arXiv:2304.11932v1 [cs.FL] 24 Apr 2023

关于词和压缩词的拱分解与子词普适性

Ph. Schnoebelen 和 J. Veron

LMF、CNRS 与巴黎-萨克雷高等师范学院，法国 *

摘要。 利用拱跳跃函数以及词的拱分解的性质，我们提出了一种计算词的子词循环普适性指标的新算法。我们还引入了词的子词普遍性签名，从而可以为 SLP 压缩词的普适性指标设计简单的算法。

1 引言

给定词的一个子词可以通过在任意位置删除若干字母得到。例如，abba 是 abracadabra 的一个子词，划线标出的字母给出了见证。子词是形式语言理论和算法学中的一个基本概念，但它们不像因子那样具有良好的性质；因子是子词的一种特殊情形，其中保留的字母对应于原词内部的一个区间。[1]

词和语言可以通过它们的子词来刻画或比较。例如，我们可以用子词 $x =$ ino 区分 $u_1 =$ nationalists 和 $u_2 =$antinationa- lists。事实上，只有 u_2 包含 x 作为子词。我们称 x 为 u_1 和 u_2之间的区分词（也称为分隔词）。注意，ino 是这两个词之间最短的区分词。[2] 在应用中，人们可能需要区分两个相似的 DNA 字符串，或某段程序执行的两条轨迹：在这些输入可能非常庞大的情形下，找到一个较短的区分子词需要高效的算法 [Sim03]。考虑词的通常一阶逻辑（即带标签的线性序）时，区分词 x可以看作一个用于区分这两个词的 Σ_1 公式。

通过子词定义。 这些思考促使 Imre Simon 在其 1972 年的博士论文 [Sim72,Sim75] 中引入了分段可测试语言：这些语言完全可以通过禁用和必需的子词来定义。

*本研究部分由 Labex DigiCosme（项目 ANR-11-LABEX-0045- DIGICOSME）资助，该项目由 ANR 负责运营，属于巴黎-萨克雷 Idex 的“未来投资”计划（ANR-11-IDEX-0003-02）。[1] 有些论文将因子称为“子词”，并将子词称为“散布子词”或“散布因子”。我们沿用 [SS83] 的术语。[2] 这在英语词汇中极为罕见，因为不同的词几乎总能通过长度为 2 的区分词来区分。首先，除非两个词所含的字母集合完全相同，否则它们已经可以通过长度为 1 的区分词来区分。

用逻辑术语来说，这对应于 $\mathcal{B}\Sigma_1$-可定义性，参见 [DGK08]。分段可测试性是一个重要且基础的概念，并且已扩展到树 [BSS12,GS16], 图像语言 [Mat98], , 以及任意稀疏线性序上的单词 [CP18]。

从描述复杂性的角度来看，一个相关的度量是定义分段可测试语言或区分两个单独的词时所用子词的长度。等价地，这些子词所需的长度就是 $\mathcal{B}\Sigma_1$ 公式所需的变量数。 [KS19] 研究了这一度量，并将其作为一种重要的新工具，用于界定可判定逻辑片段的复杂度。

子词普适性。Barker、Day 等人提出了子词普适性的概念：若一个词 u 的所有长度不超过 k 的词都是u [BFH+20,DFK+21]的子词，则称该词 u 为 k-普适的。他们进一步将 $\iota(u)$定义为使 u 成为 k-普适词的最大 k 。他们的研究动机还包括从子词重构词 [DPFD19] 以及计算编辑距离 [DFK+21]等工作；另见综述 [KKMS22]。在 [BFH+20]中，作者证明了 $\iota(u)$的若干性质，例如 u 为回文时的情形，并进一步引入循环子词普适性指数 $\zeta(u)$，其定义为$\iota(u')$的最大值，其中 u' 是 u的一个共轭词。或者， $\zeta(u)$也可视为循环词（亦称项链或循环词）的子词普适性指数 $\iota([u]_\sim)$，即按共轭关系取等价类后得到的词的等价类。

虽然计算 $\iota(u)$ 很容易，但计算 $\zeta(u)$ 则较为棘手，不过 [BFH+20]证明了若干将 $\zeta(u)$ 与 $n \in \mathbb{N}$ 的 $\iota(u^n)$ 值联系起来的界。这一点被 [FGN21] 所利用，其中给出了一种计算 $\zeta(u)$ 的 $O(|u| \cdot |A|)$ 算法。该算法相当间接，其正确性证明精细且非平凡。其他相关工作包括 [KKMS21] ，其中在已知 $\iota(u) = k$ 的情况下，研究所有长度为 $k+1$ 且不作为 u 的子词出现的单词； [FHH+22]，其中研究与 k-普适性仅相差少数几个子词的单词；以及 [KKMP22] ，其中证明了判断 u 是否具有给定长度的 k-普适因子是 NP完全的。

我们的贡献。 本文引入了研究子词（圆）普适性的新工具。首先，我们聚焦于拱分解（由 Hébrard [Héb91]引入），并展示拱跳跃函数如何为关于子词普适性指标的组合结果提供简洁证明，从而得到一种用于计算 $\zeta(u)$ 的新颖而优雅的算法。这些拱跳跃函数隐含于一些已发表的构造和证明中（例如 [FK18,FGN21,KKMS21]），但对其进行明确研究能够带来简化并提高表述的清晰度。

第二部分中，我们给出用于计算压缩词的普适性指标 ι 和 ζ 的双线性时间算法。这是通过引入一种紧凑的子词普遍性签名 实现的，该签名可以组合地计算。这些算法及其背后的思想在我们先前提到的情形中可能很有用，因为长 DNA 字符串或程序执行轨迹通常具有很强的重复性，因此以压缩形式处理它们可以大幅节省内存和通信时间。

更一般地说，这是关于计算子词并对其进行推理的算法与逻辑研究计划的一部分 [KS15,HSZ17,KS19,GLHK+20]。在这一领域，以压缩形式处理词会带来额外的困难。例如，目前尚不清楚能否高效计算两个压缩词之间最短区分词的长度。这里我们回顾一下，关于子词的推理通常比关于因子的推理更难；对于压缩词，情况确实如此：判断一个压缩 X 是否为另一个压缩 Y 的因子 属于多项式时间问题，而判断 X 是否为 Y 的子词则难以处理（属于 PSPACE 且为 PP-困难，见 [Loh12，第 8]节）。然而，在 X 或 Y 其中一个为幂词的特殊情况下，即其连接和幂运算嵌套受到限制的压缩词，子词关系可在多项式时间内判定。这一结果对 [Sch21] 中的算法至关重要；这些算法以压缩形式处理指数级长的程序执行。

论文概述。第 2 节回顾子词和普适性指标所需的所有定义。第 3 节介绍弓形跳跃函数，阐述它们与普适性指标之间的关系，并证明一些基本的组合结果。随后，第 4 节给出一个计算圆周普适性指标的简单算法。第 5 节介绍词的子词普遍性签名，并展示如何组合地计算这些签名。最后，第 6 节讨论 SLP 压缩词及其子词普适性指标。

2 基本概念

词与子词。设 $A = \{\mathsf{a}, \mathsf{b} \ldots\}$ 是一个有限字母表。我们用 $u, v, w, s, t, x, y \ldots$ 表示 A^* 中的词。连接用乘法方式表示，而 ε 表示空词。当 $u = u_1u_2u_3$ 时，我们称 u_1 是 u 的前缀，u_2 是因子，而 u_3 是后缀。当 $u = vw$ 时，我们可以用 $v^{-1}u$ 表示 w，即从 u 中移除其 v 前缀后得到的后缀。当 $u = v_0w_1v_1w_2 \cdots w_nv_n$ 时，连接 $w_1w_2 \cdots w_n$ 是 u 的子词，也就是说，它是从 u 中移除若干字母（可能一个也不移除，也可能全部移除）后得到的子序列。当 u 是 v 的子词时，我们记作 $u \preccurlyeq v$。

一个词 $u = a_1 \cdots a_\ell$ 的长度为 ℓ，记作 $|u| = \ell$，并令 $A(u) \mathrel{\mathsf{def}} = \{a_1, \ldots, a_\ell\}$ 表示其字母表，它是 A 的子集。令 $\mathsf{Cuts}(u) = \{0, 1, \ldots, \ell\} \subseteq \mathbb{N}$ 表示 u 内部的切分位置，即 u 的字母之间、可以将 u 切分的位置：对于 $0 \le i \le j \le \ell$，令 $u(i, j)$ 表示因子 $a_{i+1}a_{i+2} \cdots a_j$。采用此记号，$u(0, j)$ 是 u 的长度为 j 的前缀，而 $u(i, \ell)$ 是后缀（$u(0, i)$）^{-1}u。另请注意，只要这些因子有定义，$u(i, i) = \varepsilon$ 以及 $u(i, j) = u(i, k)\,u(k, j)$。如果 $u = u_1u_2$，我们称 u_2u_1 是 u 的共轭词。对于 $i \in \mathsf{Cuts}(u)$，u 的第 i 个共轭词为 $u(i, \ell)\,u(0, i)$，记作 $u^{\sim i}$。最后，$u^{\mathrm{R}} \mathrel{\mathsf{def}}= a_\ell \cdots a_1$ 表示 u 的镜像词。

富词与拱分解。如果一个词 $u \in A^*$ 至少包含每个字母 $a \in A$ 各一次，则称其为富词；否则称其为不完整。没有任何富词真前缀的富词称为一个拱。拱的镜像词

称为余拱（通常它并不是拱）。注意，拱（或余拱）的末尾（或开头）必然是一个在其中只出现一次的字母。

u的拱分解 由 Hebrard [Héb91], 引入，是将 u 分解为 $m+1$ 因子的分解，定义如下：— 若 u 不是富词，则 $m=0$ 且 $r=u$；— 否则，令 s_1 为 u 中最短的富词前缀（它是一个拱），并令s_2，…，s_m,r 为后缀$(s_1)^{-1}u$的拱分解。

我们用 $r(u)$表示 u的分解中的最后一个因子，称为 u的余项 。例如，当 $A=\{$为 a, b, c$\}$ 时，$u_{\mathrm{ex}}=$ baccabbcbaabacba 的拱分解为 bac · cab · bcba · abac · ba，其中 $m=4$ 且 $r(u_{\mathrm{ex}})$ = ba。因此，拱分解是将 u 分解为拱的最左分解，并带有一个最终余项 $r(u)$。

存在一种对称的余拱分解概念：将 u 分解为$u=r'\cdot s'_1\cdots s'_m$，使得 r'不完整，且每个 s'_i都是余拱，也就是说，是首字母只出现一次的富词因子。

上述所有概念都假设给定了底层字母表 A，更准确地说，我们应称之为“A-富”词、“A-拱”或“余项 $r_A(u)$”。当 A 已明确时，我们仍采用较简单的术语和记法。

子词普适性。 在 [BFH$^+$20] 中，Barker 等人定义了词 u 的子词普适性指数，记作 $\iota_A(u)$，或简写为 $\iota(u)$，它是满足以下条件的最大 $m\in\mathbb{N}$： A^* 中任意长度为 m 的词都是 u 的子词。

显然，当且仅当 u 的拱分解含有 m 个拱时，$\iota(u)=m$。因此，只需从左向右扫描 u，跟踪连续拱中字母的出现情况并计数拱，就能在线性时间内计算 $\iota(u)$，参见 [BFH$^+$20，命题 10]。使用该扫描算法计算 ι，可知对所有词 u、 v，以下等式成立：

$$\iota(u\,v)=\iota(u)+\iota\big(r(u)v\big)\,,\qquad\qquad r(u\,v)=r\big(r(u)v\big)\,. \tag{1}$$

Barker 等人进一步定义了 u 的循环子词普适性指数，记作 $\zeta(u)$，即对于 u 的 u' a 个共轭词，最大的 $\iota(u')$。显然，始终有 $\zeta(u)\geq\iota(u)$。注意， $\zeta(u)$ 可以严格大于 $\iota(u)$；例如，对于 $A=\{$a,b$\}$ 和 $u=$ aabb，有 $\iota(u)=1$，且 $\zeta(u)=2$。这些描述复杂度度量在词的镜像下保持不变，即 $\iota(u^{\mathrm{R}})=\iota(u)$ 且 $\zeta(u^{\mathrm{R}})=\zeta(u)$，并且关于子词序具有单调性：

$$u\preccurlyeq v\implies\iota(u)\leq\iota(v)\wedge\zeta(u)\leq\zeta(v)\,. \tag{2}$$

ζ 的行为可能具有迷惑性。例如，虽然 ι 是超可加的，即 $\iota(uv)\geq\iota(u)+\iota(v)$——只需结合公式 (1) 和 (2)——我们观察到 $\zeta(uv)<\zeta(u)+\zeta(v)$ 可能发生，例如对于 $u=$ ab 和 $v=$ bbaa。

3 弓形跳跃函数和普适性指标

固定一个长度为 L的词 $w=a_1a_2\cdots a_L$。现在我们引入 α 和 β弓形跳跃函数，用于描述从某个位置开始读取拱的过程

在 w中的位置。对于 $i \in Cuts(w)$，定义

$$\alpha(i) = \min\{j \mid A\big(w(i,j)\big) = A\}, \qquad \beta(j) = \max\{i \mid A\big(w(i,j)\big) = A\}.$$

这些是偏函数：当 $w(i, L)$ 或 $w(0, j)$ 分别不包含 A 中的所有字母时， $\alpha(i)$ 和 $\beta(j)$ 未定义。图1对此进行了说明。

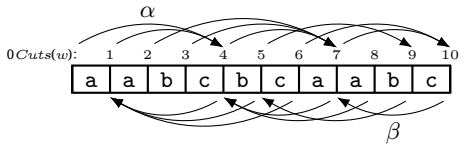


图1。 弓形跳跃函数 α、 β 对于 $A = \{\mathtt{a}、\mathtt{b}、\mathtt{c}\}$ 和 $w = \mathtt{aabcbcaabc}$。

很容易看出，以下性质对所有 i、 $j \in dom(\alpha)$ 都成立：

$$\alpha(i) \geq i + |A|\,, \qquad i \leq j \implies \alpha(i) \leq \alpha(j)\,, \tag{3}$$

$$\beta(\alpha(i)) \geq i\,, \qquad \alpha(\beta(\alpha(i))) = \alpha(i)\,. \tag{4}$$

由于 β 是 α 的镜像词版本，它具有类似的性质，本文不再赘述。

备注*3.1.* 正如本节余下部分将会看到的，弓形跳跃函数是推理拱分解时一种自然且方便的工具。文献中当然也能找到类似的概念。早在 [Héb91], Hébrard 就用 $p(n)$ 表示我们写作 $\alpha^n(0)$ 的内容，也就是 n对 0 进行 $\alpha(\alpha(\cdots(\alpha(0))\cdots))$ 次迭代： $p(n)$ 的起点是固定的，而不是可变的。在 [FK18], Fleis- cher 和 Kufleitner 使用诸如 X_a 和 Y_b 这样的秩定位器，从词中的当前位置跳到指定字母 a 和 b 的下一个（或上一个）出现位置：如果知道后续拱的最后一个字母是什么，这就可以特化为我们的 α 和 β 。在 [KKMS21] 中，minArch 与我们的 α 完全对应，但 minArch 在那里是一种用于存储信息的数据结构，而不是用于对拱进行代数推理的记号工具。

3.1 通过跳跃函数求子词普适性指数

跳跃函数 α 与子词普适性指数 $\iota(w)$ 之间的联系显而易见：

$$\iota(w) = \max\big\{n \;\big|\; \alpha^n(0) \text{ is defined}\big\}\,. \tag{5}$$

例如，图1中的 w 具有 $\alpha^3(0) = 10 = |w|$ ，因此 $\iota(w) = 3$。

我们可以推广公式（5）： $\iota(w) = n$ 当且仅当 $\alpha^p(0) \leq \beta^{n-p}(|w|)$ 对所有$p = 0, \ldots, n$都成立，反向亦然。我们可以用它证明如下结论：

命题 3.2。 $\iota(u\,v) \leq \iota(u) + \iota(v) + 1$。

证明。 设 n 和 n' 分别表示 $\iota(u)$ 和 $\iota(v)$。因此，在满足 $L = |u| + |v|$的 $w = u\,v$ 上，有 $\alpha^{n+1}(0) > |u|$ 且 $\beta^{n'+1}(L) < |u|$。见图 2。因此 $\iota(w) < n + n' + 2$。

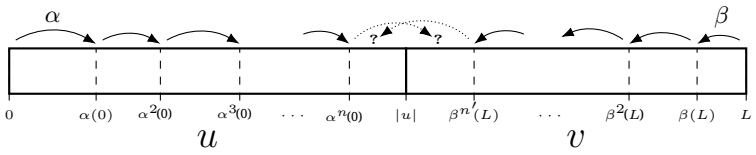


图 2。 比较 $\iota(u\,v)$ 与 $\iota(u) + \iota(v)$。

我们还可以重新证明 [BFH+20] 中的一个结果:

命题 3.3。 $\iota(u\,u^{\mathrm{R}}) = 2\iota(u)$。

证明。 记 n 为 $\iota(u)$。当 $w = u\,u^{\mathrm{R}}$ 和 $L = |w|$时，因子 $w\big(\alpha^n(0)\ \beta^n(L)\big)$ 是 $r(u)\cdot r(u)^{\mathrm{R}}$，因此不是富词。于是 $\alpha^{n+1}(0) > |u| + |r(u)^{\mathrm{R}}| = \beta^n(L)$，从而 $\iota(u\,u^{\mathrm{R}}) < 2n + 1$。

3.2 通过跳跃函数得到子词圆周普适性指标

跳跃函数可用于研究圆周普适性指标 $\zeta(u)$。为此，我们考虑通过拼接 u 的两个副本得到的词 $w = u\,u$，使得 $L = 2\ell$。现在，我们不再考虑 u 的共轭词，而是可以考虑 w 的因子 $w(i, i+\ell)$: 见图 3。

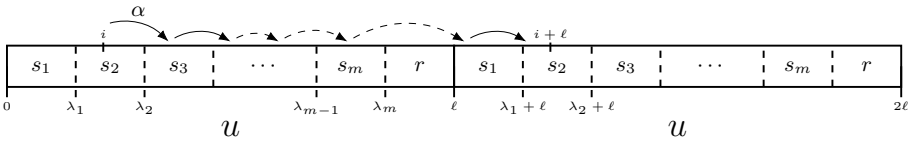


图 3. 在 $w = u^2$ 上计算 $\iota(u^{\sim i})$。

这将 $\zeta(u)$ 刻画为关于 $w = u\,u$ 上 α 的形式:

$$\zeta(u) = \max_{0\leq i<\ell} \max\big\{n \;\big|\; \alpha^n(i) \leq i + \ell\big\} \tag{6}$$

或者，使用 $u^{\sim\ell} = u^{\sim 0}$,

$$= \max_{0<i\leq\ell} \max\big\{n \;\big|\; \alpha^n(i) \leq i + \ell\big\}\,. \tag{7}$$

上界 $\zeta(u)$。对于 $k = 0, \ldots, m$，我们将 λ_k 记作 u的$\stackrel{\text{def}}{=} \alpha^k(0)$ k 第一个拱的 $|s_1 \cdots s_k|$累积长度。

下面的引理及其推论是引理 20（见[BFH+20] ）的一个版本，但我们给出了不同的证明。

引理 3.4. 设 u 和 u' 是两个共轭词。
(a) $\iota(u) - 1 \le \iota(u') \le \iota(u) + 1$. *(b)* 此外，若 $r(u) = \varepsilon$，则 $\iota(u') \le \iota(u)$。

证明。设 $s_1 \cdots s_m \cdot r$ 是 u 的拱分解，并假设 $u' = u^{\sim i}$如图 3 所示。

(a) 若位置 i 落在 u 的某个拱 s_p 内（或余项 r内），则可见 $s_{p+1} \cdots s_m \cdot s_1 \cdots s_{p-1}$是 u'的子词，因此 $\iota(u') \ge m - 1$。由此得到$\iota(u) - 1 \le \iota(u')$，另一不等式则通过交换$u$和 u'的角色得到。

(b) 此外，若 $r = \varepsilon$，则对某个 p，有 $\lambda_{p-1} \le i < \lambda_p$。将 u' 视为 $w = u^2$ 的一个因子（并假设 $\alpha^{m+1}(i)$有定义），我们可推出 $\alpha^{m+1}(i) \ge \alpha^{m+1}(\lambda_{p-1}) = \lambda_p + \ell > i + \ell$。这证明了 $\iota(u^{\sim i}) < m + 1$。

推论 3.5. *(a)* $\iota(u) \le \zeta(u) \le \iota(u) + 1$。*(b)* 此外，若 $r(u) = \varepsilon$，则 $\zeta(u) = \iota(u)$。

4 一个用于计算 $\zeta(u)$ 的 $O(|u| \cdot |A|)$ 算法

下面这个关键引理表明，计算 $\zeta(u)$ 无需检查 $0 \le i < \ell$的所有共轭词 $u^{\sim i}$。

引理 4.1. 设 $u = a_1 \cdots a_\ell$ 是一个具有拱分解 $s_1 \cdots s_m \cdot r$的丰富词。*(a)* 存在某个 $0 < d \le \lambda_1 \stackrel{def}{=} |s_1|$，使得 $\zeta(u) = \iota(u^{\sim d})$。*(b)* 此外，存在 $a \in A$ 使得 $d = \min\{i \mid a_i = a\}$，也就是说，$d$ 可以选为 u中某个字母首次出现之后紧接的位置。

证明。令 $n = \zeta(u)$。对于 (a)，只需证明存在某个 $d \in (0, \lambda_1]$，使得 $\iota(u^{\sim d}) \ge n$。

根据公式 (7)，存在某个 $0 < i_0 \le \ell$，使得 $\alpha^n(i_0) \le i_0 + \ell$。考虑由 $i_{k+1} = \alpha(i_k)$ 给出的序列 $i_0 < i_1 < \cdots < i_n$。若 $i_n \le \ell$，那么取 $d = 1$ 即可：由 α 的单调性可得 $\alpha^n(d) \le \alpha^n(i_0) \le \ell$，进而推出 $\iota(u^{\sim d}) \ge n$。显然，$d = 1$ 满足 (b)。

因此，假设 $i_n > \ell$，并令 k 为满足 $i_k \le \ell$ 的最大下标（故 $k < n$）。由于 $\alpha(\ell) = \ell + \lambda_1$（回顾一下 $\lambda_1 \stackrel{定义}{=} |s_1|$），由 α 的单调性可得 $i_{k+1} = \alpha(i_k) \le \ell + \lambda_1$，即 i_{k+1} 落在 w 中 u 第二个副本的第一个拱内。

现在令 $d \stackrel{定义}{=} i_{k+1} - \ell$，使得 $u^{\sim d} = w(d, d + \ell) = w(d, i_{k+1})$。由于 $\alpha^{n-k-1}(i_{k+1}) = i_n \le i_0 + \ell$，因此有 $\alpha^{n-k-1}(d) \le i_0$，从而 $\iota\big(w(d, i_0)\big) \ge n - k - 1$。我们还有 $\iota\big(w(i_0, i_{k+1})\big) = k + 1$，因为 $i_{k+1} = \alpha^{k+1}(i_0)$。这就得到

$$\iota(u^{\sim d}) = \iota\big(w(d, d + \ell)\big) \ge (n - k - 1) + (k + 1) = n\,,$$

从而得到 (a)。对于 (b)，注意到 $w(i_{k+1}-1, i_{k+1})$ 是一个拱的最后一个字母，该拱从 w 中第一个 u 的末尾延伸到 w中第二个 u 的开头。由于它是这个拱中该字母的首次出现，因此它也在 u 中。由于 d 被 i_{k+1}移至 u的第一个副本中，(b) 得以满足。

算法 4.2（计算 $\zeta(u)$）。对于每个满足 $u(d-1,d)$ 是 u 中某个字母首次出现的 d 位置，计算 $\iota(u^{\sim d})$（对每个 d，耗时 $O(|u|)$），并返回找到的最大值。□

该算法的正确性由引理 4.1 给出（若 u 不是富词，则 $\zeta(u)=0$，这一点会在计算 $\iota(u^{\sim 1})$ 的过程中发现）。该算法的运行时间为 $O(|A|\cdot|u|)$，因为从 $d=1$ 开始，d 最多有 $|A|$ 个取值。

有两种启发式改进可以加快算法 [3]：

– 一旦遇到两个不同的值 $\iota(u^{\sim d}) \neq \iota(u^{\sim d'})$，根据推论 3.5.(a)，就可以停止寻找最大值。例如，对于 u =aabaccb，a、b 和 c 首次出现的位置分别为 $d=1$、3 和 5。因此，首先计算 $\iota(u^{\sim 1}) = \iota$(abaccb a) $=2$。接着计算 $\iota(u^{\sim 3}) = \iota$(accb aab) $=1$。此时，由于已经遇到两个不同的值，我们可以立即得出结论：$\zeta(u) = 2$，无需计算 $\iota(u^{\sim 5})$。– 在计算某个 $\iota(u^{\sim d})$ 时，如果发现 $r(u^{\sim d}) = \varepsilon$，根据推论 3.5.(b)，就可以停止搜索。例如，仍以 $u=$ aabaccb 为例，计算 $\iota(u^{\sim 1})$ 得到了拱因式分解 $u^{\sim 1} =$ abac $\cdot$ cba $\cdot\, \varepsilon$，其中有 2 个拱，且 $r(u^{\sim 1}) = \varepsilon$。我们可以立即得出结论：$\zeta(u) = \iota(u^{\sim 1}) = 2$，无需尝试其余的共轭词。

请注意，上述算法不必显式构建 $u^{\sim d}$。任何用于 $\iota(u)$ 的朴素算法都很容易调整为从某个位置 d 开始，并在到达 u 的末尾时绕回开头。

5 子词普适性签名

在本节中，我们用 $\iota_*(u)$、$r_*(u)$ 等表示 $\iota(u)$、$r(u)$ 等的取值，其中假设 $A(u)$ 是底层字母表。这种记法比方说写成 $\iota_{A(u)}(u)$ 要简洁得多，但这是必要的，因为当 $A(u) \neq A(v)$ 时，我们将同时考虑 $\iota_*(u)$ 和 $\iota_*(v)$，也就是说，两个普适性指标是在不同的上下文中得到的。

当 u 是一个词时，我们定义一个作用于词的函数 S_u：

$$S_u(x) = \big\langle \iota_*(x\,u), A\big(r_*(x\,u)\big)\big\rangle \quad \text{for all } x \text{ such that } A(u) \not\subseteq A(x). \tag{8}$$

换言之，$S_u(x)$ 是对 $x\,u$ 的拱分解的概括：它记录了 $x\,u$ 中拱的数量以及余项 $r_*(x\,u)$ 中的字母，其中字母表为 $A(x\,u)$。

[3] 它们不会改善最坏情况 复杂度。

注意，只有当 $A(u) \not\subseteq A(x)$ 时， $S_u(x)$ 才有定义，即 u 中至少有一个字母不出现在 x 中。在此限制下，只要 $A(x) = A(x')$， $S_u(x)$ 和 $S_u(x')$ 就相同（或同时无定义）。因此，我们有时写作 $S_u(B)$，其中 B 是一个字母集合，用来表示任意满足 $A(x) = B$ 的 $S_u(x)$。

现在我们几乎可以介绍主要的新对象了：一种包含足够信息、能够计算任意参数 S_u 的紧凑数据结构。

对于一个词 u，我们关联 $e(u)$，它是一个按字母在 u 中首次出现的顺序列出 u 中字母的词。例如，在 $u = \mathtt{ccacabcbba}$ 中将每个字母的首次出现加下划线，我们得到 $e(\overline{u}) = \mathtt{\underline{c}\underline{a}\underline{b}}$。我们还将 $f(u)$ 用于表示按字母在 u 中最后一次出现的顺序列出这些字母的词：在前面的例子中， $f(u) = \mathtt{cba}$。

定义 5.1。一个词 u 的子词普遍性签名是二元组 $\Sigma(u) = \langle e(u), \mathtt{s}_u \rangle$，其中 $\mathtt{s}_u$ 限制在 $e(u)$ 的真后缀上。

例 *5.2*。对于 $u = \mathtt{aabac}$，我们有：

$$\Sigma(u) = \begin{cases} e(u) = \mathtt{abc} \\ \mathtt{s}_u = \begin{cases} \varepsilon \mapsto \langle 1, \varnothing \rangle \\ \mathtt{c} \mapsto \langle 1, \{\mathtt{a}, \mathtt{c}\} \rangle \\ \mathtt{bc} \mapsto \langle 2, \varnothing \rangle \end{cases} \end{cases} \quad \text{in view of:} \quad \begin{array}{r} \varepsilon \cdot u = \mathtt{aabac} \cdot \varepsilon \\ \mathtt{c} \cdot u = \mathtt{caab} \cdot \mathtt{ac} \\ \mathtt{bc} \cdot u = \mathtt{bca} \cdot \mathtt{abac} \cdot \varepsilon \end{array}$$

注意： $e(u)$ 的真后缀为 ε、$\mathtt{c}$ 和 $\mathtt{bc}$。

尽管有限的（而且相当小的） $\Sigma(u)$包含了在任意字母表上计算 S_u 的任意参数 x 所需的足够信息。可以使用以下算法：

算法 5.3（从 $\Sigma(u)$计算 $S_u(x)$）。给定输入 x and d $\Sigma(u) = \langle e(u), \mathtt{s}_u \rangle$ 我们继续 d 如下：

(a) 从 $e(u)$ 中提取 $A(u)$。检查 $A(u) \not\subseteq A(x)$，因为否则 $S_u(x)$ 无定义。

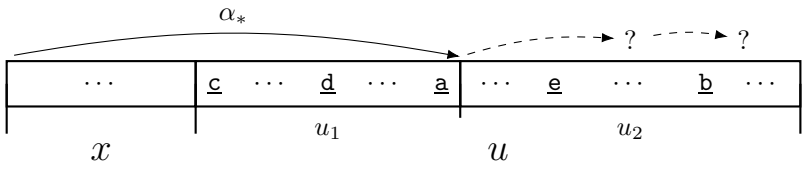
(b) 现在，令 y 为 $e(u)$ 的最长后缀，使得 $A(y) \subseteq A(x)$，其中 y 必然是 $e(u)$ 的严格后缀；然后从 $\mathtt{s}_u(y)$ 中提取 $\langle n_y, B_y \rangle$。

(c.1) 如果 $A(x) \subseteq A(u)$，则返回 $S_u(x) = \langle n_y, B_y \rangle$。

(c.2) 类似地，如果 $n_y = 1$，则返回 $S_u(x) = \langle n_y, B_y \rangle$。

(c.3) 否则返回 $S_u(x) = \langle 1, A(u) \rangle$。

证明（正确性）。假设 $x \in dom(S_u)$。由于 u 包含一个未出现在 x中的字母，$x\,u$ 的第一个拱在 u内部结束，因此我们考虑如下因式分解 $u = u_1 u_2$，使得 $x\,u_1$ 是 $x\,u$ 的第一个拱（见下图，其中 $e(u)$带有下划线）。



现在，u_1 有一个最后字母，记作 a，它在 u_1 中只出现一次，在 x中根本不出现。注意，字母 b 在 $e(u)$ 中的 a 之后出现，当且仅当它不出现在 u_1中，因此必定出现在 x中。因此，步骤 (b) 中计算出的 y 就是 $e(u)$ 在 a 之后的后缀（在上图中，y 应为 eb）。

如果 $A(x) \subseteq A(u)$，那么 $y\,u_1$ 是富词，事实上还是一个拱，因为它的最后一个字母 a 只出现了一次。因此，$S_u(x)$ 和 $S_u(y)$ 重合，步骤 (c.1) 是正确的。

如果 $A(x) \not\subseteq A(u)$，x 和 u 都包含一些另一词中没有的字母，因此必然有 $\iota_*(x\,u) = 1$ 和 $r_*(x\,u) = u_2$。接下来只需根据 $\Sigma(u)$ 计算 $A(u_2)$。我们知道 $\mathsf{s}_u(y) = \langle n_y, B_y\rangle$。如果 $n_y > 1$，这意味着 u_2 至少还包含另一个 $A(u)$-拱，因此 $A(u_2) = A(u)$，步骤 (c.3) 是正确的。如果 $n_y = 1$，这意味着 $y\,u$ 只有一个拱，即 $y\,u_1$，而 B_y 提供了 $A(u_2)$：在这种情况下，步骤 (c.2) 是正确的。

注 *5.4*（算法 *5.3* 的空间和时间复杂度）。为简化复杂度评估，我们假设字母表的大小存在一个固定上限，因此存储一个字母 $a \in A$ 需要 $O(1)$ 的空间，例如 64 位。存储 $\Sigma(u)$时，$e(u)$部分需要 $O(|A|)$ 的空间。现在，当已知 $e(u)$ 和 $f(u)$ 时，s_u 可以用 $O(|A| \log |u|)$ 的空间表示：它至多包含 $|A|$ 对$\langle n_x$、$B_x\rangle$，其中 x 是 $e(u)$ 的一个后缀，而 B_x 始终是 $f(u)$ 的某个严格后缀的字母表：因此，x 和 B_x 可以分别用$e(u)$和 $f(u)$中的一个位置（或一个字母）表示。每个 n_x 值至多需要 $\log |u|$ 位。

在时间方面，该算法的运行时间为 $O\big(|x| + |\Sigma(u)| + |A(u)|\big)$。 □

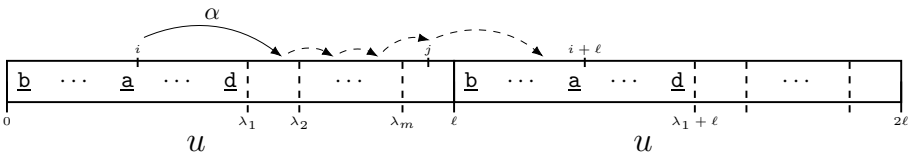
5.1 基于签名的普适性指标

显然，签名 $\Sigma(u)$包含足够的信息来恢复 $\iota_*(u)$：这可以在 $\mathsf{s}_u(\varepsilon)$ 中找到。更有趣的是，还可以恢复 $\zeta_*(u)$：

命题 5.5。设 u是一个词，并且 $\iota_*(u) = m$。则 $\zeta_*(u) = m+1$ 当且仅当存在 $e(u)$ 的一个严格后缀 x，满足 $\mathsf{s}_u(x) = \langle n_x,\ B_x\rangle$ 使得 $n_x = m+1$ 且$A(x) \subseteq B_x$。否则 $\zeta_*(u) = m$。

证明。（⇐）：假设 $\mathsf{s}_u(x) = \langle m+1, B_x\rangle$，且 $A(x) \subseteq B_x$。因此，$\iota_*(x\,u) = m+1$。将 u 分解为 $u = u_1u_2r$，使得 $x\,u_1$ 是 $x\,u$ 的第一个拱，并且$r = r_*(x\,u)$是其余项。于是 u_2 包含 m 个拱，且 $B_x = A(r)$。现在令u' $\mathrm{def}=$ $r\,u_1u_2$。我们断言，$\iota_*(u') = m+1$。事实上，$r\,u_1$ 是富词，因为 $x\,u_1$ 是富词且 $A(x) \subseteq A(r)$，所以 $\iota_*(r\,u_1u_2) \geq m+1$。由于 u' 和 u 是共轭词，根据推论 3.5.(a)，我们由此得到 $\zeta_*(u) = \iota_*(u') = m+1$。

(⇒)：假设 $\zeta_*(u) = m+1$。根据引理 4.1，我们知道，对于某个位置 $0 < i \leq \lambda_1$，$\iota_*\big(u^{\sim i}\big) = m+1$ 恰好位于 u 中某个字母的首次出现之后。像之前一样考察 $w = u\,u$ 的因子，我们有 $\alpha^{m+1}(i) \leq i+\ell$，从而得到 $j \overset{\mathrm{def}}{=} \alpha^m(i) \leq \ell$（见下图）。



现在将 x 定义为 $e(u)$ 的后缀，其中包含 $u(i,\lambda_1)$ 中的所有字母，也就是说，包含 i 右侧所有带下划线的字母。由于 $i>0$，这是一个严格后缀。现在，$x\,u(0,i)$ 是富词，而 $u(i,j)$ 恰好由 m 个拱组成，因此 $\iota_*(x\,u)=n_x=m+1$，并且 $r_*(x\,u)=u(j,\ell)$。

于是 $B_x=A\big(u(j,\ell)\big)$ 和 $w(j,i+\ell)$ 是富词，因此 $w(j,\ell)$ 包含 $w(i,i+\ell)=u(0,i)$ 中缺失的所有字母。换句话说，$B_x\supseteq A(x)$，证明完毕。

推论 5.6（从签名计算普适性指标）。可以在 $\iota_*(u)$ 和 $\zeta_*(u)$ 中根据 $\Sigma(u)$，在时间 $(|A|+\log|u|)^{O(1)}$ 内计算。

实际实现可以使用基于引理 3.4.(b) 的启发式方法：如果 $\mathsf{s}_u(\varepsilon)=\langle m,\varnothing\rangle$ 那么 $\zeta_*(u)=m$。

5.2 组合签名

子词普适性签名可以组合地计算。

算法 5.7（组合签名）。以下算法以任意两个词的签名 $\Sigma(u)$ 和 $\Sigma(v)$ 为输入，并计算 $\Sigma(u\,v)$：

(a) 从 $e(u)$ 和 $e(v)$ 中取出 $A(u)$ 和 $A(v)$，然后将 $e(u\,v)$ 计算为 $e(u)\,e'$其中，e' 是 $e(v)$ 的子词，仅保留来自 $A(v)\setminus A(u)$ 的字母。(b) 现在考虑 $e(u\,v)$ 的任意严格后缀 x，并按如下方式计算 $\mathsf{s}_{u\,v}(x)$：(b.1) 如果 $A(v)\not\subseteq A(x)\cup A(u)$，则令 $\mathsf{s}_{u\,v}(x)\ \mathsf{def}=S_v\big(x\,e(u)\big)$，使用算法 5.3。(b.2) 如果 $A(v)\subseteq A(x)\cup A(u)$，则 $A(u)\not\subseteq A(x)$。令 $\langle n,B\rangle$ 为 $\mathsf{s}_u(x)$：(b.2.1) 若此时 $A(v)\cup B\neq A(x)\cup A(u)$，则令 $\mathsf{s}_{u\,v}(x)\ \mathsf{def}=\langle n,A(v)\cup B\rangle$。(b.2.2) 否则，取出 $\mathsf{s}_v(B)=\langle n',B'\rangle$，并令 $\mathsf{s}_{u\,v}(x)\ \mathsf{def}=\langle n+n',B'\rangle$。

证明（正确性）。步骤 (a) 对 $e(u\,v)$ 是正确的。
在步骤 (b) 中，我们要计算 $S_{u\,v}(x)$。现在，$x\,(u\,v)=(x\,u)\,v$，因此 $S_{u\,v}(x)$ 与 $S_v(x\,u)$ 在后者有定义时相同。步骤 (b.1) 就是这种情况：此时通过将 $x\,u$ 替换为 $x\,e(u)$ 来计算 $S_v(x\,u)$，其中替换用的参数具有相同的字母表（回想一下，该算法无法访问 u 本身）。

在步骤 (b.2) 中，若 $S_v(x\,u)$ 未定义，计算 $S_u(x)$ 会为 $x\,u$ 的拱分解 $x\,u=s_1\cdots s_n\cdot r$ 提供 n 和 $B=A(r)$。
如果这些分解所依据的字母表相同，我们就可以继续对 $r\,v$ 进行拱分解，并合并两组拱：这就是步骤 (b.2.2)。
否则，$r\,v$ 只使用了 $x\,u$ 中字母的一个子集。不会出现新的拱，只会得到更长的余项：$r_*(x\,u\,v)=r\,v$。步骤 (b.2.1) 是正确的。
注意，算法 5.7 的运行时间为 $O\big(|A(u\,v)|+|\Sigma(u)|+|\Sigma(v)|\big)$，且结果的大小为线性大小 $|\Sigma(u\,v)|=O(|\Sigma(u)|+|\Sigma(v)|)$。

6 SLP 压缩词的普适性指标

现在我们已经可以计算 SLP 压缩词的普适性指标了。回顾一下，SLP X 是乔姆斯基层范式中的无环上下文无关文法

，此外每个非终结符只有一条产生式规则，即该文法是确定性的（参见综述 [Loh12]）。SLP 是压缩文本和文件的标准数学模型，并且除多项式时间编码外，它涵盖了实践中使用的大多数压缩方案。

形式上，具有 m 条规则的 SLP X 是一个产生式规则列表 $\langle N_1 \to \rho_1;\ \cdots;\ N_m \to \rho_m\rangle$，其中每个右侧 ρ_i 要么是来自 A 的字母 a，要么是由两个非终结符连接 $N_j\,N_{j'}$ 而成，且 j、$j' < i$。当 A 固定时，其大小为 $|X| = O(m \log m)$。

每个非终结符 N_i 都编码一个词，即其展开，通过以下方式归纳定义：

$$exp(N_i) \stackrel{\text{def}}{=} \begin{cases} a & \text{if } \rho_i = a, \\ exp(N_j)\,exp(N_{j'}) & \text{if } \rho_i = N_j\,N_{j'}. \end{cases}$$

最后，SLP 本身的展开 $exp(X)$ 是其最后一个非终结符的展开 $exp(N_m)$。这是一个长度为 $2^{O(|X|)}$ 的词（或文件）。压缩数据科学领域的主要目标之一，是开发高效方法，直接从 X 计算有关 $exp(X)$ 的相关信息，也就是说，无需实际解压该词或文件。

本着这一思路，我们可以表述如下：

定理 6.1。普适性指标 $\iota\big(exp(X)\big)$ 和 $\zeta\big(exp(X)\big)$可从 *SLP* X 在线性双线性时间内 $O\big(|A| \cdot |X|\big)$计算得出。

证明。只需为 X 的各个非终结符N_1、...、N_k计算 $\Sigma\big(\exp(N_1)\big)$、...、$\Sigma\big(\exp(N_k)\big)$。如果 N_i 对应一个产生式规则 $N_i \to N_{i_1}N_{i_2}$，我们就通过算法 5.7 将 $\Sigma\big(\exp(N_{i_1})\big)$ 和 $\Sigma\big(\exp(N_{i_2})\big)$ 合并，从而计算 $\Sigma\big(\exp(N_i)\big)$（回想一下，$i_1$、$i_2 < i$，因为文法是无环的）。如果 N_i 对应某个 $a \in A$ 的产生式 $N_i \to a$，那么 $\Sigma\big(\exp(N_i)\big) = \Sigma(a)$ 的计算是平凡的。最后，我们可以根据推论 5.6，从 $\Sigma\big(\exp(N_k)\big)$ 中提取 $\exp(X)$ 的普适性指标，其定义为 $\exp(N_k)$。注意，所有签名的大小都是 $O(|A| \cdot |X|)$，因为对于任意 $u = \exp(N_i)$，$\log|u|$ 都属于 $O(|X|)$。结合算法 5.7 和推论 5.6 的分析，这就证明了关于复杂度的论断。

7 结论

我们引入了弓形跳跃函数，并用它们来描述和分析子词普适性指标 $\iota(u)$ 和循环普适性指标 $\zeta(u)$。特别是，这由此导出了一个简单而优雅的算法，用于计算 $\zeta(u)$。

在第二部分中，我们定义了词的子词普适性签名。这是一种紧凑数据结构，包含足够的信息，可用于提取 $\iota(u)$ 和 $\zeta(u)$。由于可以通过组合 u 和 v 的签名，高效地计算 $u\,v$ 的签名，因此当 X 是 SLP压缩词时，我们得到了一种多项式时间算法，用于计算 $\iota(X)$ 和 $\zeta(X)$。尽管已知在推理子词时会遇到困难，这仍使我们对能够在压缩词上计算某些基于子词的描述复杂度度量抱有希望。

参考文献

BFH+20. L. Barker、P. Fleischmann、K. Harwardt、F. Manea 和 D. Nowotka。词的散布因子普适性。载于第*24*届语言理论发展国际会议论文集（*DLT 2020*），LNCS 第12086卷，第14–28页。Springer，2020年。BSS12. M. Bojańczyk、L. Segoufin 和 H. Straubing。分片可测试树语言。计算机科学逻辑方法，8(3)，2012年。CP18. O. Carton 和 M. Pouzet。散布词的Simon定理。载于第*22*届语言理论发展国际会议论文集（*DLT 2018*），LNCS 第11088卷，第182–193页。Springer，2018年。DFK+21. J. D. Day、P. Fleischmann、M. Kosche、T. Koß、F. Manea 和 S. Siemer。到 k-子序列普适性的编辑距离。载于第*38*届计算机科学理论方面国际研讨会论文集（*STACS 2021*），LIPiCS 第187卷，第25:1–25:19页。莱布尼茨信息学中心，2021年。DGK08. V. Diekert、P. Gastin 和 M. Kufleitner。有限词上一阶逻辑小片段综述。计算机科学基础国际期刊，19(3):513–548，2008年。DPFD19. J. D. Day、P. Fleischmann、F. Manea 和 D. Nowotka。弱 c-平衡词的 k-谱。载于第*23*届语言理论发展国际会议论文集（*DLT 2019*），LNCS 第11647卷，第265–277页。Springer，2019年。FGN21. P. Fleischmann、S. B. Germann 和 D. Nowotka。散布因子普适性——余项的力量。arXiv:2104.09063 [cs.CL]，2021年4月。FHH+22. P. Fleischmann、L. Haschke、A. Huch、A. Mayrock 和 D. Nowotka。近乎 k-普适的词——研究Simon同余的一部分。载于第*24*届形式系统描述复杂性国际会议论文集（*DCFS 2022*），LNCS 第13439卷，第57–71页。Springer，2022年。FK18. L. Fleischer 和 M. Kufleitner。Simon同余测试。载于第*43*届数学基础与计算机科学国际研讨会论文集（*MFCS 2018*），LIPiCS 第117卷，第62:1–62:13页。莱布尼茨信息学中心，2018年。GLHK+20. J. Goubault-Larrecq、S. Halfon、P. Karandikar、K. Narayan Kumar 和 Ph. Schnoebelen。良拟序闭子集计算的理想方法。载于计算、逻辑、语言与推理中的良拟序，逻辑趋势第53卷，第3章，第55–105页。Springer，2020年。GS16. J. Goubault-Larrecq 和 S. Schmitz。判定正则树语言的分片可测试可分性。载于第*43*届自动机、语言与编程国际研讨会论文集（*ICALP 2016*），LIPiCS 第55卷，第97:1–97:15页。莱布尼茨信息学中心，2016年。Héb91. J.-J. Hébrard。一种高效区分位串的子序列算法。理论计算机科学，82(1):35–49，1991年。HSZ17. S. Halfon、Ph. Schnoebelen 和 G. Zetzsche。子词序上一阶逻辑的可判定性、复杂度与表达能力。载于第*32*届*ACM/IEEE*逻辑与计算机科学研讨会论文集（*LICS 2017*），第1–12页。IEEE计算机学会出版社，2017年。KKMP22. M. Kosche、T. Koß、F. Manea 和 V. Pak。有界范围内的子序列：匹配与分析问题。载于第*16*届可达性问题国际会议论文集（*RP 2022*），LNCS 第13608卷，第140–159页。Springer，2022年。

KKMS21. M. Kosche、T. Koß、F. Manea 和 S. Siemer。词中的缺失子序列。载于第 *15* 届可达性问题国际会议论文集（*RP 2021*），*LNCS*第 13035 卷，第 115–131 页。Springer，2021。KKMS22. M. Kosche、T. Koß、F. Manea 和 S. Siemer。子序列匹配的组合算法：综述。载于第 *12* 届自动机与应用非经典模型国际研讨会论文集（*NCMA 2022*），*EPTCS*第 367 卷，第 11–27 页，2022。KS15. P. Karandikar 和 Ph. Schnoebelen。广义 Post 嵌入问题。计算系统理论，56(4)：697–716，2015。KS19. P. Karandikar 和 Ph. Schnoebelen。分片可测试语言的高度与子词逻辑的复杂度。逻辑方法与计算机科学，15(2)，2019。Loh12. M. Lohrey。SLP 压缩字符串的算法学：综述。群、复杂性与密码学，4(2)：241–299，2012。Mat98. O. Matz。关于分片可测试、无星和可识别的图像语言。载于软件科学基础与计算结构国际会议论文集（*FOSSACS '98*），*LNCS*第 1378 卷，第 203–210 页。Springer，1998。Sch21. Ph. Schnoebelen。关于平坦有损信道机。载于第 *29* 届 *EACSL* 计算机科学逻辑会议论文集（*CSL 2021*），*LIPiCS*第 183 卷，第 37:1–37:22 页。莱布尼茨信息学中心，2021。Sim72. I. Simon。点深度一的事件层次结构。博士论文，滑铁卢大学应用分析与计算机科学系，加拿大安大略省滑铁卢，1972。Sim75. I. Simon。分片可测试事件。载于第 *2* 届 *GI* 自动机理论与形式语言会议论文集，*LNCS*第 33 卷，第 214–222 页。Springer，1975。Sim03. I. Simon。由子词区分的单词。载于第 *4* 届单词国际会议论文集（*WORDS 2003*），2003。SS83. J. Sakarovitch 和 I. Simon。子词。载于 M. Lothaire（编），单词组合学，数学及其应用百科全书第 17 卷，第 6 章，第 105–142 页。剑桥大学出版社，1983。