BOX Project 2023:
Minimal Absent Words in Plasmids

K. Břinda & R. Vicedomini

karel.brinda@inria.fr
riccardo.vicedomini@inria.fr

Updates

06/12/2023 at 16h00 : update on the "Final documents" section

24/11/2023 at 10h30 : first version

General requirements

- **Deadline.** The project deadline is **December 22, 2023**.
- **How to submit.** Share your project files through your Github repository shared with us at the beginning of the semester and send us an email confirming that everything is there.
- **Groups.** You can work in pairs (up to two students) or independently if preferred or unable to find a partner.
- **Collaboration.** While distinct individuals/pairs should not collaborate on the project itself, discussing results with others is permitted.
- **Carefully review each section of this document to ensure understanding before beginning work on a solution.** If you have any doubts or need clarifications, feel free to contact both of us without hesitation.

Motivation

One of the most significant breakthroughs in computational biology has been the discovery of CRISPR (Clustered Regularly Interspaced Short Palindromic Repeats). This system functions as an adaptive immunity mechanism, employed by bacteria to defend against bacterial viruses (phages) and predatory plasmids. It shares a notable analogy with computer antiviruses – every DNA molecule entering a bacterium is compared with an "internal database" of known "malicious code," and if a match is found, the molecule is destroyed, similarly to how computer antivirus eliminates malicious code before execution.

However, to evade detection, viruses and plasmids have developed sophisticated anti-defense systems and evolved to avoid certain patterns, such as words stored in the "internal databases". Many of these mechanisms are currently heavily studied by biologists.

The project's goal is to identify minimal words absent in a major plasmid dataset, to provide insights into words potentially targeted by CRISPR-Cas across diverse bacteria.

Definitions

- **Alphabet and strings.** We consider sequences (strings) on the DNA alphabet $\Sigma = \{\texttt{A}, \texttt{C}, \texttt{G}, \texttt{T}\}$. Given a string $s \in \Sigma^*$, we denote its length by $|s|$.
- **Canonical sequence.** The *canonical* version of a sequence s is the *lexicographic* minimum between itself and its reverse complement. Recall that the reverse complement of s is obtained by reversing it and transforming `A` into `T`, `C` into `G`, and vice versa. As an example, the reverse complement of `AGGTT` is `AACCT`. The canonical version of `AGGTT` is therefore `AACCT` because `AACCT` $<_{lex}$ `AGGTT`.
- **Absent word.** Let x and s be two strings on the alphabet Σ. We say that x is an *absent word* of s if neither x nor its reverse complement occur as a substring of s. This definition can be easily extended to a set of strings: we say that x is an *absent word* of a set $\mathcal{S}$ of strings if x is an absent word with respect to *all* $s \in \mathcal{S}$.
- **Minimal absent word.** Let x be a string of length $|x| \geq 3$ and $\mathcal{S}$ be a set of strings. We say that x is a *minimal absent word* (MAW, for short) of $\mathcal{S}$ if both the following conditions hold:
  1. x is an absent word of $\mathcal{S}$;
  2. for every substring w of x such that $|w| < x$, it holds that w or its reverse complement is a substring of at least one element of $\mathcal{S}$.

The Problem

The goal of this project is to implement one (or more) approaches to enumerate all minimal absent words in an input set $\mathcal{S}$ of DNA sequences, that are shorter than a user-specified length k_{max}.

- **Input.** The program you will implement must require the following mandatory parameters:
  1. FASTA file containing the set $\mathcal{S}$ of DNA sequences.
  2. Parameter k_{max}, specifying the maximum length of MAWs to be reported.

  Feel free to add other (optional) parameters that you might find convenient to carry out your analyses.
- **Output.** The output of your program will be a TSV (Tab-separated values) file which will contain the following two fields:
  1. Length k (from 3 up to k_{max}).
  2. Lexicographically *sorted* comma-separated list of all distinct *canonical* sequences of MAWs of length k for the set $\mathcal{S}$.

  The output TSV file must be *sorted* by the first value (length k) and should exclude any value of k for which no MAW of that length exists.
- **Test datasets** The dataset for testing your programs is the EBI plasmid database, available from the following URL: `https://t.ly/uXirO`.

Final documents

The program should be accompanied by three documents, all written in **English**.

1. **Readme**. A concise document outlining your code's goal, required dependencies, and instructions for running and interpreting outputs. This should be tailored for non-expert users.

2. **Scientific report (12 pages max).** Modeled as a scientific article, this report should include the following sections:

   (a) Introduction: Provide context and objectives.

   (b) Methods: Detail the techniques employed.

   (c) Results: Present the outcomes of your work.

   (d) Discussion: Share conclusions regarding implemented methods, the quality of results, and limitations, especially concerning input parameters and data.

   (e) Bibliography: Cite relevant sources.

3. **Software report (2 pages max).** A brief report outlining the structure of your code and its implementation.

Expectations

Implement your code in Python 3 unless another language is agreed upon. This project is open-ended, allowing for various data structures without a strict requirement for efficiency. Explore one or multiple solutions through theoretical and empirical analysis. Draw comprehensive conclusions, considering implementation specifics, trade-offs, advantages, limitations, and potential improvements. Push the limits of your approach, like finding the maximum k_{max} value within reasonable time/memory.

The main challenge is in the experiments and conclusions. The report is crucial for evaluation, and originality impacts the final mark. Key points:

- Formalization of your algorithms with pseudo-code.

- Worst-case complexity analysis (both theoretically and experimentally).

- Clarity of the presentation (both code and the final documents).

- If you settle for a simple solution, we might expect a more in-depth analysis