Lecture 5: Finding Regulatory Motifs Within DNA Sequences

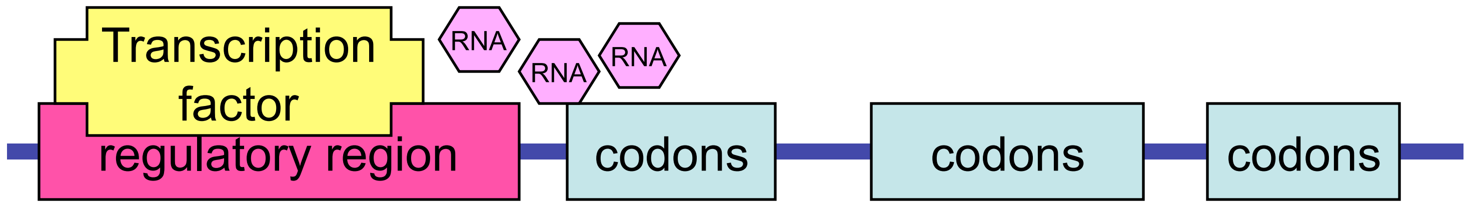
Study Chapter 4.4-4.9

Initiating Transcription

- As a precursor to transcription (the reading of DNA to construct RNAs that eventually leading to protein synthesis) special proteins bind to the DNA, separate it to enable its reading.
- How do these proteins know where the coding genes are in order to bind?
- Genes are relatively rare
  - O(1,000,000,000) bases/genome
  - O(10000) genes/genome
  - O(1000) bases/gene
- Approximately 1% of DNA codes for genes ($10^3 10^4/10^9$)

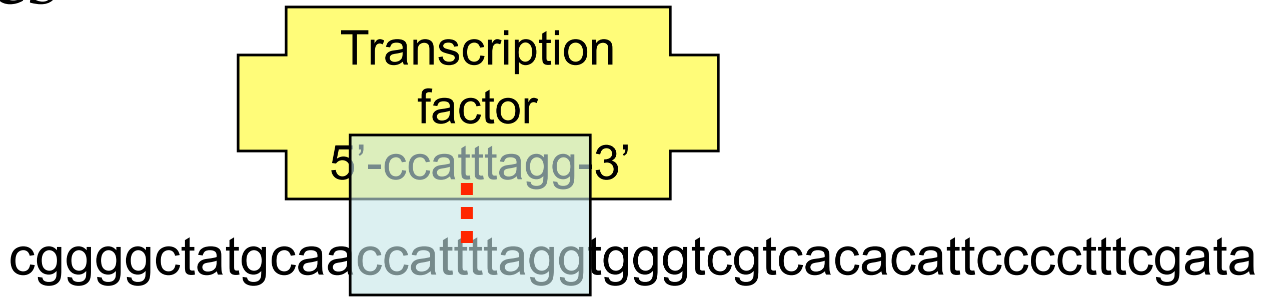
Regulatory Regions

- RNA polymerases seek out *regulatory* or *promoting* regions located 100-1000 bp upstream from the coding region
- They work in conjunction with special proteins called *transcription factors* whose presence enables gene expression
- Within these regions are the *Transcription Factor Binding Sites* (TFBS), special DNA sequence patterns known as ***motifs*** that are specific to a given transcription factor



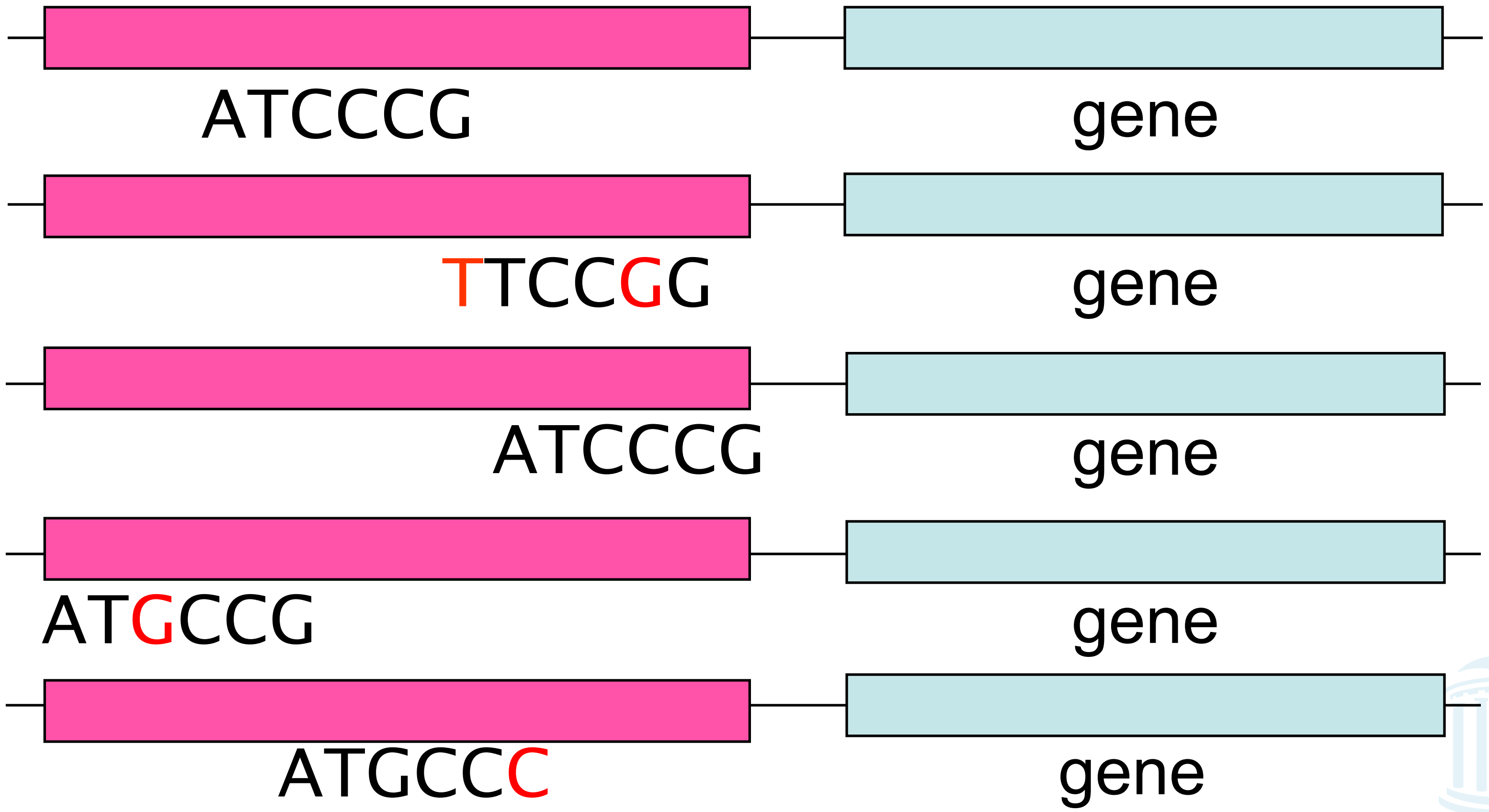
Transcription Factor Binding Sites

- A TFBS can be located anywhere within the regulatory region.
- TFBS may vary slightly across different regulatory regions since non-essential bases could mutate
- Transcription factors are robust (they will still bind) in the presence of small changes in a few bases



Motifs and Transcriptional Start Sites

Motif *(n)* - A repeated structural element in architecture or decoration



Identifying Motifs: Complications

- We do not know the motif sequence for every TF
- We do not know where it is located relative to the gene's start
- Motifs can differ slightly from one gene to the next
- We only know that it occurs frequently
- How to discern a Motif's frequent "similar" pattern from "random" patterns?

An Aside: Solving Cryptograms

- A popular form of word puzzle

  N oucgupju dlgw ynouo nwu sbu ynoho ld n jlzu dlw eupuo, xbhjb, snqup hp swhvmuo, zusuwihpuo vwlsuhp oucgupjuo.

- Based on letter, multi-letter, and word frequency it is not hard to figure out the most likely answer.

- Try solving it using

  http://rumkin.com/tools/cipher/cryptogram-solver.php

How's a Motif Like a Cryptogram?

- Nucleotides in motifs encode a message in a "genetic" language. Symbols in a cryptogram, encode messages in English
- In order to solve the problem, we analyze the *frequencies of patterns* in DNA/Cryptogram.
- Knowledge of established regulatory motifs makes the Motif Finding problem simpler. Knowledge of the words in the English dictionary helps to solve cryptograms.

Motif Finding Complications

- We don’t have a complete dictionary of motifs
- The “genetic” language doesn’t have a standard “grammar”
- Only a small fraction of nucleotide sequences encode for motifs
- The size of the genome sequence is enormous

The Motif Finding Problem

- Given a random sample of DNA sequences:

```
cctgatagacgctatctggctatccacgtacgtaggtcctctgtgcgaatctatgcgtttccaaccat
agtactggtgtacatttgatacgtacgtacaccggcaacctgaaacaaacgctcagaaccagaagtgc
aaacgtacgtgcaccctctttcttcgtggctctggccaacgagggctgatgtataagacgaaaatttt
agcctccgatgtaagtcatagctgtaactattacctgccacccctattacatcttacgtacgtataca
Ctgttatacaacgcgtcatggcggggtatgcgttttggtcgtcgtacgctcgatcgttaacgtacgtc
```

- Find the pattern that is implanted in each of the individual sequences, namely, the motif

- Additional information:
  - Assume the hidden sequence is of length 8
  - The pattern is not exactly the same in each sequence because random point mutations have been introduced

Motif Finding Example

- Finding motifs if there are no mutations
- Probability of a given 8-mer in an infinite sequence is $1/4^8 \approx 1.5\text{x}10^{-5}$ (1 every 65Kb)
- Assuming 5 strings of length 68, there are 5 (68 - 8) = 300 distinct 8-mers
- Probability of any one 8-mer is $300/4^8 \approx 0.005$
- So *any* repeat is rare

```
cctgatagacgctatctggctatccacgtacgtaggtcctctgtgcgaatctatgcgtttccaaccat
agtactggtgtacatttgatacgtacgtacaccggcaacctgaaacaaacgctcagaaccagaagtgc
aaacgtacgtgcaccctctttcttcgtggctctggccaacgagggctgatgtataagacgaaaatttt
agcctccgatgtaagtcatagctgtaactattacctgccacccctattacatcttacgtacgtataca
ctgttatacaacgcgtcatggcggggtatgcgttttggtcgtcgtacgctcgatcgttaacgtacgtc
```

acgtacgt

The Problem Becomes Harder

- Introduce 2 point mutations into each pattern:

```
cctgatagacgctatctggctatccaGgtacTtaggtcctctgtgcgaatctatgcgtttccaaccat
agtactggtgtacatttgatCcAtacgtacaccggcaacctgaaacaaacgctcagaaccagaagtgc
aaacgtTAgtgcaccctctttcttcgtggctctggccaacgagggctgatgtataagacgaaaatttt
agcctccgatgtaagtcatagctgtaactattacctgccacccctattacatcttacgtCcAtataca
ctgttatacaacgcgtcatggcggggtatgcgttttggtcgtcgtacgctcgatcgttaCcgtacgGc
```

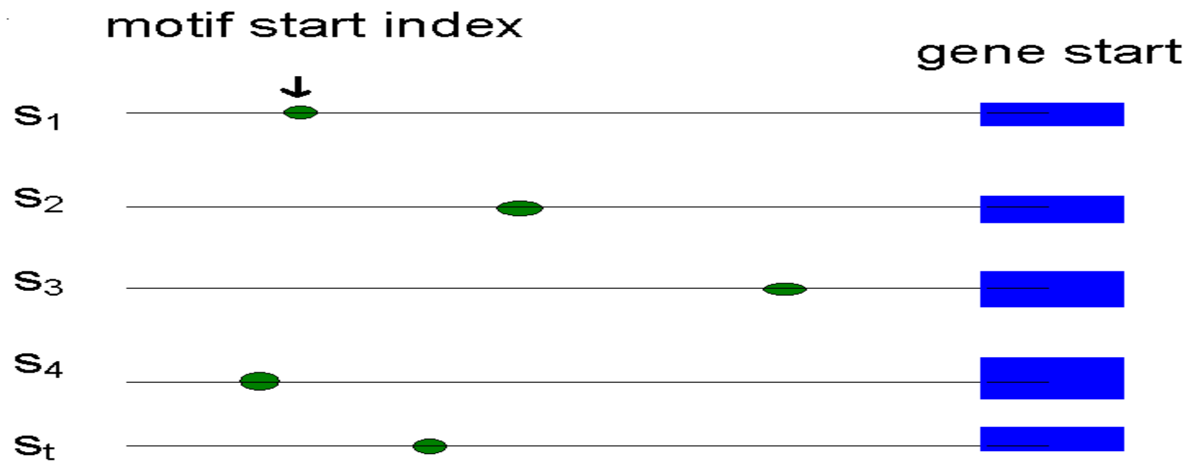
- Our original target pattern no longer appears in any sequence!

Can we still find the motif?

Defining a Motif

- To define a motif, let's assume that we know where the motif starts in each sequence
- The start positions can be represented as

$$\boldsymbol{s} = [s_1, s_2, s_3, \ldots, s_t]$$



Motifs: Profiles and Consensus

```
Alignment      a G g t a c T t
               C c A t a c g t
               a c g t T A g t
               a c g t C c A t
               C c g t a c g G
              _________________

Profile     A  3 0 1 0 3 1 1 0
            C  2 4 0 0 1 4 0 0
            G  0 1 4 0 0 0 3 1
            T  0 0 0 5 1 0 1 4
              _________________
Consensus      A C G T A C G T
```

- Line up the patterns by their start indexes

  $\mathbf{s} = (s_1, s_2, \ldots, s_t)$

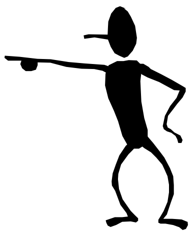
- Construct a matrix profile with the frequencies of each nucleotide in columns

- Consensus nucleotide in each position has the highest score in column

Consensus

- One can think of the consensus as an "ancestor" motif, from which mutated motifs emerged
- The *distance* between an actual motif and the consensus sequence is generally less than that for any two actual motifs
- *Hamming distance* is number of positions that differ between two strings

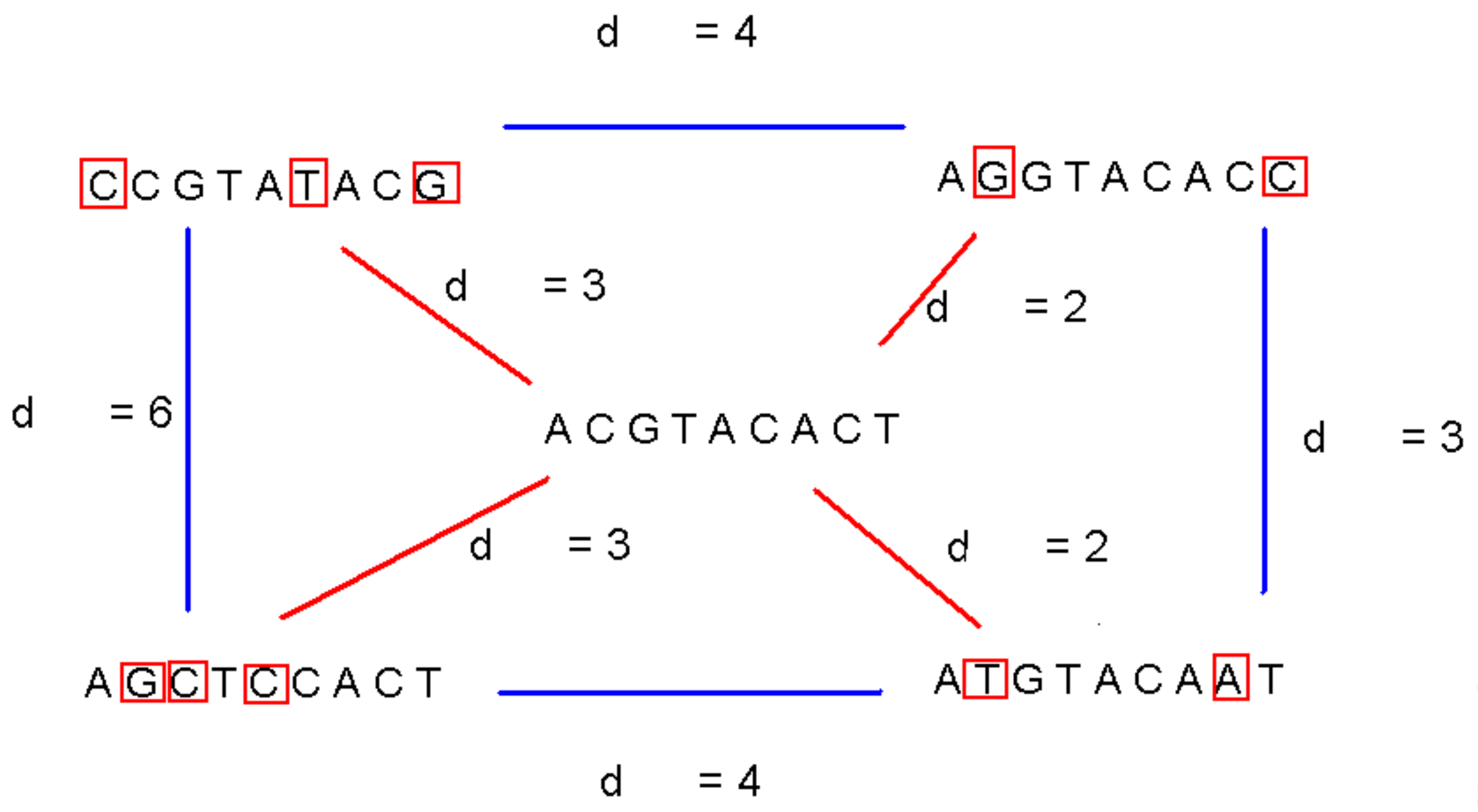
```
G A G A C T C A T
X         X
T A G A C G C A T
```



A Hamming distance of 2

Consensus Properties

- A consensus string has a minimal hamming distance to all source strings

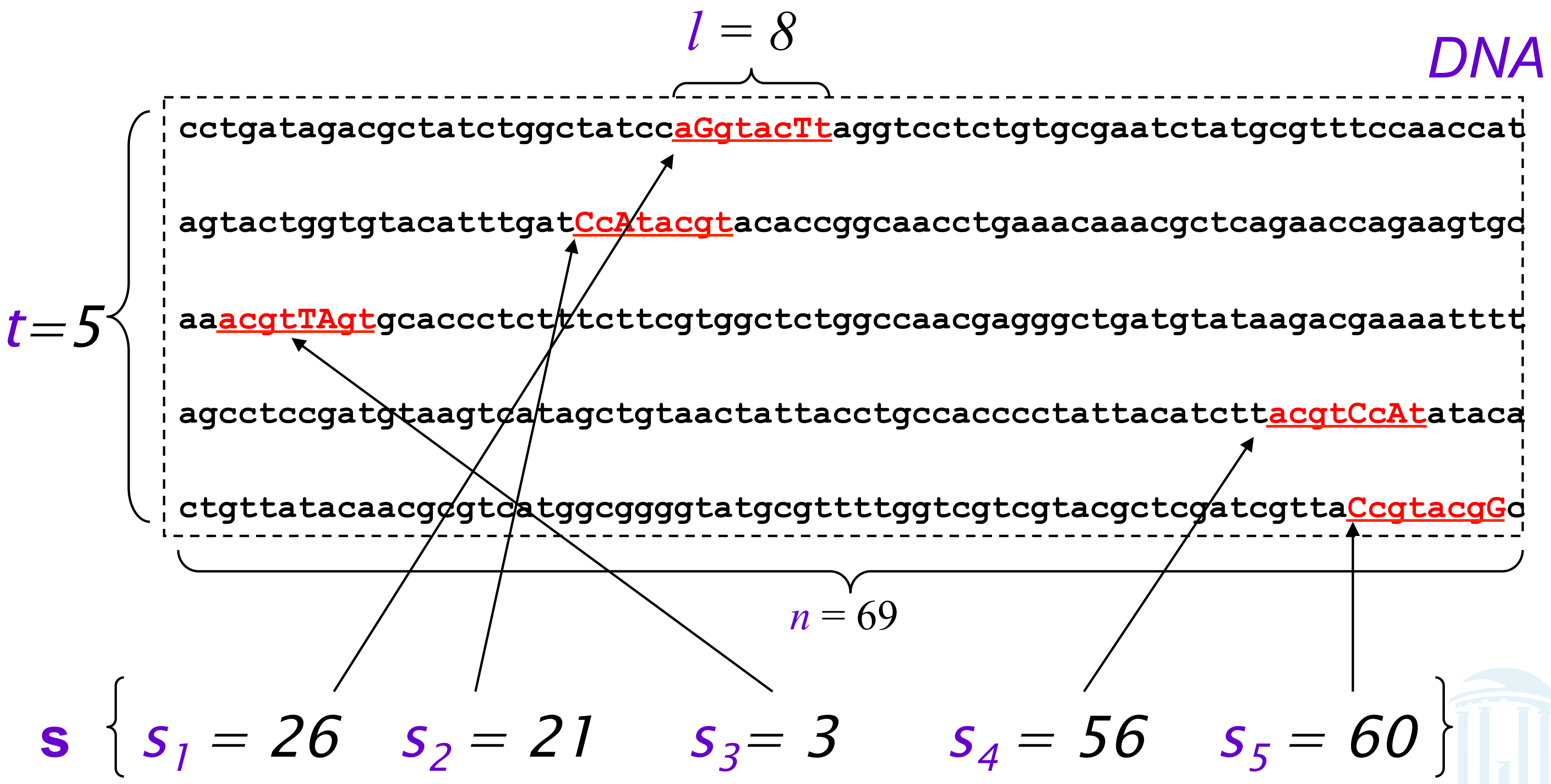


Defining Some Terms

- ***DNA*** – array of sequence fragments
- ***t*** - number of sample DNA sequences
- ***n*** - length of each DNA sequence

- ℓ - length of the motif (ℓ-mer)
- $\mathbf{s}_i$ - starting position of an ℓ-mer in sequence i
- $\mathbf{s}=(s_1, s_2,\ldots s_t)$ - array of motif's starting positions

Illustration of Terms



```
cctgatagacgctatctggctatccaGgtacTtaggtcctctgtgcgaatctatgcgtttccaaccat
agtactggtgtacatttgatCcAtacgtacaccggcaacctgaaacaaacgctcagaaccagaagtgc
aaacgtTAgtgcaccctctttcttcgtggctctggccaacgagggctgatgtataagacgaaatttt
agcctccgatgtaagtcatagctgtaactattacctgccacccctattacatcttacgtCcAtataca
ctgttatacaacgcgtcatggcggggtatgcgttttggtcgtcgtacgctcgatcgttaCcgtacgGc
```

$\mathbf{s}\ \{\, s_1 = 26 \quad s_2 = 21 \quad s_3 = 3 \quad s_4 = 56 \quad s_5 = 60 \,\}$

Scoring Motifs

- Given **s** = (s_1, … s_t) and ***DNA***:

$$\left.\overbrace{\begin{array}{cccccccc} a & G & g & t & a & c & T & t \\ C & c & A & t & a & c & g & t \\ a & c & g & t & T & A & g & t \\ a & c & g & t & C & c & A & t \\ C & c & g & t & a & c & g & G \end{array}}^{l}\right\}t$$

$$Score(\mathbf{s},\mathbf{DNA}) = \sum_{i=1}^{l} \max_{k\in\{A,C,G,T\}} count(k,i)$$

A	3	0	1	0	3	1	1	0
C	2	4	0	0	1	4	0	0
G	0	1	4	0	0	0	3	1
T	0	0	0	5	1	0	1	4

Consensus a c g t a c g t

Score 3+4+4+5+3+4+3+4=30

The Motif Finding Problem

- Goal: Given a set of DNA sequences, find a set of ℓ-mers, one from each sequence, that maximizes the consensus score

- Input: A $\boldsymbol{t} \times \boldsymbol{n}$ matrix of ***DNA***, and ℓ, the length of the pattern to find

- Output: An array of $\boldsymbol{t}$ starting positions
  $\mathbf{s} = (s_1, s_2, \ldots s_t)$ maximizing *Score*(**s**,***DNA***)

Brute Force Solution

- Compute the scores for all possible combinations of starting positions **s**
- The best score determines the best profile and the consensus pattern in ***DNA***
- The goal is to maximize *Score*(**s**,***DNA***) by varying the starting positions s_i, where:

$$s_i = [1, \ldots, \boldsymbol{n}\text{-}\boldsymbol{l}+1]$$
$$\boldsymbol{i} = [1, \ldots, \boldsymbol{t}]$$

Brute Force Pseudocode

```
1. BruteForceMotifSearch(DNA, t, n, $l$)
2. bestScore ← 0
3. for each s = ($s_1$, $s_2$, …, $s_t$) from (1, 1, …, 1)
                                  to (n-$l$+1, n-$l$+1, …, n-$l$+1)
4.    if score(s, DNA, $l$) > bestScore
5.          bestScore ← score(s, DNA, $l$)
6.          bestMotif ← ($s_1$, $s_2$, …, $s_t$)
7. return bestMotif
```

Running Time of BruteForceMotifSearch

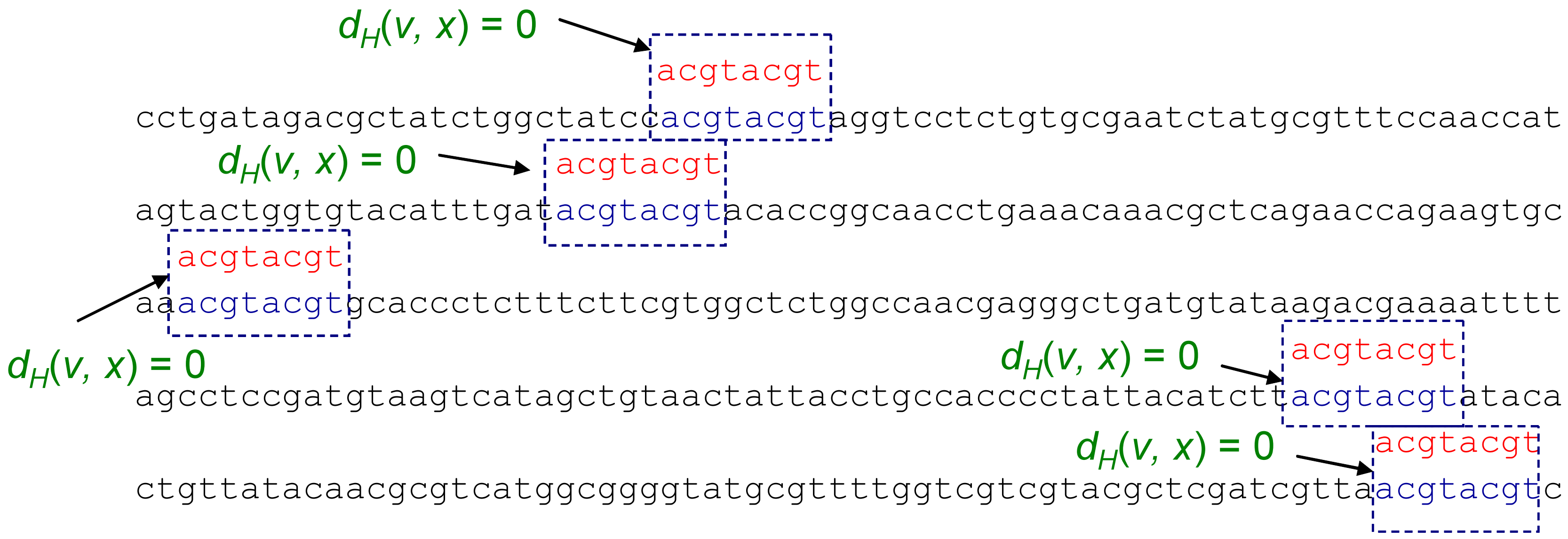
- Search $(n - \ell + 1)$ positions in each of $\boldsymbol{t}$ sequences, by examining $(n - \ell + 1)^t$ sets of starting positions
- For each set of starting positions, the scoring function makes ℓ operations, so complexity is $\ell(\mathrm{n} - \ell + 1)^t = O(\ell n^t)$
- That means that for $\boldsymbol{t} = 8$, $\boldsymbol{n} = 1000$, $\ell = 10$ we must perform approximately 10^{25} computations
- Generously assuming 10^9 comps/sec it will require only 10^{16} secs
- $10^{16}/(60 * 60 * 24 * 365) \rightarrow$ millions of years

The Median String Problem

- Given a set of ***t*** DNA sequences find a pattern that appears in ***all t*** sequences with the minimum number of mutations
- This pattern will be the motif
- Rather than finding the maximal consensus string, this approach attempts to the minimal distance string

Total Distance: An Example

- Given ***v*** = "acgtacgt" and **s**



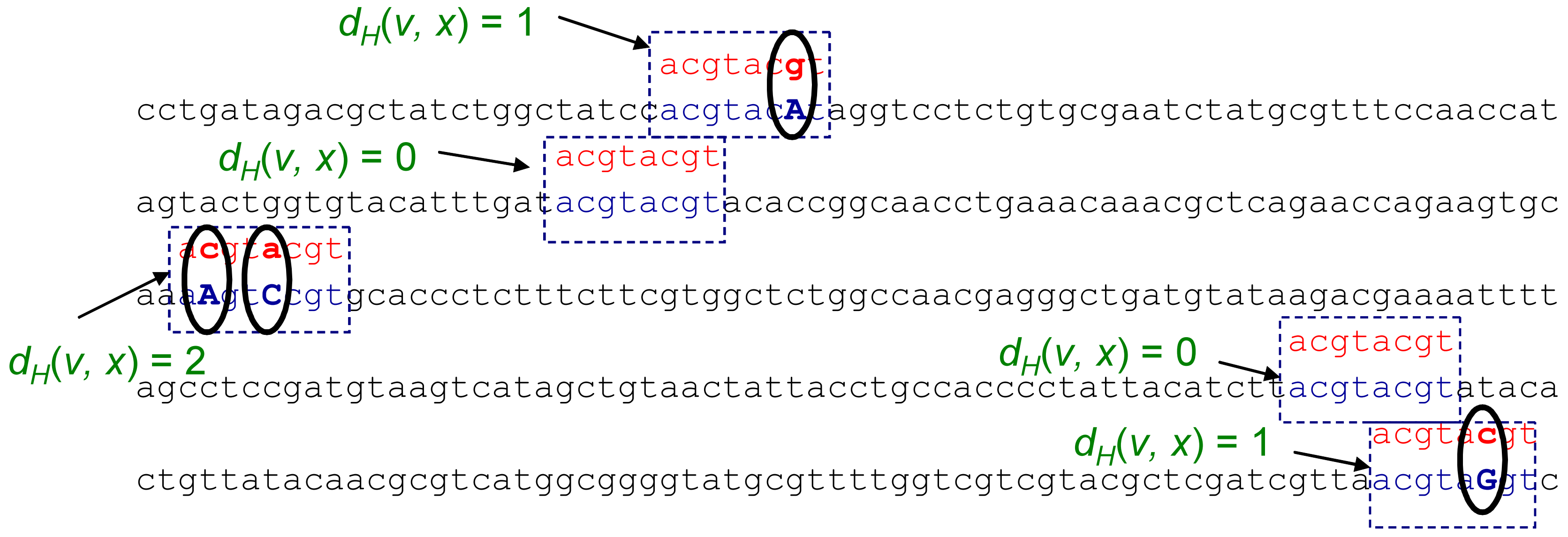
```
cctgatagacgctatctggctatccacgtacgtaggtcctctgtgcgaatctatgcgtttccaaccat
agtactggtgtacatttgatacgtacgtacaccggcaacctgaaacaaacgctcagaaccagaagtgc
aaacgtacgtgcaccctctttcttcgtggctctggccaacgagggctgatgtataagacgaaaatttt
agcctccgatgtaagtcatagctgtaactattacctgccacccctattacatcttacgtacgtataca
ctgttatacaacgcgtcatggcggggtatgcgttttggtcgtcgtacgctcgatcgttaacgtacgtc
```

v is the sequence in red, *x* is the sequence in blue

- $TotalDistance(v,DNA) = 0$

Total Distance: An Example

- Given ***v*** = "acgtacgt" and **s**



```
cctgatagacgctatctggctatccacgtacAtaggtcctctgtgcgaatctatgcgtttccaaccat
agtactggtgtacatttgatacgtacgtacaccggcaacctgaaacaaacgctcagaaccagaagtgc
aaaAgtCcgtgcaccctctttcttcgtggctctggccaacgagggctgatgtataagacgaaaatttt
agcctccgatgtaagtcatagctgtaactattacctgccacccctattacatcttacgtacgtataca
ctgttatacaacgcgtcatggcggggtatgcgttttggtcgtcgtacgctcgatcgttaacgtaGgtc
```

v is the sequence in red, *x* is the sequence in blue

- *TotalDistance(v,DNA)* = 1+0+2+0+1 = 4

Total Distance: Definition

- For each DNA sequence $\boldsymbol{i}$, compute all $d_H(\boldsymbol{v}, \boldsymbol{x})$, where $\boldsymbol{x}$ is an $\boldsymbol{\ell}$-mer with starting position s_i

  $(1 \leq s_i \leq \boldsymbol{n} - \boldsymbol{\ell} + 1)$
- Find minimum of $d_H(\boldsymbol{v}, \boldsymbol{x})$ among all $\boldsymbol{\ell}$-mers in sequence $\boldsymbol{i}$
- *TotalDistance(**v**,**DNA**)* is the sum of the minimum Hamming distances for each DNA sequence $\boldsymbol{i}$
- *TotalDistance(**v**,**DNA**)* = $min_{\mathbf{s}}\ d_H(\boldsymbol{v}, \boldsymbol{s})$, where $\boldsymbol{s}$ is the set of starting positions $s_1, s_2, \ldots s_t$

The Median String Problem

- Goal: Given a set of DNA sequences, find a median string
- Input: A $\boldsymbol{t}$ x $\boldsymbol{n}$ matrix DNA, and $\boldsymbol{l}$, the length of the pattern to find
- Output: A string $\boldsymbol{v}$ of $\boldsymbol{l}$ nucleotides that minimizes *TotalDistance(**v**,**DNA**)* over all strings of that length

Median String Search Algorithm

```
1. MedianStringSearch(DNA, t, n, $l$)
2. bestMotif ← “”
3. bestDistance ← t × $l$
4. for each $l$–mer, s, from “aaa…a” to “ttt…t”
5.    if TotalDistance(s, DNA) < bestDistance
6.        bestDistance ← TotalDistance(s, DNA)
7.        bestMotif ← s
8. return bestMotif
```

Equivalent Problems

- Motif Finding Problem ≡ Median String Problem
- The *Motif Finding* is a maximization problem while *Median String* is a minimization problem
- However, the *Motif Finding* problem and *Median String* problem are computationally equivalent (they give the same output for a common input)
- Need to show that minimizing *TotalDistance* is equivalent to maximizing *Score*

We’re looking for the same thing

ℓ

Alignment		a	G	g	t	a	c	T	t	
		C	c	A	t	a	c	g	t	
		a	c	g	t	T	A	g	t	t
		a	c	g	t	C	c	A	t	
		C	c	g	t	a	c	g	G	
Profile	**A**	3	0	1	0	3	1	1	0	
	C	2	4	0	0	1	4	0	0	
	G	0	1	4	0	0	0	3	1	
	T	0	0	0	5	1	0	1	4	
Consensus		a	c	g	t	a	c	g	t	
Score		3+4+4+5+3+4+3+4								
TotalDistance		2+1+1+0+2+1+2+1								
Sum		5	5	5	5	5	5	5	5	

- At any column ***i***
  $Score_i + TotalDistance_i = \boldsymbol{t}$

- Because there are ℓ columns
  $Score + TotalDistance = \ell * \boldsymbol{t}$

- Rearranging:
  $Score = \ell * \boldsymbol{t} - TotalDistance$

- $\ell * \boldsymbol{t}$ is constant the minimization of the right side is equivalent to the maximization of the left side

Why Bother?

- What is the point of reformulating the Motif Finding problem as the Median String problem?
  - The Motif Finding Problem needs to examine all the combinations for **s**. That is $(\boldsymbol{n} - \ell + 1)^t$ combinations!!!
  - The Median String Problem needs to examine all 4^ℓ combinations for v. This number is relatively smaller

$$\begin{tikzcd}
 & (1000-10+1)^8 \approx 9.3 \times 10^{23} \\
n=1000,\ l=10,\ t=8 \arrow[ur] \arrow[dr] & \\
 & 8(1000-10+1)4^{10} \approx 8.3 \times 10^9
\end{tikzcd}$$

Improving Motif Finding

```
1. BruteForceMotifSearch(DNA, t, n, $l$)
2. bestScore ← 0
3. for each s = ($s_1$, $s_2$, ..., $s_t$) from (1, 1, ..., 1)
                    to (n- $l$+1, n- $l$+1, ..., n- $l$+1)
4.    if score(s, DNA, l) > bestScore
5.        bestScore ← score(s, DNA, $l$)
6.        bestMotif ← ($s_1$, $s_2$, ..., $s_t$)
7. return bestMotif
```

How to Structure the Search?

- How can we perform the line

```
for each s=($s_1$,$s_2$ , . . ., $s_t$) from (1,1 . . . 1) to ($n$–$\ell$+1, . . ., $n$–$\ell$+1) ?
```

- We need a method to more efficiently examine the many possible motifs locations
- This is not very different than exploring all "*t*-digit base *(n-l+1)*" numbers

Improving Median String

```
1. MedianStringSearch(DNA, t, n, l)
2. bestMotif ← “”
3. bestDistance ← t ×l
4. for each l–mer, s, from “aaa…a” to “ttt…t”
5.    if TotalDistance(s, DNA) < bestDistance
6.        bestDistance ← TotalDistance(s, DNA)
7.        bestMotif ← s
8. return bestMotif
```

How to Best Explore Permutations?

- For the Median String Problem we need to consider all 4^{ℓ} possible ℓ-mers:

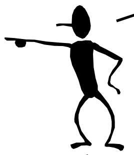
aa… aa
aa… ac
aa… ag
aa… at
aa… ca
.
.
tt… tt

How to organize this search?

Simple Code

- We've used variations of this idea before

```
def NextLeaf(a, L, k):
    # generates L^k permutations
    for i in reversed(xrange(L)):
        if (a[i] < k):
            a[i] += 1
            break
        else:
            a[i] = 1
    return a
```



The "break" here is very important is terminates the loop once any index is increased

- Each call generates a new permutation

NextLeaf Usage

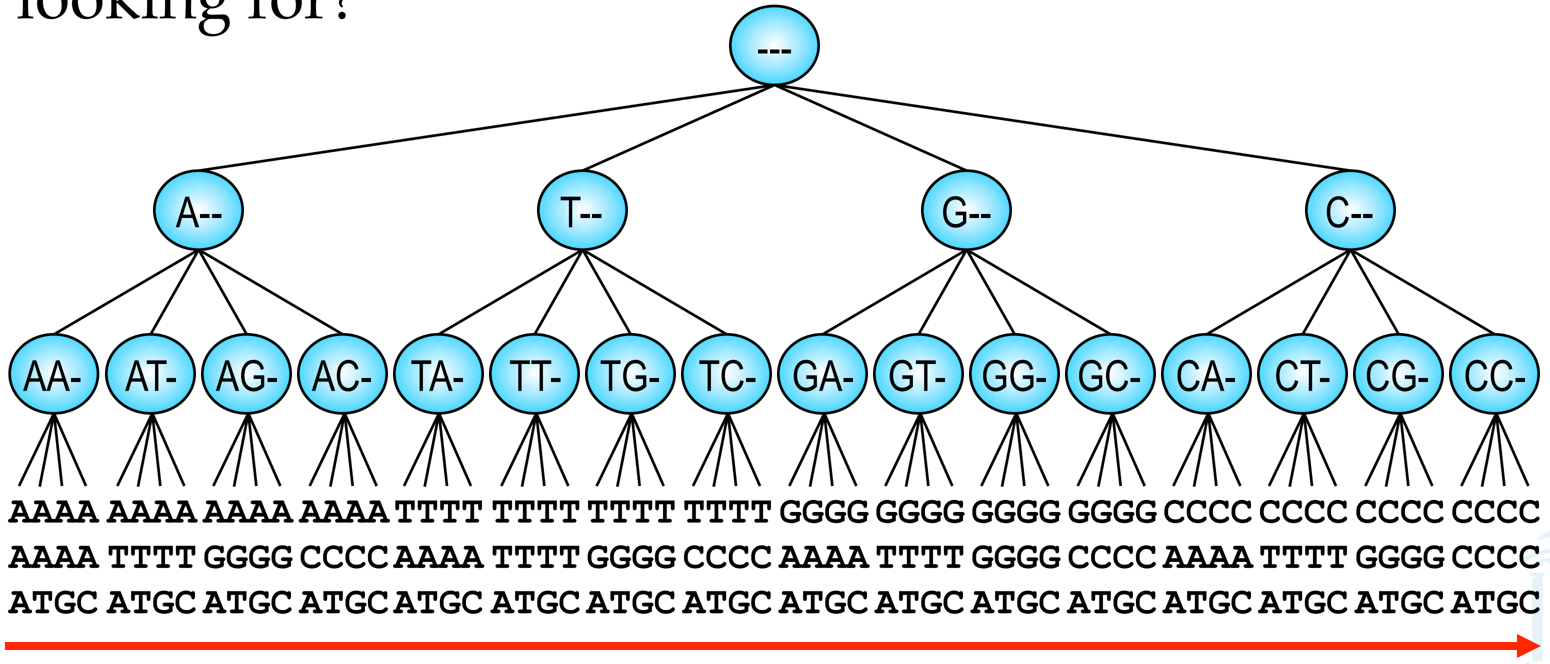
- This is the basic loop structure that we have used for many examples thus far (e. g. BruteForceChange)

```
def AllLeaves(L, k):
    a = [1 for i in xrange(L)]
    while True:
        print a
        a = NextLeaf(a, L, k)
        if (sum(a) == L)
            return
```

- Is there another way to search permutations?

Search Tree

- Our standard method for enumerating permutations just traverses the leaf nodes
- Suppose after checking the first or second letter we already know the solution could not be the one we are looking for?



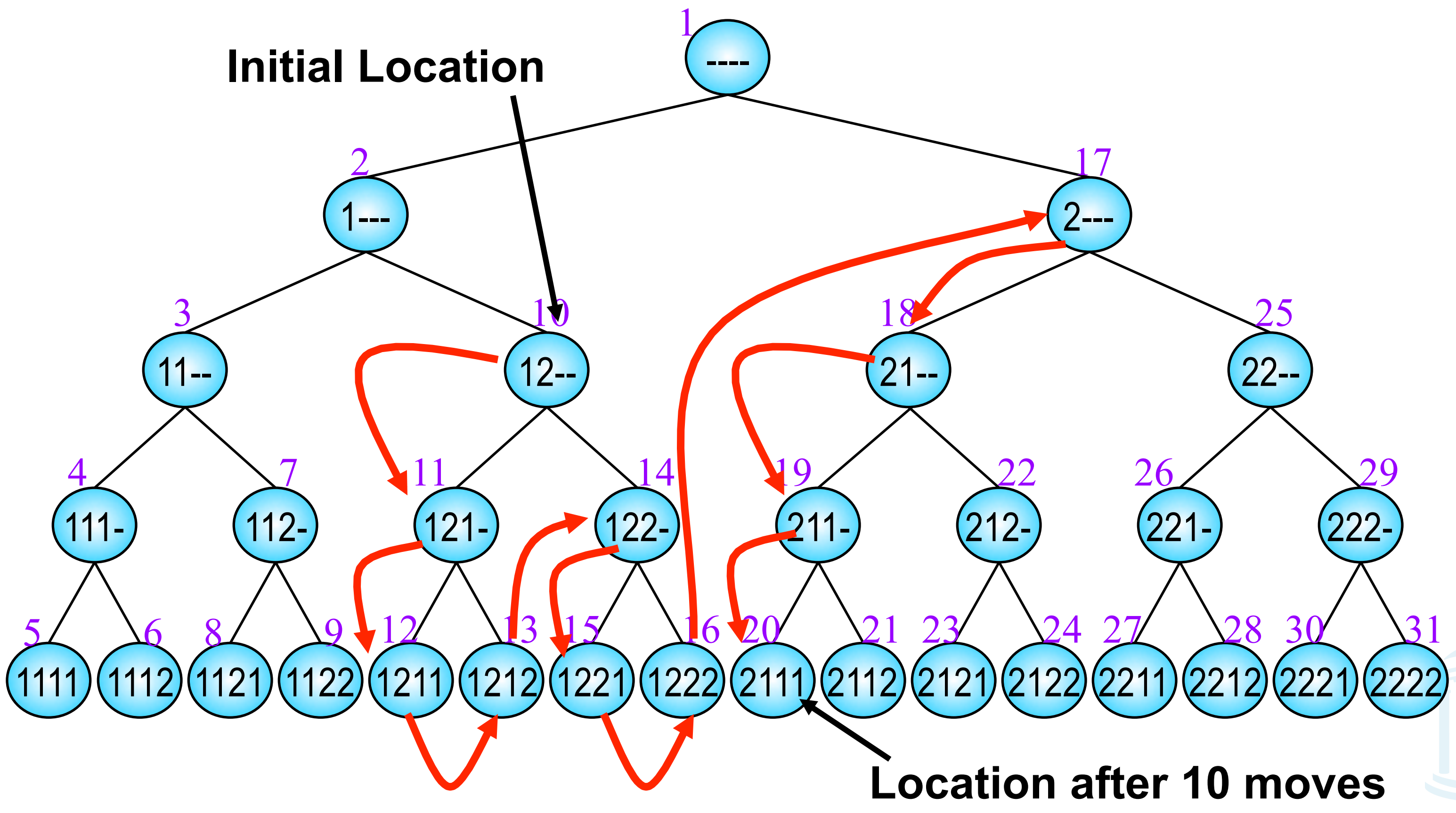
A--
T--
G--
C--
AA-
AT-
AG-
AC-
TA-
TT-
TG-
TC-
GA-
GT-
GG-
GC-
CA-
CT-
CG-
CC-
AAAA AAAA AAAA AAAA TTTT TTTT TTTT TTTT GGGG GGGG GGGG GGGG CCCC CCCC CCCC CCCC
AAAA TTTT GGGG CCCC AAAA TTTT GGGG CCCC AAAA TTTT GGGG CCCC AAAA TTTT GGGG CCCC
ATGC ATGC ATGC ATGC ATGC ATGC ATGC ATGC ATGC ATGC ATGC ATGC ATGC ATGC ATGC ATGC

Analyzing Search Trees

- Characteristics of the search trees:
  - The unique permutations reside at leaves
  - A parent node is a common prefix of its children
- How can we traverse the tree?
- Things we'd like to do:
  - Visit all the nodes (interior and leaves)
  - Visit the next node (in an ordered way)
  - Bypass the children of a node

Depth First Search

- Start from the root and explore down to the bottom one path at a time



Visiting the Next Vertex

- Uses 0s to encode unspecified part of interior nodes (the dashes in our figure)

```
def NextVertex(a, i, L, k):
    if (i < L):
        a[i] = 1
        return (a, i+1)
    else:
        for j in reversed(xrange(L)):
            if (a[j] < k):
                a[j] += 1
                return (a, j+1)
            a[j] = 0
    return (a, 0)
```

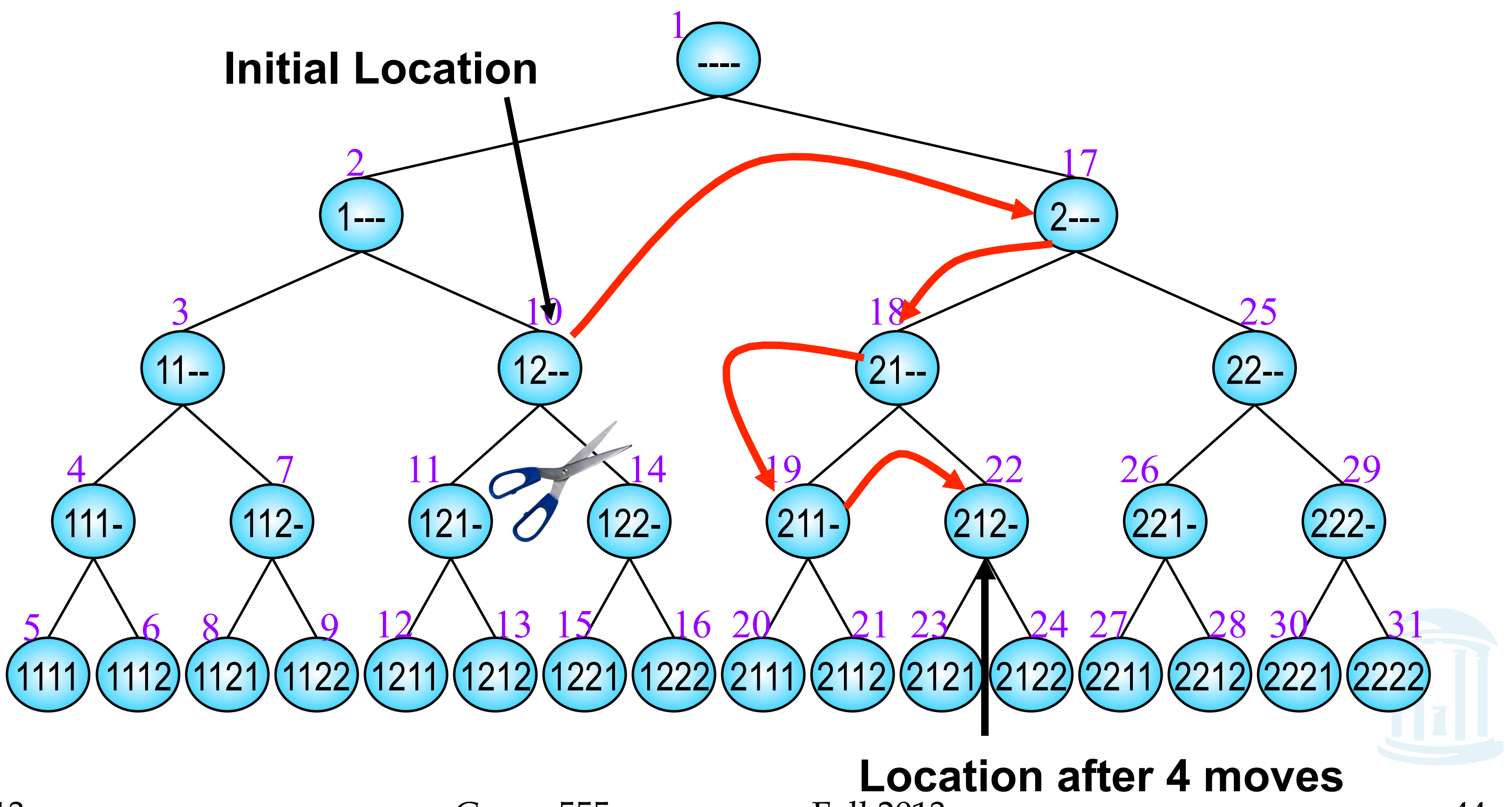
Bypass Nodes

- Given a prefix (internal vertex), find next vertex after skipping all of the current vertex’s children

```
def Bypass(a, i, L, k):
    for j in reversed(xrange(i)):
        if (a[j] < k):
            a[j] += 1
            return (a, j+1)
        a[j] = 0
    return (a, 0)
```

Bypass Example

- Bypassing descendents of nodes "12—" and "211-"



Revisiting Brute Force Search

- Now that we have method for navigating the tree, lets convert our pseudocode version of BruteForceMotifSearch to real code

```
def BruteForceMotifSearchAgain(DNA,t,n,l):
    s = [1 for i in xrange(t)]
    bestScore = Score(s, DNA)
    while (True):
        s = NextLeaf(s,t,n-l+1)
        if (Score(s, DNA) > bestScore):
            bestScore = Score(s, DNA)
            bestMotif = [x for x in s]
        if (sum(s) == t):
            break
    return bestMotif
```

Can We Do Better?

- Sets of **s**=(s_1, s_2, …,s_t) may have a weak profile for the first *i* positions (s_1, s_2, …,s_i)
- Every row of alignment may add at most ℓ to Score
- Optimism: if all subsequent $(t-i)$ positions (s_{i+1}, …s_t) add

$$(t - i) * \ell \text{ to } \mathrm{Score}(\mathbf{s}, i, DNA)$$

- If Score(**s**,*i*,*DNA*) + $(t - i) * \ell$ < **BestScore**, it makes no sense to search subtrees of the current vertex
  - Use **ByPass()**

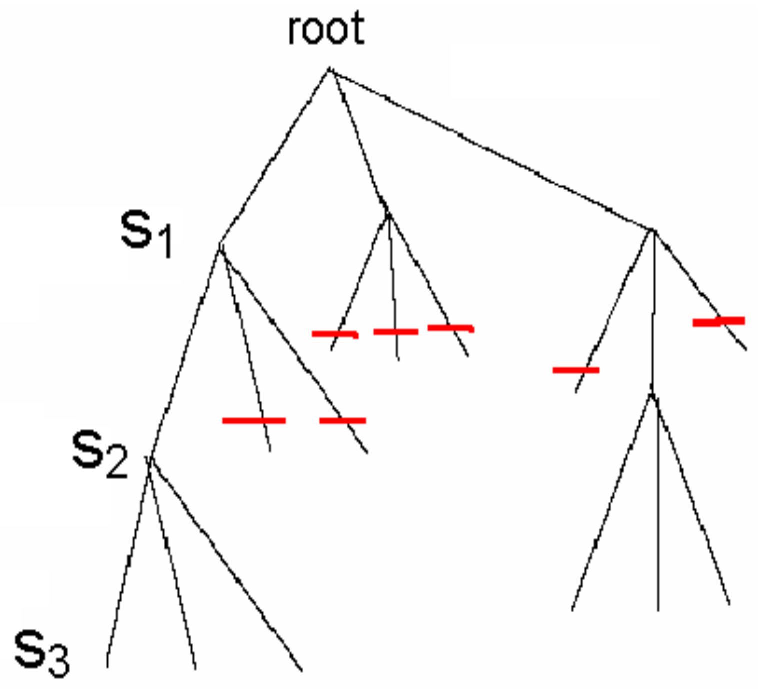
Rewrite Using Tree Traversal

- Before we apply a branch-and-bound strategy let's rewrite the brute-force algorithm using a search tree

```
def SimpleMotifSearch(DNA,t,n,l):
    s = [0 for i in xrange(t)]
    bestScore = 0
    i = 0
    while (True):
        if (i < t):
            s, i = NextVertex(s,i,t,n-l+1)
        else:
            if (Score(s, DNA, l) > bestScore):
                bestScore = Score(s, DNA, l)
                bestMotif = [x for x in s]
            s, i = NextVertex(s,i,t,n-l+1)
            if (sum(s) == 0):
                break
    return bestMotif
```

Branch and Bound Motif Search

- Since each level of the tree goes deeper into search, discarding a prefix discards all following branches



- This saves us from looking at $(n - l + 1)^{t-i}$ leaves
  - Use **NextVertex()** and **ByPass()** to navigate the tree

Branch-and-Bound Motif Code

```
def BranchAndBoundMotifSearch(DNA,t,n,l):
    s = [0 for i in xrange(t)]
    bestScore = 0
    i = 0
    while (True):
        if (i < t):
            optimisticScore = Score(s, DNA, l) + (t-i)*l
            if (optimisticScore < bestScore):
                s, i = Bypass(s,i,t,n-l+1)
            else:
                s, i = NextVertex(s,i,t,n-l+1)
        else:
            score = Score(s, DNA, l)
            if (score > bestScore):
                bestScore = score
                bestMotif = [x for x in s]
            s, i = NextVertex(s,i,t,n-l+1)
        if (sum(s) == 0):
            break
    return bestMotif
```

Improving Median Search

- Recall the computational differences between motif search and median string search
  - The Motif Finding Problem needs to examine all $(\boldsymbol{n}-\ell+1)^t$ combinations for **s**.
  - The Median String Problem needs to examine 4^ℓ combinations of $\boldsymbol{v}$. This number is relatively small
- We want to use median string algorithm with the Branch and Bound trick!

Insight for Improving Median Search

- Note that if, at any point, the total distance for a prefix is greater than that for the best word so far:

  TotalDistance (***prefix***, ***DNA***) > ***BestDistance***

  there is no use exploring the remaining part of the word

- We can eliminate that branch and BYPASS exploring that branch further

Bounded Median String Search

```
def BranchAndBoundMedianSearch(DNA,t,n,l):
    s = [1 for i in xrange(t)]
    bestDistance, bestWord = l*t, ''
    i = 1
    while (i > 0):
        if (i < l):
            prefix = NucleotideString(s, i)
            optimisticDistance = TotalDistance(prefix, DNA)
            if (optimisticDistance > bestDistance):
                s, i = Bypass(s,i,l,t)
            else:
                s, i = NextVertex(s,i,l,t)
        else:
            word = NucleotideString(s, l)
            if (TotalDistance(word, DNA) < bestDistance):
                bestDistance = TotalDistance(word, DNA)
                bestWord = word
            s, i = NextVertex(s,i,l,t)
    return bestWord
```

Today's Bad Example

- An embarrassing confession. I got bitten by a bug in the online notes for the book!

```
cctgatagacgctatctggctatccaGgtacTtaggtcctctgtgcgaatctatgcgtttccaaccat
agtactggtgtacatttgatCcAtacgtacaccggcaacctgaaacaaacgctcagaaccagaagtgc
aaacgtTAgtgcaccctctttcttcgtggctctggccaacgagggctgatgtataagacgaaaatttt
agcctccgatgtaagtcatagctgtaactattacctgccacccctattacatcttacgtCcAtataca
ctgttatacaacgcgtcatggcggggtatgcgttttggtcgtcgtacgctcgatcgttaCcgtacgGc
```

- The target motif has a consensus score of 30
- But [2, 5, 46, 4, 1] = 31 and [2, 5, 46, 6, 1] = 34
- >30 solutions with consensus of 30 or better
- Which is the real Motif?

Further Improvements

- More improvements to Motif searching
  - Why just prune based on prefixes? Can you consider suffixes too?
  - Consider a random subset of *t* strings, or *l* characters
  - Consider multiple letters at a time?
- How do you really find a TFBS?
  - Multiple answers
  - Near optimal answers
  - Motifs are just a starting point
- Next Time
  - We revisit greedy algorithms