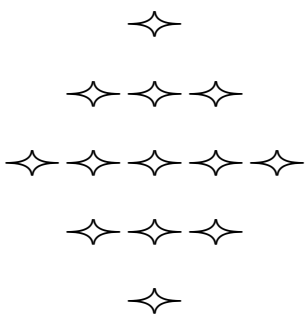
# Bioinformatics Algorithms

*— An Active-Learning Approach —*

Phillip Compeau
&
Pavel Pevzner



# Welcome!

Thank you for joining us! Please take a minute to look over the following suggestions on how to "read" this book.

We say "read" because we want it to be much more than a passive experience. First, you can find a **fully interactive** version of this book on Stepic. If you are following this PDF instead, then we would like to explain a few features of the text before you begin reading.

1. **CODE CHALLENGES** ask you to write programs to solve the problems that you encounter along the way. We have posted these challenges on the Rosalind website; click the Rosalind logo in the margin to solve them!

2. **EXERCISE BREAKS** are questions that test your understanding of the material. These questions are automatically graded on Stepic, where you can find additional exercises asking you to apply your algorithms to real data.


3. **STOP and Think** questions are invitations to slow down and contemplate the current material before continuing to the next topic.


4. **DETOUR** sections provide extra material that we like but didn't quite fit in the main text, and so we put it at the end. Click the "DETOUR" logo to learn this extra material.

We will use *Bioinformatics Algorithms* in our upcoming online class on Coursera. In this class, you'll be able to participate in open research projects with leading bioinformaticians!

In the next few pages, you can meet us, along with the directors of the open research projects and the development team who have helped us make this textbook happen.

# Contents

# About this Textbook

### Meet the Authors

Phillip Compeau is a Ph.D. candidate in the University of Calfornia, San Diego (UCSD) Department of Mathematics. He holds Master's degrees from Cambridge University and UCSD in addition to a B.S. from Davidson College. He researches the combinatorics of genome rearrangements but is passionate about the future of education, having co-founded Rosalind with Nikolay Vyahhi in 2012. He is a retired tennis player and dreams of one day going pro in golf.

Pavel Pevzner is Ronald R. Taylor Chair of Computer Science at UCSD. He holds a Ph.D. from Moscow Institute of Physics and Technology, Russia. He was named Howard Hughes Medical Institute Professor in 2006; elected an Association for Computing Machinery Fellow in 2010 for "contribution to algorithms for genome rearrangements, DNA sequencing, and proteomics"; and named an International Society for Computational Biology Fellow in 2012. He has authored two other textbooks: *Computational Molecular Biology: An Algorithmic Approach* (2000) and *An Introduction to Bioinformatics Algorithms* (2004) (jointly with Neil Jones).

## Meet the Research Directors

Mikhail Gelfand graduated from the Moscow State University and has since worked in various bioinformatics fields. He currently is studying the coevolution of transcription factors and their recognition sites, alternative splicing, and bacterial genome evolution. He is the Associate Director for Science of the Institute for Information Transmission Problems of the Russian Academy of Sciences and Professor at the Moscow State University. He was elected to Academia Europaea in 2010.

Uri Keich graduated from New York University with a Ph.D. in mathematics. He held professorships at University of California, Riverside and Cornell University before joining the faculty at University of Sydney, where he teaches the “Statistical Methods in Bioinformatics” course. His research is in regulatory motif finding, computational proteomics, and the study of replication origins. He recently co-authored a paper characterizing replication origins in yeast (Liachko et al., 2013).

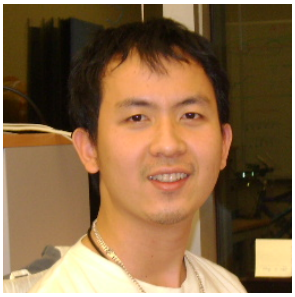
Glenn Tesler graduated from the Massachusetts Institute of Technology with a Ph.D. in Mathematics, in the area of Algebraic Combinatorics. He now holds a professorship in the Department of Mathematics at the University of California, San Diego, where his courses include “Statistical Methods in Bioinformatics." His research includes sequence assembly, genome rearrangements, and genome-wide association studies. He was the conference chair and program chair of the RECOMB Satellite Workshop on Comparative Genomics 2007, and program chair of the RECOMB Satellite Workshop on Open Problems in Algorithmic Biology 2012.

## Meet the Development Team

Nikolay Vyahhi coordinates the M.S. Program in Bioinformatics in the Academic University of St. Petersburg, Russian Academy of Sciences. In 2012, he cofounded the Rosalind online bioinformatics education project. He recently founded the Bioinformatics Institute in St. Petersburg as well as Stepic, a company focusing on content delivery for online education.

Laurence Bernstein graduated from UCSD with a degree in Computer Science. He spent 25 years working in the private sector, including 10 years running his own software development firm. Recently he has returned to UCSD to pursue a Ph.D. in Bioinformatics and Systems Biology. His thesis work involves creation of novel algorithms in the field of computational proteomics.

Olga Botvinnik is a Ph.D. candidate in Bioinformatics and Systems Biology at UCSD. She holds an M.S. in Bioinformatics from University of California, Santa Cruz and B.S. degrees in Mathematics and Biological Engineering from the Massachusetts Institute of Technology. Her research interests are data visualization and single-cell genomics. Olga enjoys yoga, photography, and playing cello.



Son Pham is a Ph.D. candidate in Computer Science and Engineering at UCSD. He holds an M.S. in Applied Mathematics from St. Petersburg State University, Russia. His research interests include graph theory, genome assembly, and comparative genomics. Besides research, he enjoys walking meditation, gardening, and trying to catch the big one.

Kai Zhang is a Ph.D. candidate in Bioinformatics and Systems Biology at UCSD. He holds an M.S. in Molecular Biology from Xiamen University, China. His research interests include epigenetics, gene regulatory networks, and machine learning algorithms. Besides research, Kai likes basketball and music.

**Acknowledgements**

The authors would like to thank the development team, who implemented coding challenges, rendered figures, and offered insightful feedback on the text.

The book was greatly improved by the efforts of Glenn Tesler who provided thorough chapter reviews, caught many errors, and even implemented some software to catch errors in the early version of this manuscript! The time and effort that he put into the task was truly gratifying.

The authors would also like to thank Glenn Tesler, Robin Betz, James Jensen, and Yu Lin for providing insightful comments on the manuscript.

Finally, they would like to thank the Stepic team for their continued support of this text in interactive form on their site: Nikolay Vyahhi, Andrey Balandin, Artem Suschev, Aleksey Kladov, and Kirill Shikhanov.

# 1

# Where Does DNA Replication Begin?

*Algorithmic Warmup*

### Introduction to DNA Replication

**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. 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.*

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, as shown in Figure 1.

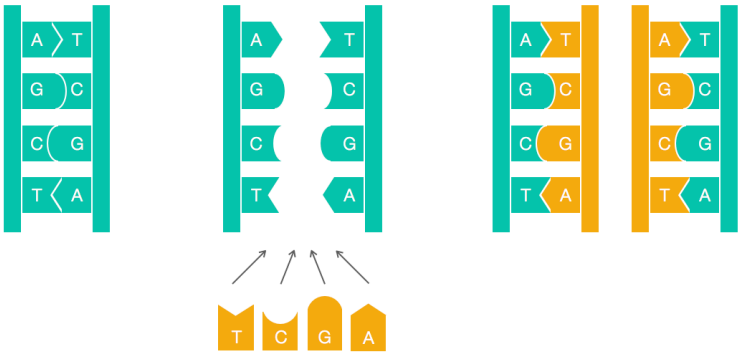



**Figure 1:** A naive view of DNA replication. Nucleotides `A` and `T` are complements of each other, as are `C` and `G`. Complementary nucleotides bind to each other in DNA.

Although Figure 1 models DNA replication on a simple level, the details of replication turned out to be much more intricate than Watson and Crick imagined; as we will see, an astounding amount of molecular logistics is required to ensure DNA replication. At first glance, a computer scientist might not imagine that these details have any computational relevance: to mimic the process in Figure 1 algorithmically, we only need to take a string representing the genome and return a copy of it! Yet if we take the time to review the underlying biological process, we will be rewarded with new algorithmic insights into analyzing replication.

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. For example, some gene therapy methods use genetically engineered mini-genomes, which are called **viral vectors** because they are able to penetrate cell walls (just like real viruses). Viral vectors carrying artificial genes have been widely used in agriculture, such as to engineer frost-resistant tomatoes and pesticide-resistant corn. In 1990, gene therapy was first successfully performed on humans when it saved the life of a four-year-old girl suffering from Severe Combined Immunodeficiency Disorder; the girl had been so vulnerable to infections that she was forced to live in a sterile environment.

The idea of gene therapy is to intentionally infect a patient who lacks a crucial gene with a viral vector containing an artificial gene that encodes a therapeutic protein. Once inside the cell, the vector replicates and eventually produces many molecules of the therapeutic protein, which in turn treats the patient's disease. To ensure that the vector actually does replicate inside the cell, biologists must know where *oriC* is in the vector's genome and ensure that the genetic manipulations that they perform do not affect it.

In the following problem, we assume that a genome has a single *oriC*.

---

**Finding *oriC* Problem**:

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

---


**STOP and Think:** Does this biological problem represent a clearly stated computational problem?

Although the Finding *oriC* 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, deleting various short segments from the genome and eventually finding a segment whose deletion stops replication. Computer scientists, on the other hand, shake their heads and demand more information before they can even start thinking about the problem.

Why should biologists care what computer scientists think? For many questions in modern biology, ***in silico*** (i.e., computational) methods are now the only realistic way to answer questions because they are often much faster than experimental approaches and because 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 an *in silico* approach to finding *oriC* so that biologists are free to spend time and money on other tasks.

**CENTRAL QUESTION**: How can we use a computer to locate the replication origin of a genome, knowing *only* the genome's nucleotide sequence?

### Hidden Messages in the Replication Origin

In the rest of this chapter, 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 an example of a bacterium in which *oriC* is known, and then determine what makes this genomic region special in order to design an *in silico* 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*:

```
atcaatgatcaacgtaagcttctaagcatgatcaaggtgctcacacagtttatccacaac
ctgagtggatgacatcaagataggtcgttgtatctccttcctctcgtactctcatgacca
cggaaagatgatcaagagaggatgatttcttggccatatcgcaatgaatacttgtgactt
gtgcttccaattgacatcttcagcgccatattgcgctggccaaggtgacggagcgggatt
acgaaagcatgatcatggctgttgttctgtttatcttgttttgactgagacttgttagga
tagacggtttttcatcactgactagccaaagccttactctgcctgacatcgaccgtaaat
tgataatgaatttacatgcttccgcgacgatttacctcttgatcatcgatccgattgaag
atcttcaattgttaattctcttgcctcgactcatagccatgatgagctcttgatcatgtt
tccttaaccctctatttttacggaagaatgatcaagctgctgctcttgatcatcgtttc
```

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 nucleo-

tides? 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 *DnaA*: "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**:
*Find a "hidden message" in oriC.*

**Input**: A string *Text* (representing *oriC* in a genome).
**Output**: A hidden message in *Text*.

---


**STOP and Think:** Does this problem represent a clearly stated computational problem? If not, specify what is unclear.

Although the Hidden Message Problem poses a legitimate intuitive question, it again makes absolutely no sense to a computer scientist because the notion of a "hidden message" is not precisely defined. 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;+?;
```

Upon seeing the paper, the narrator remarks, "Were all the jewels of Golconda awaiting me upon my solution of this enigma, I am quite sure that I should be unable to earn them". Legrand retorts, "It may well be doubted whether human ingenuity can construct an enigma of the kind which human ingenuity may not, by proper application, resolve". He reasons that the three consecutive symbols **;48** appear with surprising frequency on the parchment.

```
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**. Substituting each symbol, Legrand had a slightly easier text to decipher, which would eventually lead him to the buried treasure:

```
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
```

**EXERCISE BREAK:** Decode the entire pirate message.

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 strings of nucleotides often appear surprisingly frequently in small regions of the genome.

For example, **ACTAT** is a surprisingly frequent substring of

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

We define COUNT(*Text*, *Pattern*) as the number of times that a *k*-mer *Pattern* appears as a substring of *Text*. Following the above example,

COUNT(ACA**ACTAT**GCAT**ACTAT**CGGGA**ACTAT**CCT, **ACTAT**) = 3

Note that COUNT(CG**ATATA**TCC**ATA**G, **ATA**) is equal to 3 (not 2) since we should account for overlapping occurrences of *Pattern* in *Text*. say that *Pattern* is a **most frequent *k*-mer** in *Text* if it maximizes COUNT(*Text*, *Pattern*) among all *k*-mers. You can see that **ACTAT** is a most frequent 5-mer of ACA**ACTAT**GCAT**ACTAT**CGGGA**ACTAT**CCT, and **ATA** is a most frequent 3-mer of CG**ATATA**TCC**ATA**G.


**STOP and Think:** Is it possible for a string to have multiple most frequent *k*-mers?

We now have a rigorously defined computational problem:

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

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

A straightforward algorithm for finding the most frequent words in a string *Text* checks all $k$-mers appearing in this string (there are $|Text| - k + 1$ $k$-mers in *Text*, where $|Text|$ denotes the length of *Text*) and then computes how many times each $k$-mer appears in *Text*. To compute how many times a given $k$-mer appears in *Text*, one can check whether it appears in starting position 0, starting position 1, and so on. Since each $k$-mer requires $|Text| - k + 1$ such checks, each one requiring as many as $k$ comparisons, the overall number of steps of this algorithm is at most $(|Text| - k + 1) \cdot (|Text| - k + 1) \cdot k$. To simplify the matter, computer scientists often say that the runtime of this algorithm has an upper bound of $|Text|^2 \cdot k$ steps and refer to the **complexity** of this algorithm as $O(|Text|^2 \cdot k)$ (see **DETOUR: Big-O Notation**).

**STOP and Think:** The simple algorithm described above is not the most efficient way to solve the Frequent Words Problem. Most students solve this problem in one of five possible ways, having runtimes that can be roughly estimated as $|Text|^2 \cdot k$, $4^k + |Text| \cdot k$, $|Text| \cdot k \cdot \log(|Text|)$, $|Text| \cdot k$, and $|Text|$. What is the runtime of your solution, and how does it compare with these four approaches?

Table 1 reveals the most frequent $k$-mers in the *oriC* region from *Vibrio cholerae*.

| **k** | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|
| **count** | 26 | 12 | 8 | 8 | 5 | 4 | 3 |
| **k-mers** | tga | atga | gatca | tgatca | atgatca | atgatcaa | atgatcaag |
| | | tgat | tgatc | | | | cttgatcat |
| | | | | | | | tcttgatca |
| | | | | | | | ctcttgatc |

**Table 1:** The most frequent $k$-mers in the *oriC* region of *Vibrio cholerae* for $k$ from 3 to 9, along with the number of times each $k$-mer occurs.

**STOP and Think:** Do any of the counts in Table 1 seem surprisingly large?

For example, the 9-mer **ATGATCAAG** appears three times in the *oriC* region of *Vibrio cholerae* — is it surprising?

atcaatgatcaacgtaagcttctaagc**ATGATCAAG**gtgctcacacagtttatccacaac
ctgagtggatgacatcaagataggtcgttgtatctccttcctctcgtactctcatgacca
cggaaag**ATGATCAAG**agaggatgatttcttggccatatcgcaatgaatacttgtgactt
gtgcttccaattgacatcttcagcgccatattgcgctggccaaggtgacggagcgggatt
acgaaagcatgatcatggctgttgttctgtttatcttgttttgactgagacttgttagga
tagacggtttttcatcactgactagccaaagccttactctgcctgacatcgaccgtaaat
tgataatgaatttacatgcttccgcgacgatttacctcttgatcatcgatccgattgaag
atcttcaattgttaattctcttgcctcgactcatagccatgatgagctcttgatcatgtt
tccttaaccctctatttttacggaaga**ATGATCAAG**ctgctgctcttgatcatcgtttc

We highlight a most frequent 9-mer instead of using some other value of $k$ because bacterial *DnaA* boxes are usually 9 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 (see **DETOUR: Probabilities of Patterns in a String**). 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?


**STOP and Think:** Is any one of the four most frequent 9-mers in the *oriC* of *Vibrio cholerae* "more surprising" than the others?

### 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 as shown in Figure 1, one can imagine the synthesis of a **complementary strand** on a **template strand**, a model for replication that was confirmed by Meselson and Stahl in 1958 (see **DETOUR: The Most Beautiful Experiment in Biology**). Figure 2 shows a template strand **AGTCGCATAGT** and its complementary strand **ACTATGCGACT**.


At this point, you may think that we have made a mistake, since the complementary strand in Figure 2 reads out **TCAGCGTATCA** from left to right rather than **ACTATGCGACT**.

We did 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 2. (See **DETOUR: Chemical Basis for Directionality of DNA Strands**).

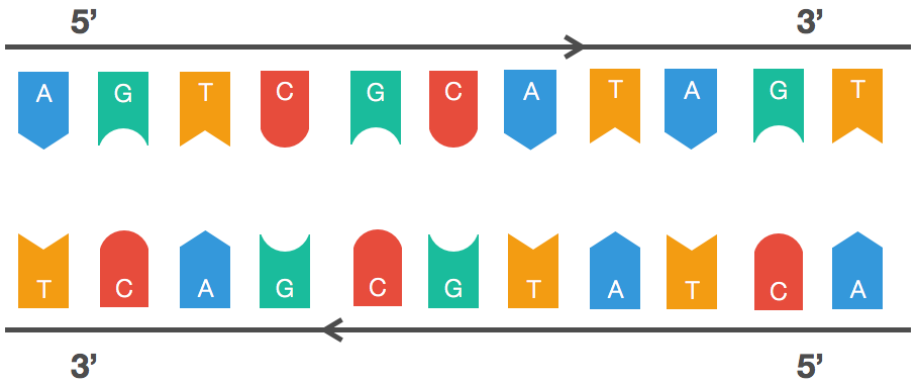



**Figure 2:** Complementary strands run in opposite directions. The beginning and end of a DNA strand are denoted 5' (pronounced "five prime") and 3' (pronounced "three prime"), respectively.

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. We will need the solution to the following problem throughout the course:

---

**Reverse Complement Problem**:
*Reverse complement a pattern.*

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

---

**STOP and Think:** Look again at the four most frequent 9-mers in the *oriC* region of *Vibrio cholerae*. Now do you notice anything surprising?

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

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

Finding a 9-mer that appears six or more 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 or more times (as itself). This statistical evidence 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. For our purposes, it doesn't really matter which of the two strings is the actual *DnaA* box: in either case, both **ATGATCAAG** and **CTTGATCAT** will appear often in the *oriC* region.

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. To this end, we need to solve the following problem:

---

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

**Input**: Two strings, *Pattern* and *Text*.
**Output**: All starting positions where *Pattern* appears as a substring of *Text*.

---

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**, i.e., 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 indeed represent the hidden message to *DnaA* to start replication.


**STOP and Think:** Is it safe to conclude that **ATGATCAAG**/**CTTGATCAT** also represents a *DnaA* box in other bacterial genomes?

### 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 . . .

Let's check the proposed *oriC* region of *Thermotoga petrophila*, a bacterium that strives in extremely hot environments; its name derives from its discovery in the water beneath oil reservoirs, where temperatures can exceed 80° Celsius (176° Fahrenheit).

```
aactctatacctccttttgtcgaatttgtgtgatttatagagaaaatcttattaactga
aactaaaatggtaggtttggtggtaggttttgtgtacattttgtagtatctgatttttaa
ttacataccgtatattgtattaaattgacgaacaattgcatggaattgaatatatgcaaa
acaaacctaccaccaaactctgtattgaccattttaggacaacttcagggtggtaggttt
ctgaagctctcatcaatagactattttagtctttacaaacaatattaccgttcagattca
agattctacaacgctgttttaatgggcgttgcagaaaacttaccacctaaaatccagtat
ccaagccgatttcagagaaacctaccacttacctaccacttacctaccacccgggtggta
agttgcagacattattaaaaacctcatcagaagcttgttcaaaaatttcaatactcgaaa
cctaccacctgcgtcccctattatttactactactaataatagcagtataattgatctga
```

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.

Applying the Frequent Words Problem to the *oriC* region above demonstrates that six different 9-mers appear in this region 3 or more times. Here they are:

| | | |
|---|---|---|
| **AACCTACCA** | **AAACCTACC** | **ACCTACCAC** |
| **CCTACCACC** | **GGTAGGTTT** | **TGGTAGGTT** |

Something peculiar is happening because it is extremely unlikely that six different 9-mers will occur so frequently within the same 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 developed by Gao and Zhang, 2008, which 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:

aactctatacctccttttgtcgaatttgtgtgatttatagagaaaatcttattaactga
aactaaaatggtaggttt**GGTGGTAGG**ttttgtgtacattttgtagtatctgatttttaa
ttacataccgtatattgtattaaattgacgaacaattgcatggaattgaatatatgcaaa
acaaa**CCTACCACC**aaactctgtattgaccattttaggacaacttcag**GGTGGTAGG**ttt
ctgaagctctcatcaatagactattttagtctttacaaacaatattaccgttcagattca
agattctacaacgctgttttaatgggcgttgcagaaaacttaccacctaaaatccagtat
ccaagccgatttcagagaaacctaccacttacctaccactta**CCTACCACC**cgggtggta
agttgcagacattattaaaaacctcatcagaagcttgttcaaaaatttcaatactcgaaa
**CCTACCACC**tgcgtcccctattatttactactactaataatagcagtataattgatctga

Now imagine that you are trying to find *oriC* in a newly sequenced bacterial genome. Searching for "clumps" of either **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*.

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.

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 (larger) 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

From our previous examples of *oriC* regions, **ATGATCAAG** forms a (500,3)-clump in the *Vibrio cholerae* genome, and **CCTACCACC** forms a (500,3)-clump in the *Thermotoga petrophila* genome. We are now ready to formulate the following 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*.

Of course, you can solve the Clump Finding Problem by simply applying your algorithm for the Frequent Words Problem to each window of length $L$ in *Genome*. However, if your algorithm for the Frequent Words Problem is not very efficient, then such an approach may be impractical. For example, recall the algorithm that we described for solving the Frequent Words Problem, which took roughly $L^2 \cdot k$ steps. Applying this algorithm to each window of length $L$ in *Genome* will take approximately $L^2 \cdot k \cdot |Genome|$ steps. Since the length of most bacterial genomes varies from 1 million to 10 million nucleotides, such an approach may take your computer too long to solve the Clump Finding Problem.


**STOP and Think:** Can you come up with a fast algorithm for solving the Clump Finding Problem that takes roughly $4^k + k \cdot |Genome|$ steps? You can assume that $L < 1000$, $k < 15$, and $|Genome| < 10^7$.

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* genome, and it is absolutely unclear which of these 9-mers might represent a *DnaA* box in the bacterium's *oriC* region.


**STOP and Think:** Should we give up? If not, what would you do now?

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 the replication process in the hope that they provide insights into a new algorithmic approach to finding *oriC*.

### The Simplest Way to Replicate DNA

We are now ready to discuss the replication process in more detail. As illustrated in Figure 3, the 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.

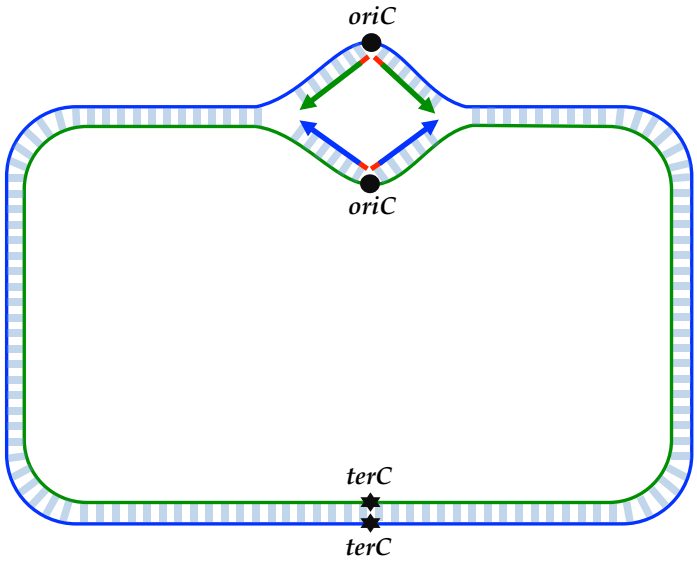



**Figure 3:** Four imaginary DNA polymerases at work replicating a chromosome as the replication fork extends from *oriC* to *terC*. The blue strand reads clockwise, whereas the green strand reads counterclockwise.

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 the *oriC* and replicate the entire chromosome as shown in Figure 3. To start replication, a DNA polymerase needs a **primer**, a very short complementary segment (shown in red in Figure 3) that binds to the parent strand and serves as an anchor for a DNA polymerase. After the strands start separating, each of the four DNA polymerases starts replication by adding nucleotides to the primer and proceeds around the chromosome from *oriC* to *terC* in either the clockwise or counterclockwise direction. When all four DNA polymerases have reached *terC*, the chromosome's DNA will have been completely replicated, resulting in two pairs of complementary strands shown in Figure 4, and the cell is ready to divide.

Yet while you were reading the description above, biology professors were writing a petition to have us fired and sent back to Biology 101. And they would be right, because

our exposition suffers from a major flaw; we only described the replication process in this way so that you can better appreciate what we are about to reveal.

The problem with our current description is that it assumes that DNA polymerases can copy DNA in *either* direction along a strand of DNA (i.e., both $5' \to 3'$ and $3' \to 5'$). However, nature has not yet equipped DNA polymerases with this ability, as they are **unidirectional**, meaning that they can only traverse a template strand of DNA in the $3' \to 5'$ direction. Notice that this is opposite from the $5' \to 3'$ direction of DNA.

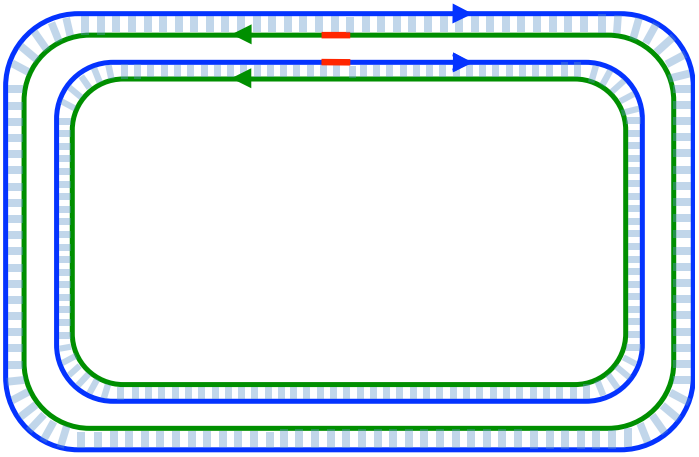


**Figure 4:** Four imaginary DNA polymerases have replicated a chromosome.


**STOP and Think:** If you were a unidirectional DNA polymerase, how would you replicate DNA? How many DNA polymerases are needed to complete this task?

The unidirectionality of DNA polymerase requires a major revision to our naive model of replication. Imagine that you decided to walk along DNA from *oriC* to *terC*. There are four different half-strands of parent DNA connecting *oriC* to *terC*, as highlighted in Figure 5. Two of these half-strands are traversed from *oriC* to *terC* in the same direction as that of DNA ($5' \to 3'$) and are thus called **forward half-strands** (represented by thin blue and green lines in Figure 5). The other two half-strands are traversed from *oriC* to *terC* in the opposite direction as that of DNA ($3' \to 5'$) and are thus called **reverse half-strands** (represented by thick blue and green lines in Figure 5).

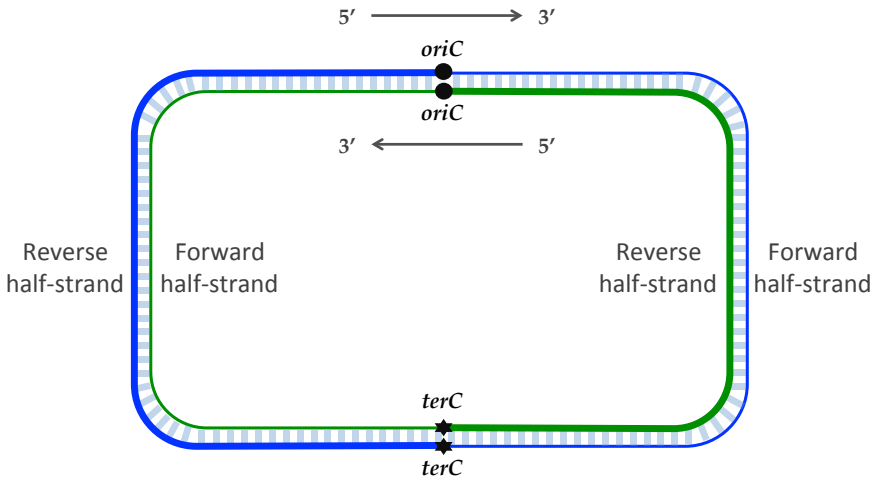



**Figure 5:** Complementary DNA strands with forward and reverse half-strands shown as thin and thick lines, respectively.

### Asymmetry of Replication

While biologists will feel at home with the following description of DNA replication, computer scientists may find it overloaded with new terms. If it seems too biologically complex, you may skim this section, as long as you believe us that the replication process is **asymmetric**, i.e., that forward and reverse half-strands have very different fates with respect to replication.

Since a DNA polymerase can only move in the direction opposite to the direction of DNA, it can copy nucleotides non-stop from *oriC* to *terC* along the reverse half-strands. However, replication on the forward half-strands is very different because a DNA polymerase cannot move in the forward ($5' \rightarrow 3'$) direction; on these half-strands, a DNA polymerase must replicate *backwards* toward *oriC*. Take a look at Figure 6 to see why this must be the case.

On a forward half-strand, in order to replicate DNA, a DNA polymerase must wait for the replication fork to open a little (approximately 2,000 nucleotides) until a new primer is formed at the end of the replication fork; afterwards, the DNA polymerase starts replicating a small chunk of DNA starting from this primer and moving back toward *oriC*. When the two DNA polymerases on forward half-strands reach *oriC*, we have the situation shown in Figure 7. Note the difference with Figure 3.

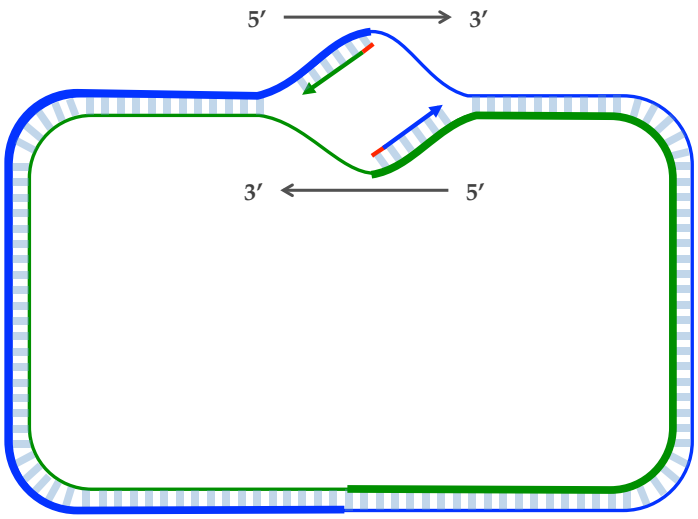



**Figure 6:** Replication begins at *oriC* (primers shown in red) with the synthesis of fragments on the reverse half-strands (shown by thick lines). A DNA polymerase must wait until the replication fork has opened some (small) distance before it starts copying the forward half-strands (shown by thin lines) back toward *oriC*.

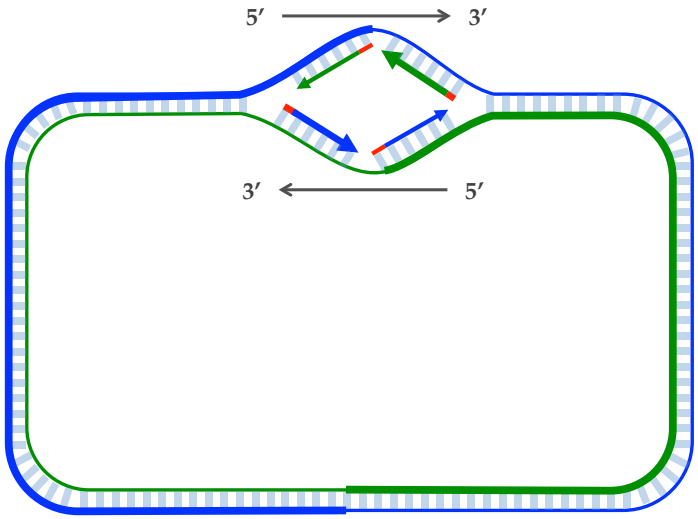



**Figure 7:** The daughter fragments are now synthesized (with some delay) on the forward half-strands, which are shown by thin lines.

After this point, replication on each reverse half-strand progresses continuously, but a DNA polymerase on a forward half-strand has no choice but to wait again until the replication fork has opened another 2,000 nucleotides or so. It then requires a new primer to begin synthesizing another fragment back toward *oriC*. On the whole, replication on a forward half-strand requires occasional stopping and restarting, which results in the synthesis of short **Okazaki fragments** that are complementary to intervals on the forward half-strand. You can see these fragments forming in Figure 8.

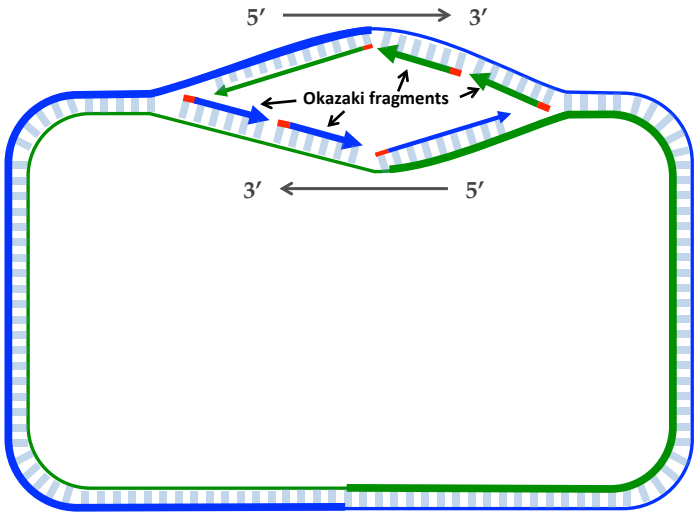



**Figure 8:** The replication fork continues growing. Only one primer is needed for each of the reverse half-strands (shown by thick lines), while the forward half-strands (shown by thin lines) require multiple primers in order to synthesize Okazaki fragments. Two of them are shown in red on each forward half-strand.

When the replication fork reaches *terC*, the replication process is almost complete, but gaps still remain between the disconnected Okazaki fragments, as shown in Figure 9.

Finally, consecutive Okazaki fragments must be sewn together by an enzyme called **DNA ligase**, resulting in two intact daughter chromosomes, each consisting of one parent strand and one newly synthesized strand, as shown in Figure 10.

Biologists call a reverse half-strand a **leading half-strand** since a single DNA polymerase traverses this half-strand non-stop, and they call a forward half-strand a **lagging half-strand** since it is used as a template by many DNA polymerases stopping and starting replication.

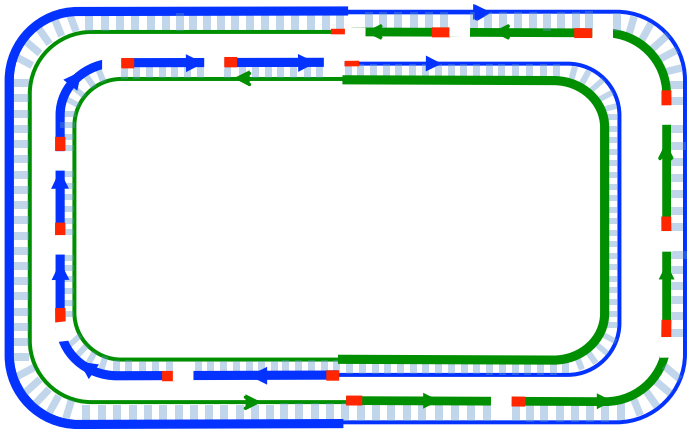


**Figure 9:** Replication is nearly complete, as all daughter DNA is synthesized. However, half of each daughter chromosome contains disconnected Okazaki fragments.

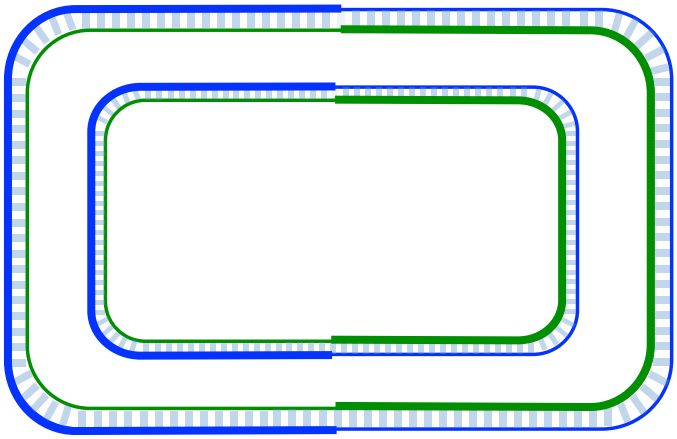


**Figure 10:** Okazaki fragments have been sewn together, resulting in two intact daughter chromosomes.

If you are confused about the differences between the leading and lagging half-strands, you are not alone: we and legions of biology students are also confused. The confusion is exacerbated by the fact that different textbooks use different terminology depending on whether the authors intend to refer to a leading *template* half-strand or a leading half-strand *that is being synthesized* from a (lagging) template half-strand. You hopefully see why we have chosen the terms "reverse" and "forward" half-strands in an attempt to mitigate some of this confusion.

### Peculiar Statistics of the Forward and Reverse Half-Strands

In the last section, we saw that as the replication fork expands, DNA polymerase synthesizes DNA quickly on the reverse half-strand but suffers delays on the forward half-strand. We will explore the asymmetry of DNA replication to design a new algorithm for finding *oriC*.

How in the world can the asymmetry of replication possibly help us locate *oriC*? Notice that since the replication of a reverse half-strand proceeds quickly, it lives doubly-stranded for most of its life. Conversely, a forward half-strand spends a much larger amount of its life single-stranded, waiting to be used as a template for replication. This discrepancy between the forward and reverse half-strands 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 has a greater tendency than other nucleotides to mutate in single-stranded DNA, then we should observe a shortage of this nucleotide on the forward half-strand.

Following up on this thought, let's compare the nucleotide counts of the reverse and forward half-strands: if these counts differ substantially, then we will design an algorithm that attempts to track down these differences in genomes for which *oriC* is unknown. The nucleotide counts for *Thermotoga petrophila* are shown in Table 2.

| | 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 |

**Table 2:** Counting nucleotides in the *Thermotoga petrophila* genome on the forward and reverse half-strands.


**STOP and Think:** Do you notice anything interesting about the nucleotide counts in Table 2?

Although the frequencies of A and T are practically identical on the two half-strands, C is more frequent on the reverse half-strand (219518) than on the forward half-strand (207901), resulting in a difference of $219518 - 207901 = \mathbf{+11617}$. Its complementary nucleotide G is less frequent on the reverse half-strand (201634) than on the forward half-strand (211607), resulting in a difference of $201634 - 211607 = \mathbf{-9973}$.

It turns out that we observe these discrepancies because cytosine (C) has a tendency to mutate into thymine (T) through a process called **deamination**, and deamination rates rise 100-fold when DNA is single-stranded. Deamination leads to a decrease in cytosine on the forward half-strand, which forms 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 (recall that a forward parent half-strand synthesizes a reverse daughter half-strand, and vice-versa).


**STOP and Think:** If deamination changes cytosine to thymine, why do you think that the forward half-strand still has some cytosine?

Let's see if we can take advantage of these peculiar statistics caused by deamination to locate *oriC*. 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 the cytosine frequency is low, and we guess that we are on the forward half-strand; on the other hand, if this difference starts *decreasing*, then the guanine frequency is low, and we guess that we are on the reverse half-strand. See Figure 11.

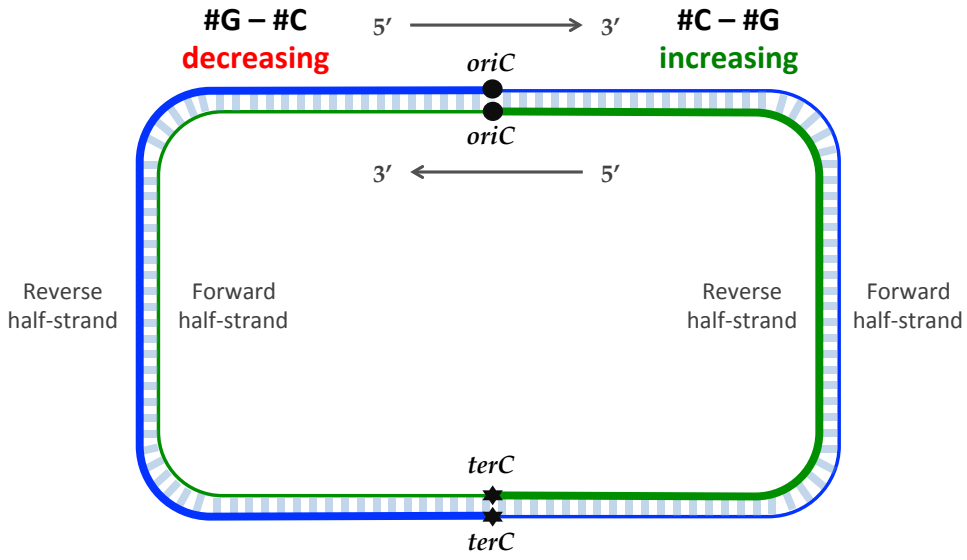



**Figure 11:** Because of deamination, each forward half-strand has more guanine than cytosine, and each reverse half-strand has more cytosine than guanine. The difference between the counts of G and C should therefore be positive on the forward half-stand and negative on the reverse half-strand.


**STOP and Think:** Imagine that you are reading through the genome and notice that the difference G − C just switched its behavior from decreasing to increasing. Where in the genome are you?

Since we don't know the location of *oriC* in a circular genome, let's linearize it (i.e., we cut the circular genome in an arbitrary place), resulting in a linear string *Genome*. Define the **skew** of *Genome*, denoted SKEW(*Genome*), as the difference between the total number of occurrences of G and C in *Genome*. Let PREFIX$_i$(*Genome*) denote the **prefix** (i.e., initial substring) of *Genome* of length $i$. A **skew diagram** is defined by plotting each value of $i$ (as $i$ ranges from 0 to $|Genome|$) against SKEW(PREFIX$_i$(*Genome*)). Figure 12 shows a skew diagram for a short DNA string.

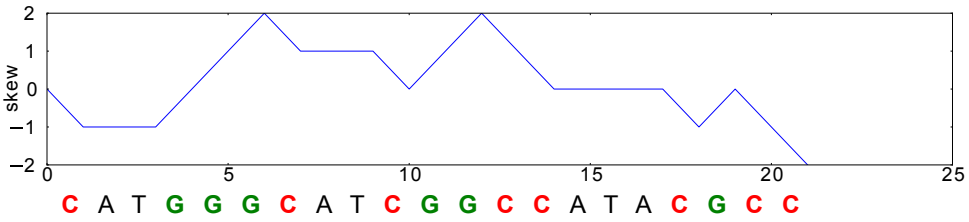



**Figure 12:** The skew diagram for *Genome* = CATGGGCATCGGCCATACGCC.

Figure 13 depicts the skew diagram for a linearized *E. coli* genome (i.e., we cut the circular genome in one place). Notice the very clear pattern! It turns out that the skew diagram for many bacterial genomes has a similar characteristic shape.

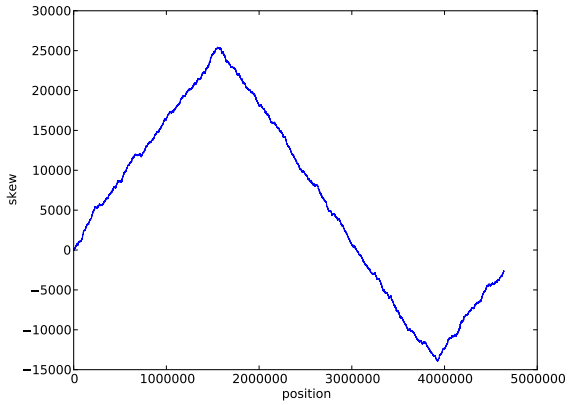



**Figure 13:** The skew diagram for *E. coli* achieves a maximum and minimum at positions 1550413 and 3923620 , respectively.


**STOP and Think:** After looking at the skew diagram in Figure 13, where do you think that *oriC* is located in *E. coli*?

Let's follow the $5' \rightarrow 3'$ direction of DNA and walk along the chromosome from *terC* to *oriC* (along a reverse half-strand) and continue on from *oriC* to *terC* (along a forward half-strand). In Figure 11, we saw that the skew is decreasing along the reverse half-strand and increasing along the forward half-strand. Thus, recalling Figure 5, 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 insight for a new algorithm for locating *oriC*: it should be found where the skew attains a minimum:

---

**Minimum Skew Problem**:
*Find a position in a genome minimizing the skew.*

**Input**: A DNA string *Genome*.
**Output**: All integer(s) $i$ minimizing $\text{SKEW}(\text{PREFIX}_i(Genome))$ among all values of $i$ (from 0 to $|Genome|$).

---


**STOP and Think:** Note that the skew diagram changes depending on where we start our walk along the circular chromosome. Do you think that the position of the minimum of the skew diagram changes as well?

### Some Hidden Messages Are More Elusive Than Others

Solving the Minimum Skew Problem now provides us with an (albeit approximate) location of *oriC* at position 3923620 in *E. coli*. In an attempt to confirm this hypothesis, 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) reveals no 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 ...

```
aatgatgatgacgtcaaaaggatccggataaaacatggtgattgcctcgcataacgcggt
atgaaaatggattgaagcccgggccgtggattctactcaactttgtcggcttgagaaaga
cctgggatcctgggtattaaaaagaagatctatttatttagagatctgttctattgtgat
```

```
ctcttattaggatcgcactgccctgtggataacaaggatccggcttttaagatcaacaac
ctggaaaggatcattaactgtgaatgatcggtgatcctggaccgtataagctgggatcag
aatgaggggttatacacaactcaaaaactgaacaacagttgttctttggataactaccgg
ttgatccaagcttcctgacagagttatccacagtagatcgcacgatctgtatacttattt
gagtaaattaacccacgatcccagccattcttctgccggatcttccggaatgtcgtgatc
aagaatgttgatcttcagtg
```


**STOP and Think:** What would you do next?

Before we 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
tagacggtttttcatcactgactagccaaagccttactctgcctgacatcgaccgtaaat
tgataatgaatttacatgcttccgcgacgatttacctCTTGATCATcgatccgattgaag
atcttcaattgttaattctcttgcctcgactcatagccatgatgagctCTTGATCATgtt
tccttaaccctctatttttacggaagaATGATCAAGctgctgctCTTGATCATcgtttc
```

Finding eight occurrences of our target 9-mer and its reverse complement in a short region (even though some of these occurrences are approximate) 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 modifications as well.

We say that position $i$ in $k$-mers $p_1 \cdots p_k$ and $q_1 \cdots q_k$ is a **mismatch** if $p_i \neq q_i$. Our observation that a *DnaA* box may appear with slight variations leads to the following generalization of the Pattern Matching Problem:

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

**Input**: Two strings *Pattern* and *Text* along with an integer $d$.
**Output**: All positions where *Pattern* appears in *Text* with at most $d$ mismatches.

Our goal is to modify our previous algorithm to find *DnaA* boxes by identifying frequent $k$-mers, possibly with mismatches. Given strings *Text* and *Pattern* as well as an integer $d$, we define $\text{COUNT}_d(\textit{Text}, \textit{Pattern})$ as the number of occurrences of *Pattern* in *Text* with at most $d$ mismatches. For example,

$$\text{COUNT}_1(\textbf{AACAA}\text{GCTG}\textbf{ATAAACA}\text{TTT}\textbf{AAAGA}\text{G}, \textbf{AAAAA}) = 4$$

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.

**EXERCISE BREAK:** Compute $\text{COUNT}_2(\text{AAAAAGCTGATAAACCTTTAAAGAG}, \text{AAAAA})$.

A **most frequent $k$-mer with up to $d$ mismatches** in *Text* is simply a string *Pattern* maximizing $\text{COUNT}_d(\textit{Text}, \textit{Pattern})$ among all $k$-mers. Note that *Pattern* does not need to actually appear as a substring of *Text*; for example, as we saw above, **AAAAA** is the most frequent 5-mer with 1 mismatch in **AACAA**GCTG**ATAAACA**TTT**AAAGA**G, even though it does not appear in this string. Keep this in mind while solving the following problem:

**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 $\overline{\textit{Pattern}}$ 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 genomic region.*

**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.

### A Final Attempt at Finding *DnaA* Boxes in *E. coli*

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 hypothesized by the minimum skew as *oriC*. Although the minimum of the skew diagram for *E. coli* is found at position 3923620, we should not assume that its *oriC* is found exactly at this position due to random fluctuations in the skew. To remedy this issue, we could choose a larger window size (e.g., $L = 1000$), but expanding the window introduces the risk that we may bring in other clumped 9-mers that do not represent *DnaA* boxes but appear in this window more often than the true *DnaA* box. It makes more sense to try a small window either starting, ending, or centered at the position of minimum skew.

Let's cross our fingers and identify the most frequent 9-mers (with 1 mismatch and reverse 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 indeed a most frequent 9-mer with 1 mismatch, along with its reverse complement **TGTGGATAA**:

aatgatgatgacgtcaaaaggatccggataaaacat**ggtgattgcctcgcataacgcggt**
**atgaaaatggattgaagcccgggccgtggattctactcaactttgtcggcttgagaaaga**
**cctgggatcctgggtattaaaaagaagatctatttatttagagatctgttctattgtgat**
**ctcttattaggatcgcactgccc**TGTGGATAA**caaggatccggcttttaagatcaacaac**
**ctggaaaggatcattaactgtgaatgatcggtgatcctggaccgtataagctgggatcag**
**aatgaggggT**TTATACACA**actcaaaaactgaacaacagttgttc**TTTGGATAA**ctaccgg**
**ttgatccaagcttcctgacagag**TTATCCACA**gtagatcg**cacgatctgtatacttattt
gagtaaattaacccacgatcccagccattcttctgccggatcttccggaatgtcgtgatc
aagaatgttgatcttcagtg

You will notice that we emphasized a large part of this sequence with bold text. This region is the experimentally verified *oriC* of *E. coli*, which starts 37 nucleotides after

position 3923620, where the skew reaches its minimum value. We were very 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 one mismatch and reverse complements.


**STOP and Think:** In this chapter, every time we find *oriC*, we seem to find some other surprisingly frequent 9-mers. Why do you think this is?

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 a later chapter. The important lesson is that existing approaches to *oriC* prediction remain imperfect and sometimes inconclusive. However, even providing a small collection of 9-mers as candidate *DnaA* boxes is a great aid to biologists as long as one of these 9-mers is correct.

Thus, the moral of this chapter is that even though *in silico* predictions can be powerful, bioinformaticians should collaborate with biologists to verify their computational predictions. Or improve these predictions . . . the next question hints at how *oriC* predictions can be carried out using **comparative genomics**, a bioinformatics approach that uses evolutionary similarities to answer difficult questions about genomes.


**STOP and Think:** *Salmonella enterica* is a close relative of of *E. coli* that causes typhoid fever and foodborne illness. After having learned what *DnaA* boxes look like in *E. coli*, how would you look for *DnaA* boxes in *Salmonella*?

You will have an opportunity to look for *DnaA* boxes in *Salmonella enterica* in the epilogue. Each chapter will feature such a "Final Challenge" asking you to apply what you have learned to a real dataset. After the epilogue is an "Open Problems" section outlining *real* open research projects.

### Epilogue: Complications in *oriC* Predictions

In this chapter, we have considered three genomes and found three different hypothesized 9-mers encoding *DnaA* boxes: **ATGATCAAG** in *Vibrio cholerae*, **CCTACCACC** in *Thermotoga petrophila*, and **TTATCCACA** in *E. coli*. We must warn you that finding *oriC* is often more complex than in the three examples we considered. Some bacteria have even fewer *DnaA* boxes than *E. coli*, making it difficult to identify them. The *terC* region is often located not directly opposite to *oriC* but may be significantly shifted, resulting in reverse and forward half-strands having substantially different lengths. The position of the skew minimum is often only a rough indicator of *oriC* position, which forces researchers to expand their windows when searching for *DnaA* boxes, bringing in extraneous repeated substrings. Finally, skew diagrams do not always look as nice as that of *E. coli*; for example, the skew diagram for *Thermotoga petrophila* is shown in Figure 14.

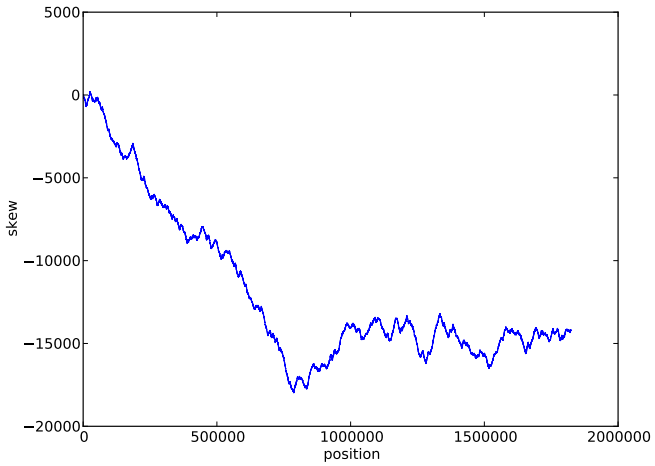



**Figure 14:** The skew diagram for *Thermotoga petrophila* achieves a minimum at position 787199 but does not have the same nice shape as the skew diagram for *E. coli*.

**STOP and Think:** What evolutionary process could possibly explain the shape of the skew diagram for *Thermotoga petrophila*?


Since the skew diagram for *Thermotoga petrophila* is complex and the *oriC* for this genome has not even been experimentally verified, there is a chance that the region predicted by Ori-Finder software as the *oriC* region for *Thermotoga petrophila* (or even for *Vibrio cholerae*) is actually incorrect!

You now should have a good sense of how to locate *oriC* and *DnaA* boxes computationally. We will take the training wheels off and ask you to solve a Final Challenge.

**FINAL CHALLENGE:** Find *DnaA* boxes in *Salmonella enterica*.

### Open Problems

*Multiple Replication Origins in a Bacterial Genome*

Biologists long believed that each bacterial chromosome has only one *oriC*. Wang et al., 2011 genetically modified *E. coli* by inserting a synthetic *oriC* a million nucleotides away from the bacterium's known *oriC*. To their surprise, *E. coli* continued business as usual, starting replication at both locations!

Following the publication of this paper, the search for naturally occurring bacteria with multiple *oriC*s immediately started: Xia, 2012 raised doubts about the "single *oriC*" postulate and gave examples of bacteria with highly unusual skews. In fact, having more than one *oriC* makes sense in the light of evolution: if the genome is long and replication is slow, then multiple replication origins would decrease the amount of time that the bacterium must spend replicating its DNA.

For example, *Wigglesworthia glossinidia*, a symbiotic bacterium living in the intestines of tsetse flies, has the atypical skew diagram shown in Figure 15. Since this diagram has at least two pronounced local minima, Xia argued that this bacterium may have two or more *oriC* regions.

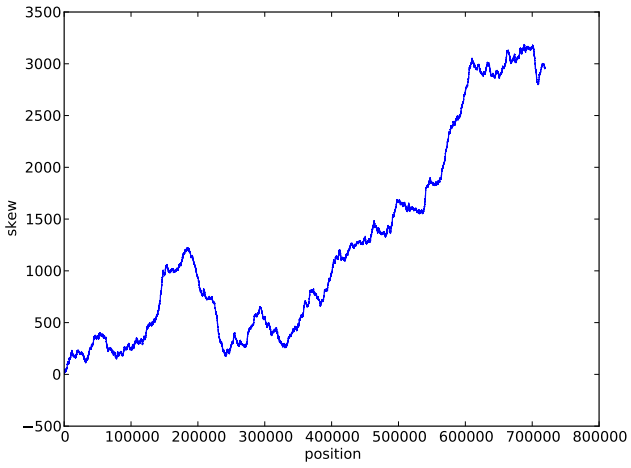



**Figure 15:** The skew diagram for *Wigglesworthia glossinidia.*

We should be careful with Xia's conclusion that this bacterium has two *oriC*s, as multiple local minima in the skew may result from outside forces. For example, **genome rearrangements** (which we will study in a later chapter) move genes within a genome and often reposition them from the forward to the reverse half-strand and vice-versa, thus

resulting in irregularities in the skew diagram. One example of a genome rearrangement is a **reversal**, which flips around a segment of chromosome and switches it to the opposite strand; Figure 16 shows what happens to the skew diagram after a reversal.

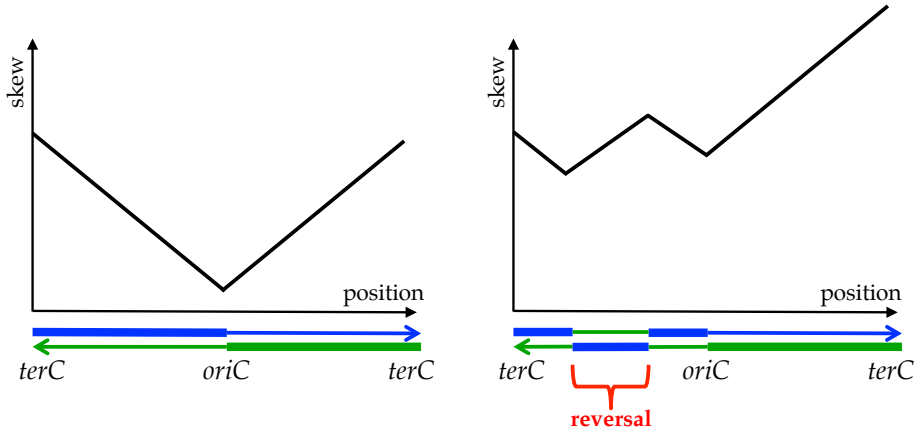



**Figure 16:** Left: An "ideal" V-shaped skew diagram that achieves minimum skew at *oriC*. The skew diagram decreases along the reverse half-strand (shown by thick line) and increases along the forward half-strand (shown by thin line). We assume that a circular chromosome was cut at *terC*, resulting in a linear chromosome that starts and ends at *terC*. Right: A skew diagram right after a reversal that repositions segments on the reverse and forward strands and alters the skew diagram. As before, the skew diagram still decreases along the segments of the genome shown by thick lines and increases along the segments shown by thin lines.

Another difficulty is presented by the fact that different species of bacteria may exchange genetic material in a phenomenon called **horizontal gene transfer**; if a gene from the forward half-strand of one bacterium is transferred to the reverse half-strand of another (or vice-versa), then we will observe an irregularity in the skew diagram. As a result, the question about the number of *oriC* regions of *Wigglesworthia glossinidia* remains unresolved.

However, if you could demonstrate that there exist two sets of identical *DnaA* boxes in the vicinity of two local minima in the skew diagram of *Wigglesworthia glossinidia*, then you would have the first solid evidence in favor of multiple bacterial *oriC*s. Maybe simply applying your solution to the Frequent Words with Mismatches and Reverse Complements Problem would reveal these *DnaA* boxes? Can you find other bacterial genomes where a single *oriC* is in doubt and check whether they indeed have multiple *oriC*s?

### *Finding Replication Origins in Draft Bacterial Genomes*

Although bacterial genome sequencing has taken off in the last few years, the locations of *oriC* in most of these newly sequenced bacteria remain unknown. The challenge with *oriC*-finding is that most of these bacteria have been sequenced as **draft genomes** because existing genome assembly algorithms have difficulties assembling entire bacterial chromosomes. Instead, algorithms assemble segments of the chromosome (called **contigs**); when taken together, the contigs cover most of the genome, but the order of the contigs remains unknown. For example, the typical draft bacterial genome may consist of 100-200 contigs ranging in length from fewer than 1,000 to over 100,000 nucleotides.

The skew approach for finding *oriC* needs to be modified for draft genomes. For example, we can compute the skew diagram for each contig and look at its minimum within each contig. This method will probably work for *E. coli*, whose skew diagram has nearly "perfect" shape and where we expect that only one contig will have a skew diagram with a V-shape. However, the skew diagram for *Thermotoga petrophila* has multiple **local minima**, or places where the skew achieves a minimum somewhere inside a window. Local minima may cause more than one contig to have a V-shape, and it is unclear which of these local minima represents the global minimum pointing to *oriC*.

As an example, let's consider a recently sequenced elusive bacterium called *TM6*, whose origin of replication remains unknown. While thousands of new bacterial species are sequenced every year, 99% of bacterial species on Earth still remain below the radar of DNA sequencing techniques. The problem is that modern DNA sequencing methods require millions of identical bacterial cells to sequence the genome. It means that the bacteria first must be grown in the lab to form **clones** (consisting of millions of cells) before they can be thrown into the sequencing machine. Since most bacteria do not survive (or refuse to replicate) in laboratory conditions, they represent a "dark matter of life" that cannot be sequenced using traditional technologies.

However, biologists recently found a way to sequence these bacteria by developing technologies that can sequence a genome from a single cell. The draft assembly of *TM6*, completed in 2013 from a single bacterial cell (McLean et al., 2013), consists of 7 contigs varying in length from 3,635 to 464,047 nucleotides. Because *TM6* is a new type of exotic bacteria, its *DnaA* boxes may be quite different from previously identified *DnaA* boxes.

The most obvious method for modifying the minimum skew approach would be to compute the skew diagram for each contig and look at the minimum within each diagram.

In this open problem, we ask you to find the *oriC* and *DnaA* boxes in draft genomes like *TM6*. Keep in mind that the *oriC* may not even be present in the assembled contigs.

### *Finding Replication Origins in Archaea*

**Archaea** are unicellular organisms so distinct from other life forms on Earth that biologists have placed them into their own **domain of life** separate from bacteria and eukaryotes. Although archaea are visually similar to bacteria, they have some genomic features that are more closely related to eukaryotes. In particular, the replication machinery of archaea is more similar to eukaryotes than bacteria. Yet they use a much greater variety of energy sources than eukaryotes, feeding on ammonia, metals, or even hydrogen gas.

Figure 17 shows the skew diagram of *Sulfolobus solfataricus*, an archaea growing in acidic volcanic springs in temperatures over 80° C. In its skew diagram, you can see at least three local minima, represented by deep valleys, in addition to many more shallow valleys.

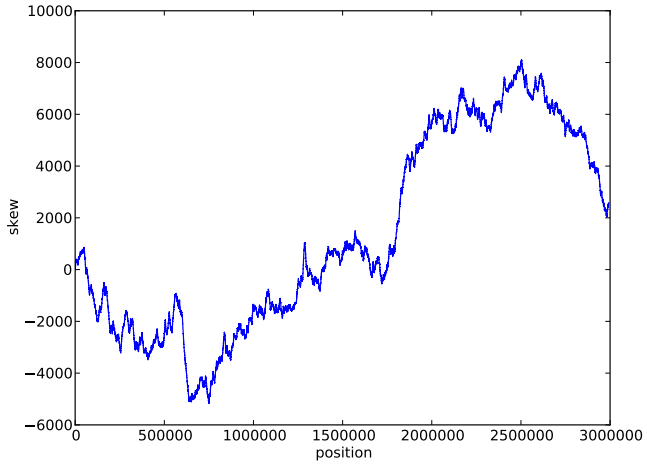



**Figure 17:** The skew diagram for *Sulfolocus salfataricus*.

Lundgren et al., 2004 demonstrated experimentally that *Sulfolocus salfataricus* indeed has three *oriC*s. Since then, multiple *oriC*s have been identified in many other archaea. However, no accurate *in silico* approach has been developed to identify multiple *oriC*s in a newly sequenced archaea. For example, the methane-producing archaea *Methanococcus jannaschii* is considered the workhorse of archaea genomics, and its genome was sequenced nearly 20 years ago, but its *oriC*(s) still remain unidentified! Its skew diagram (shown in Figure 18) suggests that it may have multiple *oriC*s: can you find them?

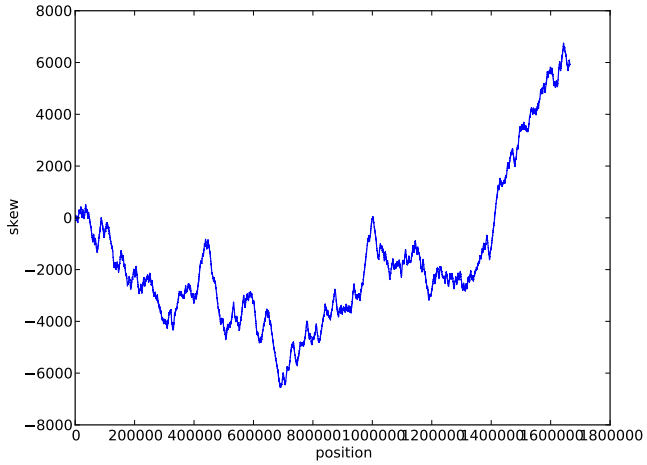



**Figure 18:** The skew diagram for *Methanococcus jannaschii*.

### *Finding Replication Origins in Yeast*

If you think that finding replication origins in bacteria is a complex problem, wait until you analyze replication origins in more complex organisms like yeast or humans, which have hundreds of replication origins. Among various yeast species, the budding yeast *Saccharomyces cerevisiae* has the best characterized replication origins. It has approximately 400 different *oriC*s, many of which may be used during the replication of any single yeast cell.

Having a large number of *oriC*s results in dozens of replication forks hurtling towards each other from different locations in the genome in ways that are not yet completely understood. However, researchers have discovered that the replication origins of *S. cerevisiae* share a (somewhat variable) pattern called the **ARS Consensus Sequence** (**ACS**). The ACS is the binding site for the so-called **Origin Recognition Complex**, which initiates the loading of additional proteins required for origin firing. Many ACSs correspond to the following "canonical" thymine-rich pattern of length 11.

**TTTAT**(**G**/**A**)**TTT**(**T**/**A**)(**G**/**T**)

Here, the notation (X/Y) indicates either nucleotide X or nucleotide Y.

However, various ACSs may differ from this canonical pattern, with lengths varying from 11 to 17 nucleotides. For example, the 11-nucleotide long pattern shown above is often part of a 17-nucleotide pattern:

(T/A)(T/A)(T/A)**TTTAT**(**G/A**)**TTT**(**T/A**)(**G/T**)(T/G)(T/C)

Furthermore, in a recent study of replication origins, Liachko et al., 2013 showed that this larger pattern is often followed by the motif (T/A)(T/A)(T/A) after a gap of length 13.

Recently, some progress has been made in characterizing the ACS in a few other yeast species. In some species like *S. bayanus*, the ACS is almost identical to that of *S. cerevisiae*, while in others such as *K. lactis*, it is very different in length (50 bp) as well as in nucleotide sequence. More alarmingly, at least for bioinformaticians, in some yeast species such as *S. pombe*, the Origin Recognition Complex binds to loosely defined AT-rich regions, which makes it next to impossible to find replication origins based on sequence analysis alone.

Despite recent efforts (Liachko et al., 2013), finding *ori*Cs in yeast remains an open problem, and no accurate software exists for predicting origins of replication from the sequence of yeast genomes. Moreover, it is unclear whether skew diagrams may inspire an *in silico* method identifying origins of replication in *S. cerevisiae* (let alone other yeast species). Can you explore this problem and devise an algorithm to predict replication origins in yeast?

### *Computing Exact Probabilities of Patterns in a String and the Overlapping Words Paradox*

In the main text, we told you that the probability that a random DNA string of length 500 contains a 5-mer appearing three or more times is *approximately* 1/1300. Since this is a rather inaccurate approximation, this research project is aimed at finding exact formulas and more accurate approximations for probabilities of patterns in strings.

We start by asking a seemingly simple question: what is the probability that a specific *Pattern* will appear (at least once) as a substring of a random string of length $N$? This simple question proved to be not so simple and was first addressed by Solov'ev, 1966 (see also Sedgwick and Flajolet, 2013).

The first surprise is that different $k$-mers *Pattern* may have different probabilities of appearing in a random string. For example, the probability that *Pattern* = "01" appears in a random binary string of length 4 is 11/16, while the probability that *Pattern* = "11" appears in a random binary string of length 4 is 8/16. This phenomenon is called the **overlapping words paradox** because different occurrences of *Pattern* can overlap each other for some patterns (e.g., "11") but not others (e.g., "01"). See **DETOUR: The Overlapping Words Paradox**.

In this open problem, we are interested in computing the following probabilities for a random $N$-letter string in an $A$-letter alphabet:

- $\Pr(N, A, Pattern, t)$, the probability that a string *Pattern* appears at least $t$ times in a random string.
- $\Pr^*(N, A, Pattern, t)$, the probabixlity that a string *Pattern* and its reverse complement $\overline{Pattern}$ appear at least $t$ total times in a random string.

Note that the above two functions are relatively straightforward to compute. Several variants of these are open; they may be computable for specific parameter values, but there is no efficient solution in general:

- $\Pr_d(N, A, Pattern, t)$, the probability that a string *Pattern* approximately appears at least $t$ times in a random string (with at most $d$ mutations).
- $\Pr(N, A, k, t)$, the probability that there exists *any* $k$-mer appearing at least $t$ times in a random string.
- $\Pr^*(N, A, k, t)$, the probability that there exists *any* $k$-mer *Pattern* such that *Pattern* and its reverse complement $\overline{Pattern}$ appear at least $t$ times total in a random string.
- $\Pr_d(N, A, k, t)$, the probability that there exists *any* $k$-mer with at least $t$ approximate occurrences in a random string (with at most $d$ mutations).

## Detours

### *Big-O Notation*

Computer scientists typically measure an algorithm's efficiency in terms of its **worst-case running time**, which is the largest amount of time an algorithm can take given the most difficult input of a fixed size. The advantage to considering the worst case running time is that we are guaranteed that our algorithm will never behave worse than our worst-case estimate.

**Big-O notation** describes the running time of an algorithm. For example, if your algorithm for sorting an array of $n$ numbers takes roughly $n^2$ operations for the most difficult dataset, we say that the running time of your algorithm is $O(n^2)$. In reality, depending on your implementation, it may use any number of operations, such as $1.5n^2$, $n^2 + n + 2$, or $0.5n^2 + 1$; all these algorithms are $O(n^2)$ because big-O notation only cares about the term that grows the fastest with respect to $n$. This is because as $n$ grows very large (which is more important than considering smaller values when analyzing algorithms), the difference in behavior between two $O(n^2)$ functions, like $999 \cdot n^2$ and $n^2 + 3n + 9999999$, is negligible when compared to the behavior of functions from different classes, say $O(n^2)$ and $O(n^6)$. Of course, we would prefer an algorithm requiring $1/2 \cdot n^2$ steps to an algorithm requiring $1000 \cdot n^2$ steps.

When we write that the running time of an algorithm is $O(n^2)$, we technically mean that it does not grow faster than a function with a leading term of $c \cdot n^2$, for some constant $c$. Formally, a function $f(x)$ is Big-O of function $g(x)$, or $O(g(x))$, when $f(x) \leq c \cdot g(x)$ for some constant $c$ and sufficiently large $x$.

### *Probabilities of Patterns in a String*

We mentioned that the probability that some 9-mer appears 3 or more times in a random DNA string of length 500 is approximately $1/1300$. We assure you that this calculation does not appear out of thin air. Specifically, we can form a **random string** modeling a strand of DNA by choosing each nucleotide for any position with probability $1/4$. The construction of random strings can be generalized to an arbitrary alphabet with $A$ symbols, where each symbol is chosen with probability $1/A$.

**EXERCISE BREAK:** What is the probability that two random strings of length $n$ in an $A$-letter alphabet will match exactly?

We now ask a simple question: what is the probability that a specific *Pattern* will appear (at least once) as a substring of a random string of length $N$? For example, say that we want to find the probability that "**01**" appears in a **binary string** ($A = 2$) of length 4. Here are all possible such strings.

0000 00**01** 0**01**0 0**01**1 **01**00 **0101** **01**10 **01**11
1000 10**01** 1**01**0 1**01**1 1100 11**01** 1110 1111

Because "01" is a substring of 11 of these 4-mers, and since each 4-mer could be generated with probability 1/16, the probability that "01" appears in a random binary 4-mer is 11/16.

**STOP and Think:** What is the probability that *Pattern* = "11" appears as a substring of a random binary 4-mer?

Surprisingly, changing *Pattern* from "01" to "11" changes the probability that it appears as a substring of a random binary string. Indeed, "**11**" appears in only 8 binary 4-mers:

0000 0001 0010 00**11** 0100 0101 0**11**0 0**111**
1000 1001 1010 10**11** **11**00 **11**01 **111**0 **1111**

As a result, the probability of "01" appearing in a random binary string of length 4 is $8/16 = 1/2$.

**STOP and Think:** Why do you think that "11" is less likely than "01" to appear as a substring of a random binary 4-mer?

Let $\Pr(N, A, Pattern, t)$ denote the probability that a string *Pattern* appears $t$ or more times in a random string of length $N$ formed from an alphabet of $A$ letters. We saw that $\Pr(4, 2, \text{"01"}, 1) = 11/16$ while $\Pr(4, 2, \text{"11"}, 1) = 1/2$. Interestingly, when we make $t$ greater than 1, we see that "01" is *less* likely to appear multiple times than "11". For example, the probability of finding "01" twice or more in a random binary 4-mer is given by $\Pr(4, 2, \text{"01"}, 2) = 1/16$ because "0101" is the only binary 4-mer containing "01" twice, and yet $\Pr(4, 2, \text{"11"}, 2) = 3/16$ because binary 4-mers "0111", "1110" and "1111" all have at least two occurrences of "11".

**EXERCISE BREAK:** Compute $\Pr(100, 2, \text{"01"}, 1)$.

We have seen that different $k$-mers have different probabilities of occurring multiple times as a substring of a random string. In general, this phenomenon is called the **overlapping words paradox** because different substring occurrences of *Pattern* can overlap each other for some choices of *Pattern* but not others (see **DETOUR: Overlapping Words Paradox**).


For example, there are two overlapping occurrences of `"11"` in `"1110"`, and three overlapping occurrences of `"11"` in `"1111"`; yet occurrences of `"01"` can never overlap with each other, and so `"01"` can never occur more than twice in a binary 4-mer. The overlapping words paradox makes computing $\Pr(N, A, \mathit{Pattern}, t)$ a rather complex problem because this probability depends heavily on the particular choice of *Pattern*. In light of the complications presented by the overlapping patterns paradox, we will try to *approximate* $\Pr(A, N, \mathit{Pattern}, t)$ rather than compute it exactly.

To approximate $\Pr(N, A, \mathit{Pattern}, t)$, we will assume that the $k$-mer *Pattern* is not overlapping. As a toy example, say we we wish to count the number of **ternary strings** ($A = 3$) of length 7 that contain `"01"` at least twice. Apart from the two occurrences of `"01"`, we have three remaining symbols in the string. Let's assume that these symbols are all `2`'s. The two occurrences of `"01"` can be inserted into `"222"` in ten different ways to form a 7-mer, as shown below.

| | | | | |
|---|---|---|---|---|
| 0101222 | 0120122 | 0122012 | 0122201 | 2010122 |
| 2012012 | 2012201 | 2201012 | 2201201 | 2220101 |

We inserted these two occurrences of `"01"` into `"222"`, but we could have inserted them into any other ternary 3-mer. Because there are $3^3 = 27$ ternary 3-mers, we obtain an approximation of $10 \cdot 27 = 270$ for the number of different possible ternary 7-mers that contain two or more instances of `"01"`. Because there are $3^7 = 2187$ ternary 7-mers, we estimate the probability $\Pr(7, 3, \texttt{"01"}, 2)$ as $270/2187 \approx 0.123$.


**STOP and Think:** Why is this an approximation for $\Pr(7, 3, \texttt{"01"}, 2)$? Is the true probability $\Pr(7, 3, \texttt{"01"}, 2)$ larger or smaller than $270/2187$?

To generalize the above method to approximate $\Pr(N, A, \mathit{Pattern}, t)$ for arbitrary parameter values, consider a string *Text* of length $N$ having at least $t$ occurrences of a $k$-mer *Pattern*. If we select exactly $t$ of these occurrences, then we can think about *Text* as a sequence of $n = N - t \cdot k$ symbols interrupted by $t$ insertions of the $k$-mer *Pattern*. If we fix these $n$ symbols, then we wish to count the number of different strings *Text* that can be

formed by inserting $t$ occurrences of *Pattern* into these $n$ symbols.

For example, consider again the problem of embedding two occurrences of "01" into 222 ($n = 3$), and note that we have added five copies of a capital X below each 7-mer.

```
0101222   0120122   0122012   0122201   2010122
X X XXX   X XX XX   X XXX X   X XXXX    XX X XX

2012012   2012201   2201012   2201201   2220101
XX XX X   XX XXX    XXX X X   XXX XX    XXXX X
```

What do the X's mean? Instead of counting the number of ways to insert two occurrences of "01" into "222", we can count the number of ways to select two of the five X's to color blue.

```
XXXXX XXXXX XXXXX XXXXX XXXXX

XXXXX XXXXX XXXXX XXXXX XXXXX
```

In other words, we are counting the number of ways to choose 2 out of 5 objects, which can be counted by the **binomial coefficient** $\binom{5}{2} = 10$. More generally, the binomial coefficient $\binom{m}{k}$ represents the number of ways to choose $k$ out of $m$ objects and is equal to $\frac{m!}{k!(m-k)!}$


**STOP and Think:** In general, how many ways are there to implant $t$ instances of a (nonoverlapping) $k$-mer into a string of length $n$ to produce a string of length $n + tk$?

To approximate $\Pr(N, A, Pattern, t)$, we want to count the number of ways to insert $t$ instances of a $k$-mer *Pattern* into a fixed string of length $n = N - t \cdot k$. We will therefore have $n + t$ X's, from which we must select $t$ for the placements of *Pattern*, giving a total of $\binom{n+t}{t}$. We then need to multiply $\binom{n+t}{t}$ by the number of strings of length $n$ into which we can insert $t$ instances of *Pattern* to have an approximate total of $\binom{n+t}{t} \cdot A^n$ (the actual number will be smaller because of over-counting). Dividing by the number of strings of length $N$, we have our desired approximation:

$$\Pr(N, A, Pattern, t) \approx \frac{\binom{n+t}{t} \cdot A^n}{A^N} = \frac{\binom{N-t\cdot(k-1)}{t}}{A^{t\cdot k}}$$

We will now find the probability that the specific 5-mer ACTAT occurs at least $t = 3$ times in a random DNA string ($A = 4$) of length $N = 30$. Since $n = N - t \cdot k = 15$, our estimated probability is

$$\Pr(30, 4, \texttt{"ACTAT"}, 3) \approx \frac{\binom{30-3\cdot 4}{5}}{4^{15}} = \frac{816}{1073741824} \approx 7.599 \cdot 10^{-7}$$

The exact probability is closer to $7.572 \cdot 10^{-7}$, illustrating that our approximation is relatively accurate for non-overlapping patterns. However, it becomes inaccurate for overlapping patterns, e.g., $\Pr(30, 4, \texttt{"AAAAA"}, 3) \approx 1.148 \cdot 10^{-3}$.

We should not be surprised that the probability of finding "ACTAT" in a random DNA string of length 30 is so low. However, remember that our original goal was to approximate the probability that there exists *some* 5-mer appearing three or more times. In general, the probability that some $k$-mer appears $t$ or more times in a random string of length $N$ formed over an $A$-letter alphabet is written $\Pr(N, A, k, t)$.


**STOP and Think:** How could we approximate $\Pr(N, A, k, t)$?

We approximated $\Pr(N, A, Pattern, t)$ as

$$p = \frac{\binom{N-t\cdot(k-1)}{t}}{A^{t\cdot k}}$$

Notice that the approximate probability that *Pattern* does *not* appear $t$ or more times is therefore $1 - p$. Thus, the probability that *all* $A^k$ patterns appear fewer than $t$ times in a random string of length $N$ can be approximated as:

$$(1 - p)^{A^k}$$

Moreover, the probability that there exists a $k$-mer appearing $t$ or more times should be 1 minus this value, which gives us the following approximation:

$$\Pr(N, A, k, t) \approx 1 - (1 - p)^{A^k}$$

Your calculator may have difficulty with this formula, which requires raising a number close to 1 to a very large power and can cause round-off errors. To avoid this, if we assume that $p$ is about the same for any *Pattern*, then we can approximate $\Pr(N, A, k, t)$ by multiplying $p$ by the total number of $k$-mers $A^k$:

$$\Pr(N, A, k, t) \approx p \cdot A^k = \frac{\binom{N-t\cdot(k-1)}{t}}{A^{t\cdot k}} \cdot A^k = \frac{\binom{N-t\cdot(k-1)}{t}}{A^{(t-1)\cdot k}}$$

We acknowledge again that this approximation is a gross over-simplification since the probability $\Pr(N, A, Pattern, t)$ varies across different choices of $k$-mers and because it assumes that occurrences of different $k$-mers are independent events. For example, in the main text, we wish to approximate $\Pr(500, 4, 9, 3)$:

$$\Pr(500, 4, 9, 3) \approx \frac{\binom{500-3\cdot 8}{3}}{4^{(3-1)\cdot 9}} = \frac{17861900}{68719476736} \approx \frac{1}{1300}$$

For more precise estimates and a further discussion of the overlapping words paradox, see Guibas and Odlyzko, 1981.

### *The Most Beautiful Experiment in Biology*

The Meselson-Stahl experiment, conducted in 1958 by Matthew Meselson and Franklin Stahl, is sometimes called "the most beautiful experiment in biology". In the late 1950s, biologists debated three conflicting models of DNA replication, illustrated in Figure 19. The **semiconservative hypothesis**, which was the one proposed by Watson and Crick (recall Figure 1) and turned out to be true, proposed that each parent strand acts as a template for the synthesis of a daughter strand. As a result, each of the two daughter molecules contains one parent strand and one newly synthesized strand. The **conservative hypothesis** proposed that the entire double-stranded parent DNA molecule serves as a template for the synthesis of a new daughter molecule, resulting in one molecule with two parent strands and another with two newly synthesized strands. The **dispersive hypothesis** proposed that some mechanism breaks the DNA backbone into pieces and splices intervals of synthesized DNA, so that each of the daughter molecules is a patchwork of old and new double-stranded DNA.

Meselson and Stahl's ingenious insight relied on the fact that one isotope of nitrogen, **Nitrogen-14** ($^{14}$N), is lighter and more abundant than **Nitrogen-15** ($^{15}$N). Knowing that the DNA molecular structure naturally contains $^{14}$N , Meselson and Stahl grew *E. coli* for many rounds of replication in a $^{15}$N medium, which caused the bacteria to gain weight as they absorbed the heavier isotope into their DNA. When they were confident that the bacterial DNA was saturated with $^{15}$N, they transferred these heavy *E. coli* cells to a less dense $^{14}$N medium.


**STOP and Think:** What do you think happened when the "heavy" *E. coli* replicated in the "light" $^{14}$N medium?

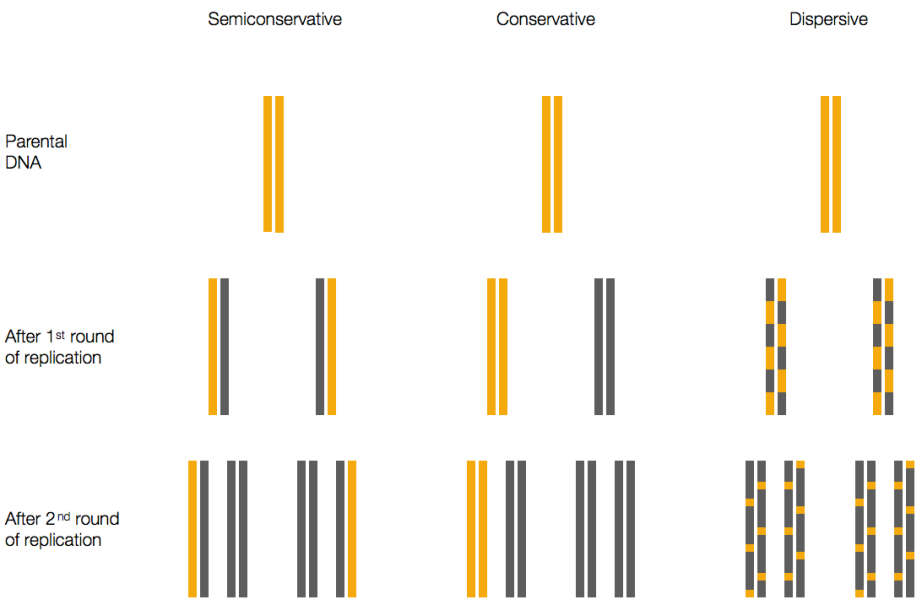



**Figure 19:** Semiconservative, conservative, and dispersive models of DNA replication make different predictions about the distribution of DNA strands after replication. Yellow strands indicate $^{15}$N (heavy) segments of DNA, and black strands indicate $^{14}$N (light) segments. The Meselson-Stahl experiment began with DNA consisting of 100% $^{15}$N.

The brilliance of the Meselson-Stahl experiment is that all newly synthesized DNA would contain exclusively $^{14}$N, and the three existing hypotheses for DNA replication predicted different outcomes for whether this $^{14}$N isotope would be incorporated into DNA. Specifically, after one round of replication, the conservative model predicted that half the *E. coli* DNA would still have only $^{15}$N and therefore be heavier whereas the other half would have only $^{14}$N and be lighter. Yet when they attempted to separate the *E. coli* DNA according to weight by using a centrifuge after one round of replication, all the DNA had the same density! Just like that, they had refuted the conservative hypothesis once and for all.

Unfortunately, Meselson and Stahl were not able to eliminate either of the other two models, as both the dispersive and semiconservative hypotheses predicted that all the DNA after one round of replication would have the same density.

**STOP and Think:** What would the dispersive and semiconservative models predict about the density of *E. coli* DNA after two rounds of replication?

Let's first consider the dispersive model, which says that each daughter strand of DNA is formed by half mashed up pieces of the parent strand, and half new DNA. If this hypothesis were true, then after two replication cycles, any daughter strand of DNA should contain about 25% $^{15}$N and about 75% $^{14}$N. In other words, all the DNA should still have the same density. And yet when Meselson and Stahl spun the centrifuge after two rounds of *E. coli* replication, this is not what they observed!

Instead, they found that the DNA divided into two different densities. This is exactly what the semiconservative model predicted: after one cycle, every cell should possess one $^{14}$N strand and one $^{15}$N strand; after two cycles, half of the DNA molecules should have one $^{14}$N strand and one $^{15}$N strand, while the other half should have two $^{14}$N strands, producing the two different densities they noticed.


**STOP and Think:** What does the semi-conservative model predict about the density of *E. coli* DNA after three rounds of replication?

Meselson and Stahl had rejected the conservative and dispersive hypotheses of replication, and yet they wanted to make sure that the semiconservative hypothesis was confirmed by further *E. coli* replication. This model predicted that after three rounds of replication, one-quarter of the DNA molecules should still have a $^{15}$N strand, causing 25% of the DNA to have an intermediate density, whereas the remaining 75% should be lighter, having only $^{14}$N. After four replication cycles, only 12.5% of the DNA should have the intermediate density, and so on. This is indeed what Meselson and Stahl witnessed in the lab, and the semiconservative hypothesis has stood strong to this day.

### *Chemical Basis for Directionality of DNA Strands*

The sugar component of a nucleotide has a ring of five carbon atoms, which are labeled as 1′, 2′, 3′, 4′, and 5′ in Figure 20. The 5′ atom is joined onto the phosphate group in the nucleotide and eventually to the 3′ end of the neighboring nucleotide. The 3′ atom is joined onto another neighboring nucleotide in the nucleic acid chain. As a result, we call the two ends of the nucleotide the **5′-end** and the **3′-end** (pronounced "five prime end" and "three prime end", respectively).

When we zoom out to the level of the double helix, we can see in Figure 21 that any DNA fragment is oriented with a 3′ atom on one end and a 5′ atom on the other end. As a standard, a DNA strand is always read in the 5′ $\rightarrow$ 3′ direction. Note that the orientations run opposite to each other in complementary strands.

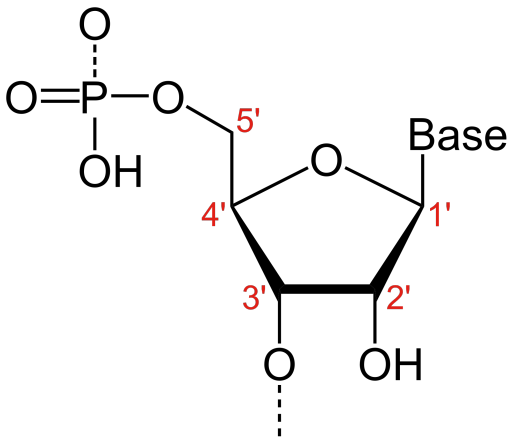



**Figure 20:** A nucleotide with sugar ring carbon atoms labeled 1', 2', 3', 4', and 5'.

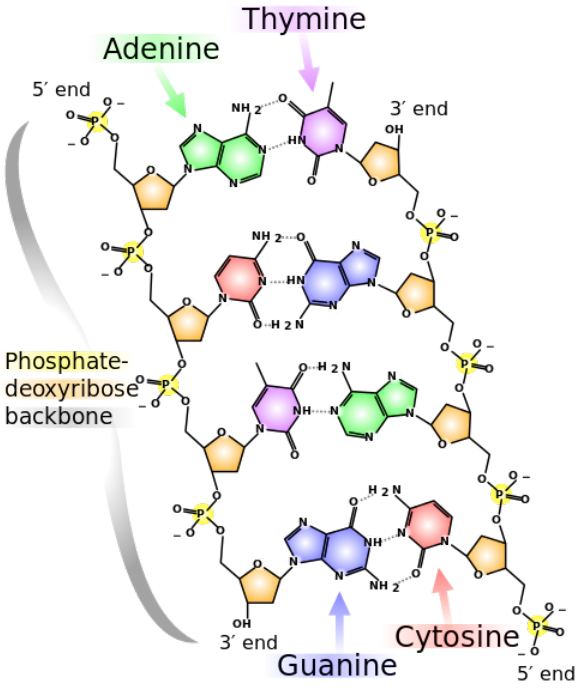



**Figure 21:** The 5' → 3' directions are down the left strand and up the right strand.

*The Overlapping Words Paradox*

We illustrate the overlapping words paradox with a two-player game called "**Best Bet for Simpletons**". Player 1 selects a binary $k$-mer $A$, and Player 2, knowing what $A$ is, selects a different binary $k$-mer $B$. The two players then flip a coin multiple times, with coin flips represented by strings of 1's ("heads") and 0's ("tails"); the game ends when $A$ or $B$ appears as a block of $k$ consecutive coin tosses.


**STOP and Think:** Do the two players always have the same chance of winning?

At first glance, you might guess that every $k$-mer has an equal chance of winning. Yet suppose that Player 1 chooses `"00"` and Player 2 chooses `"10"`. After two tosses, either Player 1 wins (`"00"`), Player 2 wins (`"10"`), or the game continues (`"01"` or `"11"`). If the game continues, then Player 1 should surrender, since Player 2 will win as soon as "tails" (0) is next flipped. Player 2 is therefore three times more likely to win!

It may seem that Player 1 should have the advantage by simply selecting the "strongest" $k$-mer. However, an intriguing feature of "Best Bet for Simpletons" is that if $k > 2$, then Player 2 can always choose a $k$-mer $B$ that beats $A$, *regardless* of Player 1's choice of $A$. Another surprise is that "Best Bet for Simpletons" is a **non-transitive game**: if $A$ defeats $B$, and $B$ defeats $C$, then we cannot automatically conclude that $A$ defeats $C$ (c.f. rock-paper-scissors).

The analysis of "Best Bet for Simpletons" is based on the notion of a **correlation polynomial**, introduced by Guibas and Odlyzko, 1981. We say that $B$ ***i*-overlaps** with $A$ if the last $i$ digits of $A$ coincide with the first $i$ digits of $B$. For example, `110110` 1-overlaps, 2-overlaps, and 5-overlaps with `011011`, as shown in Figure 22.

```
                    AB
B=110110            0
B= 11011 0          1
B=  110110          0
B=   110110         0
B=    11 0110       1
B=     1 10110      1
A=011011
```

**Figure 22:** The correlation $AB$ of $k$-mers $A = 011011$ and $B = 110110$ is the string **010011**.

Given two $k$-mers $A$ and $B$, the **correlation** of $A$ and $B$, denoted $AB = (c_0, \ldots, c_{k-1})$, is a $k$-letter binary word such that $c_i = 1$ if $A$ and $B$ $(k-i)$-overlap, and 0 otherwise. The correlation polynomial of $A$ and $B$ is defined as

$$K_{AB}(t) = c_0 + c_1 \cdot t + c_2 \cdot t^2 + \cdots + c_{k-1} \cdot t^{k-1}$$

For the strings $A$ and $B$ in Figure 22, the correlation polynomial is $K_{AB}(t) = t + t^4 + t^5$.

Next, we write $K_{AB}$ as shorthand for $K_{AB}(1/2)$. For the example above, $K_{AB} = \frac{1}{2} + \frac{1}{16} + \frac{1}{32} = \frac{19}{32}$. Mathematician John Conway suggested the following deceivingly simple formula to compute the odds that $B$ will defeat $A$:

$$\frac{K_{AA} - K_{AB}}{K_{AB} - K_{BA}}$$

Conway's proof of this formula was never published, and Martin Gardner, a leading popular mathematics writer, said the following about the formula:

> *I have no idea why it works. It just cranks out the answer as if by magic, like so many of Conway's other algorithms.*

A rather complicated proof of the formula was derived in Guibas and Odlyzko, 1981.

### Bibliography Notes

Using the skew to find replication origins was first proposed by Lobry, 1996 and also described in Grigoriev, 1998. Grigoriev, 2011 provides an excellent introduction to the skew approach, and Sernova and Gelfand, 2008 gave a review of algorithms and software tools for finding replication origins in bacteria. Lundgren et al., 2004 demonstrated that archaea may have multiple *oriC*s. Wang et al., 2011 inserted an artificial *oriC* into the *E. coli* genome and showed that it triggers replication. Xia, 2012 was the first to conjecture that bacteria may have multiple replication origins. Gao and Zhang, 2008 developed the Ori-Finder software program for finding bacterial replication origins. The *TM6* genome was sequenced in McLean et al., 2013. Liachko et al., 2013 provided the most comprehensive description of the replication origins of yeast. Solov'ev, 1966 was the first to derive accurate formulas for approximating the probabilities of patterns in a string. Guibas and Odlyzko, 1981 provided an excellent coverage of the overlapping words paradox that illustrates the complexity of computing the probabilities of patterns in a random text. Sedgwick and Flajolet, 2013 gave an overview of various approaches for computing the probabilities of patterns in a string.

# Bibliography


Gao, Feng and Chun-Ting Zhang (2008). "Ori-Finder: A web-based system for finding *oriCs* in unannotated bacterial genomes". In: *BMC Bioinformatics* 9.1, p. 79. DOI: 10.1186/1471-2105-9-79.

Grigoriev, Andrey (1998). "Analyzing genomes with cumulative skew diagrams". In: *Nucleic Acids Research* 26.10, pp. 2286–2290. DOI: 10.1093/nar/26.10.2286.

— (2011). "How do replication and transcription change genomes?" In: *Bioinformatics for Biologists*. Ed. by Pavel A. Pevzner and Ron Shamir. Cambridge, United Kingdom: Cambridge University Press, pp. 111–125.

Guibas, L.J and A.M Odlyzko (1981). "String overlaps, pattern matching, and nontransitive games". In: *Journal of Combinatorial Theory, Series A* 30.2, pp. 183 –208. DOI: 10.1016/0097-3165(81)90005-4.

Liachko, Ivan et al. (2013). "High-resolution mapping, characterization, and optimization of autonomously replicating sequences in yeast". In: *Genome Research* 23.4, pp. 698–704. DOI: 10.1101/gr.144659.112.

Lobry, J R (1996). "Asymmetric substitution patterns in the two DNA strands of bacteria". In: *Molecular Biology and Evolution* 13.5, pp. 660–665.

Lundgren, Magnus et al. (2004). "Three replication origins in *Sulfolobus* species: Synchronous initiation of chromosome replication and asynchronous termination". In: *Proceedings of the National Academy of Sciences of the United States of America* 101.18, pp. 7046–7051. DOI: 10.1073/pnas.0400656101.

McLean, Jeffrey S. et al. (2013). "Candidate phylum *TM6* genome recovered from a hospital sink biofilm provides genomic insights into this uncultivated phylum". In: *Proceedings of the National Academy of Sciences*. DOI: 10.1073/pnas.1219809110.

Sedgwick, Robert and Philippe Flajolet (2013). *An Introduction to the Analysis of Algorithms*. Reading, Massachusetts: Addison-Wesley.

Sernova, Natalia V. and Mikhail S. Gelfand (2008). "Identification of replication origins in prokaryotic genomes". In: *Briefings in Bioinformatics* 9.5, pp. 376–391. DOI: 10.1093/bib/bbn031.

Solov'ev, A.D. (1966). "A combinatorial identity and its application to the problem about the first occurence of a rare event". In: *Theory Prob. Appl.* 11.2, pp. 276–282. DOI: 10.1137/1111022.

Wang, Xindan et al. (2011). "Replication and segregation of an *Escherichia coli* chromosome with two replication origins". In: *Proceedings of the National Academy of Sciences* 108.26, E243–E250. DOI: 10.1073/pnas.1100874108.

Xia, X. (2012). "DNA replication and strand asymmetry in prokaryotic and mitochondrial genomes". In: *Curr. Genomics* 13.1, pp. 16–27. DOI: 10.2174/138920212799034776.