Bioinformatic

Where in the Genome Does DNA Replication Begins?

2025

Elnaz Farokhi
Fatemeh Shabaninejad

A journey of Thousands miles.....

Genome replication is one of the most important tasks carried out in the cell. Before a cell can divide, it must first replicate its genome so that each of the two daughter cells inherits its own copy.

A journey of Thousands miles.....

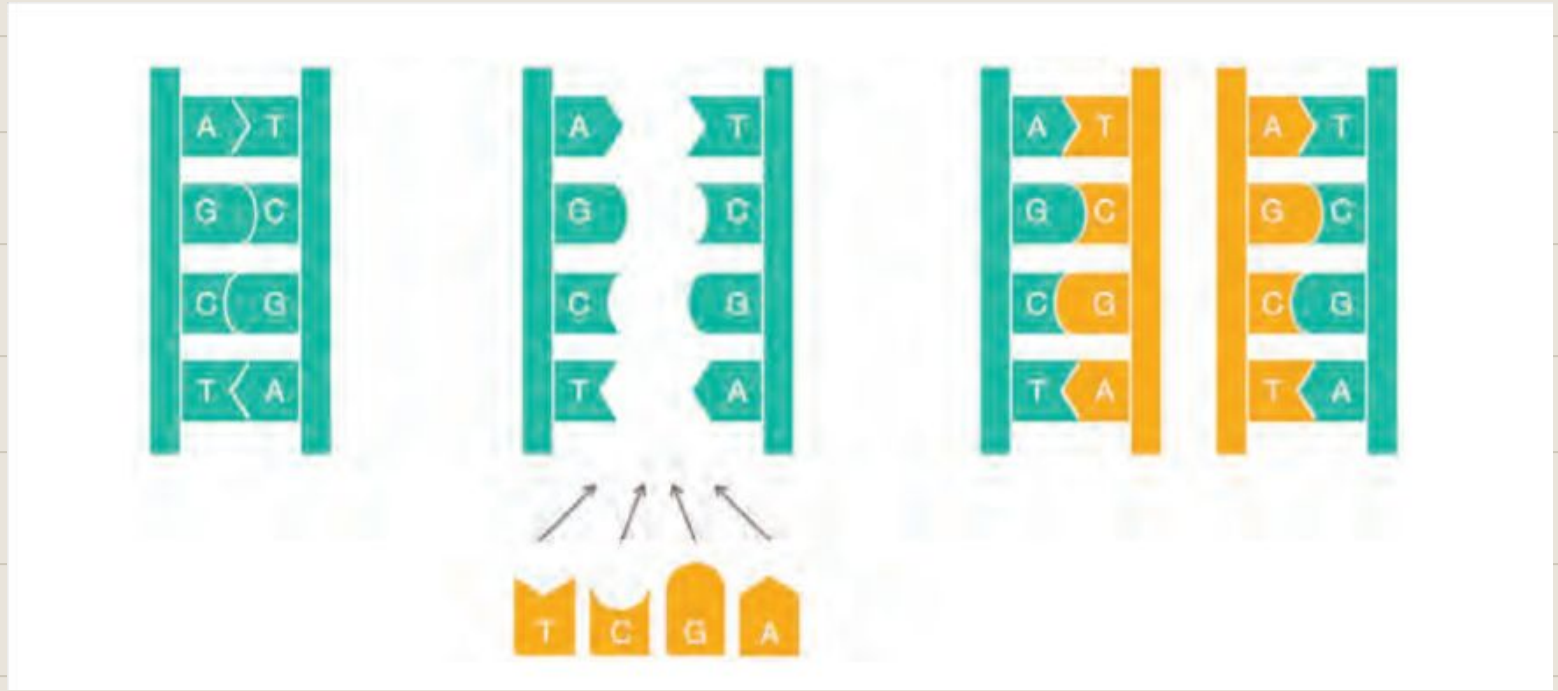
In 1953, James Watson and Francis Crick completed their landmark paper on the DNA double helix with a now-famous phrase:

> *It has not escaped our notice that the specific pairing we have postulated immediately suggests a possible copying mechanism for the genetic material.*

A journey of Thousands miles.....

They conjectured that the two strands of the parent DNA molecule unwind during replication, and then each parent strand acts as a template for the synthesis of a new strand. As a result, the replication process begins with a pair of complementary strands of DNA and ends with two pairs of complementary strands

A journey of Thousands miles.....



What is the Biological Problem ?

Replication begins in a genomic region called the replication origin (denoted oriC) and is performed by molecular copy machines called DNA polymerases. Locating oriC presents an important task not only for understanding how cells replicate but also for various biomedical problems.

What's the Computational Problem ?

Finding Origin of Replication Problem:

Input: A DNA string *Genome*.
Output: The location of *oriC* in *Genome*.

Why Should Computer Scientists Care ?

Finding Origin of Replication Problem asks a legitimate biological question, it does not present a well-defined computational problem. Indeed, biologists would immediately plan an experiment to locate oriC: for example, they might delete various short segments from the genome in an effort to find a segment whose deletion stops replication. Computer scientists, on the other hand, would shake their heads and demand more information before they can even start thinking about the problem.

Why Should Computer Scientists Care ?

Why should biologists care what computer scientists think? Computational methods are now the only realistic way to answer many questions in modern biology. First, these methods are much faster than experimental approaches; second, the results of many experiments cannot be interpreted without computational analysis. In particular, existing experimental approaches to oriC prediction are rather time consuming. As a result, oriC has only been experimentally located in a handful of species. Thus, we would like to design a computational approach to find oriC so that biologists are free to
spend their time and money on other tasks.

DnaA Boxes

we will focus on the relatively easy case of finding oriC in bacterial genomes, most of which consist of a single circular chromosome. Research has shown that the region of the bacterial genome encoding oriC is typically a few hundred nucleotides long. Our plan is to begin with a bacterium in which oriC is known, and then determine what makes this genomic region special in order to design a computational approach for finding oriC in other bacteria. Our example is Vibrio cholerae, the bacterium that causes cholera; here is the nucleotide sequence appearing in its oriC:

DnaA Boxes

atcaatgatcaacgtaagcttctaagcatgatcaaggtgctcacacagtttatccacaac
ctgagtggatgacatcaagataggtcgttgtatctccttcctctcgtactctcatgacca
cggaaagatgatcaagagaggatgatttcttggccatatcgcaatgaatacttgtgactt
gtgcttccaattgacatcttcagcgccatattgcgctggccaaggtgacggagcgggatt
acgaaagcatgatcatggctgttgttctgtttatcttgttttgactgagacttgttagga
tagacggtttttcatcactgactagccaaagccttactctgcctgacatcgaccgtaaat
tgataatgaatttacatgcttccgcgacgatttacctcttgatcatcgatccgattgaag
atcttcaattgttaattctcttgcctcgactcatagccatgatgagctcttgatcatgtt
tccttaaccctctatttttacggaagaatgatcaagctgctgctcttgatcatcgtttc

DnaA Boxes

How does the bacterial cell know to begin replication exactly in this short region within the much larger Vibrio cholerae chromosome, which consists of 1,108,250 nucleotides? There must be some "hidden message" in the oriC region ordering the cell to begin replication here. Indeed, we know that the initiation of replication is mediated by DnaA, a protein that binds to a short segment within the oriC known as a DnaA box. You can think of the DnaA box as a message within the DNA sequence telling the DnaA protein: "bind here!" The question is how to find this hidden message without knowing what it looks like in advance — can you find it? In other words, can you find something that stands out in oriC? This discussion motivates the following problem.

Hidden Message Problem

Hidden Message Problem:
Find a "hidden message" in the replication origin.

Input: A string *Text* (representing the replication origin of a genome).
Output: A hidden message in *Text*.

it again makes absolutely no sense to a computer scientist because the notion of a "hidden message" is not precisely defined.

Gold-Bug

The oriC region of Vibrio cholerae is currently just as puzzling as the parchment discovered by William Legrand in Edgar Allan Poe's story "The Gold-Bug". Written on the parchment was the following:

```
53++!305))6*;4826)4+.)4+);806*;48!8‘60))85;1+(;:+*8
!83(88)5*!;46(;88*96*?;8)*+(;485);5*!2:*+(;4956*2(5
*-4)8‘8*;4069285);)6!8)4++;1(+9;48081;8:8+1;48!85:4
)485!528806*81(+9;48;(88;4(+?34;48)4+;161;:188;+?;
```

He reasons that the three consecutive symbols ";48" appear with surprising frequency on the parchment:

Gold-Bug

```
53++!305))6*;48826)4+.)4+);806*;48!8‘60))85;1+(;:+*8
!83(88)5*!;46(;88*96*?;8)*+(;485);5*!2:*+(;4956*2(5
*-4)8‘8*;4069285);)6!8)4++;1(+9;48081;8:8+1;48!85;4
)485!528806*81(+9;48;(88;4(+?34;48)4+;161;:188;+?;
```

Legrand had already deduced that the pirates spoke English; he therefore assumed that the high frequency of ";48" implied that it encodes the most frequent English word, "THE".

```
53++!305))6*THE26)H+.)H+)TE06*THE!E‘60))E5T1+(T:+*E
!E3(EE)5*!TH6(TEE*96*?TE)*+(THE5)T5*!2:*+(TH956*2(5
*-H)E‘E*TH0692E5)T)6!E)H++T1(+9THE0E1TE:E+1THE!E5TH
)HE5!52EE06*E1(+9THET(EETH(+?3HTHE)H+T161T:1EET+?T
```

Counting Words

Operating under the assumption that DNA is a language of its own, let's borrow Legrand's method and see if we can find any surprisingly frequent "words" within the oriC of Vibrio cholerae. We have added reason to look for frequent words in the oriC because for various biological processes, certain nucleotide strings often appear surprisingly often in small regions of the genome. For example, ACTAT is a surprisingly frequent substring of :

ACA**ACTAT**GCAT**ACTAT**CGGGA**ACTAT**CCT.

Counting Words

We use the term k-mer to refer to a string of length k and define COUNT(Text, Pattern) as the number of times that a k-mer Pattern appears as a substring of Text. Following the above example :

$$\text{COUNT}(\text{ACA}\mathbf{ACTAT}\text{GCAT}\mathbf{ACTAT}\text{CGGGA}\mathbf{ACTAT}\text{CCT}, \mathbf{ACTAT}) = 3.$$

Note that COUNT(CGATATATCCATAG, ATA) is equal to 3 (not 2) since we should account for overlapping occurrences of Pattern in Text.

Counting Words

To compute COUNT(Text, Pattern), our plan is to "slide a window" down Text, checking whether each k-mer substring of Text matches Pattern.

```
PATTERNCOUNT(Text, Pattern)
    count ← 0
    for i ← 0 to |Text| − |Pattern|
        if Text(i, |Pattern|) = Pattern
            count ← count + 1
    return count
```

The Frequent Words Problem

We say that Pattern is a most frequent k-merin Text if it maximizes COUNT(Text, Pattern) among all k-mers.

Frequent Words Problem:
Find the most frequent k-mers in a string.

Input: A string *Text* and an integer *k*.
Output: All most frequent *k*-mers in *Text*.

The Frequent Words Problem

A straightforward algorithm for finding the most frequent k-mers in a string Text checks all k-mers appearing in this string (there are |Text| k + 1 such k-mers) and then computes how many times each k-mer appears in Text. To implement this algorithm, called FREQUENTWORDS, we will need to generate an array COUNT, where COUNT(i) stores COUNT(Text, Pattern) for Pattern = Text(i, k) , for example :

Text	**A**	**C**	**T**	G	**A**	**C**	**T**	C	C	C	A	C	C	C	C
COUNT	**2**	1	1	1	**2**	1	1	3	1	1	1	3	3		

FIGURE 1.2 The array COUNT for *Text* = ACTGACTCCCACCCC and $k = 3$. For example, COUNT(0) = COUNT(4) = 2 because ACT (shown in boldface) appears twice in *Text* at positions 0 and 4.

The Frequent Words Problem

The Algorithm for doing this is :

```
FrequentWords(Text, k)
    FrequentPatterns ← an empty set
    for i ← 0 to |Text| − k
        Pattern ← the k-mer Text(i, k)
        Count(i) ← PatternCount(Text, Pattern)
    maxCount ← maximum value in array Count
    for i ← 0 to |Text| − k
        if Count(i) = maxCount
            add Text(i, k) to FrequentPatterns
    remove duplicates from FrequentPatterns
    return FrequentPatterns
```

The Frequent Words Problem

NOTE , Although FREQUENTWORDS finds most frequent k-mers, it is not very efficient. Each

call to PATTERNCOUNT(Text,Pattern) checks whether the k-mer Pattern appears in posi tion 0 of Text, position 1 of Text, and so on.

To simplify the matter, computer scientists often say that the runtime of FREQUENTWORDS has an upper bound of |Text|^2 · k steps and refer to the complexity of this algorithm as O|Text|^2 · k which is :

$$\mathcal{O}(|Text|^2 \cdot k)$$

The Frequent Words Problem

k	3	4	5	6	7	8	9
count	25	12	8	8	5	4	3
k-mers	tga	atga	gatca	tgatca	atgatca	atgatcaa	atgatcaag
			tgatc				cttgatcat
							tcttgatca
							ctcttgatc

FIGURE 1.3 The most frequent *k*-mers in the *oriC* region of *Vibrio cholerae* for *k* from 3 to 9, along with the number of times that each *k*-mer occurs.

The Frequent Words Problem

For example, the 9-mer ATGATCAAG appears three times .

We highlight a most frequent 9-mer instead of using some other value of k because experiments have revealed that bacterial DnaA boxes are usually nine nucleotides long. The probability that there exists a 9-mer appearing three or more times in a randomly generated DNA string of length 500 is approximately 1/1300

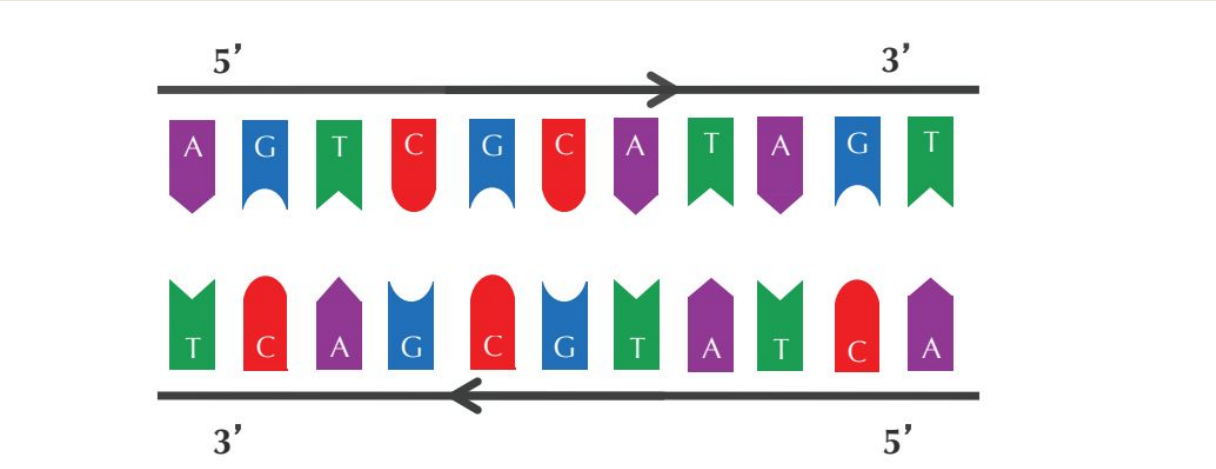
The Frequent Words Problem

In fact, there are four different 9-mers repeated three or more times in this region: ATGATCAAG, CTTGATCAT, TCTTGATCA, and CTCTTGATC. The low likelihood of witnessing even one repeated 9-mer in the oriC region of Vibrio cholerae leads us to the working hypothesis that one of these four 9-mers may represent a potential DnaA box that, when appearing multiple times in a short region, jump-starts replication …

But which one?

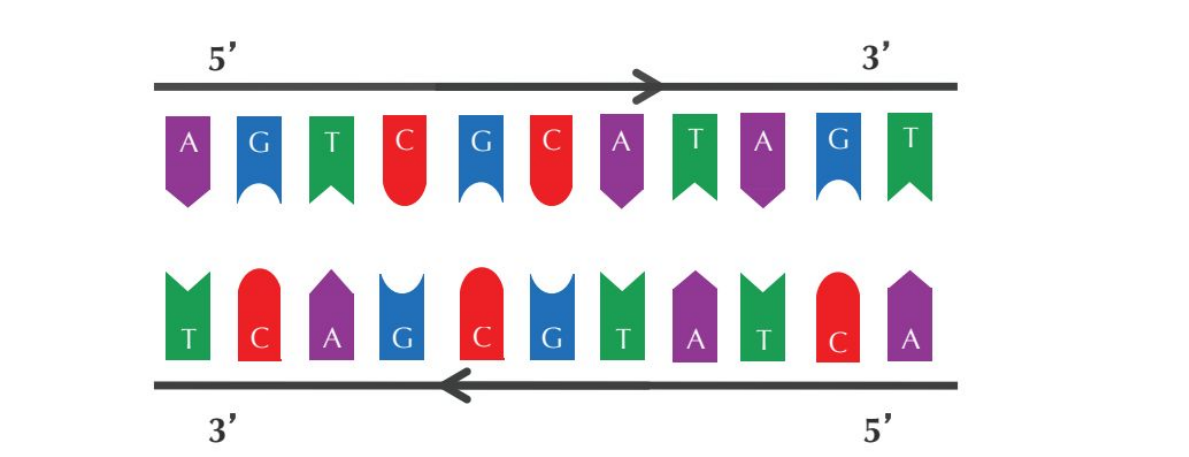
Some Hidden Messages are More Surprising than Others

Recall that nucleotides **A** and **T** are complements of each other, as are **C** and **G**. Having one strand of DNA and a supply of "free floating" nucleotides , one can imagine the synthesis of a complementary stran on a template strand. This model of replication was confirmed by **Meselson and Stahl** in 1958



Some Hidden Messages are More Surprising than Others

At this point, you may think that we have made a mistake, since the complementary strand in Figure out TCAGCGTATCA from left to right rather than ACTATGCGACT. We have not : each DNA strand has a direction, and the complementary strand runs in the opposite direction to the template strand, as shown by the arrows in Figure . Each strand is read in the 5′ to 3′ direction



Given a nucleotide p, we denote its complementary nucleotide as $\overline{p}$. The **reverse complement** of a string $Pattern = p_1 \cdots p_n$ is the string $\overline{Pattern} = \overline{p_n} \cdots \overline{p_1}$ formed by taking the complement of each nucleotide in *Pattern*, then reversing the resulting string.

And now we have this problem to follow :

Reverse Complement Problem:
Find the reverse complement of a DNA string.

Input: A DNA string *Pattern*.
Output: $\overline{Pattern}$, the reverse complement of *Pattern*.

Which is a easy problem to be solved :

```
def reverse_complement(dna):
    complement = {'A': 'T', 'T': 'A', 'C': 'G', 'G': 'C'}
    return ''.join(complement[base] for base in reversed(dna))
```

Now let's get back to our sequence that we had :

atcaatgatcaacgtaagcttctaagc**ATGATCAAG**gtgctcacacagtttatccacaac
ctgagtggatgacatcaagataggtcgttgtatctccttcctctcgtactctcatgacca
cggaaag**ATGATCAAG**agaggatgatttcttggccatatcgcaatgaatacttgtgactt
gtgcttccaattgacatcttcagcgccatattgcgctggccaaggtgacggagcgggatt
acgaaagcatgatcatggctgttgttctgtttatcttgttttgactgagacttgttagga
tagacggtttttcatcactgactagccaaagccttactctgcctgacatcgaccgtaaat
tgataatgaatttacatgcttccgcgacgatttacct**CTTGATCAT**cgatccgattgaag
atcttcaattgttaattctcttgcctcgactcatagccatgatgagct**CTTGATCAT**gtt
tccttaaccctctatttttacggaaga**ATGATCAAG**ctgctgct**CTTGATCAT**cgtttc

Interestingly, among the four most frequent 9-mers in the oriC region of Vibrio cholerae, ATGATCAAG and CTTGATCAT are reverse complements of each other, resulting in the following six occurrences of these strings.

Finding a 9-mer that appears six times (either as itself or as its reverse complement) in a DNA string of length 500 is far more surprising than finding a 9-mer that appears three times (as itself). This observation leads us to the working hypothesis that ATGATCAAG and its reverse complement CTTGATCAT indeed represent DnaA boxes in Vibrio cholerae.

This computational conclusion makes sense biologically because the DnaA protein that binds to DnaA boxes and initiates replication does not care which of the two strands it binds to. Thus, for our purposes, both ATGATCAAG and CTTGATCAT represent DnaA boxes.

However, before concluding that we have found the DnaA box of Vibrio cholerae, the careful bioinformatician should check if there are other short regions in the Vibrio cholerae genome exhibiting multiple occurrences of ATGATCAAG (or CTTGATCAT). After all, maybe these strings occur as repeats throughout the entire Vibrio cholerae genome, rather than just in the oriC region. To this end, we need to solve the following problem :

Pattern Matching Problem

Pattern Matching Problem:
Find all occurrences of a pattern in a string.

Input: Strings *Pattern* and *Genome*.
Output: All starting positions in *Genome* where *Pattern* appears as a substring.

After solving the Pattern Matching Problem, we discover that ATGATCAAG appears 17 times in the following positions of the Vibrio cholerae genome:

116556, 149355, **151913**, **152013**, **152394**, 186189, 194276, 200076, 224527,
307692, 479770, 610980, 653338, 679985, 768828, 878903, 985368

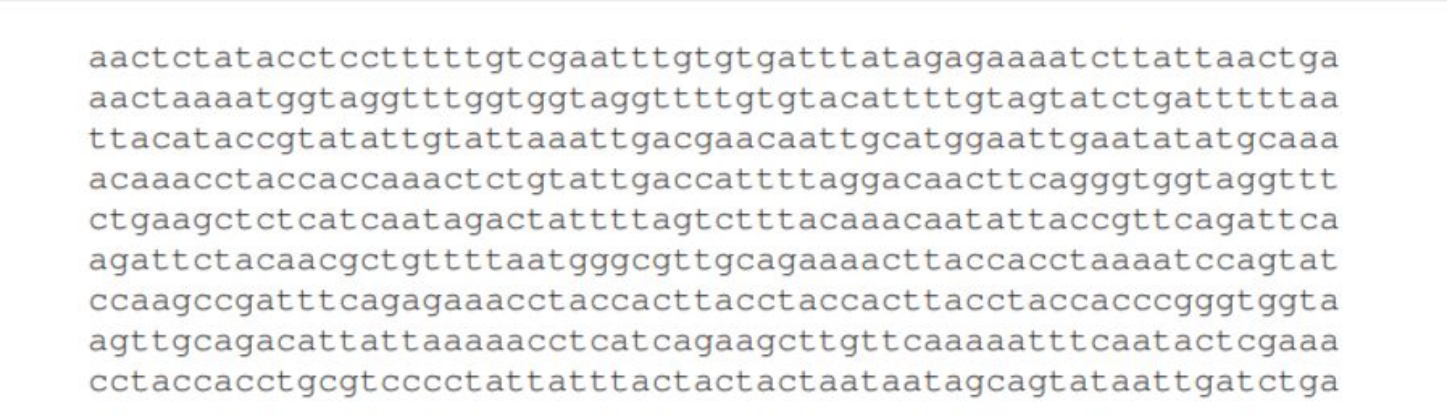
With the exception of the three occurrences of ATGATCAAG in oriC at starting positions 151913, 152013, and 152394, no other instances of ATGATCAAG form clumps, appear close to each other in a small region of the genome. You may check that the same conclusion is reached when searching for CTTGATCAT. **We now have strong statistical evidence that ATGATCAAG/CTTGATCAT may represent the hidden message to DnaA to start replication.**

An Explosion of Hidden Messages

We should not jump to the conclusion that ATGATCAAG/CTTGATCAT is a hidden message for all bacterial genomes without first checking whether it even appears in known oriC regions from other bacteria. After all, maybe the clumping effect of ATGATCAAG/CTTGATCAT in the oriC region of Vibrio cholerae is simply a statistical fluke that has nothing to do with replication. Or maybe different bacteria have different DnaA boxes . . . So

Let's check the proposed oriC region of Thermotoga petrophila

Thermotoga petrophila



This region does not contain a single occurrence of ATGATCAAG or CTTGATCAT! Thus, different bacteria may use different DnaA boxes as "hidden messages" to the DnaA protein. Application of the Frequent Words Problem to the oriC region above reveals that the following six 9-mers appear in this region three or more times:

AACCTACCA	AAACCTACC	ACCTACCAC
CCTACCACC	GGTAGGTTT	TGGTAGGTT

Something peculiar must be happening because it is extremely unlikely that six different 9-mers will occur so frequently within a short region in a random string. We will cheat a little and consult with Ori-Finder, a software tool for finding replication origins in DNA sequences. This software chooses CCTACCACC (along with its reverse complement GGTGGTAGG) as a working hypothesis for the DnaA box in Thermotoga petrophila. Together, these two complementary 9-mers appear five times in the replication origin:

aactctatacctccttttttgtcgaatttgtgtgatttatagagaaaatcttattaactga
aactaaaatggtaggtttGGTGGTAGGttttgtgtacattttgtagtatctgatttttaa
ttacataccgtatattgtattaaattgacgaacaattgcatggaattgaatatatgcaaa
acaaaCCTACCACCaaactctgtattgaccattttaggacaacttcagGGTGGTAGGttt
ctgaagctctcatcaatagactattttagtctttacaaacaatattaccgttcagattca
agattctacaacgctgttttaatgggcgttgcagaaaacttaccacctaaaatccagtat
ccaagccgatttcagagaaacctaccacttacctaccacttaCCTACCACCcgggtggta
agttgcagacattattaaaaacctcatcagaagcttgttcaaaaatttcaatactcgaaa
CCTACCACCtgcgtcccctattatttactactactaataatagcagtataattgatctga

The Clump Finding Problem

Now imagine that you are trying to find oriC in a newly sequenced bacterial genome. Searching for "clumps" of ATGATCAAG/CTTGATCAT or CCTACCACC/GGTGGTAGG is unlikely to help, since this new genome may use a completely different hidden message!

Before we lose all hope, let's change our computational focus: instead of finding clumps of a specific k-mer, let's try to find every k-mer that forms a clump in the genome. Hopefully, the locations of these clumps will shed light on the location of oriC.

The Clump Finding Problem

Our plan is to slide a window of fixed length L along the genome, looking for a region where a k-mer appears several times in short succession. The parameter value L = 500 reflects the typical length of oriC in bacterial genomes.

Definition

We defined a k-mer as a "clump" if it appears many times within a short interval of the genome. More formally, given integers L and t, a k-mer Pattern forms an (L, t)-clump inside a (longer) string Genome if there is an interval of Genome of length L in which this k-mer appears at least t times. For example, TGCA forms a (25, 3)-clump in the following Genome:

gatcagcataagggtccC**TGCA**A**TGCA**TGACAAGCC**TGCA**GTtgttttac

The Clump Finding Problem

We're now ready to formula this problem

Clump Finding Problem:
Find patterns forming clumps in a string.

Input: A string *Genome*, and integers k, L, and t.
Output: All distinct k-mers forming (L, t)-clumps in *Genome*.

Let's look for clumps in the Escherichia coli (E. coli) genome, the workhorse of bacterial genomics. We find hundreds of different 9-mers forming (500, 3)-clumps in the E. coli 15 genome, and it is absolutely unclear which of these 9-mers might represent a DnaA box in the bacterium's oriC region.

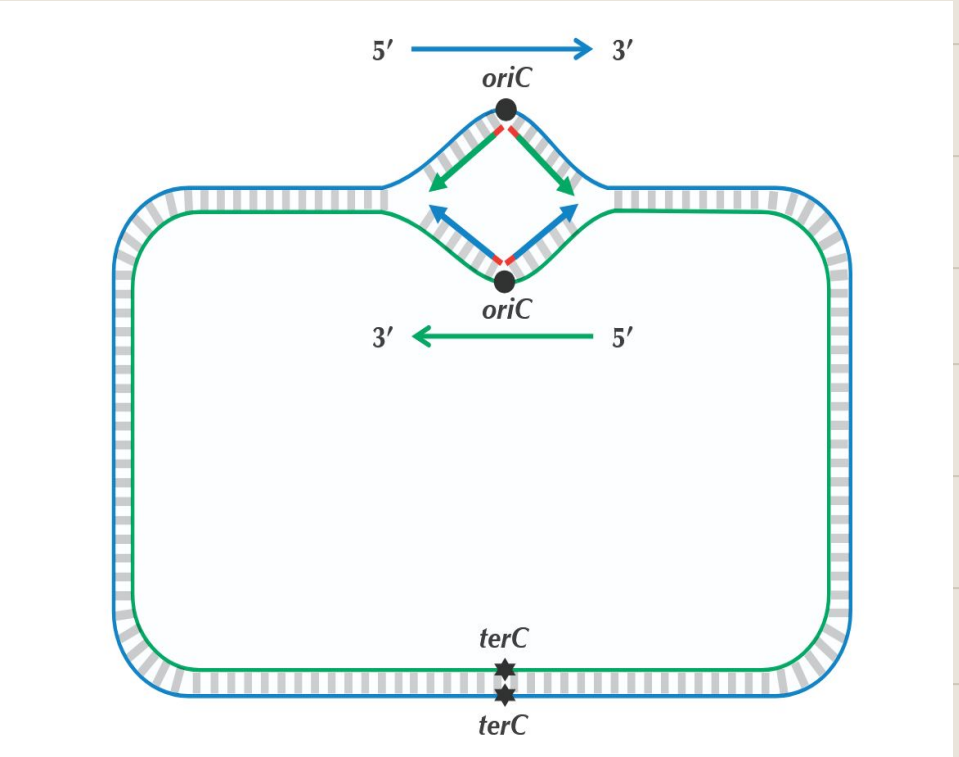
At this point, an unseasoned researcher might give up, since it appears that we do not have enough information to locate oriC in E. coli. But a fearless veteran bioinformatician would try to learn more about the details of replication in the hope that they provide new algorithmic insights into finding oriC :

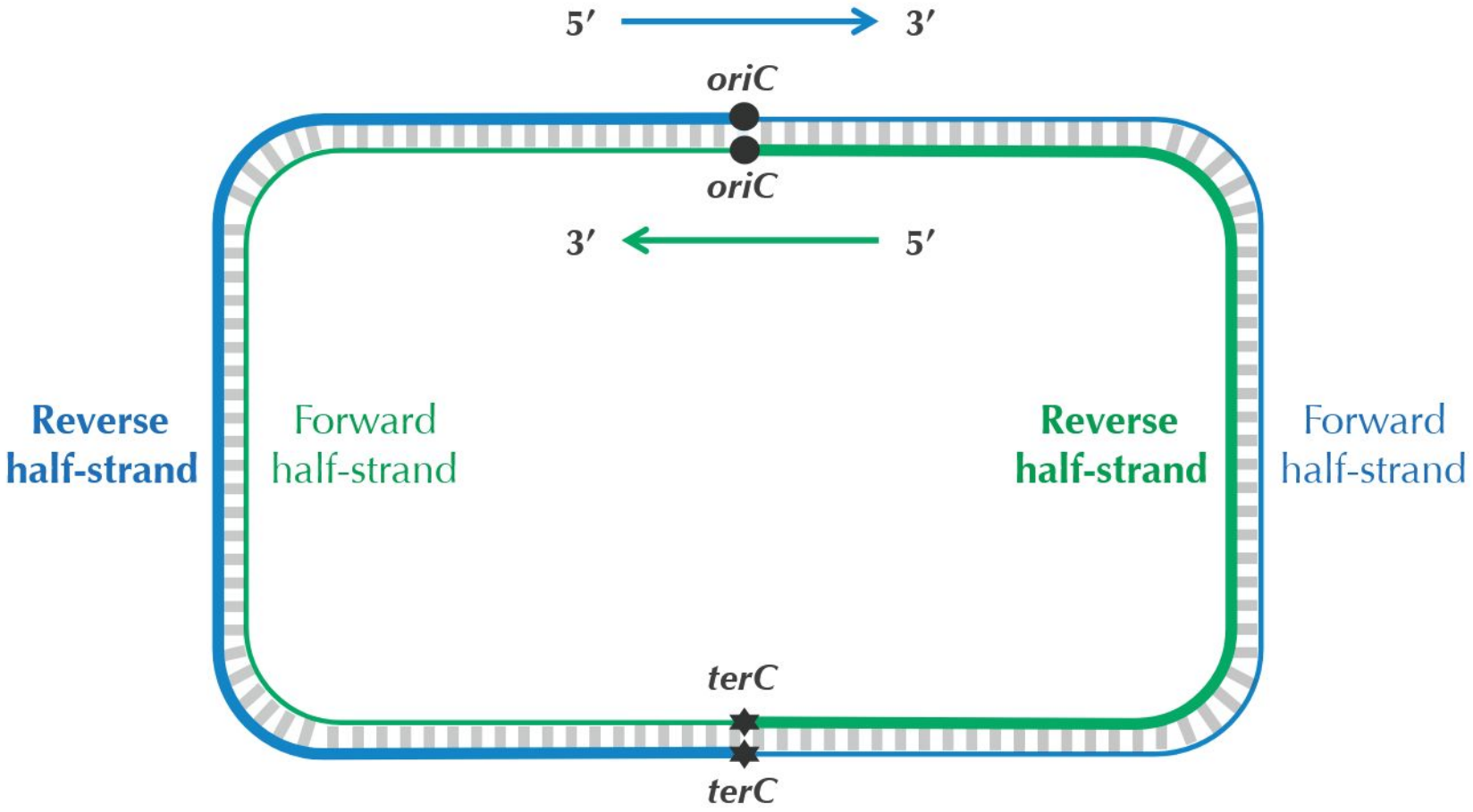
The Simplest Way to Replicate DNA

two complementary DNA strands running in opposite directions around a circular chromosome unravel, starting at oriC. As the strands unwind,they create two replication forks, which expand in both directions around the chromosome until the strands completely separate at the replication terminus (denoted terC). The replication terminus is located roughly opposite to oriC in the chromosome

An important thing to know about replication is that a DNA polymerase does not wait for the two parent strands to completely separate before initiating replication; instead, it starts copying while the strands are unraveling.

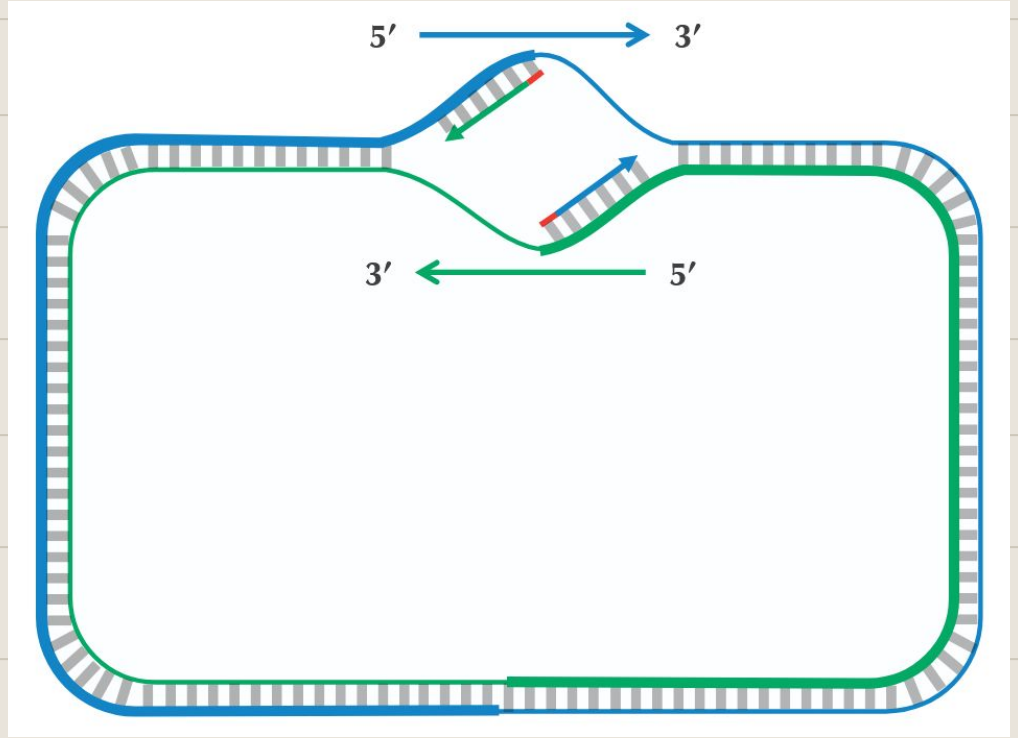
Thus, just four DNA polymerases, each responsible for one half-strand, can all start at oriC and replicate the entire chromosome.To start replication, a DNA polymerase needs a primer, a short complementary segment that binds to the parent strand and jump starts the DNA polymerase.




5′
3′
oriC
oriC
3′
5′
Reverse half-strand
Forward half-strand
Reverse half-strand
Forward half-strand
terC
terC

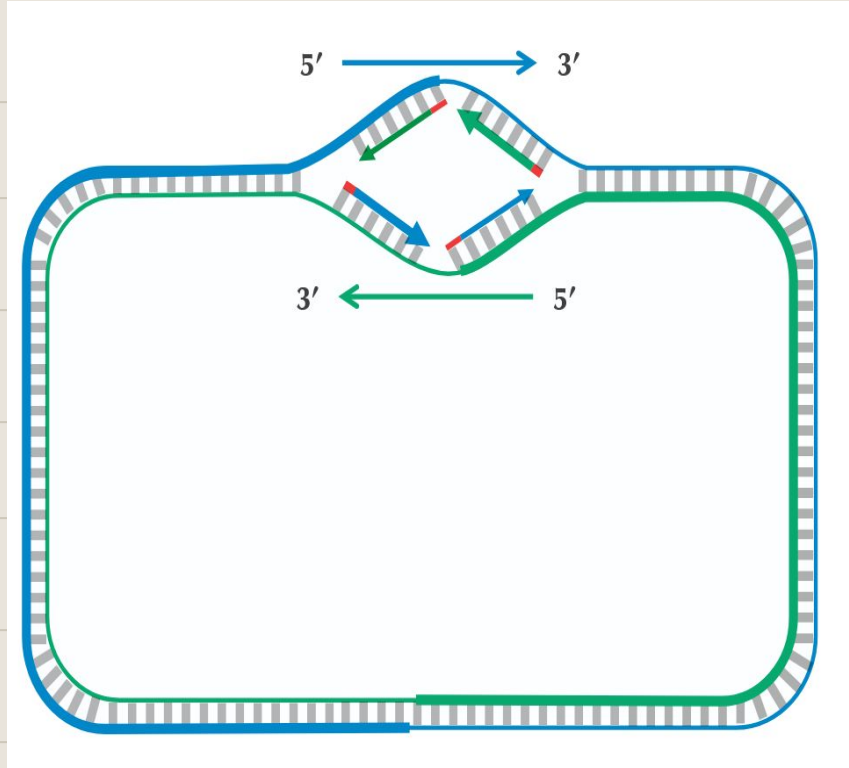
You might first think, "Well, it's simple —> the oriC region opens up, and then four DNA polymerases start moving as shown below, synthesizing the complementary strands."
However, the important point to note is that DNA polymerase has a special property: it is **unidirectional** and can only move **in the 3' → 5' direction** along the template strand. It **cannot** move in the **5' → 3' direction**

The book explains that if we move from *oriC* to *terC*, there are **two half-strands** oriented in the **5' → 3' direction**, called the **forward half-strands** (shown as **thin green and blue lines** in the figure). The **other two half-strands** are oriented in the **3' → 5' direction** and are called the **reverse half-strands** (shown as **thick lines** in the figure).



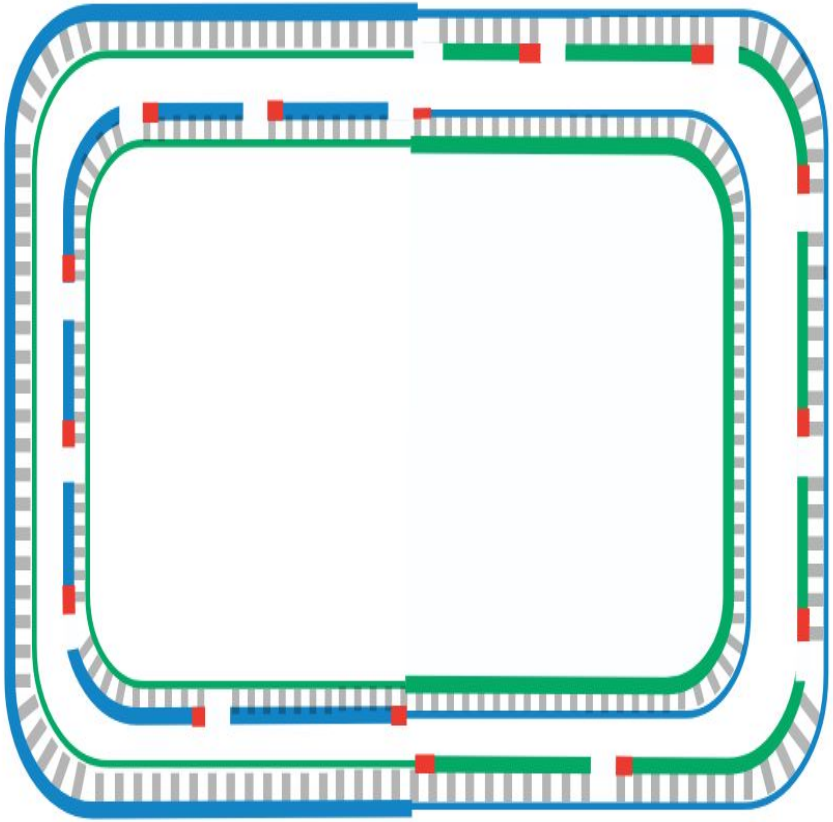
Then the book says that DNA polymerase can move smoothly along the reverse half-strands, synthesizing DNA continuously. However, it encounters a problem on the **forward half-strands**, because their direction runs from 5′ to 3′, which is opposite to the direction DNA polymerase can move

On these forward half-strands, the polymerase must wait until the replication forks open up a bit (about 2,000 nucleotides) before it can start synthesizing new DNA from a primer that forms at the fork. Then, it moves backward toward oriC while building the complementary strand in the 3′ → 5′ direction along the template.



As a result, DNA on these forward half-strands is synthesized in **small fragments**, known as **Okazaki fragments**. In contrast, DNA on the **reverse half-strands** is synthesized **continuously**.

Finally, an enzyme called **DNA ligase** joins the Okazaki fragments together, forming **two complete daughter chromosomes**, each consisting of **one original parent strand** and **one newly synthesized strand**.



Deamination!!!

Now, what good does it do us to know this?

So, we saw that DNA polymerase quickly synthesizes the complementary strand on the lagging strand, but it faces delays on the leading strand. Now, we intend to use this asymmetry in DNA replication to design a new algorithm for finding the replication origin region.

But how can this asymmetry help us find the replication origin region?
Consider this: since replication occurs faster on the lagging strand, this part of the DNA spends most of its time in a double-stranded form. In contrast, the leading strand spends a significant portion of its lifetime as a single strand because it must wait for the .DNA to unwind
This difference is important because single-stranded DNA has a much higher mutation rate than double-stranded DNA. In particular, if one of the four nucleotides in single-stranded DNA is more prone to mutation than the others, we should expect to see a lower frequency of that nucleotide on the leading strand

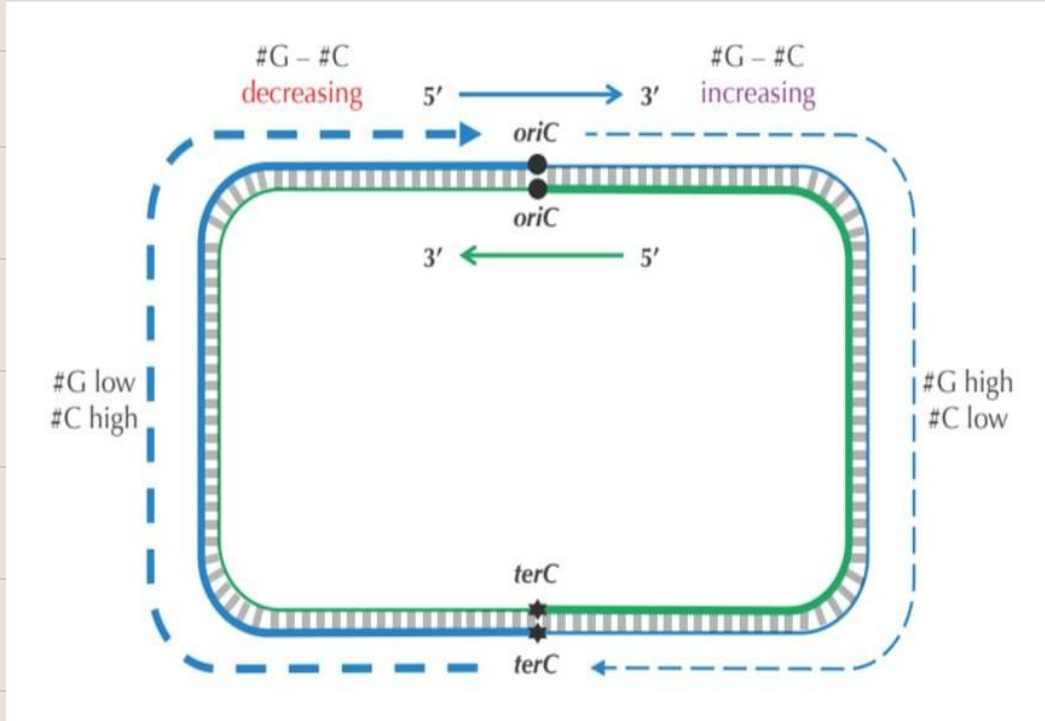
In the table below, if you look closely, the number of nucleotides in each strand is shown, and the frequencies of A and T in the two strands are approximately equal. However, the amount of C in the reverse strand is higher than in the forward strand, while the nucleotide G in the reverse strand is lower than in the forward strand.

	#C	#G	#A	#T
Entire strand	427419	413241	491488	491363
Reverse half-strand	219518	201634	243963	246641
Forward half-strand	207901	211607	247525	244722
Difference	+11617	-9973	-3562	+1919

FIGURE 1.10 Counting nucleotides in the *Thermotoga petrophila* genome on the forward and reverse half-strands.

because cytosine (C) has a tendency to mutate into thymine (T) through a process called deamination. Deamination rates rise 100-fold when DNA is single-stranded, which leads to a decrease in cytosine on the forward half-strand, thus forming mismatched base pairs T-G. These mismatched pairs can further mutate into T-A pairs when the bond is repaired in the next round of replication, which accounts for the observed decrease in guanine (G) on the reverse half-strand When the number of C nucleotides decreases in the forward strand, the number of G nucleotides also decreases in its complementary (reverse) strand.

This happens because DNA-polymerase synthesizes the complementary strand based on the template strand. Therefore, When there are fewer C nucleotides in the forward strand, DNA polymerase does not add as many G nucleotides to the reverse strand. As a result, the forward strand has fewer C's, and the reverse strand has fewer G's.



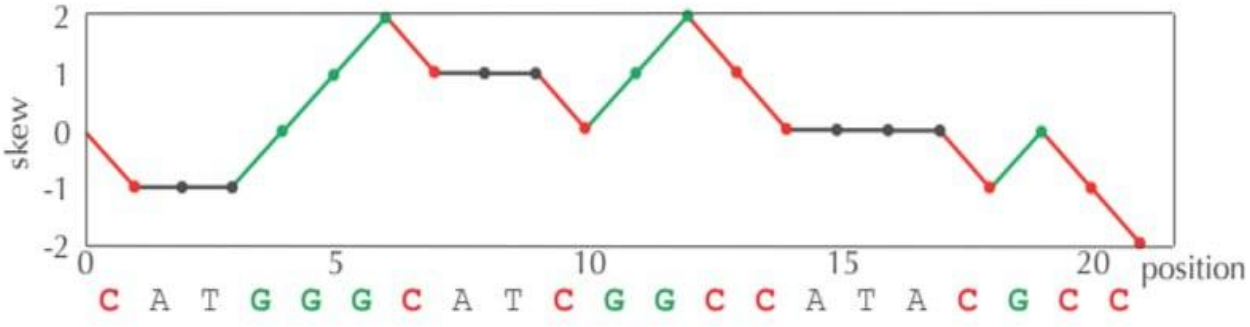
When the number of C nucleotides decreases in the forward strand, the number of G nucleotides also decreases in its complementary (reverse) strand. This happens because DNA polymerase synthesizes the complementary strand based on the template strand. Therefore, when there are fewer C nucleotides in the forward strand, DNA polymerase does not add as many G nucleotides to the reverse strand. As a result, the forward strand has fewer C's, and the reverse strand has fewer G's.

Let's see if we can take advantage of these peculiar statistics caused by deamination to locate oriC.as take below illustrates, the difference between the total amount of guanine and the total amount of cytosine is negative on the reverse half-strand (211607 207901 = 3706) and positive on the forward half-strand (201634 219518 =17884).

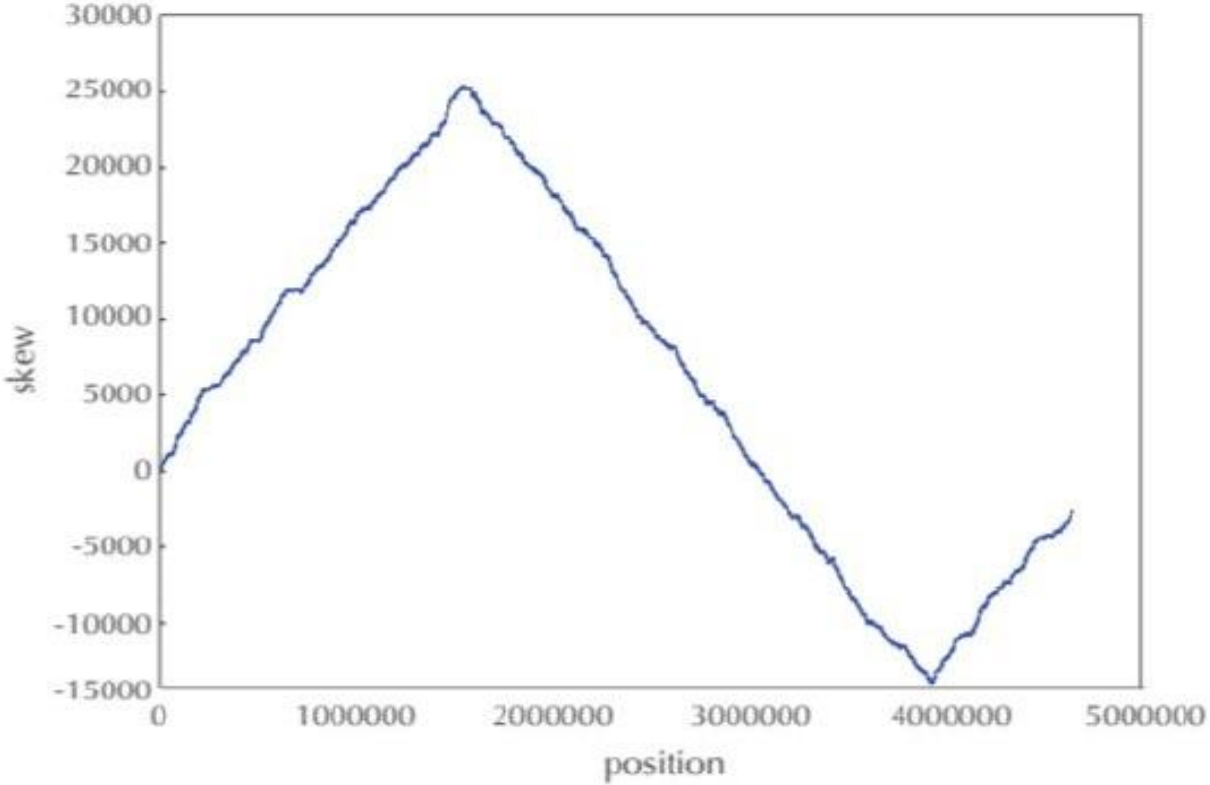
Thus, our idea is to traverse the genome, keeping a running total of the difference between the counts of G and C. If this difference starts increasing, then we guess that we are on the forward half-strand; on the other hand, if this difference starts decreasing, then we guess that we are on the reverse half-strand.

Since we don't know the location of oriC in a circular genome, let's linearize it (select an arbitrary position and pretend that the genome begins here), resulting in a linear string Genome. We define SKEW-i-(Genome) as the difference between the total number of occurrences of G and the total number of occurrences of C in the first i nucleotides
of Genome. The skew diagram is defined by plotting SKEW-i-(Genome) as i ranges from 0 to |Genome|. Figure below shows a skew diagram for a short DNA string.

SKEW-i-(genom)=#G-#C



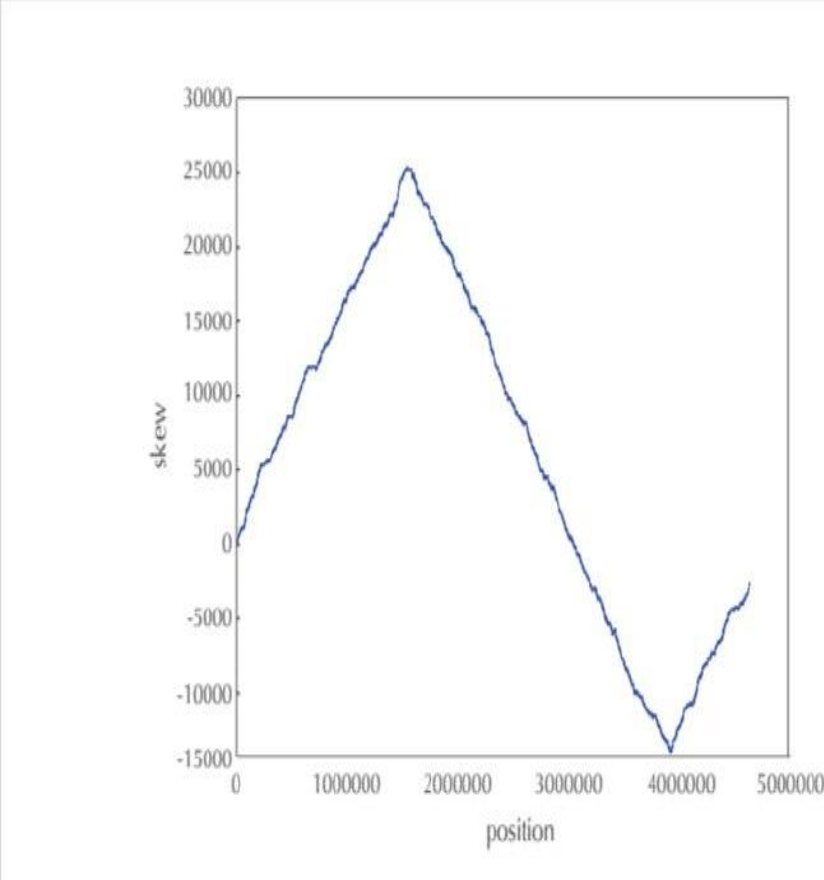
the skew diagram for a linearized E. coli genome



Let's follow the 5´—>3´ direction of DNA and walk along the chromosome from terC to oriC, then continue on from oriC to terC. In page 45, we saw that the skew is decreasing along the reverse half-strand and increasing along the forward half-strand.

Thus, the skew should achieve a minimum at the position where the reverse half-strand ends and the forward
half-strand begins, which is exactly the location of oriC!

We have just developed an algorithm for locating **oriC**: **it should be found where the skew attains a minimum**



So Solving the Minimum Skew Problem now provides us with an approximate location of oriC at position 3923620 in E. coli.

To ensure the correctness of this result let's look for a hidden message representing a potential DnaA box near this location. Solving the Frequent Words Problem in a window of length 500 starting at position 3923620 (shown below) but we cannot find any 9-mers (along with their reverse complements) that appear three or more times!

Even if we have located oriC in E. coli, it appears that we still have not found the DnaA boxes that jump-start replication in this bacterium !

SO WHAT SHOULD WE DO ??

```
aatgatgatgacgtcaaaaggatccggataaaacatggtgattgcctcgcataacgcggt
atgaaaatggattgaagcccgggccgtggattctactcaactttgtcggcttgagaaaga
cctgggatcctgggtattaaaaagaagatctatttatttagagatctgttctattgtgat
ctcttattaggatcgcactgccctgtggataacaaggatccggcttttaagatcaacaac
ctggaaaggatcattaactgtgaatgatcggtgatcctggaccgtataagctgggatcag
aatgaggggttatacacaactcaaaaactgaacaacagttgttctttggataactaccgg
ttgatccaagcttcctgacagagttatccacagtagatcgcacgatctgtatacttattt
gagtaaattaacccacgatcccagccattcttctgccggatcttccggaatgtcgtgatc
aagaatgttgatcttcagtg
```

DO NOT GIVE UP!

let's examine the oriC of *Vibrio cholerae* one more time to see if it provides us with any insights on how to alter our algorithm to find DnaA boxes in E.coli.
You may have noticed that in addition to the three occurrences of ATGATCAAG and three occurrences of its reverse complement CTTGATCAT, the Vibrio cholerae oriC contains additional occurrences of ATGATCAAC and CATGATCAT, which differ from ATGATCAAG and CTTGATCAT in only a single nucleotide

atcaATGATCAACgtaagcttctaagcATGATCAAGgtgctcacacagtttatccacaac
ctgagtggatgacatcaagataggtcgttgtatctccttcctctcgtactctcatgacca
cggaaagATGATCAAGagaggatgatttcttggccatatcgcaatgaatacttgtgactt
gtgcttccaattgacatcttcagcgccatattgcgctggccaaggtgacggagcgggatt
acgaaagCATGATCATggctgttgttctgtttatcttgttttgactgagacttgttagga
tagacggttttcatcactgactagccaaagccttactctgcctgacatcgaccgtaaat
tgataatgaatttacatgcttccgcgacgatttacctCTTGATCATcgatccgattgaag
atcttcaattgttaattctcttgcctcgactcatagccatgatgagctCTTGATCATgtt
tccttaaccctctatttttacggaagaATGATCAAGctgctgctCTTGATCATcgtttc

Finding eight approximate occurrences of our target 9-mer and its reverse complement in a short region is even more statistically surprising than finding the six exact occurrences of ATGATCAAG and its reverse complement CTTGATCAT that we stumbled upon in the beginning of our investigation. Furthermore, the discovery of these approximate 9-mers makes sense biologically, since DnaA can bind not only to "perfect" DnaA boxes but to their slight variations as well.

We say that position i in k-mers p1 · · · pk and q1 · · · qk is a mismatch if p(i) not equal to q(i). The number of mismatches between strings p and q is called the *Hamming distance* between these strings and is denoted **HAMMING_DISTANCE(p, q)**

Hamming Distance Problem:

Compute the Hamming distance between two strings.

Input: Two strings of equal length.

Output: The Hamming distance between these strings.

We say that a *k*-mer appears in a sequence with at most *d* mismatches if there exists a substring in the text that differs from the pattern in no more than *d* positions.
HAMMINGDISTANCE(Pattern, Pattern')<d

Input: Strings *Pattern* and *Text* along with an integer *d*.
Output: All starting positions where *Pattern* appears as a substring of *Text* with at most *d* mismatches.

We then define a function called **COUNT-d-(Text, Pattern)**, which represents the number of times a pattern appears in the text with at most *d* differences.
For example:

$$\text{COUNT}_1(\text{AACAAGCATAAACATTAAAGAG}, \text{AAAAA}) = 4$$

because *AAAAA* appears four times in this string with at most one mismatch: *AACAA*, *ATAAA*, *AAACA*, and *AAAGA*. Notice that two of these occurrences overlap.

Computing COUNT-d-(Text, Pattern) simply requires us to compute the Hamming distance between Pattern and every k-mer substring of Text, as follows

```
ApproximatePatternCount(Text, Pattern, d)
    count ← 0
    for i ← 0 to |Text| – |Pattern|
        Pattern' ← Text(i, |Pattern|)
        if HammingDistance(Pattern, Pattern') ≤ d
            count ← count + 1
    return count
```

Now, the problem we are addressing is as follows:
We are given an input **Text**, along with the integers **d** and **k**. Our goal is to find all the most frequent *k*-mers in **Text** that appear with at most **d** mismatches.

Frequent Words with Mismatches Problem:
Find the most frequent k-mers with mismatches in a string.

Input: A string *Text* as well as integers k and d.
Output: All most frequent k-mers with up to d mismatches in *Text*.

We now redefine the Frequent Words Problem to account for both mismatches and reverse complements. Recall that Pattern refers to the reverse complement of Pattern.

Frequent Words with Mismatches and Reverse Complements Problem:
Find the most frequent k-mers (with mismatches and reverse complements) in a string.

Input: A DNA string *Text* as well as integers k and d.
Output: All k-mers *Pattern* that maximize the sum $\text{COUNT}_d(Text, Pattern) + \text{COUNT}_d(Text, \overline{Pattern})$ over all possible k-mers.

We now make a final attempt to find DnaA boxes in E. coli by finding the most frequent 9-mers with mismatches and reverse complements in the region suggested by the minimum skew as oriC

Let's cross our fingers and identify the most frequent 9-mers (with 1 mismatch and re-verse complements) within a window of length 500 starting at position 3923620 of the E.coli genome. Bingo! The experimentally confirmed DnaA box in E. coli (TTATCCACA) is a most frequent 9-mer with 1 mismatch, along with its reverse complement TGTGGATAA:

aatgatgatgacgtcaaaaggatccggataaaacatggtgattgcctcgcataacgcggt
atgaaaatggattgaagcccgggccgtggattctactcaactttgtcggcttgagaaaga
cctgggatcctgggtattaaaaagaagatctatttatttagagatctgttctattgtgat
ctcttattaggatcgcactgcccTGTGGATAAcaaggatccggcttttaagatcaacaac
ctggaaaggatcattaactgtgaatgatcggtgatcctggaccgtataagctgggatcag
aatgaggggTTATACACAactcaaaaactgaacaacagttgttcTTTGGATAActaccgg
ttgatccaagcttcctgacagagTTATCCACAgtagatcgcacgatctgtatacttattt
gagtaaattaacccacgatcccagccattcttctgccggatcttccggaatgtcgtgatc
aagaatgttgatcttcagtg

We were fortunate that the DnaA boxes of E. coli are captured in the window that we chose. Moreover, while TTATCCACA represents a most frequent 9-mer with 1 mismatch and reverse complements in this 500-nucleotide window, it is not the only one: GGATCCTGG, GATCCCAGC, GTTATCCAC, AGCTGGGAT, and CTGGGATCA also appear four times with 1 mismatch and reverse complements.

We do not know what purpose — if any — these other 9-mers serve in the E. coli genome, but we do know that there are many different types of hidden messages in genomes; these hidden messages have a tendency to cluster within a genome, and most of them have nothing to do with replication. One example is the regulatory DNA motifs responsible for gene expression that we will study in Chapter 2. The important lesson is that existing approaches to oriC prediction remain imperfect and sometimes inconclusive. However, even providing biologists with a small collection of 9-mers as candidate DnaA boxes is a great aid as long as one of these 9-mers is correct.
Thus, the moral of this chapter is that even though computational predictions can be powerful, bioinformaticians should collaborate with biologists to verify their computational predictions

The chapter emphasizes that computational predictions of *oriC* (replication origin) are useful but imperfect — they must be verified experimentally or improved through comparative genomics.
The text also notes that other bacteria, like *Thermotoga petrophila* or *Vibrio cholerae*, may have different or fewer DnaA boxes, making prediction harder. The skew diagrams of some species are irregular, meaning the oriC position is only approximately indicated by skew minima. Finally, readers are challenged to find DnaA boxes in *Salmonella enterica*

Open Problems

Multiple Replication Origins in a bacterial Genome : Do some bacteria naturally have *multiple* replication origins (oriC) instead of the usual single oriC? Multiple oriCs could speed replication in long genomes. Wang et al. (2011) showed a bacterium can replicate from two engineered oriCs, so the question is whether multiple oriCs occur naturally.

Finding replication origins in archaea : How can we reliably predict replication origins (oriCs) from genome sequence in archaea and eukaryotes (especially yeast)? Unlike many bacteria (where skew + DnaA-box works reasonably well), archaea and yeasts show much greater variety and complexity in origin signals.

Thanks for your attention