Bioinformatics Algorithms **Dec 2024**

Chapter 1: Genome Replication

Princeton University Author: Runpeng Luo

Disclaimer: *These notes have not been subjected to the usual scrutiny.*

1.1 Introduction

Cell division is an important biological procedure that divides a parent cell into two daughter cells. Each daughter cell has same copy of DNA as the parent cell. Genome replication is the key step involved in cell division, and we want to find where does the genome replication begin.

In biology, the genome replication begins in a genomic region called the **replication origin** denoted as *ori*. Replication is performed by molecular copy machine **DNA polymerases**. DNA sequence is double-stranded, where A-T and C-G form complementary strand pairs. During genome replication, the double strands first unwind to two single strand namely forward (5'-3') and reverse strands (3'-5'), and replications are happened on each single strand by synthesizing nucleotides in the corresponding complementary strand, respectively.

Problem 1.1 *Given a DNA string Genome, the goal is to find the location of ori in Genome in a de novo approach.* i.e. *no prior knowledge of ori for similar genomes.*

For simplicity, we assume the genome is a type of bacteria with single circular chromosome.

1.2 Hidden Message in ORI

Biologist may perform experiments by deleting various short segments from the genome, one per run, and see if the deletion stops replication.

Since only the sequence context is given, the idea to extract hidden (or special) information from genome sequences that distinguishes ori region from non-ori regions, based on prior known ori locations from biological experiments.

Definition 1.2 ***DnaA box***. *From biology, a short segment DnaA box in ori region is binded by a protein DnaA, which initiates the replication. DnaA box may appear in multiple location as well.*

DnaA box serves as a message that tells DnaA protein where to bind. Thus, suppose we know the location of ori, we then want to find the DnaA box given ori.

Problem 1.3 ***Hidden Message Problem***. *Given a string Text representing the ori of Genome, find a hidden message in Text that represents DnaA box.*

[0]These notes are partially based on the textbook [1]

Since DnaA box is an important DNA segment for DNA replication, we assume that DnaA box should appear frequently such that it is less likely to destroy all of them due to mutations. Thus, the problem becomes to find the most frequent segment of size k in ori of length L, which suppose we know that the DnaA box is length k.

Problem 1.4 ***Frequent Words Problem****. Given a string Text and an integer k, find all most frequent k-mers in Text.*

The frequent words problem can be solved naively by first sliding all size k window *Pattern* in *Text* and computing **PATTERN_COUNT(Text, Pattern)**, and next selecting all k-mers with maximum counts. The naive pattern count takes steps $(|Text| - k + 1) \cdot k$, and the frequent words problem can be solved in:

$$(|Text| - k + 1)^2 \cdot k + (|Text| - k + 1) = O(|Text|^2 \cdot k) \tag{1.1}$$

Since DnaA box can be binded in either strands, both its forward and reverse complement strand are canonicalized to same pattern. Note that the most frequent k-mers may not be unique and may occur as repeats throughout entire genome (not just in ori region *Text*) the next problem is to check how the most frequent k-mer distributed along the genome.

Problem 1.5 ***Pattern Matching Problem****. Given a candidate DnaA box Pattern and Genome, find all the starting positions in Genome of substring Pattern.*

The pattern matching problem can also be naively solved by sliding a size $|Pattern|$ window along genome and checking both strands for pattern matching. This approach takes $(|Genome| - |Pattern| + 1) \cdot 2|Pattern| = O(|Genome||Pattern|)$.

Given all the starting positions of *Pattern*, we would like to find a size L region representing ori that squeezes most of *Pattern*, which forms a *clump*.

Definition 1.6 *A k-mer forms a (L, t)-clump if it appears at least t times in a size L region.*

The idea is that a significant number of same canonical k-mers all appear within a small size L region is very unlikely to be happened by chance only, thus, we have the following problem that aims to capture both *ori* region and hidden message *DnaA box*.

Problem 1.7 ***Clump Finding Problem****. Given a string Genome, integers k, L and t. Find all distinct k-mers forming (L, t)-clumps in Genome.*

Caveat: if the clump-finding algorithm outputs multiple DnaA Box, we still cannot decide the correct DnaA Box and locate ori region. Additional biological insights are required.

1.3 Asymmetry of Replication

As we know, the genome replication starts from ori region, and ends to *ter* region. A single strand consists a pair of forward half strand and reverse half strand, from ori to ter. The forward half strand is defined in direction 5' to 3', and the reverse half strand is defined in direction 3' to 5'.

$$\text{forward strand: } 5' \to ter \to \ldots \to ori \to \ldots \to ter \to 3'$$
$$\text{reverse strand: } 3' \leftarrow ter \leftarrow \ldots \leftarrow ori \leftarrow \ldots \leftarrow ter \leftarrow 5'$$

Thus, the first half strand of the forward strand is the reverse half strand, and the second half strand would be forward half strand. The DNA polymerase works from ori to ter and in reverse direction 3' to 5'. The genome replication consists following steps.

1. Helicase enzyme separates the double-stranded DNA to two single strands, served as replication fork.
2. primase enzyme creates the primer.
3. DNA polymerase enzyme binds the primer and starts making nucleotides in direction 3' to 5' continuously. (the two reverse half strands as template)
4. For the two forward strands as template, Okazaki fragments (primer plus genome segment) are made one chunk at a time progressively by primase enzyme and DNA polymerase enzyme, until ter is reached. This step works in the start&stop fashion.
5. Gaps between consecutive Okazaki fragments are sewn together by DNA ligase enzyme.

The forward half strands of parent chromosomes are also called lagging strands, where the reverse half strands are called leading strands. The different strategy causes asymmetry of replication in different half-strands.

1.4 Peculiar Statistics of Half Strands

Given a sequence starts from and ends with terminus of replication partitioned into multiple small fragments, we can compute the frequency of bases per fragment. In page 23, it shows the following pattern:

$$5' : ter \to \text{high C, low G} \to ori \to \text{high G, low C} \to ter : 3'$$

The reverse half-strand (ter to ori) expresses high C frequency, where the forward half-strand (ori to ter) expresses low C frequency. We have:

$$5' : ter \to \text{low (G-C)} \to ori \to \text{high (G-C)} \to ter : 3'$$

1.4.1 Deamination

DNA sequence stays stable if it is in double-stranded structure. Due to 5' to 3' direction of DNA polymerase, it synthesizes DNA quickly on reverse half-strand (or leading strand) but delays in forward half-strand (or lagging strand). Reverse half-strand is mostly stay in double-stranded structure, while the forward half-strand is mostly in single-stranded structure, during replication.

Single stranded DNA has much higher mutation rate than double stranded DNA, in which C has higher tendency to be mutated to T through a process called **deamination**. The deamination rates rise 100x if the DNA is in single-stranded. The mutation from C to T converts C-G pair to T-G pair, and consequently

mutated to T-A pair when the bond is repaired in the next round of replication, which leads to decreased G frequency in reverse half-strand.

Thus, the deamination explains that the forward half-strand has higher G and lower C (due to deamination), where the reverse half-strand has lower G (due to bond repair) and higher C.

1.5 Final Attempts of ORI Finding

Consider an imaginary walk along the genome from ter to ori to ter in direction 5' to 3', which walks along reverse half-strand and then forward half-strand. By accumulating the counts of $G - C$, one would expect to observe decreasing G (due to bond repair) and leads to decreased $G - C$ upon ori. And from ori to ter following the forward half-strand, one would expect to observe decreasing C (due to deamination) and leads to increased $G - C$. Thus, we have:

$$5' : ter \rightarrow \text{decreasing \#G-\#C} \rightarrow ori \rightarrow \text{increasing \#G-\#C} \rightarrow ter : 3'$$

Since we don't have prior knowledge of the location of ori and ter, the walk starts from a random genome position on a circular genome and follows a fixed position from left to right. It accumulates the count $G-C$. When $G - C$ turns from decreasing trend to increasing trend (turning point), we know that the ori region is at the surrounding positions. When $G - C$ turn from increasing to decreasing, we discovered ter region instead. Regardless of the starting position, the accumulated counts reached minimum value when it visits the ori region. The plot of accumulated counts of $G - C$ is called *skew diagram*.

Problem 1.8 ***Minimum Skew Problem***. *Given a DNA string Genome, find the positions that minimized the accumulated counts of $G - C$ from left-most to right-most position.*

Given the set of approximated positions of ori region, we can then finds the clumps in nearby regions with increasing window size. Since real data always come with sequencing error, we should account for mismatches and counts non-perfect k-mers as well. The number of mismatches between 2 k-mers are called *Hamming distance*. And we can similarly formulate the approximate pattern matching problem by allowing at most d mismatches. (omitted here)

Caveat 1: The most frequent words within the text may not necessarily appears in the text, if mismatches are allowed. The naive approach is to perform k-mer counting among all 4^k possible k-mers, which is extremely inefficient.

1.6 Conclusion

In real dataset, noises occur and skew diagram is imperfect in most case, such as multiple turning points. The skew-based ori finding approach is powerful, but imperfect and may discover multiple DnaA Box. Since there are other types of hidden message in genome that also appears to form a clump such as motifs but not due to duplications. Nevertheless, biologist can pick the correct one out of a handful of potential ones.

Currently, there is hypothesis that there might be more than 1 ori in bacteria genome, and consequently leads to more than one turning points in skew diagram. This hypothesis is unsolved due to following reasons:

First, genome rearrangement such as reversal can reposition gene to different strands, which causes fluctuation of $G - C$ near the reversal region. E.g., a segment from reverse half-strand been switched to forward half-strand would cause increasing G and decreasing C in the skew of forward half-strand.

Second, different species of bacteria may exchange genetic material in **horizontal gene transfer**, which can also cause irregularity in the skew diagram.

Challenge: Find the ori in Salmonella enterica.

Appendix

A1.1 DNA Basics

DNA sequence is a string $S \in \Sigma^*$ of alphabet $\Sigma = \{A, C, T, G\}$. A-T and C-G are complement pairs. A sequence can either be represented in its forward strand or reverse complemented strand, in the equivalent format. Suppose S is represented in the forward strand, read from left (5') to right (3'), the reverse complemented strand of S as $\overline{S}$ is defined as reverse all characters and transform all A to T, C to G, and vice versa.

Formally, let $\bar{p}$ denotes the complementary nucleotide of p, we have:

$$\overline{S} = \overline{p_1 p_2 \dots p_n} = \overline{p_n} \dots \overline{p_1}$$

For DNA synthesis, given one strand of DNA served as template strand, DNA polymerase will generate free floating nucleotides in the complementary strand and attach to the template strand, forms the double-stranded DNA. Check page 11 for illustrations.

$$5' \to \dots \to 3'$$
$$3' \leftarrow \dots \leftarrow 5'$$

A1.2 k-mer Counting

Given a string of length n, to count for all observed k-mers in the string, a naive approach would take $O(n^2)$ by counting each k-mer through sliding all k-mer windows, or takes $O(4^{n-k+1})$ storage for count array and $O(n-k+1)$ for single-traversal.

k-mer to index translation

To make k-mer counting more efficient, first, we need to find an efficient translation function that maps a k-mer to an integer, and vice versa. Namely, we need to construct a bijective function $T : \Sigma^k \to \{0, \dots, 4^k - 1\}$ and its inverse T^{-1}. T should be as efficient as possible.

Suppose we have an ordering $A < C < G < T$, and we order all k-mers lexicographically into $S = (s_0, s_1, \dots, s_{4^k-1})$. If we remove the last nucleotide in all k-mers of S and leads to S', S' remains sorted. This is similar to the fact when we sort numbers by comparing digits from left to right, a number is less than another number if the former has smaller higher significant digits compared to the latter. Let m be a function maps A, C, G, T to $0, 1, 2, 3$, respectively, we can define T as follows:

$$T(s,k) := \begin{cases} 0 & \text{if } k = 0, \\ 4 \cdot T(s[0 : |s| - 2]) + m(s[|s| - 1]) & \text{otherwise.} \end{cases}$$

Similarly, we can define T^{-1} as follows:

$$T^{-1}(v,k) := \begin{cases} \textbf{invalid} & \text{if } k \leq 0, \\ m^{-1}(v) & \text{if } k = 1, \\ T^{-1}(\text{quotient}(v,4), k-1) || m^{-1}(\text{remainder}(v,4)) & \text{otherwise.} \end{cases}$$

Note that both functions take $O(k)$ to compute.

Efficient approach

Given T, we can achieve k-mer counting using following steps:

1. Construct index array A with $A_i := T(i\text{th } k\text{-mer in string})$.
2. Sort A lexicographically.
3. Keep a counter $c := 0$ and an index $s := A_1$, iterate through A, increment c, and assign $s := A_i$.
4. If $s \neq A_i$, report c and re-assign $c := 1$ and $s = A_i$, back to step 3 until all indices are visited.

Note that this approach takes $O(nk) + O(n \log n) + O(n) = O(nk)$ time complexity (assumes $n \leq b^k$ with log base b) and $O(n)$ space complexity, which works more efficient than naive approach and takes linear space.

A1.3 Clump Finding Problem

A naive approach would slide a size L window along the size n genome. For each window, it performs k-mer counting and asks if there are any k-mers have counts $\geq t$, which suggests current window to be a (L,t)-clump. One motivation for improvement indicates that the k-mer counting result for adjacent size L window only varies along the 2 k-mers at the boundary, we can rather reuse the k-mer counting result from previous iteration and slightly modify for current iteration. For instance, we decrement the count for left-most k-mer and increment the count for the right-most k-mer by 1, respectively. The modification takes constant time. Thus, we only need to check if the right-most k-mer in the size L window forms a new (L,t)-clump by examining (L,t)-clump k-mers from very first iteration and compute k-mer counting once only. Roughly speaking, the improvement would leads to $O(nk) + O(n - L + 1) = O(nk)$ time complexity and $O(n)$ space to store the (L,t)-clump k-mers and a k-mer counting array.

A1.4 k-mer Counting with Mismatches

To allow mismatches in k-mer counting, we define the d-neighborhood of a k-mer as follows:

Definition 1.9 *The d-neighborhood NEIGHBORS(Pattern,d) is a set of k-mers that all have hamming distance at most d compared to k-mer Pattern.*

As an example, the d-neighborhood of any 3-mer with $d = 1$ would have size:

$$1 + \binom{3}{1} 3 = 10$$

During k-mer traversing, we increment the count by 1 for all the k-mers in its d-neighborhood.

caveat: The k-mer count may be nonzero even if it does not appear in the text.

Compute d-neighborhood

The d-neighborhood can also be computed in a recursive way similar to k-mer indexing. Given a k-mer *Pattern*, and we want to compute **NEIGHBORS**$(Pattern, d)$. Given $x \in$ **NEIGHBORS**$(Pattern[0 : k-2], d)$, we have:

$$\textbf{NEIGHBORS}(Pattern, d) \ni \begin{cases} x||Pattern[k-1] & \text{if } hamming(Pattern, x) = d, \\ x||(\forall z \in \Sigma^*) & \text{otherwise.} \end{cases}$$

Challenge: estimate the runtime of **NEIGHBORS**$(Pattern, d)$ given **NEIGHBORS**$(Pattern[0 : k-2], d)$.

Algorithm sketch

We can adapt the efficient approach defined in k-mer counting section, and add one additional step before k-mer indexing. Since every visited k-mer leads to additional count of $|$**NEIGHBORS**$(Pattern, d)|$, before the indexing step, we can compute the d-neighborhood for each k-mer in the text, and flatten the k-mer array. The rest of the steps remain the same. Note that in the end, the frequency of a k-mer accounts for all k-mers in the text that has at most d mismatches, since the former would appear in the d-neighborhood set of the all the latters of k-mers.

A1.5 Probabilities of Patterns in a Random String

In Statistics, we always consider the significance of an observation by comparing with the probability of seeing such observation from a random population. We consider the probability of k-mers here.

Lemma 1.10 *Given alphabet $\Sigma = \{A, C, T, G\}$, the probability of two randomly generated k-mers are identical is as follows:*

$$p = 4^k \cdot (4^{-k})^2 = 4^{-k}$$

Lets define the following probability:

$$\textbf{Pr}(N, A, Pattern, t) := \textbf{Pr}(Pattern \text{ appears} \geq t \text{ times in a random size } N \text{ string with } |\Sigma| = A)$$

Note that $\textbf{Pr}(N, A, Pattern, t)$ can only be approximated due to Overlapping Words Paradox, which suggests that the choice of Pattern affects the probability. Thus, we assumes that all k-mers are non-overlapping and obtains an approximation instead. Note that the probability can be computed via dividing the number of size N string with $\geq t$ occurrence of Pattern versus the total number of size N strings. We can count the numerator based on stars and bars model. A string has t instances of Pattern with length k would have $N - kt$ positions that free to change. Thus, we have:

$$p := \textbf{Pr}(N, A, Pattern, t) \approx \frac{\binom{N-kt+t}{N-kt} A^{N-kt} 1^t}{A^N} = \frac{\binom{N-kt+t}{t}}{A^{kt}}$$

And we have:

$$\textbf{Pr}(\text{Pattern appears} < t \text{ times}) = 1 - p$$

$$\textbf{Pr}(\text{all possible Pattern appears} < t \text{ times}) = (1-p)^{A^t}$$

$$\textbf{Pr}(\text{at least 1 Pattern appears} \geq t \text{ times}) = 1 - (1-p)^{A^t} \leq pA^t \qquad \text{By union bound}$$

TODO: Extension materials page 58-66

References

[1] Compeau, P., and Pevzner, P. *Bioinformatics algorithms: an active learning approach.* Active Learning Publishers, 2015.