How Do We Find Regulatory Motifs in DNA?

Greedy and Randomized Algorithms

Phillip Compeau and Pavel Pevzner.

Bioinformatics Algorithms: An Active Learning Approach

Outline

- **From Implanted Patterns to Regulatory Motifs**
  - **Implanting Patterns into Strings**
  - Do We Have a Clock Gene?
  - Implanted Motif Problem
  - Motif Finding Problem (Music Break)
  - Median String Problem
  - Greedy Motif Search
- How Rolling Dice Helps Us Find Regulatory Motifs
  - Randomized Motif Search
  - How do Bacteria Hibernate?
  - Gibbs Sampling
  - Pseudocounts

Generate Ten Random Sequences

atgaccgggatactgataccgtatttggcctaggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg

acccctatttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaacttttccgaatactgggcataaggtaca

tgagtatccctgggatgacttttgggaacactatagtgctctcccgatttttgaatatgtaggatcattcgccagggtccga

gctgagaattggatgaccttgtaagtgttttccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga

tccctttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaatggcccacttagtccacttatag

gtcaatcatgttcttgtgaatggatttttaactgagggcatagaccgcttggcgcacccaaattcagtgtgggcgagcgcaa

cggttttggcccttgttagaggcccccgtactgatggaaactttcaattatgagagagctaatctatcgcgtgcgtgttcat

aacttgagttggtttcgaaaatgctctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta

ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcatttcaacgtatgccgaaccgaaagggaag

ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttctgggtactgatagca

Implant a Pattern **AAAAAAAGGGGGGG** at Randomly Chosen Positions

atgaccgggatactgat**AAAAAAAAGGGGGGG**ggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg

acccctatttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaacttttccgaata**AAAAAAAAGGGGGGG**a

tgagtatccctgggatgactt**AAAAAAAAGGGGGGG**tgctctcccgatttttgaatatgtaggatcattcgccagggtccga

gctgagaattggatg**AAAAAAAAGGGGGGG**tccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga

tccctttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaat**AAAAAAAAGGGGGGG**cttatag

gtcaatcatgttcttgtgaatggattt**AAAAAAAAGGGGGGG**gaccgcttggcgcacccaaattcagtgtgggcgagcgcaa

cggttttggcccttgttagaggcccccgt**AAAAAAAAGGGGGGG**caattatgagagagctaatctatcgcgtgcgtgttcat

aacttgagtt**AAAAAAAAGGGGGGG**ctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta

ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcat**AAAAAAAAGGGGGGG**accgaaagggaag

ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagctt**AAAAAAAAGGGGGGG**a

Where Are the Implanted Patterns Hiding?

atgaccgggatactgataaaaaaaaggggggggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg

acccctatttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaacttttccgaataaaaaaaaaggggggga

tgagtatccctgggatgacttaaaaaaaaggggggtgctctcccgatttttgaatatgtaggatcattcgccagggtccga

gctgagaattggatgaaaaaaaaggggggtccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga

tccctttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaataaaaaaaaggggggcttatag

gtcaatcatgttcttgtgaatggatttaaaaaaaaggggggggaccgcttggcgcacccaaattcagtgtgggcgagcgcaa

cggttttggcccttgttagaggcccccgtaaaaaaaaggggggcaattatgagagagctaatctatcgcgtgcgtgttcat

aacttgagttaaaaaaaaggggggctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta

ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcataaaaaaaaggggggggaccgaaagggaag

ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttaaaaaaaaggggggga

The implanted patterns can be found by the Frequent Words algorithm.

Implant a Pattern **AAAAAAGGGGGGG** with 4 Random Mutations at Random Positions

atgaccgggatactgat**A**g**AA**g**AAAGG**tt**GGG**ggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg

acccctatttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaacttttccgaata**c**AA**t**AAAA**c**GG**c**GGGa

tgagtatccctgggatgactt**AAAA**t**AA**t**GG**a**G**t**GG**tgctctcccgatttttgaatatgtaggatcattcgccagggtccga

gctgagaattggatg**c**AAAAAAAGGG**att**Gtccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga

tccctttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaat**A**t**AA**t**AAAGG**aa**GGG**cttatag

gtcaatcatgttcttgtgaatggattt**AA**c**AA**t**AAGGG**ct**GG**gaccgcttggcgcacccaaattcagtgtgggcgagcgcaa

cggttttggcccttgttagaggcccccgt**A**t**AAA**c**AAGG**a**GGG**ccaattatgagagagctaatctatcgcgtgcgtgttcat

aacttgagtt**AAAAAA**t**AGGG**a**G**cc**ctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta

ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcat**A**ct**AAAAAGG**a**G**c**GG**accgaaagggaag

ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagctt**A**ct**AAAAAGG**a**G**c**GG**a

A *(k,d)-motif:* a *k*-mer that appears in each sequence with at most *d* mismatches. **AAAAAAGGGGGGG is a (15,4)-motif**

Now Where are the Implanted Patterns?

atgaccgggatactgatagaagaaaggttgggggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg

acccctatttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaacttttccgaatacaataaaacggcggga

tgagtatccctgggatgacttaaaataatggagtggtgctctcccgatttttgaatatgtaggatcattcgccagggtccga

gctgagaattggatgcaaaaaaagggattgtccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga

tccctttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaatataataaaggaagggcttatag

gtcaatcatgttcttgtgaatggatttaacaataagggctgggaccgcttggcgcacccaaattcagtgtgggcgagcgcaa

cggttttggcccttgttagaggcccccgtataaacaaggagggccaattatgagagagctaatctatcgcgtgcgtgttcat

aacttgagttaaaaaatagggagccctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta

ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcatactaaaaaggagcggaccgaaagggaag

ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttactaaaaaggagcgga

Would the Frequent Words algorithm find this hidden message?

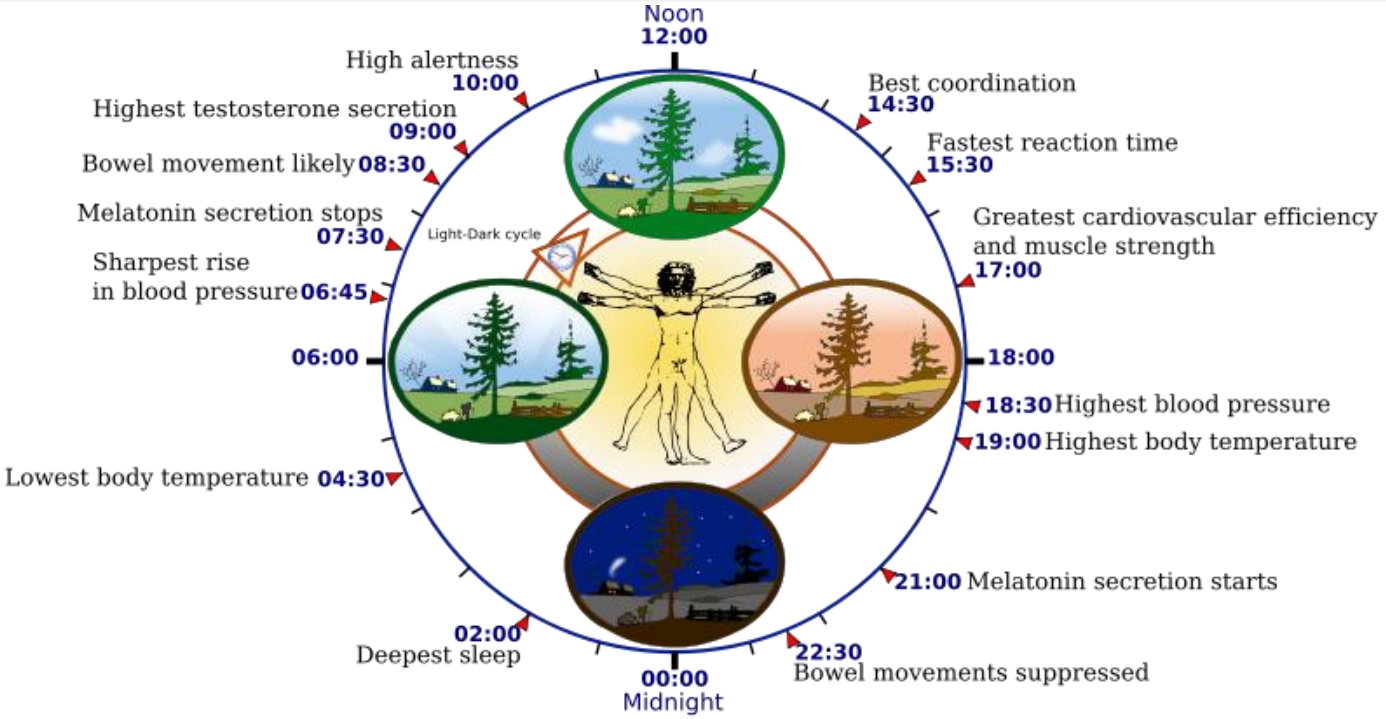
Why Would a Biologist Care?

Outline

- **From Implanted Patterns to Regulatory Motifs**
  - Implanting Patterns into Strings
  - **Do We Have a Clock Gene?**
  - Implanted Motif Problem
  - Motif Finding Problem (Music Break)
  - Median String Problem
  - Greedy Motif Search
- How Rolling Dice Helps Us Find Regulatory Motifs
  - Randomized Motif Search
  - How do Bacteria Hibernate?
  - Gibbs Sampling
  - Pseudocounts

Circadian Clock

Our schedules are controlled by a molecular timekeeper called the **circadian clock**.



How does each cell in our body know when it should decrease or increase **gene expression** of circadian genes?

How do Plants Know Whether it is Day or Night?

Plants need to **change gene expression** of 1000+ genes (photosynthesis, flowering, frost resistance, etc.) switching from day to night and vice versa

Who are the "molecular managers" telling genes to switch expression on and off in billions of cells?

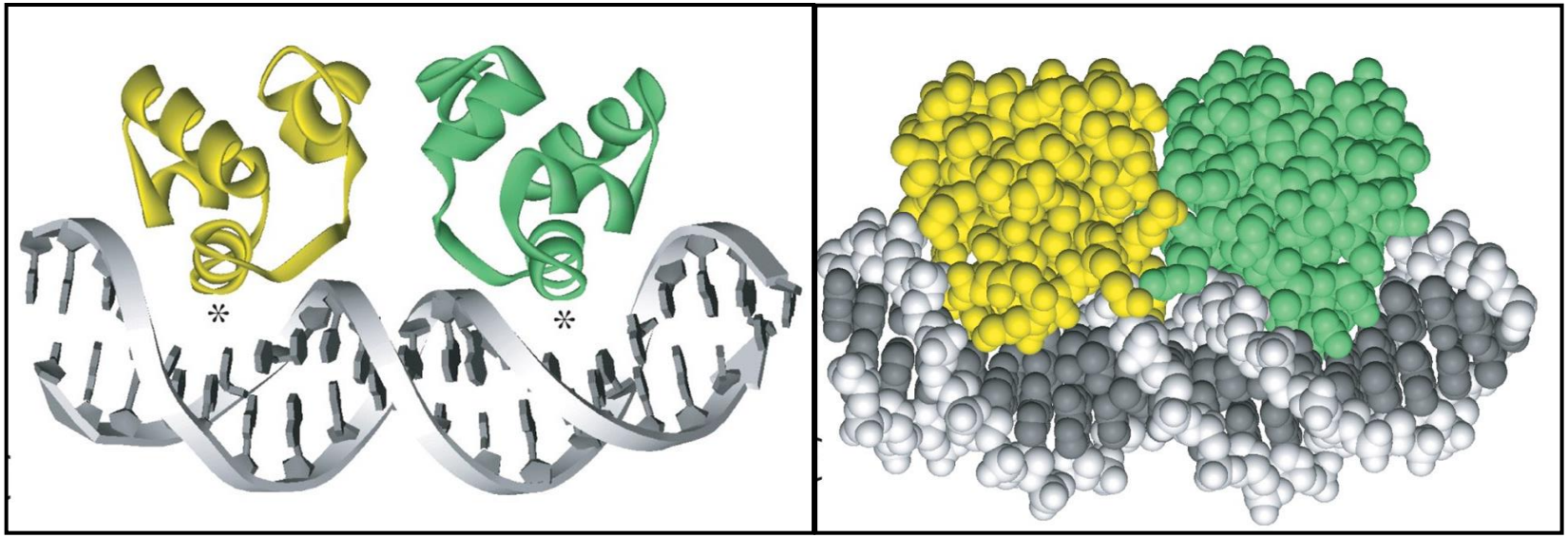
Just 3 genes call the shots: CCA1, LCY, and TOC1.

Night Blooming Cereus
only blooms at night

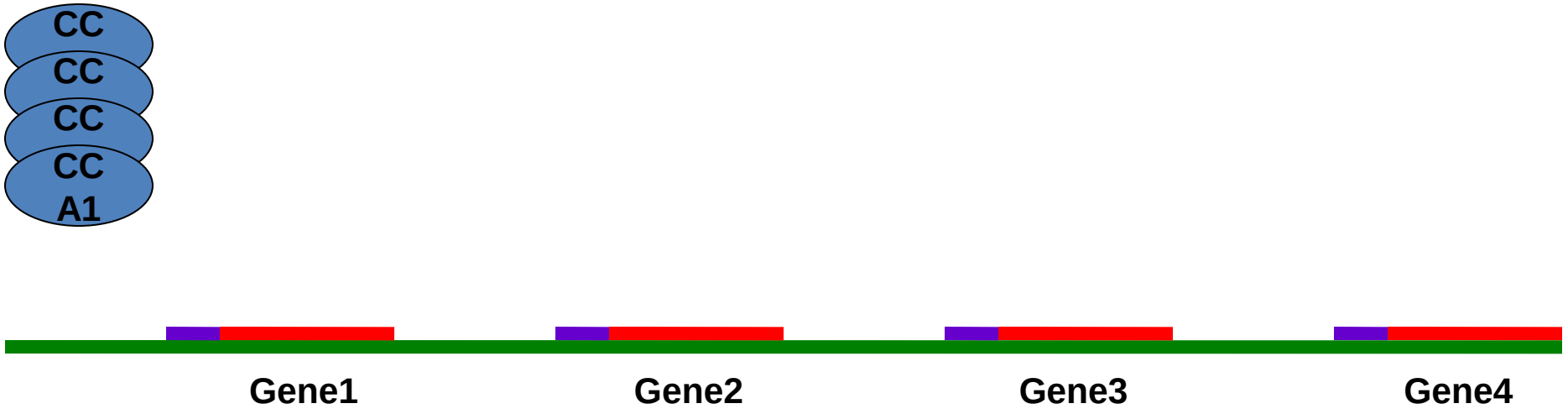
Sunflower
follows the sun

Transcription Factors and Their Binding Sites

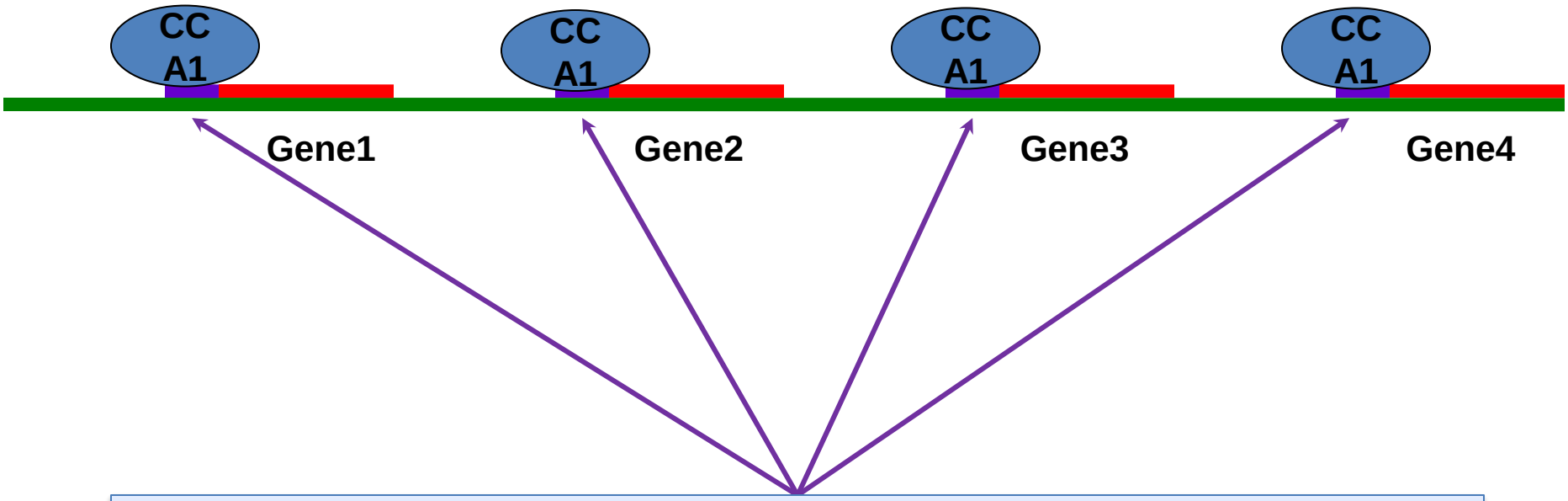
CCA1, LCY, and TOC1 are **regulatory proteins** (a.k.a. **transcription factors**) control other genes by binding to short DNA fragments (**transcription factor binding sites**) in the upstream regions of these genes.



Transcription Factors Bind to Their Binding Sites

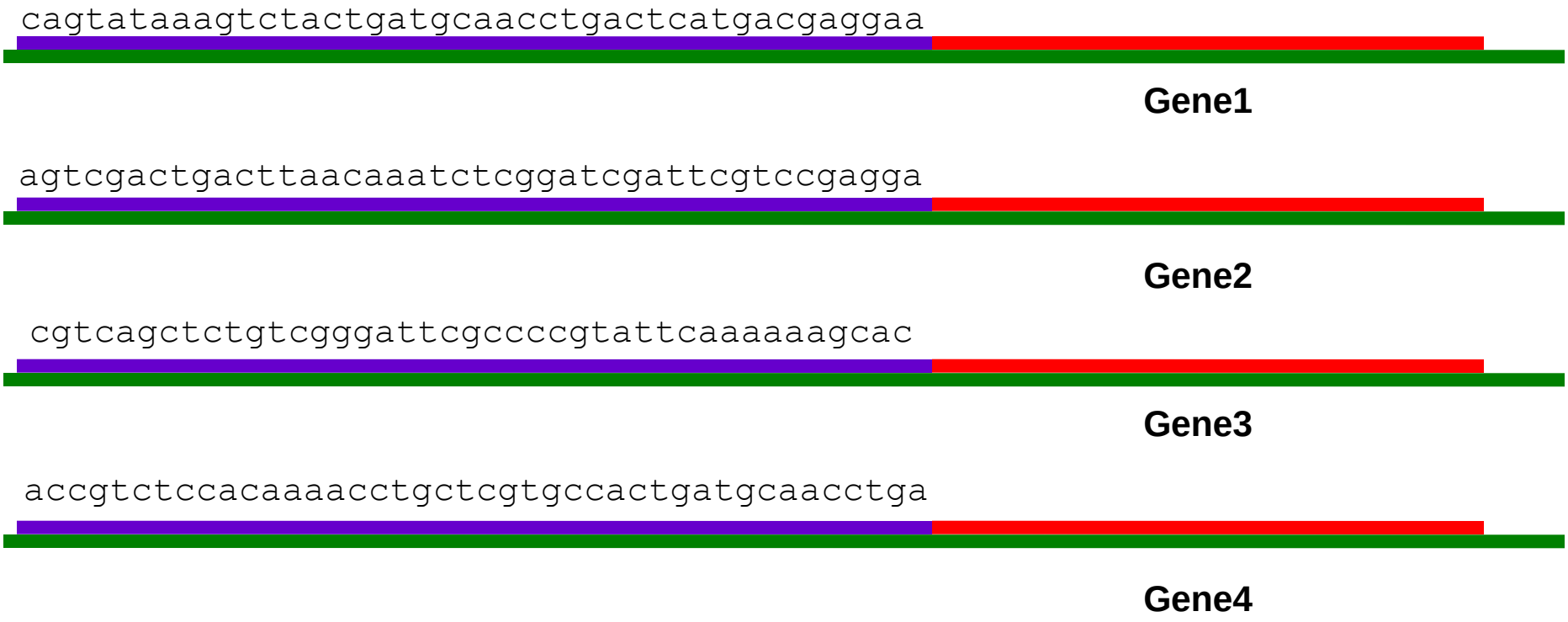


How Does CCA1 Know Where to Bind?

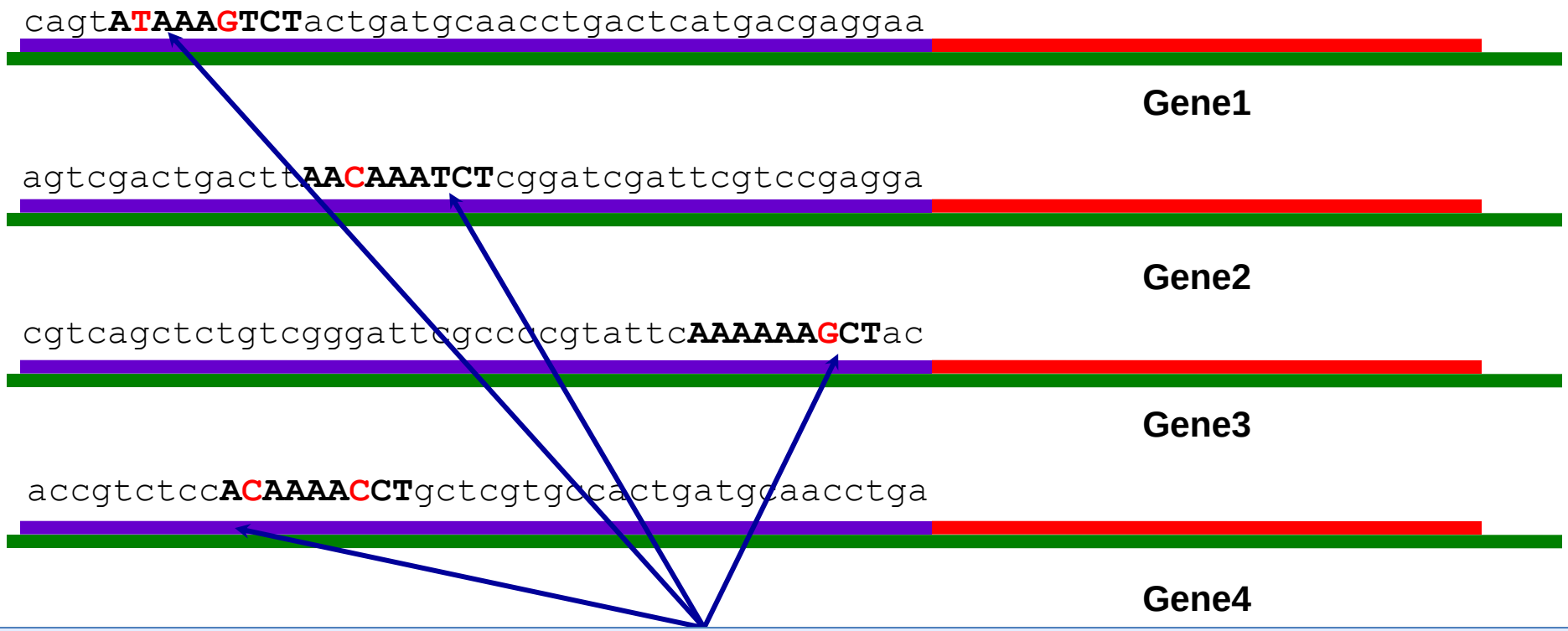


There must be some hidden messages in these regions that tells CCA1 **WHERE** to bind.

Where are the Hidden Messages Hiding?



Transcription Factor Binding Sites



The hidden messages (motif **AAAAAATCT**):
transcription factor binding sites of CCA1

Outline

- **From Implanted Patterns to Regulatory Motifs**
  - Implanting Patterns into Strings
  - Do We Have a Clock Gene?
  - **Implanted Motif Problem**
  - Motif Finding Problem (Music Break)
  - Median String Problem
  - Greedy Motif Search
- How Rolling Dice Helps Us Find Regulatory Motifs
  - Randomized Motif Search
  - How do Bacteria Hibernate?
  - Gibbs Sampling
  - Pseudocounts

Randomly Implant **AAAAAAGGGGGGG** with 4 Random Mutations at 4 Random Positions

atgaccgggatactgat**A**g**AA**g**AAAGG**tt**GGG**ggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg

acccctatttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaacttttccgaata**c**AA**t**AAAA**c**GG**c**GGGa

tgagtatccctgggatgactt**AAAA**t**AA**t**GG**a**G**t**GG**tgctctcccgatttttgaatatgtaggatcattcgccagggtccga

gctgagaattggatg**c**AAAAAAAGGG**att**Gtccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga

tccctttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaat**A**t**AA**t**AAAGG**aa**GGG**cttatag

gtcaatcatgttcttgtgaatggattt**AA**c**AA**t**AAGGG**ct**GG**gaccgcttggcgcacccaaattcagtgtgggcgagcgcaa

cggttttggcccttgttagaggcccccgt**A**t**AAA**c**AAGG**a**GGG**ccaattatgagagagctaatctatcgcgtgcgtgttcat

aacttgagtt**AAAAAA**t**AGGG**a**G**cc**ctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta

ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcat**A**ct**AAAAAGG**a**G**c**GG**accgaaagggaag

ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagctt**A**ct**AAAAAGG**a**G**c**GG**a

Would the Frequent Words algorithm find this (15,4) motif?

Implanted Motifs Problem

Implanted Motif Problem. Finding *(k,d)*-motifs in a set of strings.

- **Input:** A set of strings *Dna*, and integers *k* (motif length) and *d* (maximal number of mismatches in a motif).

- **Output:** All *(k,d)*-motifs in *Dna*

Should we explore ALL 4^k *k*-mers to solve the Implanted Motif Problem?

Finding Implanted Motifs by Pairwise Comparison

atgaccgggatactgat**AgAAgAAAGGttGGG**ggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg

acccctatttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaacttttccgaata**cAAtAAAAcGGcGGG**a

tgagtatccctgggatgactt**AAAAtAAtGGaGtGG**tgctctcccgatttttgaatatgtaggatcattcgccagggtccga

gctgagaattggatg**cAAAAAAAGGGattG**tccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga

tccctttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaat**AtAAtAAAGGaaGGG**cttatag

gtcaatcatgttcttgtgaatggattt**AAcAAtAAGGGctGG**gaccgcttggcgcacccaaattcagtgtgggcgagcgcaa

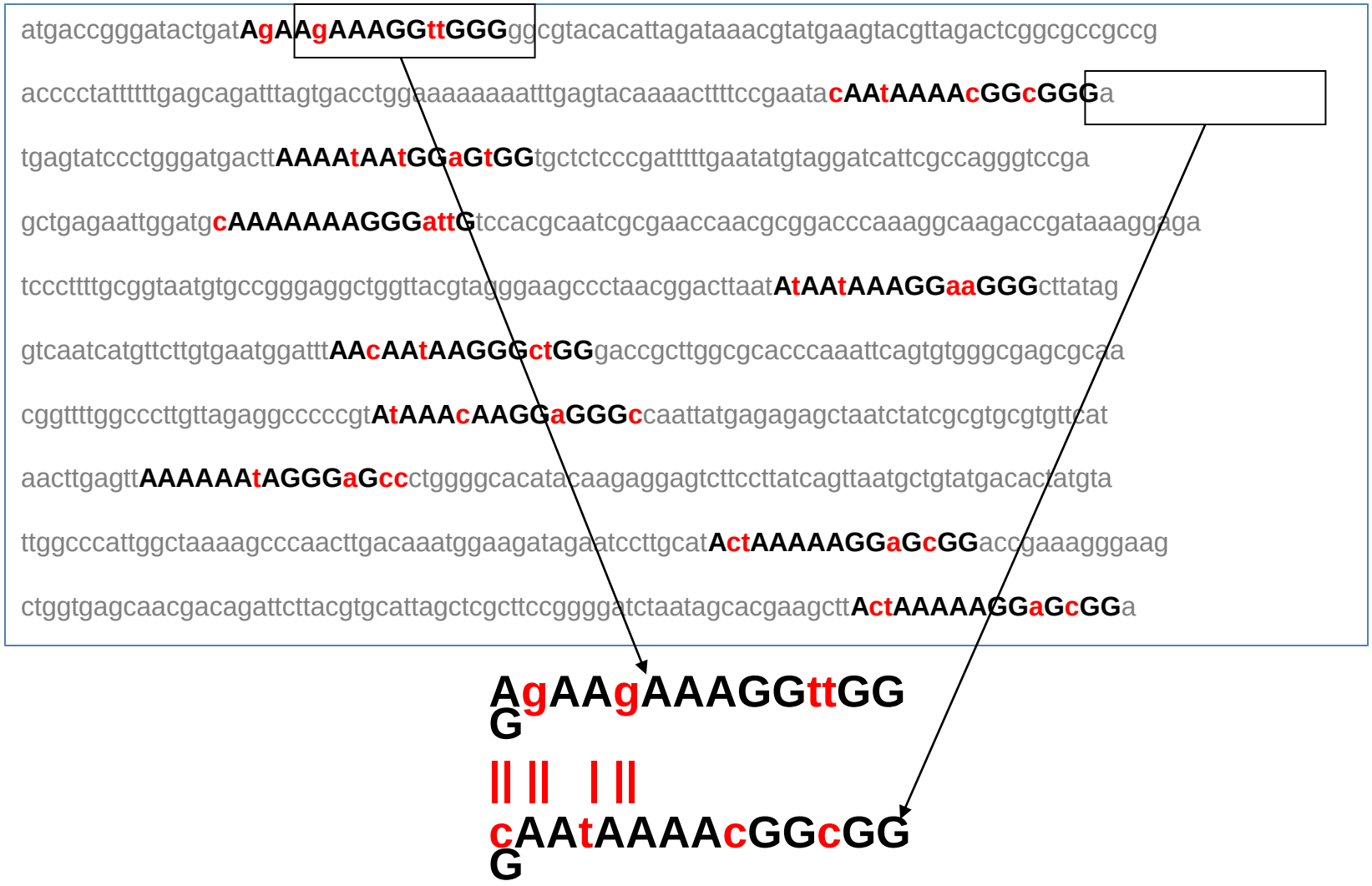
cggttttggcccttgttagaggcccccgt**AtAAAcAAGGaGGGc**caattatgagagagctaatctatcgcgtgcgtgttcat

aacttgagtt**AAAAAAtAGGGaGcc**ctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta

ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcat**ActAAAAAGGaGcGG**accgaaagggaag

ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagctt**ActAAAAAGGaGcGG**a

Finding Implanted Motifs by Pairwise Comparison



Why Pairwise Comparison Won't Work

AgAAgAAAGGttGGG *Implanted instance 1*

| | || 4 mismatches

AAAAAAAAGGGGGGGG *Pattern*

| | | | 4 mismatches

cAAtAAAAcGGGGGc *Implanted instance 2*

The implanted instances have up to **4 + 4 = 8 mutations** – **they don't even look like similar strings!**

Resorting to Motif Enumeration Instead

MotifEnumeration(*Dna*, *k*, *d*)
for each *k*-mer *a* in *Dna*
  for each *k*-mer *a'* differing from *a* by at most *d* mutations
    if *a'* is a *(k,d)*-mer
      output *a'*

Would this simple (albeit slow) algorithm work for finding **real** biological motifs?

MotifEnumeration assumes that **EACH** sequence has a *k*-mer similar to an (unknown) implanted pattern. This condition does not hold for noisy biological datasets.

Outline

- **From Implanted Patterns to Regulatory Motifs**
  - Implanting Patterns into Strings
  - Do We Have a Clock Gene?
  - Implanted Motif Problem
  - **Motif Finding Problem (Music Break)**
  - Median String Problem
  - Greedy Motif Search
- How Rolling Dice Helps Us Find Regulatory Motifs
  - Randomized Motif Search
  - How do Bacteria Hibernate?
  - Gibbs Sampling
  - Pseudocounts

From Motif to Consensus String

	T	**C**	**G**	**G**	**G**	**G**	G	**T**	**T**	**T**	T	T
	C	**C**	**G**	**G**	T	**G**	**A**	C	**T**	**T**	A	**C**
	A	**C**	**G**	**G**	**G**	**G**	**A**	**T**	**T**	**T**	T	**C**
	T	T	**G**	**G**	**G**	**G**	**A**	C	**T**	**T**	T	T
Motifs		A	A	**G**	**G**	**G**	**G**	**A**	C	**T**	**T**	**C**	**C**
	T	T	**G**	**G**	**G**	**G**	**A**	C	**T**	**T**	**C**	**C**
	T	**C**	**G**	**G**	**G**	**G**	**A**	**T**	**T**	C	A	T
	T	**C**	**G**	**G**	**G**	**G**	**A**	**T**	**T**	C	**C**	T
	T	A	**G**	**G**	**G**	**G**	**A**	A	C	**T**	A	**C**
	T	**C**	**G**	**G**	**G**	T	**A**	**T**	A	A	**C**	**C**

From Motif to Consensus String

	T C G G G G g T T T T t
	c C G G t G A c T T a C
	a C G G G G A T T T t C
	T t G G G G A c T T t t
Motifs	a a G G G G A c T T C C
	T t G G G G A c T T C C
	T C G G G G A T T c a t
	T C G G G G A T T c C t
	T a G G G G A a c T a C
	T C G G G t A T a a C C

Most **popular** nucleotide in each column is in **uppercase** and **bold**. Other (**unpopular**) nucleotides in each column are shown in **lowercase**.

Consensus(Motifs) **T C G G G G A T T T C C**

most popular symbol in each column

Score(Motifs) **3+ 4+ 0+ 0+ 1+ 1+ 1+ 5+ 2+ 3+ 6+ 4=30**

#unpopular symbols

A Different Way to Compute *Score(Motifs)*

	T C G G G G g T T T T t	3
	c C G G t G A c T T a C	4
	a C G G G G A T T T t C	2
	T t G G G G A c T T t t	4
Motifs	a a G G G G A c T T C C	3
	T t G G G G A c T T C C	2
	T C G G G G A T T c a t	3
	T C G G G G A T T c C t	2
	T a G G G G A a c T a C	4
	T C G G G t A T a a C C	3
		=30

#unpopular symbols (counted ROW-BY-ROW)

Hamming distance from *Consensus* to each motif:

$d(Consensus, Motif_i)$

Consensus(Motifs) T C G G G G A T T T C C

most popular symbol each column

Score(Motifs) 3+ 4+ 0+ 0+ 1+ 1+ 1+ 5+ 2+ 3+ 6+ 4=30

#unpoplular symbols

Hamming distance: number of mismatches between *k*-mers.

GATTCTCA
|| |
GACGCTGA

d(GATTCTCA, GACGCTGA) = **3**

The Motif Finding Problem

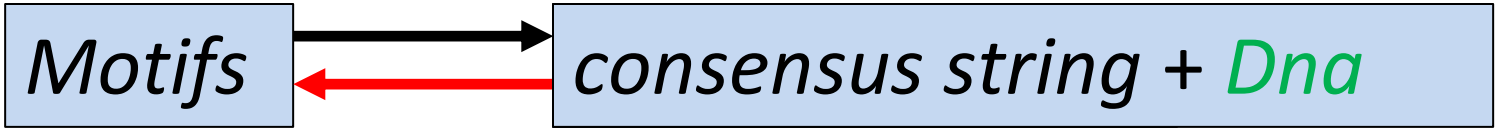
Motif Finding Problem. Given a set of sequences, find a set of *k*-mers (one from each sequence) with minimal score among all choices of *k*-mers.

- **Input:** A set of sequences *Dna* and integer *k*.

- **Output:** A set of *k*-mers *Motifs*, one from each sequence in *Dna*, minimizing *Score(Motifs)*.

BruteForceMotifSearch:
iterates through all sets of *k*-mers (**runtime:** $n^t \cdot k \cdot t$)

Reformulating the Motif Finding Problem



What is the distance between a *k-mer* and $Motifs=\{Motif_1, \ldots, Motif_t\}$?

Define *d*(*k-mer*, *Motifs*) as the sum of Hamming distances between a *k-mer* and each $Motif_i$:

$$d(k\text{-}mer, Motifs) = \sum_{i=1,t} d(k\text{-}mer, Motif_i)$$

What is *d(consensus(Motifs), Motifs)?*

Score(Motifs) = d(Consensus(Motifs), Motifs)

	T C G G G G g T T T T t	3
	c C G G t G A c T T a C	4
	a C G G G G A T T T t C	2
	T t G G G G A c T T t t	4
Motifs	a a G G G G A c T T C C	3
	T t G G G G A c T T C C	2
	T C G G G G A T T c a t	3
	T C G G G G A T T c C t	2
	T a G G G G A a c T a C	4
	T C G G G t A T a a C C	3
		=30

Consensus(Motifs) T C G G G G A T T T C C

Score(Motifs) 3+ 4+ 0+ 0+ 1+ 1+ 1+ 5+ 2+ 3+ 6+ 4=30

#unpopular symbols (ROW-BY-ROW)

Hamming distance from *Consensus* to each motif:

distance(Consensus,Motif$_i$)

#unpopular symbols (COLUMN-BY-COLUMN)

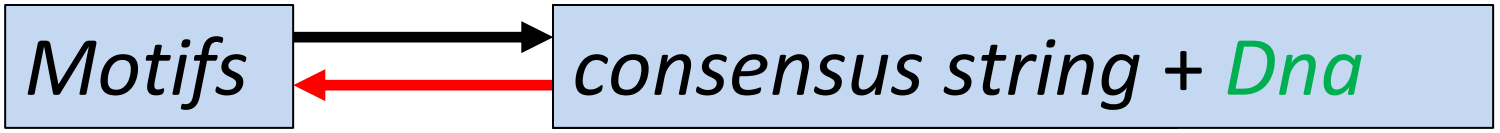
Score(*Motifs*)=

unpopular symbols counted column-by-column =

unpopular symbols counted row-by-row=

d(*Consensus*(*Motifs*), *Motifs*)

Yet Another (Equivalent) Motif Finding Problem



Search for *Motifs in Dna* minimimizing *Score(Motifs)*

Search for a *k*-mer minimizing *d(k-mer,Motifs)* among all possible *k*-mers and all possible *Motifs in Dna*.

Equivalent Motif Finding Problem. Find a *k*-mer *Pattern* and a set of *k*-mers *Motifs* in *Dna* that minimizes the distance between all possible choices of *Pattern* and *Motifs*.

- **Input:** A set of sequences *Dna* and integer *k*.
- **Output:** A *k*-mer *Pattern* and a set of *k*-mers *Motifs* in *Dna* minimizing *d(Pattern, Motifs)*.

Have We Just Made Our Task More Difficult?

Motif Finding Problem. Find a set of *k*-mers *Motifs*, one from each sequence in *Dna*, minimizing *Score(Motifs)*.

Equivalent Motif Finding Problem. Find *k*-mer *Pattern* **AND** a set of *k*-mers *Motifs*, one from each sequence in *Dna*, minimizing *d(Pattern, Motifs)*.

The key insight for solving the Equivalent Motif Finding Problem: given *Pattern*, we can quickly find *Motifs* minimizing *d(Pattern,Motifs)*.

No need to explore all *Motifs* like in **Motif Finding Problem**!
Instead, the **Median String Problem** explores all *k*-mers *Pattern*

Outline

- **From Implanted Patterns to Regulatory Motifs**
  - Implanting Patterns into Strings
  - Do We Have a Clock Gene?
  - Implanted Motif Problem
  - Motif Finding Problem (Music Break)
  - **Median String Problem**
  - Greedy Motif Search
- How Rolling Dice Helps Us Find Regulatory Motifs
  - Randomized Motif Search
  - How do Bacteria Hibernate?
  - Gibbs Sampling
  - Pseudocounts

Distance between a *k*-mer and a (longer) String

Distance *d*(GATTCTCA, GCAAAGACGCTGACCAA) = ?

Distance: **7**

d(*Pattern*, *String*):
minimum distance between *Pattern* and all *k*-mers in *String*

Distance between a *k*-mer and a (longer) String

d(GATTCTCA, GCAAAGACGCTGACCAA) = ?

Distance: 7 **6**

d(*Pattern*, *String*):
minimum distance between *Pattern* and all *k*-mers in *String*

Distance between a *k*-mer and a (longer) String

d(GATTCTCA, GCAAAGACGCTGACCAA) = ?

Distance: 7 6 **7**

d(*Pattern*, *String*):
minimum distance between *Pattern* and all *k*-mers in *String*

Distance between a *k*-mer and a (longer) String

d(GATTCTCA, GCAAAGACGCTGACCAA) = ?

Distance: 7 6 7 **5**

d(*Pattern*, *String*):
minimum distance between *Pattern* and all *k*-mers in *String*

Distance between a *k*-mer and a (longer) String

d(GATTCTCA, GCAAAGACGCTGACCAA) = ?

Distance: 7 6 7 5 **8**

d(*Pattern*, *String*):
minimum distance between *Pattern* and all *k*-mers in *String*

Distance between a *k*-mer and a (longer) String

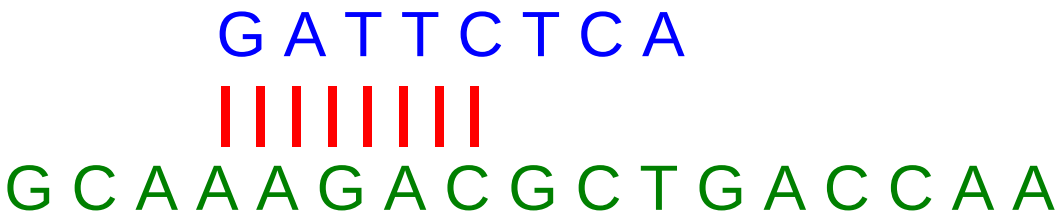
d(GATTCTCA, GCAAAGACGCTGACCAA) = ?

Distance: 7 6 7 5 8 **3**

d(*Pattern*, *String*):
minimum distance between *Pattern* and all *k*-mers in *String*

Distance between a *k*-mer and a (longer) String

d(GATTCTCA, GCAAAGACGCTGACCAA) = ?



Distance: 7 6 7 5 8 3 **8**

d(*Pattern*, *String*):
minimum distance between *Pattern* and all *k*-mers in *String*

Distance between a *k*-mer and a (longer) String

d(GATTCTCA, GCAAAGACGCTGACCAA) = ?

Distance: 7 6 7 5 8 3 8 **7**

d(*Pattern*, *String*):
minimum distance between *Pattern* and all *k*-mers in *String*

Distance between a *k*-mer and a (longer) String

d(GATTCTCA, GCAAAGACGCTGACCAA) = ?

Distance: 7 6 7 5 8 3 8 7 **4**

d(*Pattern*, *String*):
minimum distance between *Pattern* and all *k*-mers in *String*

Distance between a *k*-mer and a (longer) String

d(GATTCTCA, GCAAAGACGCTGACCAA) = ?

Distance: 7 6 7 5 8 3 8 7 4 **6**

d(*Pattern*, *String*):
minimum distance between *Pattern* and all *k*-mers in *String*

Distance between a *k*-mer and a (longer) String

d(GATTCTCA, GCAAAGACGCTGACCAA) = **3**

Distance: 7 6 7 5 8 **3** 8 7 4 6

d(*Pattern*, *String*):

minimum distance between *Pattern* and all *k*-mers in *String*

Distance between a *k*-mer and a **Set** of Strings

Distance between a *k-mer* and a *set of strings* $Dna = \{Dna_1, \ldots, DNA_t\}$:

$$d(k\text{-}mer, Dna) = \sum_{\text{all strings in } Dna} d(k\text{-}mer, Dna_i)$$

Pattern = AAA

ttaccttAAc **1**
gAtAtctgtc **1**
Acggcgttcg **2**
ccctAAAgag **0**
cgtcAgAggt **1**

d(AAA, *Dna*)= **5**

A **median string** for the set of strings *Dna:*
a *k-mer* minimizing distance *d*(*k-mer*, *Dna*) over all possible *k*-mers.

Median String Problem

Median String Problem. Finding a median string.

- **Input:** A set of sequences *Dna* and an integer *k*.
- **Output:** A *k-mer* minimizing distance *d(k-mer, Dna)* among all *k*-mers.

```
MedianString(Dna, k)
   best-k-mer ← AAA · · · AA
   for each k-mer from AAA · · · AA to TTT · · · TT
      if d(k-mer, Dna) < distance(best-k-mer, Dna)
         best-k-mer ← k-mer
   return(best-k-mer)
```

Runtime: $4^k \cdot n \cdot t \cdot k$ (for *Dna* with *t* sequences of length *n*).

Motif Finding Problem
versus
Median String Problem

Runtime: $n^t \cdot k \cdot t$

Runtime: $4^k \cdot n \cdot t \cdot k$

Motif Finding Problem = Median String Problem

Motif Finding Problem
versus
Median String Problem

Runtime: $n^t \cdot k \cdot t$

Runtime: $4^k \cdot n \cdot t \cdot k$

Although **MedianString** is much faster than **BruteForceMotifSearch**, it is still slow for large *k.*

Outline

- **From Implanted Patterns to Regulatory Motifs**
  - Implanting Patterns into Strings
  - Do We Have a Clock Gene?
  - Implanted Motif Problem
  - Motif Finding Problem (Music Break)
  - Median String Problem
  - **Greedy Motif Search**
- How Rolling Dice Helps Us Find Regulatory Motifs
  - Randomized Motif Search
  - How do Bacteria Hibernate?
  - Gibbs Sampling
  - Pseudocounts

From Motifs to Profile

Motifs

T C G G G G g T T T t t
c C G G t G A c T T a C
a C G G G G A T T T t C
T t G G G G A c T T t t
a a G G G G A c T T C C
T t G G G G A c T T C C
T C G G G G A T T c a t
T C G G G G A T T c C t
T a G G G G A a c T a C
T C G G G T A T a a C C

Profile(Motifs)

A: .2 .2 0 0 0 0 **.9** .1 .1 .1 .3 0
C: .1 **.6** 0 0 0 0 0 .4 .1 .2 **.4** **.6**
G: 0 0 **1** **1** **.9** **.9** .1 0 0 0 0 0
T: **.7** .2 0 0 .1 .1 0 **.5** **.8** **.7** .3 .4

frequency of nucleotide *i* in column *j*

T C G G G G A C T T C C

Each column of a profile represents a biased 4-sided **die** with A, C, G, and T on its sides. Thus, a profile corresponds to *k* dice.

What is the probability that *k* rolls of such dice generate a given string ?

Scoring *k*-mers with a *Profile*

Given the following *Profile*:

A	1/2	7/8	3/8	0	1/8	0
C	1/8	0	1/2	5/8	3/8	0
T	1/8	1/8	0	0	1/4	7/8
G	1/4	0	1/8	3/8	1/4	1/8

The probability of the consensus string:

Pr(**AAACCT**|*Profile*) = ???

Scoring *k*-mers with a *Profile*

Given the following *Profile*:

A	**1/2**	**7/8**	**3/8**	0	1/8	0
C	1/8	0	1/2	**5/8**	**3/8**	0
T	1/8	1/8	0	0	1/4	**7/8**
G	1/4	0	1/8	3/8	1/4	1/8

The probability of the consensus string:

Pr(**AAACCT**|*Profile*) =

1/2 x 7/8 x 3/8 x 5/8 x 3/8 x 7/8 =

0.033646

Scoring *k*-mers with a *Profile*

Given the following *Profile*:

A	**1/2**	7/8	**3/8**	0	**1/8**	0
C	1/8	0	1/2	**5/8**	3/8	0
T	1/8	**1/8**	0	0	1/4	7/8
G	1/4	0	1/8	3/8	1/4	**1/8**

The probability of another string:

Pr(**AAACCT**|*Profile*) =

1/2 x 7/8 x 3/8 x 5/8 x 3/8 x 7/8 = 0.033646

Pr(**ATACAG** |*Profile*) =

The closer *k*-mer is to the consensus string, the larger Pr(*k*-mer| *Profile*) is.

What is the *Profile*-most probable 6-mer in CTATAAACCTTACAT?

A	**1/2**	**7/8**	**3/8**	0	1/8	0
C	1/8	0	1/2	**5/8**	**3/8**	0
T	1/8	1/8	0	0	1/4	**7/8**
G	1/4	0	1/8	3/8	1/4	1/8

***Profile*-most probable *k*-mer in a sequence**: the *k*-mer with the highest

Pr(*k-mer* | *Profile*)

among all *k*-mers in this sequence.

6-mer	**Prob(6-mer|*Profile*)**	
CTATAAACCTTACAT	1/8 x 1/8 x 3/8 x 0 x 1/8 x 0	0
C**TATAAA**CCTTACAT	1/2 x 7/8 x 0 x 0 x 1/8 x 0	0
CT**ATAAAC**CTTACAT	1/2 x 1/8 x 3/8 x 0 x 1/8 x 0	0
CTA**TAAACC**TTACAT	1/8 x 7/8 x 3/8 x 0 x 3/8 x 0	0
CTATAAACCTTACAT	1/2 x 7/8 x 3/8 x 5/8 x 3/8 x 7/8	**.0336**
CTATA**AACCTT**ACAT	1/2 x 7/8 x 1/2 x 5/8 x 1/4 x 7/8	.0299
CTATAA**ACCTTA**CAT	1/2 x 0 x 1/2 x 0 1/4 x 0	0
CTATAAA**CCTTAC**AT	1/8 x 0 x 0 x 0 x 0 x 1/8 x 0	0
CTATAAAC**CTTACA**T	1/8 x 1/8 x 0 x 0 x 3/8 x 0	0
CTATAAACC**TTACAT**	1/8 x 1/8 x 3/8 x 5/8 x 1/8 x 7/8	.0004

GreedyMotifSearch

Given a set of *k*-mers $Motif_1, \ldots, Motif_{i-1}$ in the first *i-1* sequences in *Dna*:

- form *Profile* from them
- form $Motif_1, \ldots Motif_{i-1}, Motif_i$ by adding the *Profile*-most probable *k*-mer $Motif_i$ in the *i*-th sequence to the growing set of *k*-mers

ITERATE

Outline

- From Implanted Patterns to Regulatory Motifs
  - Implanting Patterns into Strings
  - Do We Have a Clock Gene?
  - Implanted Motif Problem
  - Motif Finding Problem (Music Break)
  - Median String Problem
  - Greedy Motif Search
- **How Rolling Dice Helps Us Find Regulatory Motifs**
  - **Randomized Motif Search**
  - How do Bacteria Hibernate?
  - Gibbs Sampling
  - Pseudocounts

From Motifs to Profile to Motifs to Profile to Motifs…

Given *Motifs*, we can construct

Profile(*Motifs*)

Given an arbitrary *Profile* and a set of sequences *Dna*, we can construct

Motifs(*Profile*, *Dna*)

as a set of *Profile*-most probable *k*-mers in each sequence from *Dna*.

Iterate!

Motifs(*Profile*(*Motifs*(*Profile*(*Motifs*),*Dna*)),*Dna*)),*Dna*)…

RandomizedMotifSearch

```
RandomizedMotifSearch(Dna, k, t)
    randomly select k-mers Motifs = (Motif_1, … ,Motif_t) in each  string from DNA
    bestMotifs  ← Motifs
    while forever
      Profile ←  Profile(Motifs)
      Motifs ← Motifs(Profile, Dna)
      if Score(Motifs) < Score(bestMotifs)
        bestMotifs ← Motifs
      else
        return(bestMotifs)
```

How in the World Can This Algorithm Find Anything?

When we form *k*-mers *randomly*, it results in a **uniform** expected *Profile* with every entry 1/4. Such a uniform profile is useless because it does not provide any clues about the implanted motif.

RandomizedMotifSearch in Action

Dna with implanted (4,1)-motif ACGT	Randomly select *Motifs* (in **bold**)	*Motifs*	*Profile*(*Motifs*)

ttACCTtaac
gATGTctgtc
ccgGCGTtag
cactaACGAg
cgtcagAGGT

ttACCT**taac**
gAT**GTct**gtc
ccgGCGTtag
c**acta**ACGAg
cgtcag**AGGT**

t	a	a	c
G	**T**	c	t
c	c	g	**G**
a	c	t	a
A	**G**	**G**	**T**

A:	1/5	1/5	2/5	1/5
C:	1/5	2/5	1/5	1/5
G:	1/5	1/5	2/5	1/5
T:	1/5	1/5	1/5	2/5

.0016/ttAC .0016/tACC **.0128**/ACCT .0064/CCT**t** .0016/Ct**ta** .0016/T**taa** .0016/**taac**

.0016/gAT**G** **.0128**/AT**GT** .0016/T**GTc** .0032/**GTct** .0032/**Tct**g .0032/**ct**gt .0016/**t**gtc

.0064/**ccgG** .0036/**cgG**C .0016/**gG**CG **.0128**/**GCGT** .0032/CGTt .0016/Gtta .0016/Ttag

.0032/c**act** .0064/**acta** .0016/**cta**A .0016/**ta**AC .0032/**a**ACG **.0128**/ACGA .0016/CGAg

.0016/cgtc .0016/gtca .0016/tcag .0032/cag**A** .0032/ag**AG** .0032/g**AGG** **.0128**/**AGGT**

tt**ACCT**taac
g**ATGT**ctgtc
ccg**GCGT**tag
cacta**ACGA**g
cgtcag**AGGT**

Motifs (*Profile* (*Motifs*), *Dna*)

How in the World Can This Algorithm Find Anything?

When we select *k*-mers *randomly*, it results in a **uniform** expected *Profile* where every entry is 1/4. Such a uniform profile is useless because it does not provide any clues about the implanted motif.

But strings in *Dna* are not truly random since they include implanted motifs!

These implanted motifs result in a **biased** expected *Profile.*

Where does this statistical bias point to?

RandomizedMotifSearch in Action

DNA with implanted (4,1)-motif ACGT	Randomly select ***Motifs*** (in bold)	***Motifs***	*Profile*(***Motifs***)

```
ttACCTtaac
gATGTctgtc
ccgGCGTtag
cactaACGAg
cgtcagAGGT
```

```
ttACCTtaac
gATGTctgtc
ccgGCGTtag
cactaA
cgtcaAGGT
```

```
t a a c
G T c t
c c g G
a c t a
A G G T
```

A:	**2/5**	1/5	1/5	1/5
C:	1/5	**2/5**	1/5	1/5
G:	1/5	1/5	**2/5**	1/5
T:	1/5	1/5	1/5	**2/5**

The statistical bias points in the direction of the implanted (4,1)-motif ACGT because one of the randomly selected 4-mers happened to be the implanted pattern AGGT simply by chance.

Outline

- **From Implanted Patterns to Regulatory Motifs**
  - Implanting Patterns into Strings
  - Do We Have a Clock Gene?
  - Implanted Motif Problem
  - Motif Finding Problem (Music Break)
  - Median String Problem
  - Greedy Motif Search
- How Rolling Dice Helps Us Find Regulatory Motifs
  - Randomized Motif Search
  - **How do Bacteria Hibernate?**
  - Gibbs Sampling
  - Pseudocounts

Entropy of a Motif

	A:	.2	.2	0	0	0	0	.9	.1	.1	.1	.3	0
	C:	.1	.6	0	0	0	0	0	.4	.1	.2	.4	.6
Profile	G:	0	0	1	1	.9	.9	.1	0	0	0	0	0
	T:	.7	.2	0	0	.1	.1	0	.5	.8	.7	.3	.4

Every column of the profile matrix is a **probability distributio**

Entropy of a probability distribution $(p_1, \ldots, p_n)$ is

$$-\sum p_j \cdot \log_2(p_j)$$

through all probabilities

Entropy of **uniform** distribution **(¼, ¼, ¼, ¼)** is $-4 \cdot \frac{1}{4} \log_2(\frac{1}{4}) = 2$ **(maximal entropy).**

Entropy of a **completely conserved** column like **(1, 0, 0, 0)** is **0 (minimal**

The entropy varies from **0** to **2**, entropy of the 1st column (0.2, 0.1, 0.0, 0.7):

$$-(0.2 \cdot \log_2 0.2 + 0.1 \cdot \log_2 0.1 + 0.0 \cdot \log_2 0.0 + 0.7 \cdot \log_2 0.7) \approx 1.147$$

Entropy of a motif – sum of entropies of columns of its profile matrix.

Motif Logo

	A:	.2	.2	0	0	0	0	**.9**	.1	.1	.1	.3	0
	C:	.1	**.6**	0	0	0	0	0	.4	.1	.2	**.4**	**.6**
Profile(Motifs)	G:	0	0	**1**	**1**	**.9**	**.9**	.1	0	0	0	0	0
	T:	**.7**	.2	0	0	.1	.1	0	**.5**	**.8**	**.7**	.3	.4

Consensus(Motifs) **T C G G G G A T T T C C**

motif logo

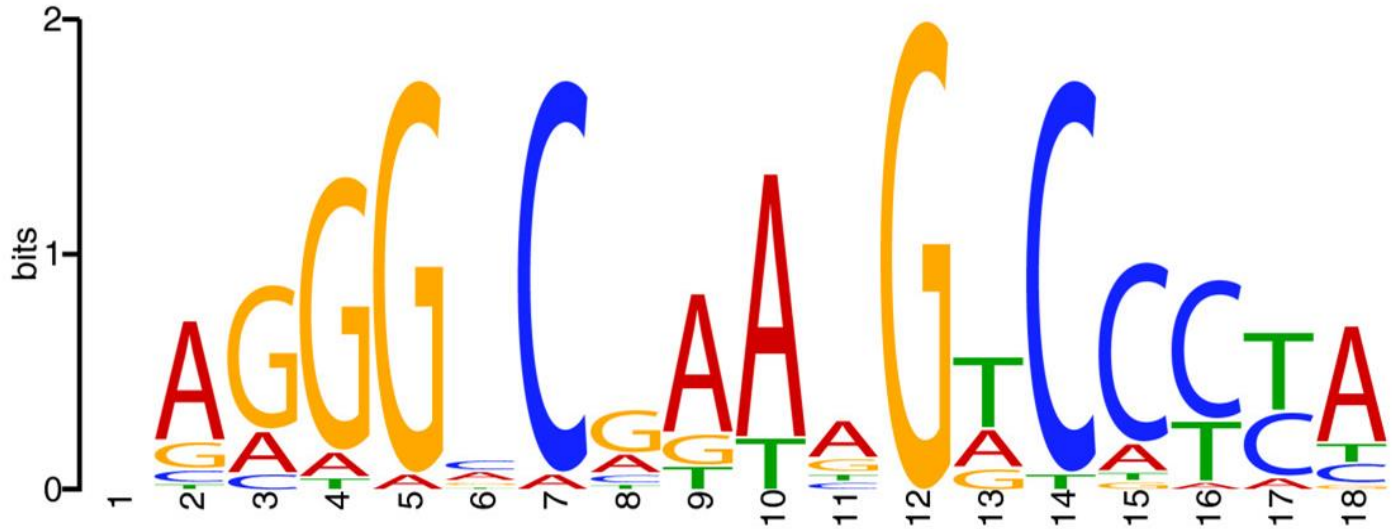
The **total height** of 4 nucleotides in each position: **2 - entropy.**

The **relative sizes** of 4 nucleotides in each position: frequencies in the column.

Predicted vs. Real Dormancy Survival Regulator Binding Site

Length 20 motif returned by **RandomizedMotifSearch**:

CGGGGGACCTACGGCCCTAG



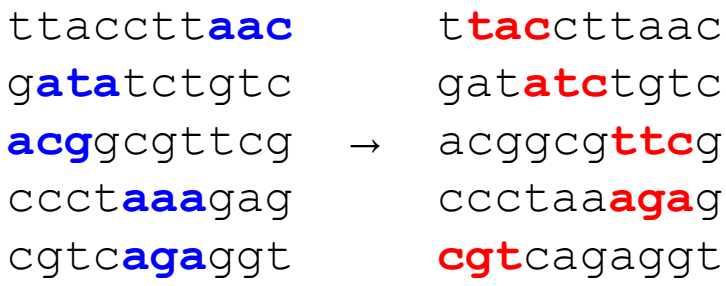
Motif logo of real Dormancy Survival Regulator binding sites

Outline

- **From Implanted Patterns to Regulatory Motifs**
  - Implanting Patterns into Strings
  - Do We Have a Clock Gene?
  - Implanted Motif Problem
  - Motif Finding Problem (Music Break)
  - Median String Problem
  - Greedy Motif Search
- How Rolling Dice Helps Us Find Regulatory Motifs
  - Randomized Motif Search
  - **Gibbs Sampling**
  - Pseudocounts

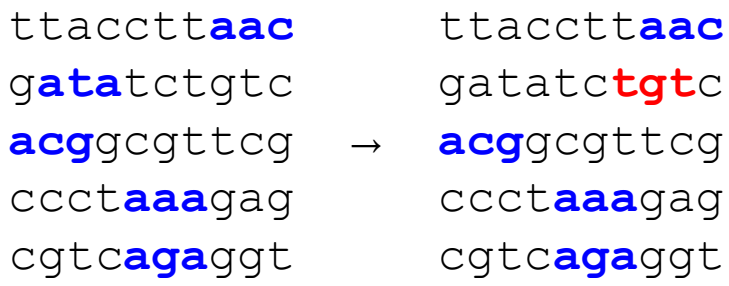
Gibbs Sampling

RandomizedMotifSearch may replace all *k*-mers in a single iteration and thus may potentially discard a nearly correct motif.



RandomizedMotifSearch
(may change **all** *k*-mers in 1 iteration)

Gibbs sampling replaces a single *k*-mer at each iteration and thus moves with more caution in the space of all motifs.



GibbsSampler
(changes a **single** *k*-mer in 1 iteration)

GibbsSampler in Action

Randomly select ***Motifs*** (in **bold**)

ttACCT**taac**
gAT**GTct**gtc
ccg**GCGT**tag
c**acta**ACGAg
cgtcag**AGGT**

Randomly remove a *k*-mer in ***Motifs***

ttACCT**taac**
gAT**GTct**gtc
---
c**acta**ACGAg
cgtcag**AGGT**

Motifs matrix

t	**a**	**a**	**c**
G	**T**	**c**	**t**
a	**c**	**t**	**a**
A	**G**	**G**	**T**

Choose a new starting position in the deleted sequence based on rolling the die

Count matrix

A:	**2**	1	1	1
C:	**0**	1	1	1
G:	1	1	1	**0**
T:	1	1	1	**2**

Calculate the probabilities of all *k*-mers in the deleted string **ccgG**CGTtag

Profile matrix

A:	1/4			
C:	**0**	1/4	1/4	1/4
G:	1/4	1/4	1/4	**0**
T:	1/4	1/4	1/4	2/4

0 (**ccgG**) **0** (**cgG**C) **0** (**gGCG**) **1/128** (**G**CGT) **0** (CGTt) **0** (GTta) **0** (Ttag)

Roll a 7-sided die with probabilities of sides proportional to proba
mers

ITE

Gibbs Sampler Outline

1. Form *Motifs* by randomly selecting a *k*-mer in each sequence
2. Randomly choose one of selected *k*-mers (from *RemovedSequence)* and remove it from *Motifs*.
3. Create *Profile* from the remaining *k*-mers in *Motifs*.
4. For each *k*-mer in *RemovedSequence*, calculate *Pr(k-mer|Profile)* resulting in *n-k+1* probabilities:

   $$p_1, p_2, \ldots, p_{n-k+1}.$$

5. Roll a die (with *n-k+1* sides) where probability of ending up at side *i* is proportional to p_i.
6. Choose a new starting position based on rolling the die. Add the *k*-mer starting at this position in *RemovedSequence* to *Motifs*.
7. **Repeat steps 2-6.**

Gibbs Sampler in Action

Randomly select ***Motifs*** (in **bold**)

ttACCT**taac**
gAT**GTct**gtc
ccgGCGTtag
c**acta**ACGAg
cgtcag**AGGT**

Randomly remove a *k*-mer in ***Motifs***

ttACCT**taac**
gAT**GTct**gtc
---
c**acta**ACGAg
cgtcag**AGGT**

Motifs matrix

t	**a**	**a**	**c**
G	**T**	**c**	**t**
a	**c**	**t**	**a**
A	**G**	**G**	**T**

Count matrix

A:	**2**	1	1	1
C:	**0**	1	1	1
G:	1	1	1	**0**
T:	1	1	1	**2**

Calculate the probabilities of all *k*-mers in the deleted string **ccgG**CGTtag

Profile matrix

A:	1/4	1/4	1/4	**0**
C:	**0**	1/4	1/4	1/4
G:	1/4	1/4	1/4	**0**
T:	1/4	1/4	1/4	**2/4**

0 (**ccgG**) **0** (**cgG**C) **0** (**gGCG**) **1/128** (**G**CGT) **0** (CGTt) **0** (GTta) **0** (Ttag)

Should we roll a one-sided

Outline

- **From Implanted Patterns to Regulatory Motifs**
  - Implanting Patterns into Strings
  - Do We Have a Clock Gene?
  - Implanted Motif Problem
  - Motif Finding Problem (Music Break)
  - Median String Problem
  - Greedy Motif Search
- How Rolling Dice Helps Us Find Regulatory Motifs
  - Randomized Motif Search
  - How do Bacteria Hibernate?
  - Gibbs Sampling
  - **Pseudocounts**

Laplace's Rule of Succession

In small datasets, there is always a chance that a possible event does not occur (e.g., zeroes in *Count* matrix).

Randomized algorithms do not like zeroes and introduce **pseudocounts** that inflate the probabilities of rare events and eliminate empirical zero-frequencies.

Laplace calculated the probability that the sun will not rise tomorrow, given that it has risen every day for the past 5000 years (1 in 1826251)

His estimate was ridiculed because his opponents failed to understand its importance.

If we repeat an experiment that can result in a success or failure ***n*** times and get ***s*** successes, what is the probability that the next repetition will be a success?

Pseudocounts

If $X_1, \ldots, X_{n+1}$ are *conditionally* independent random boolean variables (failure 0 or success 1) then:

$$\Pr(X_{n+1}=1 \mid X_1+\ldots+X_n=s)=(s+1)/(n+2)$$

Since we have the prior knowledge that both success and failure are possible, we essentially made ***n+2*** rather than ***n*** observations since we (implicitly) observed one success and one failure before we even started the experiments.

Thus, we have made ***n+2*** observations (known as **pseudocounts**) with ***s+1*** successes.

What would Laplace Do to Get Rid of Zeroes?

```
ttACCTtaac
gATGTctgtc
ccgGCGTtag
cactaACGAg
cgtcagAGGT
```

```
ttACCTtaac
gATGTctgtc
----------
cactaACGAg
cgtcagAGGT
```

```
t    a    a    c
G    T    c    t
---------------
a    c    t    a
A    G    G    T
```

Choose a new starting position in the deleted sequence based on rolling the die

Update *Count* Matrix with pseudocounts

```
A:   2+1  1+1  1+1  1+1
C:   0+1  1+1  1+1  1+1
G:   1+1  1+1  1+1  0+1
T:   1+1  1+1  1+1  2+1
```

Recalculate the probabilities of all 4-mers in the deleted string **ccgG**CGTtag

Update *Profile* Matrix

```
A:   3/8  2/8  2/8  2/8
C:   1/8  2/8  2/8  2/8
G:   2/8  2/8  2/8  1/8
T:   2/8  2/8  2/8  3/8
```

1/1024 (ccgG) 2/1024 (cgGC) 2/1024 (gGCG) **6/1024 (GCGT)** 3/1024 (CGTt) 4/1024 (GTta) 4/1024 (Ttag)

Roll a 7-sided rather than 1-sided dice!

Gibbs Sampler Continues

Randomly select ***Motifs*** (in bold)

```
ttACCTtaac
gATGTctgtc
ccgGCGTtag
cactaACGAg
cgtcagAGGT
```

Randomly remove a *k*-mer in ***Motifs***

```
----------
gATGTctgtc
ccgGCGTtag
cactaACGAg
cgtcagAGGT
```

Motifs matrix

```
------------
G   T   c   t
G   C   G   T
a   c   t   a
A   G   G   T
```

Choose a new starting position in the deleted sequence based on rolling the die

Pseudocount Matrix

```
A:  2+1 0+1 0+1 1+1
C:  0+1 2+1 1+1 0+1
G:  2+1 1+1 2+1 0+1
T:  0+1 1+1 1+1 3+1
```

Calculate the probabilities of all *k*-mers in the deleted string ttACCT**taac**

Construct *Profile* using Laplace's Rule

```
A:  3/8 1/8 1/8 2/8
C:  1/8 3/8 2/8 1/8
G:  3/8 2/8 3/8 1/8
T:  1/8 2/8 2/8 4/8
```

2/4096 (ttAC) 2/4096 (tACC) 72/4096 (ACCT) 24/4096 (CCT**t**) 8/4096 (C**Tta**) 4/4096 (**Ttaa**) 1/4096 (**taac**)

Roll a 7-sided die

Gibbs Sampler Continues and Continues

Randomly select *Motifs* (in **bold**)

```
ttACCTtaac
gATGTctgtc
ccgGCGTtag
cactaACGAg
```

Randomly remove a *k*-mer in ***Motifs***

```
ttACCTtaac
gATGTctgtc
ccgGCGTtag
----------
```

Motifs matrix

```
A   C   C   T
G   T   c   t
G   C   G   T
------------
```

We have already captured four out of five correct 4-mers and will almost surely find all implanted motifs at one of the next iterations.

Calculate the probabilities of all *k*-mers in the deleted string ttACCT**taac**

Create *Profile* using Laplace's Rule

```
A: 3/8 1/8 1/8 1/8
C: 1/8 3/8 3/8 1/8
G: 3/8 2/8 3/8 1/8
T: 1/8 2/8 1/8 5/8
```

2/4096 (ttAC) 2/4096 (tACC) 72/4096 (ACCT) 24/4096 (CCT**t**) 8/4096 (C**Tta**) 4/4096 (**Ttaa**) 1/4096 (**taac**)

Roll a 7-sided die