sigma \in O(1)$（即输入字母表大小为常数），那么我们求解带间隙长度约束的 LCS 的大多数算法都是最优的（除非 SETH 不成立），误差仅为多对数因子；例外是两种情况：我们没有对间隙约束元组施加任何单调性或同步条件。若 σ 不是常数，那么对于根据界定间隙的两个字母来定义约束的情况，上述结论也不再成立。对于有界范围内子序列的 LCS，[1] 证明了即使 B 相对于 n为多对数级，也存在平方级下界。因此，除非 $B \in O(\text{polylog } N)$，我们算法给出的上界与现有下界之间存在超对数级差距。

自然会问，当我们有非平凡约束时会发生什么，例如，形如(a, b)且带有 $a, b \in O(1)$ 的约束。在 [21], 中，研究表明，判断是否存在字符串 v 作为另一个字符串 w的子序列的嵌入，使得该嵌入满足一系列形如 (a, b)且带有 $a, b \leq 6$ 的 $|v|$ 个约束，无法在 $O(N^{1-c})$时间内完成，其中 $c > 0$，除非SETH 为假；此外，相应的归约可以修改为：对于 $O(1)$ 个不同的间隔约束，嵌入也满足我们的同步性质。相应的判定问题还可以通过检查 v 和 w的最长公共约束子序列的长度是否为 $|v|$来求解，其中公共子序列的间隔长度约束恰好是为 v 嵌入 w所定义的那些约束。因此，来自 [21] 的相同下界（它与经典最长公共子序列问题的下界一致）也适用于受约束的最长公共子序列问题，即使我们只有常数数量的常数间隔长度约束，并且这些约束还满足上述同步性质。由于篇幅限制，本版本省略了一些证明；请参见完整版 [4]。

2 预备知识

设 $\mathbb{N} = \{1, 2, \ldots\}$ 为自然数集合，$[n] = \{1, \ldots, n\}$，且 $[m : n] = [n] \setminus [m-1]$，对于 m、$n \in \mathbb{N}$。对于(a, b)、$(c, d) \in \mathbb{N}^2$，我们记 $(a, b) \subseteq (c, d)$当且仅当 $c \leq a$ 且 $b \leq d$。本文中使用的所有对数均以 2 为底。

对于有限字母表 Σ，Σ^+ 表示由 Σ 构成的非空词（或字符串）集合，$\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$（其中 ε 为空词）。对于词 $w \in \Sigma^*$，$|w|$ 表示其长度（特别地，$|\varepsilon| = 0$）。我们规定 $w^0 = \varepsilon$，并且对于每个 $k \geq 1$，$w^k = ww^{k-1}$。对于长度为 n 的词 w 以及某个 $i \in [n]$，我们用 $w[i]$ 表示 w 中位于 i^{th} 位置的字母，因此 $w = w[1]w[2] \cdots w[n]$。对于每个 i，$j \in [|w|]$，我们定义 $w[i : j]$ 若 $i \leq j$，则为 $w[i]w[i+1] \ldots w[j]$；若 $i > j$，则为 $w[i : j] = \varepsilon$。对于 $w \in \Sigma^*$，我们定义 $\text{alph}(w) := \{b \in \Sigma \mid b \text{ 在 } w\}$ 中至少出现一次。字符串 $w[i : j]$ 称为字符串 w 的因子；若 $i = 1$（分别为 $j = |w|$），则 $w[i : j]$ 称为 w 的前缀（分别为后缀）。为简便起见，对于词 w 和两个自然数 $m \leq n$，若 $m \leq |w| \leq n$，我们记作 $w \in [m, n]$。

对于一个 $m \times n$ 矩阵 $M = (M[i,j])_{i\in[m],\ j\in[n]}$ 以及两个集合 $I \subset [m]$、$J \subset [n]$，令 $M[I,J]$ 为子矩阵 $(M[i,j])_{i\in I,\ j\in J}$，它由位于行 $M[i,\cdot]$ 和列 $M[\cdot,j]$ 交点处的元素组成，其中 $i \in I$ 且 $j \in J$。

此外，我们遵循 [21] 定义子序列以及具有间隔长度约束的子序列的概念。我们的定义基于嵌入的概念。对于长度为 n 的字符串 w 和自然数 $k \in [n]$，嵌入是一个函数 $e\colon [k] \to [n]$，满足：对于所有 i、$j \in [k]$，若 $i < j$，则 $e(i) < e(j)$。如果 $e(k) = n$，则称 e 是一个匹配嵌入。对于满足 $|u| \le |w|$ 的字符串 u、$w \in \Sigma^*$，如果嵌入 $e\colon [|u|] \to [|w|]$ 满足 $u = w[e(1)]w[e(2)]\dots w[e(k)],$，则称 e 是 u 的嵌入，并称 u 为 w 的子序列。

对于嵌入 $e\colon [k] \to [|w|]$ 和任意 $j \in [k-1]$，由 e 诱导的 w 的 j^{th} 间隙是字符串间隙$_e(w,j) = w[e(j)+1 :\ e(j+1)-1]$。一个 t 元组的间隙长度约束是一个 t 元组 $gc = (C_1, C_2, \dots, C_t)$，其中 $C_i = (\ell_i, u_i)$，且对每个 $i \in [t]$都有$0 \le \ell_i \le u_i \le n$。对于每个 $i \in [t]$，我们设 $gc[i] = C_i$，并设$gc[1 :\ i] = (C_1, C_2, \dots, C_i)$。若嵌入 e 相对于字符串 w 具有如下形式 $e\colon [k] \to [|w|]$，且对每个 $i \in [k-1]$，$\ell_i \le |$间隙$_e(w,i)| \le u_i$（即，间隙$_e(w,i) \in C_i$），则称它满足一个 $(k-1)$ 元组的间隙长度约束 gc。

如果存在满足间隔约束 gc的 u 到 w 的嵌入 e，我们记为 $u \preceq_{gc} w$。对于一个由$(k-1)$个间隔约束组成的元组 gc，令 $SubSeq(gc, w)$表示由满足 gc的嵌入所诱导的 w 的所有子序列的集合，即$SubSeq(gc, w) = \{u \mid u \preceq_{gc} w\}$。$SubSeq(gc, w)$的元素也称为 w的 gc-子序列。更多详情请参见 [21]。

我们希望在带间隔约束的子序列背景下，定义并研究最长公共子序列问题（简称最长公共子序列）。然而，在 [21], 引入的框架中，间隔约束取决于子序列的（长度），而对于最长公共子序列问题，这一点尚不明确。因此，我们提出一系列问题，引入能够适应间隔长度约束的最长公共子序列变体。在我们所有的问题中，都有两个输入字符串 v 和 w，其中 $|v| = m$ 和 $|w| = n$ 和 $m \le n$，这些字符串属于字母表 $\Sigma = \{1, 2, \dots, \sigma\}$，其中 $\sigma \le m$。本文余下部分，令 $N = mn$。不失一般性地，我们还假设，当输入包含间隔长度约束时，每个单独的约束 $C = (\ell,\ u)$ 均满足 $0 \le \ell \le u \le n$。

首先，再给出一些定义。令 v、$w \in \Sigma^*$ 为两个词；如果 s 是 v 和w的子序列，则称词 s 是 v 和 w 的公共子序列。令 gc 为一个 $(k-1)$-元组的间隔长度约束。长度为 k 的词 s 是公共 gc-子序列的 v 和 w，当且仅当 $s \preceq_{gc} w$ 和 $s \preceq_{gc} v$ 均成立。令$ComSubSeq(gc, v, w)$ 表示 $SubSeq(gc, v) \cap SubSeq(gc, w)$。

如果对于所有 $i \in [k-2]$，$gc[i] \subseteq gc[i+1]$，则称间隙长度约束的 $(k-1)$-元组 gc 为递增的。设 gc 为间隙长度约束的一个递增的 $(k-1)$-元组，并设 $i \in [k-2]$。假设 s 是一个 $gc[1 :$ 嵌入在$w[1 :\ i']$中的 $i]$-子序列，且 s 的最后一个位置映射到 i'；同时，t 也是一个 $gc[1:$ 嵌入在 $w[1 :\ i']$ 中的 $j]$-子序列，且 t 的最后一个位置映射到 i'，并且 $j > i$。如果存在 $a \in \Sigma$，使得 s在 $w[1 :\ i']$ 中的嵌入可以扩展为 sa 在 $w[1:\ i'']$中的嵌入，且该嵌入满足

$gc[1:$ 对于某个 $i'' > i'$, $i+1]$, 那么 t在 $w[1:\ i']$中的嵌入也可以扩展为 ta 在 $w[1:\ i'']$ 中的嵌入，使其满足 $gc[1:\ i+1]$。

一个 $(k-1)$-元组 gc 间隙长度约束被称为同步的，当其满足以下性质：对于所有 i、$j \in [k-1]$, 如果 $gc[i] = gc[j]$ 且 $i \leq j$，那么$gc[i+e] \subseteq gc[j+e]$ 对于所有满足 $i+e \leq j+e \leq m-1$的 $e \geq 0$ 均成立；例如，元组 ((0, 5)(0, 1)(0, 2)(0,3)(0, 1)(0, 5)(0,3)(0,4)) 是同步的。令 gc 为一个同步的 $(k-1)$-元组间隙长度约束，并令 $i \in [k-2]$。假设s 是一个 $gc[1:$ 嵌入在 $w[1$ 中的 $i]$-子序列: $i']$，使得s 的最后一个位置映射到 i'，并且 t 也是一个 $gc[1:$ 嵌入在 $w[1$ 中的 $j]$-子序列: $i']$，使得 t 的最后一个位置映射到 i'，且 $j > i$ 和 $gc[i+1] = gc[j+1]$。现在，如果存在一个字母 a，使得将 s 嵌入 $w[1:\ i']$ 可以扩展为将 sa 嵌入 $w[1:\ i'']$，并且该嵌入满足 $gc[1:\ i+1]$（对于某个 $i'' > i'$），那么将 t 嵌入 $w[1:\ i']$ 也可以扩展为将 ta 嵌入 $w[1:\ i'']$，并且该嵌入满足 $gc[1:\ i+1]$。

最长公共子序列问题（LCS）定义如下。

问题 *1*（LCS）。给定 v、w，计算最大的 $k \in [m]$，使得存在一个 s，它是 v 和 w 的公共子序列，且满足 $|s| = k$。

现在，我们将 LCS 扩展到带间隙约束的子序列情形（有关此问题各种变体的更详细讨论，请参阅完整版 [4]）。首先，我们考虑约束为局部的情况，如 [21] 所示：这些约束仅涉及子序列中两个连续符号之间的间隙。

问题 *2*（LCS-MC）。给定 v、$w \in \Sigma^*$ 和一个由间隙长度约束组成的（$m-1$）元组 gc，计算最大的 $k \in \mathbb{N}$，使得存在 v 和 w 的公共 $gc[1:\ k-1]$-子序列 s，并满足 $|s| = k$。也就是说，求最大的 k，使得 $ComSubSeq$（$gc[1:\ k-1]$、v、w）非空。

显然，最长公共子序列是 LCS-MC 的一个特例，其中 $gc = ((0, n) \ldots, (0, n))$。

在 LCS-MC 中，输入的元组间隙长度约束包含任意多个约束（因此问题名称中包含缩写 MC），其数量最多可达公共子序列 v 和 w 中间隙的最大数量（即 $m-1$）。我们还考虑间隙长度约束的递增元组 gc 对应的 LCS-MC；这一变体称为 LCS-MC-INC。

我们考虑问题 LCS-MC 的两种特殊情况：所有这些约束要么完全相同（即带有一个约束的最长公共子序列），要么取自一个大小为常数的集合（即带有 $O(1)$ 个约束的最长公共子序列）；我们认为这两种情况都很有趣。

问题 *3*（LCS-1C）。给定 v、$w \in \Sigma^*$ 和一个由相同间隔长度约束组成的 $(m-1)$-元组 $gc = ((\ell, u) \ldots, (\ell, u))$，计算最大的 $k \in \mathbb{N}$，使得存在一个公共 $gc[1:\ k-1]$-子序列 s，其属于 v 和 w，且满足 $|s| = k$。

问题*4 (*LCS-O(1)C*).* 给定 v、$w \in \Sigma^*$ 以及一个间隔长度约束的 $(m-1)$-元组 $gc = ((\ell_1, u_1) \ldots, (\ell_{m-1}, u_{m-1}))$，其中 $|\{(\ell_i, u_i) \mid i \in [m-1]\}|$（$gc$的不同约束数量）属于 $O(1)$，计算最大的 $k \in \mathbb{N}$，使得存在一个公共的 $gc[1:\ k-1]$-子序列 s，其来自 v和 w，并满足 $|s| = k$。

通过仅考虑同步间隙长度约束这一受限情形，改进了针对 LCS-MC 得出的一般结果，以适用于 LCS-O(1)C。由此得到的问题称为 LCS-O(1)C-SYNC。

在目前为止介绍的问题中，所搜索子序列中两个连续符号之间的间隔取决于这些符号在相应子序列中的位置（即，i^{th} 与 $i+1^{th}$ 符号之间的间隔始终相同）。也就是说，子序列的实际符号在定义约束时不起作用；重要的只是子序列的长度。对于接下来的问题，连续符号之间间隔的约束由限定相应间隔的一个或两个符号决定，并且不取决于该间隔在子序列中的位置。

为此，我们首先需要修改我们的设定。设 $left\colon\ \Sigma \to [n] \times [n]$ and$right\colon\ \Sigma \to [n] \times [n]$ 是两个函数，用于定义间隔约束。对于一个嵌入 $e\colon [k] \to [n]$，如果对于每个 $i \in [k-1]$，都有 gap$_e(x,i) \in left(x[e(i)]) \cap right(x[e(i+1)])$，我们就称 e 满足相对于字符串 x 由 $(left, right)$ 定义的间隔约束；换句话说，gap$_e(x, i)$ 必须同时满足约束 $left(x[e(i)])$ 和 $right(x[e(i+1)])$，它们由界定该间隔的符号定义。如果存在一个将字符串 y 嵌入 x 且满足间隔约束 $(left, right)$ 的嵌入 e，我们记作 $y \preceq_{left,right} x$，并称 y 是 x 的 $(left, right)$-子序列。在接下来的算法问题中，函数 $g\colon\ \Sigma \to [n] \times [n]$ 给定为由 σ 个元组 $(a, g(a))_{a\in\Sigma}$ 组成的序列。

问题 *5*（最长公共子序列-Σ）。给定两个词 v、$w \in \Sigma^*$ 和两个函数 $left\colon\ \Sigma \to [n] \times [n]$ 以及 $right\colon\ \Sigma \to [n] \times [n]$，计算最大的数 $k \in \mathbb{N}$，使得存在 v 和 w 的一个公共（$left$，$right$）-子序列 s，且 $|s| = k$。

当对于所有 $a \in \Sigma$，$left(a) =$(0,{v4)（分别地，对于所有 $a \in \Sigma$，$right(a) = (0, n)$）时，间隙约束仅由函数 $right$（分别为 $left$）定义，此时问题 LCS-Σ 记为 LCS-ΣR（分别记为 LCS-ΣL）。

在目前为止介绍的问题中，约束是局部的（从它们由子序列中的连续问题定义这一意义上来说）。在我们介绍的最后一个问题中，我们基于已有工作 [1,43], 并考虑出现在长度有界的输入字符串的因子中的子序列。特别地，对于给定整数 B，词 s 是 B-子序列 w，如果存在一个因子$w[i+1\colon\ i+B]$，其属于 w 并包含 s 作为子序列；我们要寻找两个输入字符串的最大公共 B-子序列。该问题在 [1], 中被称为Local-2-最长公共子序列，但由于约束现在作用于整个子序列，我们更倾向于将其称为 LCS-BR（有界范围内的最长公共子序列）。

问题 *6*（LCS-BR）。给定 v、$w \in \Sigma^*$ 和 $B \in [n]$，计算最大的 $k \in \mathbb{N}$，使得存在 v 和 w 的一个公共 B-子序列 s，满足 $|s| = k$。

我们注意到，在所有定义中，我们给定的都是一组间隔长度约束，这意味着 v 和 w 的公共子序列的嵌入都应满足相同的约束。或者，我们也可以输入一组用于 v 的间隔长度约束 gc_v，以及一组用于 w 的间隔长度约束 gc_w，分别约束公共子序列在 v 和 w 中的嵌入。在这种

在这些设置下，嵌入将同时取决于子序列和目标词，而不仅仅取决于子序列，就像论文当前使用的模型那样。

首先，我们注意到，包含单个间隙约束元组的模型对我们而言似乎更自然，因为在词上嵌入子序列时所允许的间隔，似乎更应对应于（或由）子序列的性质决定，而不是对应于嵌入它的文本的性质。例如，在 [21] 以及该论文所依据的工作 [41], 中，间隔是针对希望作为子序列嵌入更长字符串的字符串定义的。

其次，当输入的是两组间隔约束而不是一组间隔约束时，我们的大多数结果仍然原样成立。唯一不能以完全相同形式成立的结果，是那些依赖二维区间最值查询数据结构的结果，即在 $O(N \log N)$时间内运行的最长公共子序列-ΣL/R的解，以及在 $O(N\sigma \log N)$ 时间内运行的最长公共子序列-Σ 的解；在所有这些情况下，我们需要扩展二维区间最值查询结构，以支持对矩形子矩阵（而不仅仅是正方形子矩阵）的查询，这会使复杂度增加一个 $\log n$ 因子。

我们简要介绍用于描述算法的计算模型，以便高效求解第 2 节中所述的问题。该模型是标准的单位代价 RAM，字长为对数级：对于大小为 L 的输入，每个内存字可容纳 $\log L$ 位。因此，对数值 $[1 : L]$ 执行算术和位运算，均假定耗时为 $O(1)$。此外，作为输入给定的数值（例如，用于描述间隙约束的数值）均采用二进制编码。在所有问题中，我们假设给定两个词 w 和 v，其长度分别为 $|w| = n$ 和 $|v| = m$ （因此输入大小为 $L = n + m$），它们取自字母表 $\Sigma = \{1, 2, \ldots, \sigma\}$，且满足 $2 \leq |\Sigma| = \sigma \leq m$。也就是说，我们假设所处理的词是由整数构成的序列（称为字母或符号），每个整数都能装入 $O(1)$ 个内存字中。这是字符串算法中的常见假设：称输入字母表为整数字母表。此外，由于这些问题涉及公共子序列，因此不失一般性地，我们可以假设两个词的字母表相同。关于该计算模型的更多细节，请参见例如 [18]。

3 带有局部间隙约束的最长公共子序列

针对 LCS-MC的一种初始方法。对于所有约束是局部的LCS 变体（即约束取决于子序列中间隔的位置，或取决于界定该间隔的字母），作为 s 候选的子序列集合在最坏情况下相对于 N可能具有指数规模。因此，分别计算两个输入字符串对应的集合、求它们的交集，然后在该交集中寻找最长字符串，将导致一个指数时间算法。不过，最长公共子序列可以通过动态规划方法在 $O(N)$时间内求解（这被视为常识）。类似地，LCS-MC（及其特例LCS-1C 和 LCS-O(1)C，以及 LCS-Σ）也可以通过动态规划

一种在多项式时间内运行的方法。我们仅针对带最大匹配数约束的最长公共子序列描述这一通用思路（因为它很容易调整以适用于所有其他问题）。在某种程度上，这一思路体现了最长公共子序列经典算法的一种效率较低的实现。

对于输入字符串 v、w 和约束 $gc = (C_1, \ldots, C_{m-1})$，我们为每个 $p \in [m]$ 定义一个矩阵 $M_p \in \{0,1\}^{m \times n}$，其中 $M_p[i,j] = 1$ 当且仅当存在字符串 s_p，其长度为 $|s_p| = p$，并且分别嵌入 $v[1:\ i]$ 和 $w[1:\ j]$ 的匹配嵌入 e_v、e_w 满足 $gc[1:\ p-1]$。我们通过设置 $M_1[i,j] = 1$ 当且仅当 $v[i] = w[j]$ 来计算 M_1。接着，我们通过动态规划递归计算 M_p：令 $C_{p-1} = (\ell, u)$，并注意到 $M_p[i,j] = 1$ 当且仅当 $v[i] = w[j]$，且存在位置 i' 满足 $\ell \le i - i' - 1 \le u$，以及位置 j' 满足 $\ell \le j - j' - 1 \le u$，使得存在一个长度为 $p-1$ 的字符串 s_{p-1}，其分别嵌入 $v[1:\ i']$ 和 $w[1:\ j']$ 的匹配嵌入满足 $gc[1:\ p-2]$。也就是说，$M_p[i,j] = 1$ 当且仅当子矩阵 $M_{p-1}[I,J]$ 中存在一个 **1**，其中 $I = [i-u-1:\ i-\ell-1]$ 且 $J = [j-u-1:\ j-\ell-1]$。最后，满足 gc 的 v 和 w 的最长公共子序列的长度 k，等于最大的 p，使得 M_p 不是零矩阵。此方法的朴素实现运行时间为 $O(N^2 k)$。

下面给出了一个更高效的实现。

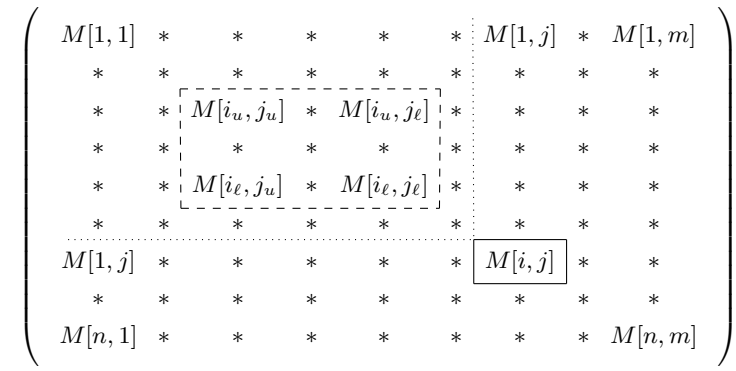


图 1：在引理 1 中，针对 $x \in \{i,j\}$，使用 $x_u = x - u - 1$ 和 $x_\ell = x - \ell - 1$ 对 $M[i,j]$ 的计算。

引理 1. *LCS-MC* 可在 $O(Nk)$ 时间内求解，其中 k 是满足存在公共 $gc[1:\ k-1]$-子序列 s 属于 v 和 w的最大数。

证明。如上所述，我们可以在 $O(N)$ 时间内计算 M_1。因此，设 $2 \le p \le m$, $C_{p-1} = (\ell, u)$，且 $d = |C_{p-1}| = u - \ell + 1$。我们希望计算 M_p 的元素，并假设 M_{p-1} 已经计算完成。为方便起见，当 $i < 1$ 或 $j < 1$ 时，我们将 $M_{p-1}[i,j] = 0$ 视为.....。

我们使用一对 $m \times n$ 矩阵 A 和 B，其中 $A[i,j]$ 存储 M_{p-1}各行中 d 个连续条目 $M_{p-1}[i, j-d+1]$、.....、$M_{p-1}[i,j]$ 的和（或等价地，**1** 的数量）。然后 $A[i,1] = M_{p-1}[i,1]$ 和 $A[i,j] =$

$A[i,j-1] - M_{p-1}[i,j-d] + M_{p-1}[i,j]$ 对所有 $i \in [m]$ 和 $j \in [2:\ n]$。条目 $B[i,j]$ 存储所有满足 $0 \le i - i' < d$ 且 $0 \le j - j' < d$ 的条目 $M_{p-1}[i',j']$ 之和。同样，为方便起见，如果 $i < 1$ 或 $j < 1$，我们将所有条目 $A[i,j]$ 视为 0。然后按如下方式计算 $B[i,j]$。我们设定 $B[1,1] = M_{p-1}[1,1]$、$B[1,j] = B[1,j-1] - M_{p-1}[1,j-d] + M_{p-1}[1,j]$ 和 $B[i,j] = B[i-1,j] - A[i-d,$ 对所有 $i \in [2:\ m]$ 以及 $j \in [2:\ n]$。

由于计算 A 或 B 中的每个元素需要 $O(1)$ 时间，因此我们可以在 $O(N)$ 时间内计算矩阵 A 和 B。现在，当且仅当子矩阵 $M_{p-1}[I,J]$ 中存在一个 1 时，$M_p[i,j] = 1$ 成立；而这当且仅当 $B[i-\ell-1, j-\ell-1] > 0$ 成立。因此，我们可以在常数时间内计算 $M_p[i,j]$，在 $O(N)$ 时间内计算 M_p，并在 $O(Nk)$ 时间内计算序列 $M_1, \ldots, M_{k+1}$。□

一种 $O(N \log^2 N)$ 时间的算法，用于 LCS-MC-INC。现在我们考虑 LCS-MC-INC，它是 LCS-MC 的一个变体，其中元组 gc 按递增顺序排列。我们先介绍一个用于描述数据结构的引理，然后利用该数据结构求解 LCS-MC-INC。

引理 2。给定一个初始时所有元素均为0的 $m \times n$ 矩阵 M，我们可以维护一个数据结构（二维线段树）$\mathcal{T}$，用于 M，从而能够高效地执行以下操作：

– ***update***$_{\mathcal{T}}$***(***i', i'', j', j'', x***)***：对所有 $i \in [i'$，将 $i'']$ 和 $j \in [j'$ 设置为 $j'']$；其中，x 是一个自然数。时间复杂度：$O(\log n \log m)$。– ***query***$_{\mathcal{T}}$***(***i, j***)***：返回 $M[i,j]$。时间复杂度：$O(\log m)$。

证明（概要）。其思路是维护一个二维（简称 2D）线段树 $\mathcal{T}$，用于 M（有关线段树的详细信息，请参阅 [10,9] ）。矩阵 M 的二维线段树 $\mathcal{T}$ 定义如下（例如参见 [48,36],；另请注意，这是一种在竞赛编程中相当常见的数据结构）。

– 我们为范围 $[1:n]$ 定义一棵线段树 T。– 在段树 T 的每个节点 α 中，我们都有一棵线段树 T_α，用于范围 $[1:m]$。– 在树 T_α 的每个节点 β 中，我们存储一个整数值 $val(\beta)$，其初始值为 0。

注意，T 的节点对应于范围 $[1:\ n]$ 的子范围 $[x:y]$。因此，假设我们有一个节点 α，它对应于范围 $[a,b]$。那么，T_α 的节点按规范对应于范围 $[a:b]$（因为它们依赖于 α），同时也对应于 $[1:\ m]$ 中的一个范围 $[c:\ d]$（因为它们是针对 $[1:\ m]$ 的线段树）。因此，T_α 的节点对应于 M 的子矩阵 $M[c:\ d][a:\ b]$。

此外，树 T_α 的叶子与元素 $M[i,j]$ 之间存在一一对应关系，其中 α 是 T 的叶子，且满足 $i \in [m]$ 和 $j \in [n]$。按照上述方式构造结构 $\mathcal{T}$ 时，我们可以为每个 $M[i,j]$ 计算并存储一个指向与其对应叶子的指针。

现在，我们说明这些操作是如何执行的（不详细介绍线段树的标准用法及其相关结果）。

首先考虑 update$_{\mathcal{T}}(i', i'', j', j'', x)$。我们先使用线段树 T_1 找出节点 $\alpha_1, \ldots, \alpha_e$，其中 $e \leq \log\ n$，这些节点对应于区间 $[j' : j'']$的一个划分。找出这些节点的过程采用标准方法实现，耗时 $O(\log n)$。然后，对于每棵树 T_α（其中 $\alpha \in \{\alpha_1, \ldots, \alpha_e\}$），我们找出节点 $\beta_1^\alpha, \ldots, \beta_{e_\alpha}^\alpha$，其中 $e_\alpha \leq \log\ m$，这些节点对应于区间 $[i' : i'']$的一个划分。同样，可以在$O(\log\ m)$时间内找出这些节点。现在，对于 $\alpha \in \{\alpha_1, \ldots, \alpha_e\}$，节点 $\beta_1^\alpha, \ldots, \beta_{e_\alpha}^\alpha$对应于一组 $O(\log\ n \log\ m)$ 个子矩阵，它们将子矩阵 $M[i' : i''][j' : j'']$（其元素需要更新）划分开来。此外，对于 $\alpha \in \{\alpha_1, \ldots, \alpha_e\}$ 和 $\beta \in \{\beta_1^\alpha, \ldots, \beta_{e_\alpha}^\alpha\}$，我们令 $val(\beta) = \max\{val(\beta)\ x\}$。

此更新操作的时间复杂度为 $O(\log\ n \log\ m)$。

要获取 $M[i, j]$的当前值并回答查询$_{\mathcal{T}}(i, j)$，我们按如下方式进行。首先注意，实际条目 $M[i, j]$ 可能并未发生变化。因此，我们需要考虑我们在树上执行的更新。不过，这并不复杂。我们获取与 $M[i, j]$ 对应的叶节点（假设这是节点 T_α中的一个叶节点）。然后，我们沿树向上移动，直到到达这棵树的根节点，并计算一个值 ret。初始时，$ret = M[i, j]$。然后，当我们到达 T_α的节点 β 时，我们更新 $ret \leftarrow \max\{ret, val(\beta)\}$。处理完 T_α 的根节点后，我们停止，并返回 ret 作为 $M[i, j]$的正确值。

此查询操作的时间复杂度为 $O(\log m)$。

该方法的正确性直接源自线段树的性质。□

基于这一数据结构，我们可以证明以下引理。

引理 3. *LCS-MC-INC* 可在 $O(N \log^2 N)$ 时间内求解。

证明。主要思想。我们的算法逐一计算一个 $m \times n$矩阵 M（其元素最初均设为 0）。具体而言，对于满足 $v[i] = w[j]$ 的每一对位置 $(i, j) \in [m] \times [n]$，定义 $M[i, j]$为满足以下条件的最长字符串 s_p 的长度 p：它分别嵌入 $v[1 : i]$和 $w[1 : j]$，且满足 $gc[1 : p - 1]$。由于 gc 是递增的（对所有 $i \in [m - 2]$，$gc[i] \subseteq gc[i + 1]$均成立），因此可按如下方式确定 p。只需将符号 $a = v[i] = w[j]$（在 v 中的位置为 i，在 w中的位置为j）添加到最长子序列 $s_{p'}$ 中，其中 $|s_{p'}| = p'$，并且$s_{p'}$ 在 v 和 w 中的嵌入分别以位置 i' 和 j'结束，同时 i' 与 i 之间的间隔以及 j' 与 j 之间的间隔均满足间隔约束 $gc[p' + 1]$。这是充分的：由于 gc 是递增的，因此与其他任何具有相同性质的较短子序列一样，这个最长子序列也能以完全相同的方式扩展。然后，为了得到 $s_{p'}$，只需设置 $p = p' + 1$，并将字母a添加到 $s_{p'}$ 中；该字母在两个嵌入中分别映射到 $v[i]$ 和 $w[j]$。

然而，在考虑位置 (i, j) 时，我们不知道 p' 的值，因此也不知道需要在哪里查找 i' 和 j' 的范围。因此，我们需要设法绕开这一问题。

动态规划方法。现在我们展示如何计算 M 的元素。对于求解 LCS 的动态规划算法，矩阵 M 的元素 $M[i, j]$ 是通过查看某些元素 $M[i', j']$ 来计算的，

其中 $i' \le i$、$j' \le j$、$(i,\ j) \neq (i',\ j')$。根据上述论证，这种方法似乎无法直接用于 LCS-MC-INC。然而，如果我们知道某个条目 $M[i,j]$ 的值 p，那么对于所有满足 $i+\ell+1 \le i'' \le i+u+1$ 和 $j+\ell+1 \le j'' \le j+u+1$ 的 i'' 与 j''，我们可以确定 $M[i'',j''] \ge p+1$，其中 $gc[p+1] = (\ell,\ u)$；我们会存储这一信息。此外，如果我们已经知道所有 $M[i,j]$ 的值，其中 $i \le i'$、$j \le j'$、$(i,\ j) \neq (i',\ j')$，那么我们已经看到了（并存储了）$M[i',j']$ 的所有可能值（换句话说，即我们可以扩展以获得 $M[i,j]$ 的所有可能子序列），因此我们只需将 $M[i',j']$ 设为这些可能值中的最大值。

因此，我们通过遍历 M 的元素来逐个计算元素 $M[i,j]$：令 i 从 1 到 m，令 j 从 1 到 n，并按如下方式进行。当遍历 M 到达元素 $M[i,j]$ 时，我们只需将其永久设为当前值。然后，如果我们将 $M[i,j]$ 设为某个值 p，且 $gc[p] = (\ell, u)$，就将子矩阵 $M[I,J]$ 中的每个元素 $M[i',j']$ 更新为 $M[i',j'] = \max\{M[i',j'], p+1\}$，其中 $I = [i+\ell+1 :\ i+u+1]$ 且 $J = [j+\ell+1 :\ j+u+1]$。

算法。首先，我们定义矩阵 M，并将其所有条目初始化为 0。然后，我们根据引理 2 为 M 构建数据结构 $\mathcal{T}$。在初始步骤中，我们设定所有满足 $v[1] = w[j]$ 的值 $M[1,j] = 1$，以及满足 $v[i] = w[1]$ 的值 $M[i,1] = 1$；这是通过对 $\mathcal{T}$ 执行更新操作完成的（将条目 $M[i,j] = x$ 设为某个 $x > 0$，鉴于 $M[i,j]$ 等于 0，只需执行 update(i,i,j,j,x)）。此外，我们执行以下过程。

1: 对 $i = 2$ 至 m 执行2: 对 $j = 2$ 至 n 执行 3: 设置 $M[i,j] =$查询$_{\mathcal{T}}(i,j) = p$; $M[i,j]$ 始终等于 p; 4: 对于 (ℓ,u) 更新$_{\mathcal{T}}(i+\ell+1, i+u+1, j+\ell+1, j+u+1, p+1)$。

LCS-MC-INC 的解是 M 中的最大元素。

结论。我们的算法的正确性可由上述论证得出。算法的时间复杂度为 $O(N \log^2 N)$，因为预处理（设置 $\mathcal{T}$ 并对其执行初始更新）需要 $O(N \log^2 N)$ 时间。然后，上述四步过程同样需要 $O(N \log^2 N)$ 时间，因为在内层循环的每次迭代中，最耗时的操作是对 $\mathcal{T}$ 进行更新。我们的结论由此得证。□

综上，我们证明了关于 LCS-MC 的以下定理。

定理 1. *$LCS\text{-}MC$ 可在 $O(Nk)$ 时间内求解，其中 k 是存在公共 $gc[1 :\ k-1]$-子序列 s 的最大值，v 和 w分别为.....。$LCS\text{-}MC\text{-}INC$ 可在 $O(N \log^2 N)$时间内求解。*

$O(N)$ 关于 LCS-1C 和 LCS-O(1)C的解。尽管这个问题已在 [37], 得到解决，我们仍简要介绍我们对此问题的解法。我们的方法基于以下数据结构引理，该引理也用于解决我们在此讨论的其他一些问题。

引理 4. *设 $\Psi\colon [m] \times [n] \to \{0,1\}$ 是一个预定义函数，使得可在 $O(1)$ 时间内检索到 $\Psi(i,j)$。给定一个 $m \times n$ 矩阵 M，其所有元素*

初始值均为 0，以及四个正整数 $\ell_1 \le u_1, \ell_2 \le u_2$，我们可以维护一个用于 M 的数据结构 $\mathcal{D}$，使得以下过程可在 $O(N)$ 时间内运行：

1：对于 $i = 1$ 到 m，执行 *2*：更新 $\mathcal{D}$（为处理第 i 行做准备）；
3：对于 $j = 1$ 到 n，执行 *4*：更新 $\mathcal{D}$（为计算 $M[i, j]$ 做准备）；
5：使用 $\mathcal{D}$ 获取 $\mathtt{m}$，即子矩阵 $M[I, J]$ 中的最大值，其中 $I = [i - u_1 : i - \ell_1]$ 且 $J = [j - u_2 : j - \ell_2]$；当 I 或 J 为空时，$\mathtt{m}$ 设为0 。*6*：如果 $\Psi(i, j) = 1$，则设置 $M[i, j] = \mathtt{m} + 1$。

证明。预处理阶段。在预处理中，我们定义 $\mathcal{D}$。该数据结构包含一个双端队列（deque） Q_f，对应矩阵 M 的每一列 $M[\cdot, f]$，其中 $f \in [n]$，以及一个包含 n 个元素的数组 Max。

主要思想。一般而言，我们对双端队列Q_f维护以下不变性质：双端队列 Q_f 的内容是 Q_f中的第一个元素到 Q_f中的最后一个元素之间（按顺序）的矩阵条目列表 $M[i_{1,f}, f]$、.....、 $M[i_{e_f,f}, f]$，其中 $M[\cdot, j]$满足以下条件：

$-i_{1,\ f} < i_{2,\ f} < \ldots < i_{e_f,\ f}$，$-\ M[i_{1,f}, f] > \ldots > M[i_{e_f,f}, f]$，以及–对于所有$i_{g-1,\ f} < h < i_{g,\ f}$，对于 $g \in [2 : e_f]$。– 对于某些值 i 和 j执行上述过程的第4步后，队列 Q_f（其中 $f \in J$）仅包含子矩阵 $M[I, J]$中的元素。

实现。最初，$\mathcal{D}$ 中的所有双端队列均为空。
数据结构 $\mathcal{D}$ 按如下方式更新（在我们流程的第 2 步中）：对于f从 1 到 n，如果元素 $M[i - 1 - d_1, f]$包含在双端队列 Q_f中，我们就将其从中移除。由于我们对所有值 i都执行此步骤，显然，当我们针对 $i = a$执行第 2 步时，Q_f的第一个元素为某个$e \ge i - 1 - d_1$对应的 $M[e, f]$。此外，此步骤完成后，我们重新计算数组Max，使得 $M[f]$为 Q_f的第一个元素（即 Q_f中最大的元素）与 $M[i - d_1 + \ell_1, f]$之间的最大值。对于数组 Max，我们在 O（n）时间内构建数据结构，以便在O（1）时间内回答区间最大值查询（见 [8]）。此时，我们可以在 O（1）时间内检索任意子数组 $Max[a : b]$ 中的最大元素，其中 $1 \le a \le b \le n$。

此外，$\mathcal{D}$ 按如下方式更新（在我们流程的第 4 步中）。

– 当我们针对某些值 i 和 j执行算法的第 3 步时，会将 $M[i - d_1 + \ell_1, j - d_2 + \ell_2]$ 插入 $Q_{j-d_2-\ell_2}$中。– 对于某个 f，将值 x 插入双端队列 Q_f 的处理方式如下：从最后一个元素开始向前遍历 Q_f，并移除所有小于或等于 x的值。当遇到严格大于 x 的元素，或 Q_f已为空时，我们将值 x 存储为 Q_f的最后一个元素。现在，这个x 是 Q_f 中最小的元素（而 Q_f 中的第一个元素是 Q_f中最大的元素）。

请注意，执行此更新步骤后，对于所有 $f \in J$，双端队列Q_f的第一个元素恰好是元素 $Max[f]$。

因此，利用第 2步中构建的区间最大值查询结构，我们只需查询即可将第 5步实现为查找范围 J 上 Max 中的最大元素。该操作可在 $O(1)$ 时间内返回结果。

结论. 此实现的正确性可由上述解释得出。要分析算法的复杂度，我们首先注意到，步骤 2 需要 $O(n)$ 时间（并执行 m 次），步骤 5 和 6 需要 $O(1)$ 时间（并执行 mn 次）。要了解在算法的整个执行过程中，步骤 4 总共耗费多少时间，只需注意到 M 中的每个元素都会被插入存储在 $\mathcal{D}$ 中的某个双端队列中一次，而步骤 4 所耗费的时间与从这些双端队列中移除的元素数量成正比。因此，此步骤的总体执行时间至多为插入双端队列的元素数量。由此，步骤 4 至多为算法的总体复杂度增加 $O(mn)$ 时间。总之，这意味着按照此处所述的实现，相应过程的运行时间为 $O(N)$。□

现在我们可以立即证明以下结果。

定理 2。 $LCS\text{-}1C$ 可在 $\mathrm{O}(N)$ 时间内求解。

证明。 令 (ℓ, u) 为 gc 中出现的唯一间隔长度约束。我们定义 $m \times n$矩阵 M，其中，当且仅当 p 是满足以下条件的最大数时， $M[i, j] = p$：存在一个子序列 s_p，满足 $|s_p| = p$，使得存在 e_v 和 e_w 两个将 s_p 嵌入 $v[1$ 的匹配嵌入：分别为 $i]$ 和 $w[1:$ $j]$，且二者都满足 $gc[1:\ p-1]$。我们有，当且仅当 $v[i] = w[j]$ 时， $M[i, j] = p$；并且 $p-1$ 是满足 $i-u-1 \le i' \le i-\ell-1$、 $j-u-1 \le j' \le j-\ell-1$ 和 $M[i', j'] = p-1$ 的 i' 与 j' 存在时的最大数。因此，可以利用引理 4 计算 M 的各项，对于 $u_1 = u_2 = u+1$、 $\ell_1 = \ell_2 = \ell+1$ 和 Ψ（i、 j），当且仅当 $v[i] = w[j]$时， $= 1$。□

这一结果可以扩展到 LCS-O(1)C-SYNC 的情形。不过，对于无约束问题 LCS-O(1)C，是否也存在类似结果仍未解决。

定理 3。 $LCS\text{-}O(1)C\text{-}SYNC$ 可以在 $\mathrm{O}(N)$ 时间内求解，其中 O-表示法隐藏的常数与 gc的不同间隙长度约束的数量 h 呈线性关系。

事实上，定理 3 的结果适用于更大范围的约束族，即其元素可以划分为 $h \in O(1)$ 个类的约束族，并且对于所有 $1 \le i < j \le k-1$，如果 i、 j属于该划分的同一类，那么 $gc[i] = gc[j]$，且对于所有满足 $i+e \le j+e \le m-1$的 $e \ge 0$， $gc[i+e] \subseteq gc[j+e]$。

证明。预处理阶段。 假设 $(\ell'_1, u'_1) \ldots, (\ell'_h, u'_h)$ 是集合 $\{(\ell_i, u_i) \mid i \in [m-1]\}$ 中不同约束的一个枚举，其中 h 是常数，且 $(\ell'_g, u'_g) \le (\ell'_{g+1}, u'_{g+1})$（按照自然数对的规范顺序）。在 $O(m+n)$ 时间内，我们可以对 (ℓ_i, u_i) 的约束列表进行基数排序，且

$i \in [m-1]$，并获得列表（ℓ'_1, u'_1）.. .，（ℓ'_h, u'_h），以及一个包含 $m-1$ 个元素的数组 $label$，其中当且仅当（ℓ_i, u_i）=（ℓ'_j, u'_j）时，$label[i] = j$；为简便起见，我们称 $label[i]$ 定义了间隔（$\ell'_{label[i]}, u'_{label[i]}$）。现在我们继续介绍主要算法。

主要思想。我们的方法是：对于每对位置（i, j）$\in [m] \times [n]$，使得 $v[i] = w[j] = a$，以及每个 $r \in [h]$，计算 $v[1: i]$ 和 $w[1: j]$ 的最长公共子序列 s_p，其中 $|s_p| = p$ 和 $label(p) = r$，并满足 $gc[1: p-1]$，且 s_p 的最后一个符号在 s_p 嵌入到 $v[1$ 的过程中映射到 $v[i]: i]$，并在 s_p 嵌入到 $w[1$ 的过程中映射到 $w[j]: j]$。这是通过扩展 v 和 w 的一个公共子序列 s_{p-1} 得到的，其中 $|s_{p-1}| = p-1$，满足 $gc[1: p-2]$，且其最后一个符号分别映射到 v 的位置 i' 和 w 的位置 j'，使得 i' 与 i 之间的间隔以及 j' 与j 之间的间隔满足由 $r' = label[p-1]$ 定义的间隙约束。

这里的主要观察是，由于 gc 的同步性质（对于所有 i、$j \in [m-1]$，如果 $gc[i] = gc[j]$ 且 $i \le j$，那么对于所有 $e \ge 0$ 都有 $gc[i+e] \subseteq gc[j+e]$），$p$ 可以如下确定。对于每个 $r' \in [h]$，只需将符号 a（映射到 v 的位置 i 以及 w 的位置 j）扩展到满足 $label(m_{r'}) = r'$ 的最长子序列 $s_{m_{r'}}$，并且 $s_{m_{r'}}$ 在 v 和 w 中的嵌入分别结束于位置 $i'_{r'}$ 和 $j'_{r'}$，其中 i' 与 i 之间的间隔以及 j' 与 j 之间的间隔满足 r' 所定义的间隔约束。事实上，这已经足够：由于 gc 的同步性，该最长子序列可以以与具有相同性质的任何其他较短子序列完全相同的方式进行扩展。然后，为了得到 s_p，只需扩展满足 $label[m_{r'}+1] = r$ 的子序列 $s_{m_{r'}}$（对于 $r' \in [h]$）中最长的一个。这自然导出了一个动态规划算法，我们将在下文中对其进行描述。

动态规划。对于每个 $r \in [h]$，我们定义一个 $m \times n$ 矩阵 M_r，其中 $M_r[i, j] = p$ 当且仅当 $v[i] = w[j]$，并且 p 是满足以下条件的最大数：r，且存在一个 $gc[1: p-1]$-子序列 s_p，它属于 $v[1: i]$ 和 $w[1: j]$，并且存在该子序列嵌入到 $v[1: i]$ 和 $w[1: j]$ 的两种嵌入 e_v 和 e_w，分别满足 $e_v(p) = i$，$e_w(p) = j$。

根据 M_r 的定义以及上述观察，我们有：$M_r[i, j] = p$ 当且仅当 $v[i] = w[j]$，并且 $p-1$ 是满足以下条件的最大数：$label[p] = r$，且存在 i' 和 j'，使得 $i - u'_{r'} - 1 \le i' \le i - \ell'_{r'} - 1$ 和 $j - u'_{r'} - 1 \le j' \le j - \ell'_{r'} - 1$，从而对于某个 $r' \in [h]$，有 $M_{r'}[i', j'] = p-1$。也就是说，$M_r[i, j] = p$ 当且仅当存在一个长度为 $p-1$ 的子序列嵌入，该嵌入满足间隙约束，并且可以扩展为（同时仍满足约束）一个长度为 p 的子序列，其最后一个符号分别嵌入到 $v[i]$ 和 $w[j]$ 中；此外，p^{th} 间隙（即新找到的 p^{th} 符号之后的间隙）的标签为 r。

因此，要计算 $M_r[i, j]$，我们首先必须为所有 $r' \in [h]$ 计算 $M_{r'}[I_{r'}, J_{r'}]$ 的最大值 $m_{r'}$，其中 $I_{r'} = [i - u'_{r'} - 1: i - \ell'_{r'} - 1]$ 且 $J_{r'} = [j - u'_{r'} - 1: j - \ell'_{r'} - 1]$。然后，我们只需设置 $M_r[i, j] = 1 + \max_{r' \in [h], label[m_{r'}+1] = r} m_{r'}$。

现在需要说明的是，如何在 $I_{r'} = [i - u'_{r'} - 1$ 为 $i - \ell'_{r'} - 1]$ 且 $J_{r'} = [j - u'_{r'} - 1$ 为 $j - \ell'_{r'} - 1]$ 时，求出 $M_{r'}[I_{r'}, J_{r'}]$ 的最大值 $p_{r'}$。为此，我们采用如下所示的引理 4。

算法。我们将 $m \times n$ 矩阵 M_r 初始化为 $r \in [h]$，使其所有元素均为 0，并按照引理 4 为每个矩阵定义一个数据结构 $\mathcal{D}_r$，其中四个输入数分别为 ℓ'_r+1、u'_r+1、ℓ'_r+1、u'_r+1。然后，我们对引理 4 中的过程进行调整，以如下方式计算这些矩阵：

1：对于 $i=1$ 到 m，执行 2：更新 $\mathcal{D}_{r'}$，对于 $r' \in [h]$；3：对于 $j=1$ 到 n，执行 4：更新 $\mathcal{D}_{r'}$，对于 $r' \in [h]$；5：对于 $r' \in [h]$，使用 $\mathcal{D}_{r'}$ 检索 m_r，即满足 $I_{r'} = [i-u'_{r'}-1 :\ i-\ell'_{r'}-1]$ 且 $J_{r'} = [j-u'_{r'}-1 :\ j-\ell'_{r'}-1]$ 的 $M_{r'}[I_{r'}, J_{r'}]$ 中的最大值；当 $I_{r'}$ 或 $J_{r'}$ 为空时，$m_{r'}$ 设为 0。6：计算集合 $max_r = \max_{r' \in [h],\ label[m_{r'}+1]=r} m_{r'}$，对于所有 $r \in [h]$；7：对于 $r \in [h]$，如果 $v[i] = w[j]$，则设置 $M_r[i,j] = 1 + max_r$。

LCS-O(1)-SYNC 的结果是计算结束时存储在某个矩阵 M_r 中的最大值。

结论。 算法的正确性可由以上说明得出。根据引理 4 的结果以及 $h \in O(1)$ 这一事实，其时间复杂度为 $O(N)$。□

首先，我们分析 LCS-ΣR 问题。该问题的输入由两个词 v 和 w 以及一个函数 $right :\ \sigma \to [n] \times [n]$ 组成，其中对于所有 $a \in \Sigma$，$right(a) = (\ell_a, u_a)$（也就是说，对于 $right$-函数，我们有 $right(a) = (0, n)$，对所有$a \in \Sigma$均成立）。我们采用的方法基于引理 4，与 LCS-O(1)C-SYNC 的解法相同。

引理 5。 $LCS\text{-}\Sigma R$ 可在 $O(N\sigma)$ 时间内求解。

证明。主要思想。我们的方法是针对每一对位置 $(i, j) \in [m] \times [n]$，其中 $v[i] = w[j] = a$，计算两个对象v 和 w的最长（$right$）子序列 s_p，其中 $|s_p| = p$，且 s_p 的最后一个符号在 s_p 于 $v[1$ 中的嵌入中映射到 $v[i]$，并在 s_p 于 $w[1$ 中的嵌入中映射到 $w[j] :\ j]$。只需在长度为 r、最后一个符号分别映射到 v 中的位置 i'和 w中的位置 j' 的最长（$right$）子序列 s_r后接上符号 $a = v[i] = w[j]$，即可得到该结果；其中 i' 与 i 之间的间隔以及 j' 与 j 之间的间隔满足由 $right(a)$ 定义的间隔约束。显然，p 定义为 $r+1$。

动态规划。 我们计算一个 $m \times n$ 矩阵 M，其中 $M[i,j]$ 是两个对象 v 和 w 的最长（$right$）子序列 s_p 的长度，其中 $|s_p| = p$，且 s_p 的最后一个符号在 s_p 于 $v[1$ 中的嵌入中映射到 $v[i]$，并在 s_p 于 $w[1$ 中的嵌入中映射到 $w[j] :\ j]$；当且仅当 $v[i] \neq w[j]$时，$M[i,j] = 0$ 成立。为计算 $M[i,j]$，我们需要找出子矩阵 $M[I_a, J_a]$中的最大元素 M_a，其中 $I_a = [i-u_a-1 :\ i-\ell_a-1]$，且 $J_a = [j-u_a-1 :\ j-\ell_a-1]$，并令 $M[i,j] = M_a + 1$。我们可以按以下步骤进行。

算法。我们将 $m \times n$矩阵 M初始化，使其所有条目均为0，并为每个 $a \in \Sigma$ 定义一个数据结构 $\mathcal{D}_a$，如引理4中所述，用于矩阵 M，其四个输入数字分别为 ℓ_a+1、u_a+1、ℓ_a+1、u_a+1，其中$right(a) = (\ell_a, u_a)$。然后，我们调整引理 4 的过程，使其按如下方式工作：

1:对 $i = 1$ 到 m 执行2: 更新 $\mathcal{D}_b$，对于 $b \in \Sigma$； 3:对 $j = 1$ 到 n 执行4:更新 $\mathcal{D}_b$，对于 $b \in' \Sigma$； 5: 如果 $w[j] = v[i]$，令 $a = w[j]$； 5: 使用 $\mathcal{D}_a$ 获取 M_a，即 $M[I_a, J_a]$的最大值，其中 $I_a = [i - u_a - 1 : i - \ell_a - 1]$ 且 $J_a = [j - u_a - 1 : j - \ell_a - 1]$；$M_a$ 在 I_a 或 J_a 为空时设为 0。 6: 设置 $M[i, j] = 1 + M_a$。

结论。 算法的正确性可由上述解释得出。根据引理 4 的结果，以及我们需要维护 σ 个数据结构 $\mathcal{D}_a$（针对 $a \in \Sigma$）这一事实，其时间复杂度为 $O(N\sigma)$；每个数据结构的总维护时间均为 $O(N)$。□

现在，我们介绍另一种用于 LCS-ΣR 的算法，其运行时间为 $O(N \log m)$。

该算法使用二维范围最大值查询数据结构（简称 RMQ）的一种特殊情形。该结构能够在 $O(\log(m))$ 时间内，查询矩阵 M中大小为 $m \times n$ 的正方形子矩阵的最大值，并支持我们在 LCS-ΣR 的解法中所需的一种特殊更新操作。该结构是对 [8] 中稀疏表方法的扩展。

二维范围最大值查询结构。 在基础层面，我们的 RMQ 数据结构维护一个 $m \times n \times (1 + \lceil \log\ m \rceil)$ 数组 $RMQ[\cdot, \cdot, \cdot]$，使得对于 $1 \le i \le m$ 和 $1 \le j \le n$ 以及 $0 \le q \le \lceil \log\ m \rceil$，$RMQ[i, j, q]$ 存储子矩阵 $M[I, J]$ 的最大值，其中 $I = [i - 2^q + 1 : i]$，且 $J = [j - 2^q + 1 : j]$。为简单起见，我们假设当 $i \le 0$ 或 $j \le 0$ 时，$M[i, j] = 0$；当 $i \le 0$ 或 $j \le 0$ 时，$RMQ[i, j, q]$ 定义为 0。此外，在 O(n) 时间的预处理阶段，我们可以计算数组 $Q[1 : n]$；对于 $h \in [m]$，有 $Q[h] = \max\{q \in \mathbb{N} \cup \{0\} \mid 2^q \le h\}$（换言之，$Q[h] = \lfloor \log h \rfloor$）。现在来看，在计算出三维数组 RMQ 后，如何在常数时间内求出 $M[I, J]$ 中任意正方形子矩阵的最大值，其中 $I = [i' : i'']$，且 $J = [j' : j'']$（$|i'' - i'| = |j'' - j'|$）。

引理 6. 给定 $RMQ[\cdot, \cdot, \cdot]$，我们可以在 $O(1)$ 中检索 最大值 $M[I, J]$。

证明。如前所述，假设我们要计算满足 $I = [i' : i'']$ 和 $J = [j' : j'']$ 的最大值 $M[I, J]$。首先，我们需要确定最大的 q，使得大小为 2^q 的正方形完全包含在正方形 $M[I, J]$ 中。也就是说，找到满足 $2^q \le |i' - i''| = |j' - j''| < 2^{q+1}$ 的最大 q，使得 $q = Q[|i' - i''|]$。

我们现在断言，值 $RMQ[i' + 2^q, j' + 2^q, q]$、$RMQ[i' + 2^q, j'', q]$、$RMQ[i'', j' + 2^q, q]$、$RMQ[i'', j'', q]$ 中的最大值是 $M[I, J]$ 中的最大值。我们区分两种情况。

情况 1： $2^q = |i' - i''| = |j' - j''|$。在这种情况下，我们在全部四种情况下都用 $|i' - i''|$ 替换 2^q，立即得到四倍的查找 $RMQ[i'', j'', q]$；根据 $RMQ[\cdot, \cdot, \cdot]$ 的定义，这就是所需的答案。

情况 2： $2^q < |i' - i''| = |j' - j''| < 2^{q+1}$。让我们来看一下由单次 RMQ 查询操作所考虑的值集合。我们有

$M[I_1, J_1], M[I_1, J_2], M[I_2, J_1], M[I_2, J_2]$，其中 $I_1 = [i' : \ i' + 2^q - 1]$, $J_1 = [j' : j' + 2^q - 1]$, $I_2 = [i'' - 2^q + 1 : \ i'']$，以及 $J_2 = [j'' - 2^q + 1 : \ j'']$。由于我们有 $2^q < |i' - i''| < 2^{q+1}$ 和 $2^q < |j' - j''| < 2^{q+1}$，因此我们有 $I_1 \cap I_2 \neq \emptyset$ 和$J_1 \cap J_2 \neq \emptyset$。因此，我们完全涵盖了 $M[I, J]$的所有值。□

现在我们已经了解了数组 RMQ 的用途，接下来看看如何初始化计算所有 q 的 $RMQ[i, j, q]$。显然，对于所有 $i \in [m]$ 和 $j \in [n]$, $RMQ[i, j, 0] = M[i, j]$。然后我们采用动态规划方法。假设 $1 \leq i \leq m$ 和 $1 \leq j \leq n$ 的所有条目 $RMQ[i, j, q-1]$ 都已经计算完毕。那么，$RMQ[i, j, q] = \max\{RMQ[i - 2^{q-1}, j - 2^{q-1}, q-1], RMQ[i, j - 2^{q-1}, q-1], RMQ[i - 2^{q-1}, j, q-1], RMQ[i, j, q-1]\}$。显然，计算 $RMQ[\cdot, \cdot, q]$ 需要 $O(N)$ 时间。因此，计算整个数组 RMQ 需要 $O(N \log m)$ 时间。

基于*RQM*的解法最长公共子序列-ΣR。下面介绍我们针对最长公共子序列-ΣR提出的第二种解法。

引理 7。最长公共子序列-ΣR 可以在 $O(N \log m)$ 时间内求解。

证明。主要思想。我们计算一个 $m \times n$ 数组 $M[\cdot, \cdot]$，使得 $M[i, j] = p$ 当且仅当 p 是满足以下条件的最大数：存在一个 $v[1$ 的 $(right)$-子序列s_p，以及 $w[1$ 的 $j]$，使得存在 s_p 到 $v[1$ 的两个嵌入 e_v 和 e_w，分别满足 $i]$ 和 $j]$，且 $e_v(p) = i$、$e_w(p) = j$。

主要观察是，$M[i, j] = p$ 当且仅当 $v[i] = w[j] = a$，其中$right(a) = (\ell_a, u_a)$，而对于 $I = [i - u_a - 1 : i - \ell_a - 1]$ 和 $J = [j - u_a - 1 : \ j - \ell_a - 1]$，$M[I, J]$中的最大值为 $p - 1$。也就是说，存在$i' \in I$和 $j' \in J$，使得 $M[i', j'] = p - 1$，这表明存在一个 $(right)$-子序列 s_{p-1}，它属于 $v[1 : \ i']$ 和 $w[1 : \ j']$，满足 $|s_{p-1}| = p - 1$，且其最后一个符号分别映射到 $v[i']$ 和 $w[j']$；此外，该子序列 s_{p-1} 可以扩展为一个 $(right)$-子序列 s_p，它属于 $v[1 : \ i]$ 和$w[1 : j]$，且其最后一个符号分别映射到 $v[i]$ 和 $w[j]$。

预处理。首先，我们将 M 的所有值设为 0。然后，对于从 1 到 n 的 j，如果 $v[1] = w[j]$，则将 $M[1, j] = 1$ 设为相应值；对于从 1 到 m 的 i，如果 $v[i] = w[1]$，则将 $M[i, 1] = 1$ 设为相应值。我们还为 M 计算数据结构 $RMQ[\cdot, \cdot, \cdot]$。

动态规划算法。我们按如下方式计算 $M[\cdot, \cdot]$ 的各项：

1: 对于 $i = 2$ 到 m 执行 2: 对于 $j = 2$ 到 n 执行 3: 获取 $a = v[i] = w[j]$ 的约束 $right(a) = (\ell_a, u_a)$; 4: 设置 $I = [i - u_a - 1 : \ i - \ell_a - 1]$ 和 $J = [j - u_a - 1 : \ j - \ell_a - 1]$; 5: 设置 $M[i, j]$为 $M[I, J]$的最大值，使用 RMQ 计算； 6: 设置 $RMQ[i, j, 0] = M[i, j]$; 7: 对于 $q = 1$ 到 $\lceil$记录 $m\rceil$8: 设置 $RMQ[i, j, q] = \max\{RMQ[i - 2^{q-1}, j - 2^{q-1}, q-1], RMQ[i, j - 2^{q-1}, q-1], RMQ[i - 2^{q-1}, j, q-1], RMQ[i, j, q-1]\}$

在计算出 M 的所有条目后，我们只需在 $O(N)$ 时间内计算其最大值，并将其作为最长公共子序列-ΣR 的输出值返回。

结论。我们的算法的正确性可以如下证明。显然，数组 $M[1,\cdot]$ 和 $M[\cdot,1]$ 中的各项均已正确计算，且数据结构 RMQ 已正确初始化。现在假设 $M[i',j']$ 已正确计算，并且对于所有满足 $(i',j') \neq (i,j)$ 的 $i' \leq i$ 和 $j' \leq j$，数据结构 RMQ 均能正确返回 $RMQ[i',j',q]$。根据上述观察，$M[i,j]$ 第 5 步的计算因此是正确的。此外，一旦计算出 $M[i,j]$，第 6、7、8 步便能确保我们的 RMQ 数据结构对于所有满足 $(i',j') \neq (i,j)$ 的 $i' \leq i$ 和 $j' \leq j$，均能正确返回 $RMQ[i',j',q]$。因此，根据归纳法可知，M 中的所有项都已正确计算。

该算法的总体时间复杂度显然为 $O(N \log m)$。 □

针对 最长公共子序列-ΣL 和 最长公共子序列-Σ。此外，我们考虑问题最长公共子序列-ΣL，对于该问题，$right(a) = (0,n)$ 对所有 $a \in \Sigma$ 均成立。该问题的输入由两个词 v, w 以及一个函数 $left$ 组成，其中 $left(a) = (\ell_a, u_a)$ 对所有 $a \in \Sigma$ 均成立。我们可以立即将该问题转换为最长公共子序列-ΣR，其输入字符串为 v^R 和 w^R（即输入字符串的镜像），并使用定义间隙约束的函数 $right'$，其中 $right'(a) = left(a)$，对所有 $a \in \Sigma$ 均成立。然后，我们可以使用在引理 5 和 7 中见到的解法来获得最长公共子序列-ΣL 的解法。

定理4. 最长公共子序列-ΣR，最长公共子序列-ΣL 可在 $O(\text{最大}\{N\sigma, N\text{对数 } m\})$ 时间内求解。

我们的方法可以推广到求解一般问题 LCS-Σ。

定理 5. 最长公共子序列-Σ 可在 $O(\text{最小}\{N\sigma^2, N\sigma\text{对数 } m\})$ 时间内求解。

证明。主要思想。与 LCS-ΣR 的情况一样，我们的方法是针对每对位置 $(i,j) \in [m] \times [n]$，使得 $v[i] = w[j] = a$，计算两者的最长 $(left, right)$-子序列 s_p：v 和 w，其中 $|s_p| = p$，且 s_p 的最后一个符号在 s_p 嵌入 $v[1$ 时映射到 $v[i]$：$i]$，并在 s_p 嵌入 $w[1$ 时映射到 $w[j]$：$j]$。这可以通过将符号 $a = v[i] = w[j]$ 扩展到最长的 $(left, right)$-子序列 s_r 来实现，其长度为 r，最后一个符号（记为 b）映射到 v 的位置 i'，并映射到 w 的位置 j'，且 i' 与 i 之间的间隔以及 j' 与 j 之间的间隔满足由数对（$left$（b） $right$（a））定义的间隔约束；显然，p 随后定义为 $r+1$。

为了求出某个 i 和 j 使得 $v[i] = w[j]$ 时，能够通过添加符号 $a = v[i] = w[j]$ 扩展为更长的 $(left, right)$-子序列的最长 $(left, right)$-子序列 s_r，我们按如下方式进行。令 $right(a) = (\ell_a, u_a)$。对于每个字母 $b \in \Sigma$，令 $left(b) = (\ell'_b, u'_b)$。我们计算这对值 $(\ell_{ab}, u_{ab}) = (\max\{\ell_a, \ell'_b\}, \min\{u_a, u'_b\})$，并令 $I_{ab} = [i - u_{ab} - 1 : i - \ell_{ab} - 1]$ 且 $J_{ab} = [j - u_{ab} - 1 : j - \ell_{ab} - 1]$。然后，我们计算 M_{ab} $M[I_{ab}][J_{ab}]$ 中满足 $v[i'] = w[j'] = b$ 的最大项 $M[i'][j']$。因此，我们需要一种机制，能够从 M 中提取子矩阵的最大值，但只考虑该子矩阵中对应于某个特定字母的项。

动态规划。为此，对于所有 $b \in \Sigma$，我们计算 σ $m \times n$ 矩阵 M_b，以及一个 $m \times n$矩阵 M。我们将 $M[i,j]$定义为 v 和 w的最长（$left$,$right$）-子序列 s_p，其中 $|s_p| = p$，且 s_p 的最后一个符号在 s_p 嵌入到 $v[1$ 中时映射到 $v[i]$： $i]$；在 s_p嵌入到 $w[1$ 中时映射到 $w[j]$： $j]$；显然， $M[i,j] \neq 0$ 当且仅当 $v[i] = w[j]$。然后，对于所有 $b \in \Sigma$，将 $M_b[i,j]$ 初始化为 0；每当我们设置 $M[i,j] = x$时，如果 $v[i] = w[j] = b$，则也设置 $M_b[i,j] = x$。

现在，为了计算 $M[i,j]$，如果 $v[i] = w[j] = a$，我们需要针对所有$b \in \Sigma$，获取子矩阵 $M_b[I_{ab}, J_{ab}]$中的最大元素 M_{ab}，并将 $M[i,j] =$设置为 $\max_{b \in \Sigma} M_{ab} + 1$。

为了避免重复描述相同的算法，我们只非正式地说明其做法。一方面，可以采用引理 5 中的方法，并为每个矩阵 M_b 维护数据结构 $\mathcal{D}_{ab}$，对所有 a，$b \in \Sigma$ （因此总共维护 σ^2 个这样的数据结构）。然后，使用引理 4 中概述的相同算法方法，可以在 $O(N\sigma^2)$ 时间内计算所有元素 $M[i,j]$。另一方面，可以采用引理 7 中的方法，并为每个矩阵 M_b 维护 RMQ 数据结构，对所有 $b \in \Sigma$。使用引理 7 中相同的算法方法，可以在 $O(N\sigma \log\ m)$ 时间内计算所有元素 $M[i,j]$。

结论。该方法的正确性可由上述解释以及关于最长公共子序列-ΣR 的讨论和证明得出。其时间复杂度为 $O(\min\{N\sigma^2, N\sigma \log\ m\})$。 □

4 带全局约束的最长公共子序列

在本节中，我们介绍 LCS-BR 的解法。首先，我们注意到朴素解法：对于两个输入字符串，分别考虑它们中每一对长度为 B 的因子 $(v[i+1: i+B], w[j+1: j+B])$，并使用经典动态规划算法计算它们的最长公共子序列。由于长度为 n 的每个词都有 $n-B+1$ 个长度为 B 的因子，因此该方法需要对 $O((m-B)(n-B)) \subseteq O(N)$ 个词求解最长公共子序列，每次最长公共子序列计算需要 $O(B^2)$ 时间。由此得到总时间复杂度为 $O(NB^2)$。

这里，我们通过 Charalampopoulos 等人 [16]提供的对齐预言机，给出一个 $O(NB^{o(1)})$时间算法，从而改进了这一结果。每个这样的预言机都针对一对词 v 和 w 构建，其中 $|v| = m$、$|w| = n$ 和 $N = mn$，并且能够回答查询，返回两个因子 $v[i: i']$ 和 $w[j$ 之间的最长公共子序列长度： $j']$。 [16] 的结果之一是以下定理。

定理 6（[16]）。我们可以在 $N^{1+o(1)}$ 时间内为词 v 和 w 构造一个对齐预言机，其查询时间为 $\log^{2+o(1)} N$ 。

定理 6 可以直接用于在 $O(N^{1+o(1)})$ 时间内为输入字符串 w 和 v 构建一个对齐预言机 $\mathcal{A}$。利用此预言机，我们可以通过对 $\mathcal{A}$ 进行 $O(N)$ 次查询来求解 LCS-BR，遍历每一对索引 $i \leq m$、$j \leq n$，每次查询需要 $O(\log^{2+o(1)} N)$ 时间。因此，这种直接方法的总时间复杂度为 $O(N^{1+o(1)})$。我们通过创建一组更小的

预言机使我们能够避免为回答超出有界范围的最长公共子序列查询所需的额外工作。为简单起见，不失一般性地假设 m 和 n 是 $2B$ 的倍数。对于所有 i、$w_i = w[iB+1$，令：$(i+2)B]$；或者令 $w_i = w[iB+1:\ n]$，如果 $(i+2)B > n$，并令 $v_i = v[iB+1:$ $(i+2)B]$；或者令 $v_i = v[iB+1:\ n]$，如果 $(i+2)B > m$。注意，w 中每个长度为 B 的因子至少出现在一个子词 w_i 中，而 v 中每个长度为 B 的因子至少出现在一个子词 v_i 中。因此，对每个 $i \in [0, m/B]$、$j \in [0, n/B]$，以 v_i、w_j 为输入求解 LCS-BR，也就能得到以输入字符串 v、w 为输入的 LCS-BR 解。

要找到输入 v_i, w_j的 LCS-BR 解，我们使用定理 6 构造一个用于 v_i 和 w_j的预言机$\mathcal{A}_{i,\ j}$。由于 $|v_i| = |w_j| = 2B$，预言机$\mathcal{A}_{i,\ j}$可以在 $O(B^{2+o(1)})$时间内构造。随后，可以通过向$\mathcal{A}_{i,\ j}$进行 $O(B^2)$次查询来确定输入字符串v_i, w_j的 LCS-BR 解，每次查询需要$O(\log^{2+o(1)} B)$时间。因此，求输入字符串 v_i, w_j的 LCS-BR 的总时间复杂度为 $O(B^{2+o(1)})$。由于有 $O(N/B^2)$对因子 v_i, w_j，求词 v, w的 LCS-BR 的时间复杂度为 $O\left(\frac{N}{B^2} B^{2+o(1)}\right) = O(NB^{o(1)})$。

定理 7。 $LCS\text{-}BR$ 可在 $O(NB^{o(1)})$时间内求解。

我们用一个运行时间为 $O(N)$ 的常数因子近似算法补充精确算法。和之前一样，我们使用子词 $v_i = v[iB+1:(i+2)B]$和 $w_j = w[jB+1:(j+2)B]$。设 s 为 v 与 w之间长度为 B 的有界范围内的最长公共子序列，注意到 $|s| \le \max_{i,j\in[0,n/B]} LCS(v_i, w_j) \le 3|s|$。因此，$\max_{i,j\in[0,n/B]} LCS(v_i, w_j)/3$ 至多是 v 与 w之间最长公共 B-子序列的一个 $(1/3)$-近似。

定理 8. 给定一对词 v，$w \in \Sigma^*$，其长度至多为 n，并令$v_i = v[iB+1:(i+2)B]$ 和，分别为 $w_j = w[jB+1:(2+1)B]$。v 与 w 之间的最长公共 B-子序列 s 的长度满足以下界：

$$\max_{i,j\in[n/B]} LCS(v_i, w_j)/3 \le |s| \le \max_{i,j\in[n/B]} LCS(v_i, w_j)$$

其中，$LCS(v_i, w_j)$ 是字符串 v_i和 w_j的最长公共子序列的长度。此外，这些界可以在 $O(N)$ 时间内求得。

证明。令 $x, y \in [B+1, n]$ 为使 $LCS(v[x-B:x], w[y-B:\ y])$ 最大化的索引对，并令 s为 $v[x-B:x]$ 与 $w[y-B:\ y]$之间的最长公共子序列。注意，s 必须是 v_i 和 w_j的子序列，其中 $i = \lfloor \frac{x}{B} \rfloor$ 且 $j = \lfloor \frac{y}{B} \rfloor$。因此，由于 $LCS(v_i, w_j) \ge |s|$，s 的长度至多为 $\max_{i,j\in[n/B]} LCS(v_i, w_j)$。

从相反方向来看，反设存在某些 v_i, w_j，使得 $|s| < LCS(v_i, w_j)/3$。设 s' 是 v_i 和 w_j 的最长公共子序列，且满足 $|s'| > 3|s|$。现在，令 (x_1, y_1)、(x_2, y_2)、.....、$(x_{|s'|}, y_{|s'|})$ 为一组索引元组，使得 $v[x_\ell] = w[y_\ell] = s'[\ell]$。注意，存在一组索引，使得对于每个 $\ell \in [2, |s'|]$、$x - B \le x_{\ell-1} < x_\ell \le x$ 和 $y - B \le y_{\ell-1} < y_\ell \le y$。因此，如果存在某对索引 $x_\ell > iB + B$, $y_\ell \le jB + B$, 则不可能存在任何

对 $x_{\ell'} \leq iB+B, y_{\ell'} > jB+B$。反之，如果存在某一对 $x_\ell \leq iB+B$, $y_{\ell'} > jB+B$，那么就不可能存在任何一对 $x_{\ell'} \leq iB+B$, $y_{\ell'} > jB+B$。因此，必然存在一个序列集合 s'_1, s'_2, s'_3，使得 $s' = s'_1, s'_2, s'_3$，并且：

- s'_1 是 $v_i[1:B]$ 和 $w_j[1:B]$的子序列。
- s'_2 是以下任一组合的子序列：$v_i[1:B]$ 和 $w_j[B+1:2B]$，或 $v_i[B+1:2B]$和 $w_j[1,B]$。
- s'_3 是 $v_i[B+1:2B]$ 和 $w_j[B+1:2B]$的子序列。

由于 $|s'_1|+|s'_2|+|s'_3| = |s'|$，那么 $LCS(v_i[1:B], w_j[1:B])$、$LCS(v_i[1:B], w_j)[B+1:2B]$、$LCS(v_i[B+1:2B], w_j[1:B])$或 $LCS(v_i[B+1:2B], w_j[B+1:2B])$ 中至少一个的长度必须至少为 $|s'|/3 \leq |S|$。因此，

$$\max_{i,j\in[n/B]} LCS(v_i, w_j)/3 \leq |s| \leq \max_{i,j\in[n/B]} LCS(v_i, w_j).$$

时间复杂度可直接得出。 □

5 未来工作

我们的工作仍留下了一系列有待解决的问题。关于 LCS-MC 的结果能否改进，至少对其特殊情形 LCS-O(1)C 而言是否如此？如果不能，能否为这些问题证明紧的复杂度下界？能否消除 Σ 对 LCS-Σ 及其变体的解的依赖？我们并未着重于削减算法时间复杂度中的多对数因子，但若能实现这一点也会很有意思。尽管如此，对于所有已讨论的问题，如何高效计算实际的最长公共约束子序列，也值得研究。

参考文献

1. Abboud,A.,Backurs,A., Williams, V.V.：LCS及其他序列相似性度量的紧致困难性结果。载于：Guruswami, V.（编）：IEEE第56届计算机科学基础年会，FOCS2015，美国加利福尼亚州伯克利，2015年10月17日至20日。第59–78页。IEEE计算机学会（2015）。https://doi.org/10.1109/FOCS.2015.14, https://doi.org/10 142 1109 2015 . /FOCS. ..
Abboud,A.,Rubinstein, A.：快速且确定性的常数因子近似算法若适用于LCS，则可推出新的电路下界。载于：第9届理论计算机科学创新会议，ITCS 2018，2018年1月11日至14日，美国马萨诸塞州剑桥。第35:1–35:14页（2018）。https://doi.org/10.4230/LIPIcs.ITCS.2018.35, https://doi.org/10 35 3 42302018 . / LIPIcs.ITCS. . . Abboud,A.,Williams,V.V., Weimann, O.：更快速的序列比对所带来的后果。载于：自动机、语言与编程——第41届国际学术会议，ICALP2014，丹麦哥本哈根，2014年7月8日至11日，会议录，第I部分。第39–51页（2014）。https://doi.org/10.1007/978-3-662-43948-7_4

4. Adamson, D., Kosche, M., Koß, T., Manea, F., Siemer, S.：带间隙约束的最长公共子序列。CoRR **即将发表**（2023） 5. Artikis, A., Margara, A., Ugarte, M., Vansummeren, S., Weidlich, M.：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，DEBS 2017，23，2017 19 7 10 2017 10 1145 3093742 巴塞罗那，西班牙，6月 - 。第 – 页（）。https://doi.org/ . / .3095106，https://doi.org/10 3095106 6 1145 3093742 . / . . Baeza-Yates, R.A.：搜索子序列。理论计算机科学 **78**（2），363–376（1991） 7. Barker, L., Fleischmann, P., Harwardt, K., Manea, F., Nowotka, D.：词的散布因子全称性。载于：DLT 2020论文集。《计算机科学讲义》，第12086卷，第14–28页。Springer（2020） 8. Bender, M.A., Farach-Colton, M.：重新审视 LCA 问题。载于：Gonnet, G.H., Panario, D., Viola, A.（编）：LATIN 2000：理论信息学，第4届拉丁美洲研讨会，乌拉圭埃斯特角城，2000年4月10–14日，论文集。《计算机科学讲义》，第1776卷，第88–94页。Springer（2000）。https://doi.org/ 10 1007 10719839_9 10 1007 10719839_9 。 / ，9 https://doi.org/ 。 / 。de Berg, M., Cheong, O., van Kreveld, M.J., Overmars, M.H.：计算几何：算法与应用，第3版。Springer（2008），https://www.worldcat.org/oclc/227584184 10. de Berg, M., Cheong, O., van Kreveld, M.J., Overmars, M.H.：更多几何数据结构。Springer（2008）。https://doi.org/10.1007/978-3-540-77974-2_10，https://www.worldcat.org/oclc/227584184 11. Bergroth, L., Hakonen, H., Raita, T.：最长公共子序列算法综述。载于：de la Fuente, P.（编）第七届国际字符串处理与信息检索研讨会，SPIRE 2000，西班牙拉科鲁尼亚，2000年9月27–29日。第39–48页。IEEE 计算机学会（2000）。https://doi.org/10.1109/SPIRE.2000.878178，https://doi.org/10 878178 12 1109 2000 . /SPIRE. . . Bille, P., Gørtz, I.L., Vildhøj, H.W., Wind, D.K.：具有可变长度间隔的字符串匹配。理论计算机科学 **443**，25 34 2012 10 1016 2012 03 –（）。https://doi.org/ . /j.tcs. . .029，https://doi.org/10 029 13 1016 2012 03 . /j.tcs. . . . Bringmann, K., Chaudhury, B.R.：加权最长公共子序列的草图化、流式处理与细粒度复杂性。载于：FSTTCS 2018论文集。LIPIcs，第122卷，第40:1–40:16页（2018） 14. Bringmann, K., Künnemann, M.：最长公共子序列的多变量细粒度复杂性。载于：SODA 2018论文集。第1216–1235页（2018） 15. Buss, S., Soltys, M.：将平方进行解乱是 NP-困难的。《计算机系统科学杂志》**80**（4），766–776（2014）。https://doi.org/10.1016/j.jcss.2013.11.002，https://doi.org/10 002 16 1016 2013 11 . / j.jcss. . . . Charalampopoulos, P., Gawrychowski, P., Mozes, S., Weimann, O.：近乎最优的编辑距离预言机。载于：Bansal, N., Merelli, E., Worrell, J.（编）第48届自动机、语言与编程国际学术会议，ICALP 2021，2021年7月12–16日，苏格兰格拉斯哥（虚拟会议）。LIPIcs，第198卷，第48:1–48:20页。Schloss Dagstuhl—莱布尼茨信息学中心（2021）。https://doi.org/10.4230/LIPIcs.ICALP.2021.48，https://doi.org/10.4230/LIPIcs.ICALP.2021.48

17. Chvátal, V., Sankoff, D.：两个随机序列的最长公共子序列。《应用概率学杂志》 **12**(2), 306–315 (1975), http://www.jstor.org/stable/3212444 18. Crochemore, M., Hancart, C., Lecroq, T.：《字符串算法》。剑桥大学出版社 (2007) 19. Crochemore, M., Melichar, B., Tronícek, Z.：有向无环子序列图概述。《离散算法杂志》**1**(3-4), 255–280 (2003) 20. Day, J.D., Fleischmann, P., Kosche, M., Koß, T., Manea, F., Siemer, S.：到 k-子序列普适性的编辑距离。载于：第38届计算机科学理论基础国际研讨会 STACS 2021，2021年3月16日至19日，德国萨尔布吕肯（线上会议）。第25:1–25:19页 (2021)。 https://doi.org/10.4230/LIPIcs.STACS.2021.25, https://doi.org/10.4230/LIPIcs.STACS.2021.25 21. Day, J.D., Kosche, M., Manea, F., Schmid, M.L.：带间隙约束的子序列：匹配与分析问题的复杂性界。载于：Bae, S.W., Park, H.（编）第33届算法与计算国际研讨会 ISAAC 2022，2022年12月19日至21日，韩国首尔。LIPIcs，第248卷，第64:1–64:18页。Schloss Dagstuhl—莱布尼茨信息学中心 (2022)。https://doi.org/10.4230/LIPIcs.ISAAC.2022.64, https://doi.org/10.4230/LIPIcs.ISAAC.2022.64 22. Fleischer, L., Kufleitner, M.：Simon同余测试。载于：MFCS 2018论文集。LIPIcs，第117卷，第62:1–62:13页 (2018) 23. Freydenberger, D.D., Gawrychowski, P., Karhumäki, J., Manea, F., Rytter, W.：测试 k-二项等价性。载于：《多学科创造力》，献给 G. Păun 65岁生日的论文集。第239–248页 (2015)，CoRR abs/1509.00622 中可查 24. Ganardi, M., Hucke, D., König, D., Lohrey, M., Mamouras, K.：滑动窗口上的自动机理论。载于：STACS。LIPIcs，第96卷，第31:1–31:14页。Schloss Dagstuhl—莱布尼茨信息学中心 (2018) 25. Ganardi, M., Hucke, D., Lohrey, M.：在滑动窗口上查询正则语言。载于：FSTTCS。LIPIcs，第65卷，第18:1–18:14页。Schloss Dagstuhl—莱布尼茨信息学中心 (2016) 26. Ganardi, M., Hucke, D., Lohrey, M.：正则语言的随机化滑动窗口算法。载于：ICALP。LIPIcs，第107卷，第127:1–127:13页。Schloss Dagstuhl—莱布尼茨信息学中心 (2018) 27. Ganardi, M., Hucke, D., Lohrey, M.：正则语言的滑动窗口算法。载于：LATA。《计算机科学讲义》，第10792卷，第26–35页。Springer (2018) 28. Ganardi, M., Hucke, D., Lohrey, M., Starikovskaya, T.：正则语言的滑动窗口性质测试。载于：ISAAC。LIPIcs，第149卷，第6:1–6:13页。Schloss Dagstuhl—莱布尼茨信息学中心 (2019) 29. Garel, E.：两个词的极小分隔符。载于：CPM 1993论文集。《计算机科学讲义》，第684卷，第35–53页 (1993) 30. Gawrychowski, P., Kosche, M., Koß, T., Manea, F., Siemer, S.：高效测试Simon同余。载于：第38届计算机科学理论基础国际研讨会 STACS 2021，2021年3月16日至19日，德国萨尔布吕肯（线上会议）。第34:1–34:18页 (2021)。 https://doi.org/10.4230/LIPIcs.STACS.2021.34, https://doi.org/10.4230/LIPIcs.STACS.2021.34

31. Giatrakos, N., Alevizos, E., Artikis, A., Deligiannakis, A., Garofalakis, M.N.：大数据时代的复杂事件识别：综述。VLDB J. **29**(1), 313–352 (2020)。 https://doi.org/10.1007/s00778-019-00557-w, https://doi.org/10.1007/s00778-019-00557-w 32. Halfon, S., Schnoebelen, P., Zetzsche, G.：子词序上一阶逻辑的可判定性、复杂性和表达能力。载于：LICS 2017论文集。第1–12页 (2017) 33. Hebrard, J.J.：一种高效区分位字符串的算法。Theor. Comput. Sci. **82**(1), 35–49 (1991年5月22日) 34. Hirschberg, D.S.：最长公共子序列问题的算法。《J. ACM》**24**(4), 664–675 (1977)。https://doi.org/10.1145/322033.322044, https://doi.org/10.1145/322033.322044 35. Hunt, J.W., Szymanski, T.G.：一种计算最长子序列的快速算法。Commun. ACM **20**(5), 350–353 (1977)。 https://doi.org/10.1145/359581.359603, https://doi.org/10.1145/359581.359603 36. Ibtehaz, N., Kaykobad, M., Rahman, M.S.：多维线段树可在多对数时间内执行区间更新。Theor. Comput. Sci. **854**, 30–43 (2021)。https://doi.org/10.1016/j.tcs.2020.11.034, https://doi.org/10.1016/j.tcs.2020.11.034 37. Iliopoulos, C.S., Kubica, M., Rahman, M.S., Walen, T.：计算最长参数化公共子序列的算法。载于：Ma, B., Zhang, K.（编）组合模式匹配，第18届年度研讨会 CPM 2007，2007年7月9日至11日，加拿大伦敦，论文集。《计算机科学讲义》，第4580卷，第265–273页。Springer (2007)。 https://doi.org/10.1007/978-3-540-73437-6_27, https://doi.org/10.1007/978-3-540-73437-6_2738. Karandikar, P., Kufleitner, M., Schnoebelen, P.：关于分段可测试性的Simon同余指数。Inf. Process. Lett. **115**(4), 515–519 (2015) 39. Karandikar, P., Schnoebelen, P.：分段可测试语言的高度及其在逻辑复杂性中的应用。载于：CSL 2016论文集。LIPIcs，第62卷，第37:1–37:22页 (2016) 40. Karandikar, P., Schnoebelen, P.：分段可测试语言的高度与子词逻辑的复杂性。Log. Methods Comput. Sci. **15**(2) (2019) 41. Kleest-Meißner, S., Sattler, R., Schmid, M.L., Schweikardt, N., Weidlich, M.：从轨迹中发现事件查询：为带通配符和间隙大小约束的子序列查询奠定基础。载于：第25届数据库理论国际会议 ICDT 2022。LIPIcs，第220卷，第18:1–18:21页。Schloss Dagstuhl—莱布尼茨信息学中心 (2022)。 https://doi.org/10.4230/LIPIcs.ICDT.2022.18, https://doi.org/10.4230/LIPIcs.ICDT.2022.18 42. Kleest-Meißner, S., Sattler, R., Schmid, M.L., Schweikardt, N., Weidlich, M.：从轨迹中发现多维子序列查询——从理论到实践。载于：König-Ries, B., Scherzinger, S., Lehner, W., Vossen, G.（编）面向商业、技术与网络的数据库系统（BTW 2023），德国信息学会数据库与信息系统专业组第20届会议，2023年3月6日至10日，德国德累斯顿，论文集。LNI，第P-331卷，第511–533页。Gesellschaft für Informatik e.V. (2023)。https://doi.org/10.18420/BTW2023-24, https://doi.org/10.18420/BTW2023-24

43. Kosche, M., Koß, T., Manea, F., Pak, V.：有界范围内的子序列：匹配与分析问题。载于：Lin, A.W., Zetzsche, G., Potapov, I.（编）：可达性问题——第16届国际会议，RP 2022，凯泽斯劳滕，德国，2022年10月17日至21日，论文集。《计算机科学讲义》，第13608卷，第140–159页。Springer（2022）。 https://doi.org/10.1007/978-3-031-19135-0_10, https://doi.org/10.1007/978-3-031-19135-0_1044. Kosche, M., Koß, T., Manea, F., Siemer, S.：词中的缺失子序列。载于：Bell, P.C., Totzke, P., Potapov, I.（编）：可达性问题——第15届国际会议，RP 2021，利物浦，英国，2021年10月25日至27日，论文集。《计算机科学讲义》，第13035卷，第115–131页。Springer（2021）。 https://doi.org/10.1007/978-3-030-89716-1_8, https://doi.org/10.1007/978-3-030-89716-1_845. Kosche, M., Koß, T., Manea, F., Siemer, S.：子序列匹配的组合算法：综述。载于：Bordihn, H., Horváth, G., Vaszil, G.（编）：第12届非经典自动机模型与应用国际研讨会论文集，NCMA 2022，德布勒森，匈牙利，2022年8月26日至27日。EPTCS，第367卷，第11–27页（2022）。 https://doi.org/10.4204/EPTCS.367.2，https://doi.org/10.4204/EPTCS.367.2 46. Kuske, D.：子迹序与一阶逻辑计数。载于：Proc. CSR 2020。《计算机科学讲义》，第12159卷，第289–302页（2020） 47. Kuske, D., Zetzsche, G.：按子词序排列的语言。载于：Proc. FOS-SACS 2019。《计算机科学讲义》，第11425卷，第348–364页（2019） 48. Lau, J., Ritossa, A.：多维范围更新与查询的算法及复杂性。载于：Lee, J.R.（编）：第12届理论计算机科学创新会议，ITCS 2021，2021年1月6日至8日，线上会议。LIPIcs，第185卷，第35:1–35:20页。Schloss Dagstuhl—莱布尼茨信息学中心（2021）。 https://doi.org/10.4230/LIPIcs.ITCS.2021.35, https://doi.org/10.4230/LIPIcs.ITCS.2021.35 49. Lejeune, M., Leroy, J., Rigo, M.：计算Thue-Morse词的 k-二项式复杂度。载于：Proc. DLT 2019。《计算机科学讲义》，第11647卷，第278–291页（2019） 50. Leroy, J., Rigo, M., Stipulanti, M.：词的二项式系数的广义帕斯卡三角形。Electron. J. Combin. **24**(1.44)，36页（2017） 51. Li, C., Wang, J.：利用间隔约束高效挖掘闭子序列。载于：SDM。第313–322页。SIAM（2008） 52. Li, C., Yang, Q., Wang, J., Li, M.：间隔约束子序列的高效挖掘及其多种应用。ACM Trans. Knowl. Discov. Data **6**(1)，2:1–2:39（2012） 53. Maier, D.：关于子序列和超序列的一些问题的复杂度。J. ACM **25**(2)，322–336（1978年4月） 54. Masek, W.J., Paterson, M.：一种更快的字符串编辑距离计算算法。J. Comput. Syst. Sci. **20**(1)，18–31（1980）。 https://doi.org/10.1016/0022-0000(80)90002-1, https://doi.org/10.1016/0022-0000(80)90002-1 55. Mateescu, A., Salomaa, A., Yu, S.：子词历史与Parikh矩阵。J. Comput. Syst. Sci. **68**(1)，1–21（2004） 56. Nakatsu, N., Kambayashi, Y., Yajima, S.：一种适用于相似文本字符串的最长公共子序列算法。Acta Informatica **18**，171–179（1982）。https://doi.org/10.1007/BF00264437, https://doi.org/10.1007/BF00264437

57. Riddle, W.E.：《软件系统建模与分析的一种方法》。计算机语言 **4**(1)，49–66（1979）。https://doi.org/10.1016/0096-0551(79)90009-2，https://doi.org/10 2 58 1016 00960551 79 90009 ./-() -。Rigo,M., Salimov, P.：阿贝尔等价的另一种推广：无限词的二项式复杂度。理论计算机科学**601**，47–57（2015）59.Salomaa, A.：子词与某些矩阵映射之间的联系。理论计算机科学 **340**(2)，188–203（2005）60. Seki, S.：子词不等式的绝对性是不可判定的。理论计算机科学**418**，116–120（2012）61.Shaw, A.C.：带流表达式的软件描述。IEEE软件工程汇刊 **4**(3)，242–254（1978）。https://doi.org/10.1109/TSE.1978.231501，https://doi.org/10 231501 621109 1978 . / TSE. .。Simon, I.：点深度一的事件层次结构——博士学位论文。滑铁卢大学（1972）62.Simon,I.：分段可测试事件。载于：自动机理论与形式语言，第二届 GI会议。《计算机科学讲义》，第33卷，第214–222页（1975）63. Simon,I.：由其子词区分的词（扩展摘要）。载于：WORDS2003 会议论文集。TUCS 通用出版物，第27卷，第6–13页（2003）64. Tronícek,Z.：公共子序列自动机。载于：CIAA 2002 会议论文集（修订论文）。《计算机科学讲义》，第 2608卷，第270–275 页（2002）65.Zetzsche, G.：向下闭包比较的复杂度。载于：ICALP2016会议论文集。LIPIcs，第 55 卷，第 123:1–123:14 页（2016）66. Zhang, H.,Diao,Y.,Immerman, N.：复杂事件处理中高昂查询的复杂度与优化。载于：数据管理国际会议，SIGMOD 2014，美国犹他州斯诺鸟，2014 年 6 月 22–27 日。第217–228页（2014）。https://doi.org/10.1145/2588555.2593671，https://doi.org/10.1145/2588555.2593671