arXiv:2304.02657v1 [cs.DS] 5 Apr 2023

多个不确定字符串中近似弱公共区间的快速计算

丹尼尔·多尔[1] 和伯纳德·M.E.·莫雷特[2]

[1] *Institute for Medical Biometry and Bioinformatics, Heinrich Heine University, Düsseldorf, Germany*[2] *School of Computer and Communication Sciences, EPFL, CH-1015, Lausanne,Switzerland*

摘要

在正在进行的旨在定义同线区块发现与结构化原则性方法的工作中，该工作基于同源性衍生约束和公共区间的推广，我们面临一个基本的计算问题：如何在一组不确定字符串（其元素由字符子集组成的字符串）中快速确定连续区间，这些区间将共享其元素的绝大多数，但允许共享被其他子集包含的字符子集，并且允许某些元素在某些基因组中缺失。Doerr 等人描述了该问题在确定字符串特殊情况下的算法（其中每个元素是字母表中的单个字符，即“正常”字符串），但如果推广到不确定字符串，其运行时间将会爆炸。在本文中，我们描述了一种计算这些特殊公共区间的算法，其时间接近 Doerr 等人更简单算法的时间，并表明对于大型集合（数十到数百个）细菌基因组，可以在短短几个小时内计算出这些区间。

1 引言

公共存储库中的全基因组序列数量正在迅速增加，这使得人们越来越需要一种有效的方法来比较多个全基因组。此类比较不仅能增进我们对现今生物体的了解，还能为探索它们的进化历史打开一扇窗口。许多受到关注的基因组（例如脊椎动物基因组）规模过大，无法进行逐碱基对比较；而当需要考虑多个基因组时，这类比较就更加困难。因此，大规模全基因组比较研究受到两个因素的限制：(i) 基因组的大小（脊椎动物的基因组通常包含数十亿个碱基对）以及 (ii) 希望进行比较的基因组数量。在过去15年中，针对第一个问题的标准方法一直是将基因组分解为同线区块。在最常见的形式中，同线区块是从 10^4 到 10^7 碱基对不等的连续区块，它们在不同基因组之间构成明显保守的单元——这一点可由密切相关的基因组标记集合的保守性得到证明。与其比较

逐个碱基对地比较基因组，甚至逐个标记地比较基因组，人们会按同线区块逐一比较它们。换言之，拥有数十亿个碱基对的基因组可以由数千个同线区块表示。由于此类分解的大部分工作都是使用基因作为基因组标记完成的，因此可以将沿每个基因组排列的同线区块视为沿每个基因组排列的基因的一种推广：正如每个基因组中的基因与其他基因组中的基因具有同源性并形成基因家族一样，同线区块也具有这种性质。（另一个问题，即必须比较多个基因组，或许最好在系统发育背景下进行，在这种背景下，随着基因组数量增加而呈二次增长的直接比较会被摒弃，转而比较后代与共同祖先 [7], 但这并非本文的重点。）

在持续开展的工作中，为公共区间的发现与结构化定义一种有原则的方法，我们基于同源性衍生约束 [4] 以及对公共区间 [5], 的推广，面临一个根本性的计算问题：如何在一组不确定字符串（其元素由字符子集构成的字符串）中快速确定这样的连续区间——这些区间的大多数元素应彼此共有，但允许某些字符子集被其他字符子集所包含，同时还允许某些基因组缺失某些元素。Doerr *et al.* [3], 描述了一种针对确定字符串这一特殊情形（其中每个元素都是字母表中的单个字符，即“普通”字符串）的该问题算法；然而，如果以“显而易见”的方式将其推广到不确定字符串，其运行时间将呈爆炸式增长。在本文中，我们描述了一种计算这些特殊公共区间的算法，其运行时间接近于 Doerr *et al.* [3] 提出的更简单算法，从而能够在完整的脊椎动物基因组尺度上计算这些区间。

2 背景

我们先简要概述在多个基因组中发现同线区块的方法，以便恰当地理解本文所关注的计算问题。

识别多个基因组间的大型保守区域，即同线区块，本身就很有用，因为这使我们能够界定感兴趣的区域，并将研究重点放在这些区域上。不过，同线区块也可用于以更高层次的抽象来描述整个基因组，从而能够在这一新层次上开展进一步分析。事实上，分解（或抽象）的层次数量不必局限于两三层：在一个极端情况下，基因组序列中的每个位置都可视为一个同线区块；而在另一个极端情况下，每个完整基因组都可视为一个单独的同线区块。这两个极端都不太有意义，而且对短子序列的研究已相当成熟，因此我们的设计从基因组标记这一层次开始，也就是说，从较短的序列开始：它们既要短到足以在其他基因组中以完全相同或非常相似的形式找到，又要长到足以使其保守性在统计上显著。这类标记可通过基因组分割 [6] 获得；更常见的做法是利用现有的基因组序列注释，这些注释标出了非编码和蛋白质编码基因等保守区域。

我们的方法依赖于这些基因组标记之间预先定义的同源关系。（请注意，同源关系是一种等价关系，表示共同的 an-

cestry。）基于基因组标记之间的同源关系，我们使用 Ghiurcuta 和Moret [4]首次提出的形式化标准，推导出同线区块之间的同源关系。非正式地说，根据他们的标准，如果两个同线区块中的每个组成基因组标记都与对方区块中的某个基因组标记同源，则这两个同线区块是同源的。

从概念的角度来看，共享起源是基因组标记的一个共同特征；事实上，一个完美的同源性预言机不会提供非常有用的信息——它会缺乏可区分性，因此对同线性分析无效。因此，在某种程度上幸运的是，序列通过突变变化的分化使得在一定时间后同源性推断变得不可能。不幸的是，这种同样的分化使得同源性推断过程容易出错，并且不太可能产生等价关系。大多数当前的同源性推断算法依赖于某种类型的聚类，通常几乎完全基于序列相似性，但同源性还应考虑每个基因组内的相对位置。换句话说，标记之间的同源性分配是同线性分析所必需的，然而它们在同线区块中的嵌入提供了真正同源性的有力指标。我们通过不要求推断的同源性描述等价关系来打破这种循环：我们放弃传递性的要求。这种放松在实践中带来了决定性的优势：基因组标记不再需要聚类成家族，因此同线性分析可以直接在成对序列相似性上执行。

我们通过在不确定字符串中识别近似弱公共区间（AWCI）集合，来解决在弱同源性条件下识别同线区块的问题——这也是本文所要解决的计算问题。简单来说，不确定字符串是指每个位置都关联一个（非空）字母表字符子集的序列。如果两个不确定字符串中的两个区间满足以下条件，则称它们为弱公共区间：两个区间中的每个位置都至少包含一个也出现在另一个区间中的字符。如果两个不确定字符串的字符编码表示两个基因组序列中标记之间的（非传递）同源关系，那么弱公共区间就等同于同源同线区块。借助 AWCI，我们可以识别轻微违反共线性标准的同线区块，允许存在数量有限的同源关系缺失声明。这样做是为了应对：（i）推断同源关系时出现的假阴性错误；（ii）基因组演化导致的一个或少数几个基因组标记的插入或缺失。

3 第一个算法

3.1 基本定义

我们回顾来自Doerr*et al.* [3]的不确定字符串中公共区间的一些基本定义。一个不确定字符串 S 具有 n 个索引位置是字母表 Σ的幂集 $\mathcal{P}(\Sigma) \setminus \emptyset$ 上的字符串。即，对于每个 i, $1 \leq i \leq n$，我们有 $S[i] \subseteq \Sigma$ 和 $S[i] \neq \emptyset$，其中 $S[i]$ 表示与 S中第 i个位置相关联的字符集。我们用 $|S| \equiv n$ 表示不确定字符串 S 具有n 个索引位置的长度，用 $\|S\| \equiv \sum_{i=1}^{n} |S[i]|$表示其基数，即 S中所有 元素的数量。两个位置 i 和 j, $1 \leq i \leq j \leq |S|$诱导（不确定）子字符串 $S[i,j] \equiv S[i]\,S[i+1]\,\ldots\,S[j]$。为了区分区间

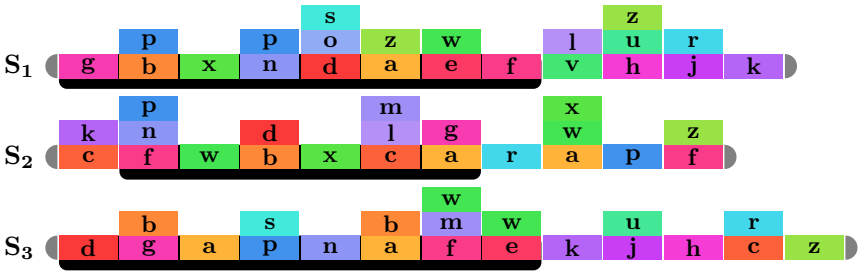


图1：三个不确定字符串 S_1、S_2、S_3的示例。黑色下划线标出的区间构成一个1-近似弱公共区间的闭集。

在不同的不确定字符串中，我们用下标记号 $[i,j]_S$ 表示区间 $[i,j]$ 对不确定字符串 S 的隶属关系。

公共区间最初定义在排列 [5] 上，随后扩展到普通字符串，然后是不确定字符串 [2, 3]。公共区间背后的思想是基于它们的字符集来比较子字符串。不确定字符串 S 的字符集 定义为 $\mathcal{C}(S) \equiv \bigcup_{i=1}^{n} S[i]$。给定两个不确定字符串 S和 T，两个区间， $[i,j]$ 在 S中和 $[k,l]$ 在 T中，是弱公共区间，具有公共字符集 $C = \mathcal{C}(S[i,j]) \cap \mathcal{C}(T[k,l])$如果对于每个 i', $i \le i' \le j$，我们有 $C \cap S[i'] \neq \emptyset$ 并且对于每个 k', $k \le k' \le l$，我们有 $C \cap T[k'] \neq \emptyset$。我们研究弱公共区间的一种变体，该变体容忍有限数量的插入和删除。给定某个阈值$\delta \ge 0$，两个区间，$[i,j]$ 在不确定字符串 S中和 $[k,l]$ 在不确定字符串 T中，是(δ-)近似弱公共区间，具有公共字符集 $C = \mathcal{C}(S[i,j]) \cap \mathcal{C}(T[k,l])$ 如果与 C 没有交集的位置 数量受 δ限制，即，如果我们能写出

$$|\{x \mid i \le x \le j \colon S[x] \cap C = \emptyset\}| + |\{y \mid k \le y \le l \colon T[y] \cap C = \emptyset\}| \le \delta$$

我们将这些位置称为插入缺失。一个区间集合是近似弱公共区间，当且仅当其中每一对成员都满足该性质。

最后，我们将 [3] 中成对互相闭合区间的性质推广到由两个或更多区间组成的集合:

定义 1（区间的闭集）一组 $\mathcal{I}$ 区间是闭 的，当每个区间 $[i,j]_S \in \mathcal{I}$ 均没有带有 $S[p] \cap \mathcal{C}$（$S'[k,l]$） 的紧邻左侧或右侧位置 p，对于每个 $[k,l]_{S'} \in \mathcal{S} \setminus \{[i,j]_S\}$。

请注意，闭性是区间集合的一种非遗传性质：闭区间集合的子集本身不一定是闭的。图 1 展示了一个闭的 1-近似弱公共区间集合的例子。

4 识别 AWCI 的极大闭集

如果一个集合无法通过纳入更多有效成员来扩展，则称其为极大的。

问题 1（极大闭集）给定 m 个不确定字符串 $S_1, \ldots, S_m$ 和插入缺失阈值 $\delta \ge 0$，找出 δ-近似弱闭集的所有极大闭集。

在 m 个不确定字符串中的至少 q 个字符串中具有成员的公共区间

由于区间集合的闭合性无法延伸至子集，因此枚举过程中识别出的每个 AWCI 集合都需要单独检验是否具有这一性质。出于同样的原因， [3] 中介绍的发现算法所返回的 AWCI 闭合对不能用于构建多个序列中的更大 AWCI 集合，因为被省略的非闭合对可能有助于构成更大的闭合 AWCI 集合。

由于在给定的不确定字符串集合中可能存在指数数量的 AWCI集合，我们不显式枚举这些集合，而是构造一种紧凑表示，从中可以提取出这些集合。给定一个不确定字符串集合 $\{S_1, \ldots, S_m\}$，我们构造一个图 $G(S_1, \ldots, S_m)$，其中每个顶点对应于某个不确定字符串中的一个区间，并且当且仅当它们所对应的区间构成一对 AWCI 时，任意两个顶点之间由一条无向边连接。识别 AWCI 的极大闭集合随后可归结为：在该图中寻找由顶点构成的极大团，且这些顶点所对应的 AWCI是闭的。在构造该图时，我们已经省略了那些无法参与大小至少为 q 的 AWCI集合的区间。我们检验每个顶点是否可能参与某个 AWCI闭集合。也就是说，如果顶点 v 的邻居都与某个顶点 u 相连，使得与顶点 v 关联的区间I_v 是对应于顶点 u的区间 I_u 的真子区间，并且区间 I_u 向 I_v 的左侧和/或右侧延伸至多一个位置，且该位置与顶点 I_v 的邻居共享字符，那么顶点 v 就可以被丢弃。所得图中加入了 AWCI 的极大闭团。

4.1 MACSI——ACSI 的扩展

现在我们介绍 MACSI，这是一种用于枚举不确定字符串中的 AWCI 的简单方法，见算法 1。MACSI 是 Doerr等人 [3]所述算法 ACSI 的简单扩展；不过，与 ACSI 不同，它会枚举任意成对的 AWCI，而不只是闭合的 AWCI。

我们不再反复确定不同不确定字符串各位置上的字符集交集，而是在开始时识别并存储字符集相交的位置。给定两个不确定字符串S_x 和 S_y，交集表 Pos_{xy} 中的每个位置 j, $1 \le j \le |S_y|$ 都关联着一个有序位置列表，其中列出 S_y 中与字符集 $S_x[j]$存在非空交集的位置。

给定一组序列 S_1、.....、 S_m，其中每个成员都依次作为参考序列（第 2–19 行），算法遍历该组序列，并据此固定一个左边界$i = 1$、.....、 $|S_x|$（第 3–20 行）。随后，算法确定右边界候选项 $J \subseteq \{i+1, \ldots, |S_x|\}$；每个候选区间 $[i, j]_{S_x}$, $j \in J$都可能属于一组 AWCI，这组 AWCI 至少在 m 个序列中的 q 个序列里得到保留。固定参考字符串 S_x 中的区间 $[i, j]$ 后，算法遍历其余 $m - x$ 个序列，枚举所有此类候选区间（第 7–14 行）。

构造表示 S_x中潜在右区间界的集合 J （第4 行）可以在线性于 $|S_x|$的时间内轻松完成。对于 S_x中的每个固定区间 $[i, j]$，可以在 $\mathcal{O}(|S_y| \cdot \|S_y\|)$ 的摊销运行时间内枚举 S_y 中所有与 $[i, j]_{S_x}$ 构成 AWCI 的区间。该算法首先收集并排序

所有位置 $P_y \in S_y$，此类区间可能位于这些位置周围（第 8 行）。对于每个这样的位置 $p \in P_y$，都会识别出左界大于之前处理位置的所有区间（第10–13 行）。通过跟踪 $[i,j]_{S_x}$ 的已观测索引，可以在 $\mathcal{O}(1)$ 中测试区间对 $[i,j]_{S_x}$，$[k,l]_{S_y}$，$k \le p \le l$ 是否具有近似公共区间属性，

$$C_{kl} = \{i' \in \mathrm{Pos}_{yx}[k] \cup \mathrm{Pos}_{yx}[l] \mid i \le i' \le j\}$$

以及 $[k,l]_{S_y}$ 中字符集与 $[i,j]_{S_x}$ 不相交的位置数 d_{kl}。那么，$[i,j]_{S_x}$，$[k,l]_{S_y}$ 是 AWCI，当且仅当 $j-i+1-|C_{kl}|+d_{kl} \le \delta$ 成立；并且所有满足 $[i,j]_{S_x}$ 的 AWCI 区间都可以在 $\mathcal{O}(|S_y| \cdot \|S_y\|)$ 时间内枚举。

算法 1的总体运行时间为 $\mathcal{O}(m^2 \cdot n^2 \cdot N^2)$，其中 $n = \max_{x=1..m} |S_x|$ 且 $N = \max_{x=1..m} \|S_x\|$。这接近最优，因为成对AWCI的数量可能达到区间对的总数，而其本身为 $\mathcal{O}(m^2 \cdot n^4)$。在此，我们作出合理假设：不确定字符串 S的基数 $\|S\|$ 为 $a \cdot |S|$ 量级，其中 $a \ll \Sigma$。

5 加速共线性分析中 AWCI 的枚举

现在我们更详细地研究集合 J（第 4 行）的构造，该集合包含 S_x 中潜在的右区间界，并介绍一种利用基因组两项特性的方案，可显著缩短运行时间。首先，存在许多仅由所有基因组的一个子集共享的字符。

算法 1 MACSI 枚举至少由 m 个不确定字符串中的 q 个共享的所有近似弱公共区间对。输入：不确定字符串 $S_1, \ldots, S_m$，插入缺失阈值 δ，法定人数 q。输出：$S_1, \ldots, S_m$ 中至多含有 δ 个插入缺失且在至少 q 个字符串中保守的所有近似弱公共区间。1: 为所有序列对 $\{x,y\} \subseteq [1,m]$ 构建字符集交集表 Pos_{xy} 2: 对 $x \leftarrow 1$ 到 $m-1$ 执行 3: 对 $i \leftarrow 1$ 到 $|S_x|$ 执行 4: 确定 AWCI 对的潜在右边界集合 J，其左边界为 i，位于 S_x 中 5: 对每个 $j \in J$ 执行 6: 初始化空列表 $\textsc{Ints}$ 7: 对 $y \leftarrow x+1$ 到 m 执行 8: 构建位置 $\mathrm{Pos}_{xy}[i] \cup \cdots \cup \mathrm{Pos}_{xy}[i+\delta]$ 的有序集合 P_y 9: $p_{prev} \leftarrow -1$ 10: 对每个 $p \in P_y$ 执行 11: 将 S_y, $p_{prev} < k \le p \le l$ 中所有满足以下条件的区间 $[k,l]$ 追加到列表 $\textsc{Ints}$：它们是近似弱公共区间，且 $[i,j]$ 位于 S_x 中 12: $p_{prev} \leftarrow p$ 13: 结束循环 14: 结束循环 15: 如果 $\textsc{Ints}$ 包含来自至少 $q-y+1$ 个序列的区间，则 16: 报告 $\{([i,j]_{S_x}, [k,l]_{S_y}) \mid [k,l]_{S_y} \in \textsc{Ints}\}$ 17: 结束条件判断 18: 结束循环 19: 结束循环 20: 结束循环

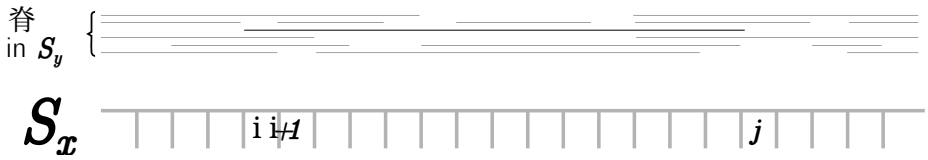


图 2：数据结构 RIDGE^t_{xy}；线条表示脊，即跨二进制向量 $\text{RIDGE}^t_{xy}[i']$ 对于某些 $i' \in [0, |S_x|]$ 的 1。

序列；其次，数据集中的每个序列都对应一个单独的基因组，该基因组由一个或多个染色体或重叠群串接而成，但 AWCI 不允许跨越多个染色体或重叠群。因此，我们的序列中存在数量不可忽略的插入缺失，可轻易识别这些插入缺失，并利用它们提前终止或结束对 AWCI 集合的无效搜索。

回顾一下，我们关注的是识别这样的 AWCI 集合：其成员存在于至少 $q \le m$ 个字符串中。AWCI 集合确保传递性，但 AWCI性质本身并不具备传递性。因此，即使我们识别出一组区间 $I_{x+1..m}$，其成员存在于至少 $q-1$ 个字符串中，并且这些成员与给定区间 $[i,j]_{S_x}$分别构成 AWCI 对，我们也无法保证区间 $I_{x+1..m}$彼此之间构成AWCI 对。因此，对于每个区间 $[i,j]_{S_x}$，在构建集合 $I_{x+1..m}$之前，我们希望评估它是否能够参与至少一个满足法定人数参数的 AWCI 集合。

观察1给定序列 S_1、...、S_m、一个插入缺失阈值 δ 和一个法定人数 q，S_x 中的区间 $[i,j]$ 对于 $1 \le x \le m$，不是 *AWCI* 对的候选，如果我们有 $|\{|\mathcal{C}(S_x[i,j]) \setminus \mathcal{C}(S_y) | \le \delta \mid y \in [1,m] \setminus x\}| < q-1$。

为利用这一观察，我们为 $\{x,y\} \subseteq [1,m]$构建了两类表格：RIDGE^c_{xy} 和 RIDGE^t_{xy}，其中上标 c 和 t 分别表示顺式 和反式。第一类表格 RIDGE^c_{xy}保存交集表 POS_{xy} 中的信息，用于判定 $[i,j]_{S_x}$ 是否推定参与某个AWCI 对；而 RIDGE^t_{xy} 保存来自 RIDGE^c_{yx}的信息，从而可以检验区间 $[i,j]_{S_x}$ 是否与 S_y中的区间相关联，而它有可能与这些区间形成 AWCI 对。表 RIDGE^c_{xy} 中的每个位置 $0 \le p < |S_x|$ 表示这样的位置 $0 \le p' \le p$ 的数量：对于这些位置，$S_x[p']$ 不包含 $\mathcal{C}$（S_y）中的任何字符。因此，如果$\text{RIDGE}^c_{xy}[j] - \text{RIDGE}^c_{xy}[i-1] > \delta$，字符串 S_x 中的区间 $[i,j]$ 就无法参与任何 AWCI 对。

我们现在借助表 RIDGE^t_{xy} 为参考区间 $[i,j]_{S_x}$ 的反式对构造类似的测试。我们称字符串 S_x 中的两个位置 i、j（对于 $0 \le i < j \le |S_x|$）位于同一脊上，当且仅当 $\text{RIDGE}^c_{xy}[j] - \text{RIDGE}^c_{xy}[i-1] \le \delta$ 成立。对于反式字符串 S_y、$x \ne y$，注意到 $\mathcal{C}$（$S_x[i,j]$）的字符可以出现在 S_y 中的许多不同位置，因此可以与形成 AWCI 对的不同区间相关联，这些区间与 $[i,j]_{S_x}$形成 AWCI 对。对于 S_y 中的每个候选区间 $[k,l]$，我们必须有 $\text{RIDGE}^c_{yx}[k] - \text{RIDGE}^c_{yx}[l-1] \le \delta$。为了快速测试此属性，我们构造二进制向量来识别 S_y 中包含 $\mathcal{C}$（$S_x[i,j]$）字符的脊；图 2 中给出了一个示例。与 S_x 中位置 $i..j$ 相关联的任何向量中的每个索引都对应于 S_y 中的一个不同脊。为了测试任意两个位置 i'、j'、$i \le i' < j' \le j$ 是否包含位于 S_y 中公共脊上的字符，只需对其关联的二进制向量执行简单的按位与运算即可。此外，

通过对 S_y 中 $i..j$ 位置关联的所有向量进行按位与运算，我们可以判定 S_y 中是否至少存在一条脊，该脊包含来自所有字符集 $S_x[i']$、 $i \le i' \le j$ 的字符。

数据结构 $\textsc{Ridge}^t_{xy}$的大小为 $\mathcal{O}\left(\delta+1\cdot|S_x|\cdot\|S_y\|\right)$ ，其使用会使 MACSI 的总体运行时间增加 $(\delta+1)$ 倍。在实践中，其大小远未达到渐近上界，因此 MACSI 的运行速度得到了显著提升。为了最小化 $\textsc{Ridge}^t_{xy}$中这些二进制向量的大小，我们放宽了以下要求：每条脊在所有二进制向量中都必须被分配一个唯一的索引。由于我们并不会将任意二进制向量相互比较，而只比较那些在 S_x 中的位置相距不超过 δ 个平凡插入缺失的二进制向量，因此这样做不会破坏数据结构的一致性。于是，我们在二进制向量中复用脊的索引。具体而言，在从左到右构建数据结构$\textsc{Ridge}^t_{xy}$ 时，对于给定位置 j ，如果与某个索引关联的脊尚未在 S_x 中此前距离不超过 δ 个插入缺失的任何位置出现过，我们就将新观察到的脊 $\textsc{Ridge}^c_{yx}[k]$ （对于某个 $k \in \textsc{Pos}_{xy}[j]$）分配给二进制向量中此前使用过的索引。

在算法2中，我们扩展了该方法，以允许在 $S_x[i,j]$ 或 S_y中的任何脊中总共有 δ 插入缺失， $[1,k]\setminus x$。为此，我们使用了两个数据结构 $\textsc{Active}^t$ 和 Δ^t，它们在每次迭代中更新和查询，用于测试 S_x 中具有固定左边界 i 和递增右边界 j的区间 $i..j$ 。

算法 2 测试字符串 S_x 中的位置 j 是否存在某个区间 $[i',j']_{S_x}$, $i' \le i \le j \le j'$——即，在至少 $q-1$ 个其他不确定字符串中具有区间的 AWCI。输入：字符串 S_x 中的起始位置 i 和当前位置 j，表 $\textsc{Ridge}^c_{xy}$ 和 $\textsc{Ridge}^t_{xy}$, 临时数据结构 $\textsc{Active}^t$ 和 Δ^t，插入缺失阈值 δ，法定人数 q，参考字符串 x。输出：如果 $S_x[i,j]$ 是一个候选 AWCI 集合，且该集合在至少 q 个字符串中具有少于 δ 个插入缺失，则返回 True，否则返回 False。1: $candidates \leftarrow 0$2: 对于 y 中的 $[1,k]\setminus x$ 执行 *// Determine the location of currently masked (trans-) ridges in* $\Delta^t[y]$3: $d \leftarrow \max(0, \delta+1+\textsc{Ridge}^c_{xy}[i]-\textsc{Ridge}^c_{xy}[j])$ *// Ridges that have been observed before but are not present at this position*4: $a_{off} \leftarrow \textsc{Active}^t[y] \wedge \neg(\textsc{Ridge}^t_{xy}[j] \vee \Delta^t[y][d])$ *// Ridges that have not been observed before but are present in* $\textsc{Ridge}^t_{xy}[j]$5: $a_{new} \leftarrow \textsc{Ridge}^t_{xy}[j] \wedge \neg(\textsc{Active}^t[y] \vee \Delta^t[y][d])$ *// Update* $\Delta^t[y]$ *for all ridges in* a_{off} *and* a_{new} 6: 对于 $d' \leftarrow 0$ 到 d 执行 7: $tmp \leftarrow \Delta^t[y][d'] \wedge a_{off}$8: $\Delta^t[y][d'] \leftarrow \Delta^t[y][d'] \vee a_{off}$ 9: $a_{off} \leftarrow tmp$10: 如果 $d' < j-i$ 则 11: $\Delta^t[y][d'] \leftarrow \Delta^t[y][d'] \vee a_{new}$12: 结束 if 13: 结束 for *// Carry ridges of* a_{off} *already marked in* $\Delta^t[y][d']$ *over to* $\Delta^t[y][d'+1]$ *// Add ridges of current position to* $\textsc{Active}^t[y]$14: $\textsc{Active}^t[y] \leftarrow \textsc{Active}^t[y] \vee \textsc{Ridge}^t_{xy}[j]$ 15: 如果 *// Test if* $S_x[i,j]$ *has at least one ridge* S_y *with less than* δ *indels* $\textsc{Active}^t[y] \wedge \neg\Delta^t[y][d] \neq 0$ 则 16: $candidates \leftarrow candidates+1$17: 结束 if 18: 结束 for 19: 返回 $candidates \ge q-1$

$\textsc{Active}^t[y]$ 是一个二进制向量，用于捕获 S_y 中包含来自 $\mathcal{C}$的字符的所有脊（$S_x[i,j]$）。 $\Delta^t[y]$ 是由 $\delta+1$ 个向量组成的数据结构，这些向量充当 S_y中脊的插入缺失计数器——$\Delta^t[y][d']$ 记录 S_y 中累积了 $0<d'\le\delta$ 个插入缺失的所有脊。显然，如果在 $S_x[i,j]$ 中已观察到 $d^*=\textsc{Ridge}^c_{xy}[j]-\textsc{Ridge}^c_{xy}[i]$ 个平凡插入缺失（第3行），则 S_y中的任何脊只允许发生 $\delta-d^*$ 个插入缺失。算法2使用 $\Delta^t[y]$中的（$\delta+1-d^*$）th向量作为掩码，以在后续计算中忽略所有超过 δ 个插入缺失的脊。

在每次迭代 j 中，S_y 的脊可分为三类：(i) 在任意 $i..j-1$ 次迭代中先前观察到的 $\textsc{Ridge}^t_{xy}[j]$ 中的脊；(ii) 先前观察到但不在 $\textsc{Ridge}^t_{xy}[j]$中的脊；(iii) $\textsc{Ridge}^t_{xy}[j]$的新脊。后两类需要更新 Δ^t，因此分别赋给向量 a_{off} 和a_{new}（第4、5行）。a_{off} 所有脊的插入缺失计数器均加一，而对于 a_{new}，计数器则增加 $\min(j-i,\delta)$；这两处更新均在第 6-13 行的 **for** 循环中完成。更新 $\textsc{Active}^t[y]$向量后（第 14 行），算法 2 可测试 S_y 中是否至少存在一条满足与 $S_x[i,j]$ 的插入缺失约束的脊（第 15-17 行）。仅当 $q-1$ 条字符串满足插入缺失约束时，算法 2 才返回 **True**；否则，参考字符串 S_x 中的当前位置 j 会产生集合 $J=[i,j]$（算法 1，第 4 行）。

进一步提升速度的方法是：针对集合 P_y 中的每个位置，分别细化字符集合 J；集合 P_y 在算法 1 的第 8 行构造。通过这样做，我们改进了 [3] 中已经描述的一项技术。注意，P_y 并不依赖于 S_x 中区间的候选右边界集合 J，而是依赖于左侧候选区间边界 i，因此可以在遍历集合 J 之前计算。对于不确定字符串 S_y 的每个位置 $p\in P_y$，我们收集所有集合 $\textsc{Pos}_{yx}[k^*]..\textsc{Pos}_{yx}[l^*]$ 中满足以下条件的索引 $J_p\subseteq J$：k^* 和 l^*、$k^*\le p\le l^*$ 与 p 相距至多 δ 个平凡插入缺失，即 $J_p=(\textsc{Pos}_{yx}[k^*]\cup\cdots\cup\textsc{Pos}_{yx}[l^*])\cap J$。因此，对于从 p 向 k^* 方向移动时访问到的不确定字符串 S_y 的每个位置 k'，我们都使用算法 2。这样，我们检验是否可能存在这样的 AWCI 集合：其中的成员至少位于除 S_y 和 S_x 之外的 $q-2$ 个字符串中，并且包含某个区间$[k,l]_{S_y}$、$k\le k'\le p\le l$。如果算法 2 返回**False**，我们就停止向 k^* 推进。此过程同样适用于位置 l'、$p\le l'\le l^*$。

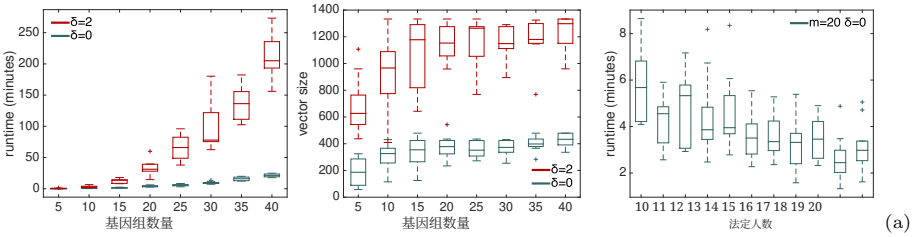
正如在 [3], 集合 J_p、$p\in P_y$中所述，随后可用于细化 J 的 S_x右区间界候选集合。不同于 [3], ，不过，这一细化是在多个序列之间执行的。对于任意位置集合P_y，我们确定在满足 S_x中的插入缺失阈值的任意$p\in P_y$ 邻域内找到的最右侧区间界，即，

$$j_y^\star:=\max_{p\in P_y}\{j_p\mid|J_p\cap[i,j_p]|\le\delta\}$$

然后，下一次迭代的新的右区间界候选集是

$$J=\big[i,\max_{\substack{Z\subseteq[1,m]\setminus x\\|Z|\ge q-1}}\big(\min_{z\in Z}j_z^\star\big)\big]$$

我们重复此过程，以细化每个集合 J_p。在实践中，不到 3 次迭代即可收敛到稳定的集合 J 和 J_p。



(b) (c) (a)

图3：(a) MACSI的运行时间和(b) 数据结构中位向量大小（每个位置），该数据结构 $\textsc{Ridge}^t_{xy}$ 应用于具有不同插入缺失阈值$\delta \in \{0, 2\}$的不同大小的基因组集合；(c) 在法定人数设置不同的20个基因组采样集合上，插入缺失阈值为 $\delta = 0$ 时 MACSI 的运行时间。

6 结果与讨论

我们展示了MACSI在一个包含93个细菌基因组的数据集上比较基因组序列的适用性；该数据集也曾用于评估Doerr 等人 [3] 提出的发现 AWCI 对的算法，这些数据最初由 Ciccarelli 等人 [1]获得。Doerr 等人通过使用蛋白质编码基因的注释并计算基因之间成对的 BLASTP 命中，将基因组序列转换为不确定字符串。所得不确定字符串的平均长度为2726个元素，并且每对不确定字符串包含 6499 个字符集相交的位置。

MACSI 使用 Python 实现；我们的实现支持并行计算，但为了评估方法的运行时间，所有计算均在单线程中执行。除了插入缺失参数δ 和法定人数之外，我们的实现还允许为报告的AWCI 设置最小大小，其中大小通过包含共享字符的位置数量来衡量AWCI对之间的距离。在所有实验中，我们将最小大小设为 10。实验在一台配备 64 个 2.3 吉赫兹核心的机器上运行。为了评估实现的运行时间，我们从包含93 个细菌基因组的池中抽取不同大小的基因组集合，然后使用不同的参数设置运行 MACSI。在所有实验中，抽样重复进行了10次。图3展示了结果。由于我们的加速效果取决于插入缺失的频率，插入缺失阈值 δ 对算法运行时间的影响最为显著；相比之下，降低法定人数的影响较为温和。比较 40 个完整的细菌基因组平均只需约 3.5 小时的计算时间。

所报告的 AWCI 对数量以及数据结构 $\textsc{Ridge}^t_{xy}$ 对我们算法的空间需求影响最大。由于前者较难通过算法改进，我们在实验中只监测后者。图3（b）展示了实验中 $\textsc{Ridge}^t_{xy}$ 的位向量大小，并表明在 $\delta = 0$ 的运行中所需位数少于 500，而在 $\delta = 2$ 的运行中少于 1400 位。

7 结论

我们提出了一种快速算法，用于在多个不确定字符串中发现近似弱公共区间。我们的主要贡献是设计了数据结构，可以高效地筛选出不可能成为具有给定最小大小的 AWCI 集合成员的区间。我们利用了基因组特性，尤其是这样一个事实：我们的不确定字符串将呈现许多平凡的插入缺失，从而为 AWCI 提供易于识别的边界。在不存在平凡插入缺失的情况下，如何加速通用算法仍然是一个悬而未决的问题。

利益冲突

作者声明不存在任何竞争利益。

作者贡献

BM 发起并指导了该研究项目，DD 设计并实现了本文介绍的算法并开展了实验；两位作者共同撰写了稿件，并阅读和批准了其最终版本。

参考文献

[{"id":0,"output":"[1] F. D. Ciccarelli、T. Doerks、C. von Mering、C. J. Creevey、B. Snel 和 P. Bork。迈向高度解析生命之树的自动重建。*Science*, 311(5765):1283–1287，2006。[2] G. Didier、T. Schmidt、J. Stoye 和 D. Tsur。字符串的字符集。*J.Discr. Alg.*，5(2):330–340，2007。[3] D. Doerr、J. Stoye、S. B¨ocker 和 K. Jahn。通过在不确定字符串中发现公共区间来识别基因簇。载于 第*12*届*RECOMB*比较基因组学卫星研讨会论文集 *RECOMB-CG'14*，*BMC Genomics*第15卷（增刊6），第S2页，2014年。[4] C. Ghiurcuta 和 B. Moret。评估共线性以改进比较研究。载于第*22*届分子生物学智能系统研讨会论文集 *ISMB'14*，*Bioinformatics*第30卷（第12期），第i9–18页，2014年。[5] T. Uno 和 M. Yagiura。枚举两个排列的所有公共区间的快速算法。*Algorithmica*，26(2):290–309，2000年。[6] M. Visnovsk´a、T. Vina˘r 和 B. Brejov´a。基于局部相似性的DNA序列分段。载于 第*13*届信息技术——应用与理论会议论文集 *ITAT'13*，第36–43页，2013年。[7] X. Zhang 和 B. Moret。通过系统发育信息传递细化调控网络。*IEEE/ACM*计算生物学与生物信息学汇刊，9(4):1032–1045，2012年。"}]