Finding Regulatory Motifs in DNA Sequences

Outline

- Implanting Patterns in Random Text
- Gene Regulation
- Regulatory Motifs
- The Gold Bug Problem
- The Motif Finding Problem
- Brute Force Motif Finding
- The Median String Problem
- Search Trees
- Branch-and-Bound Motif Search
- Branch-and-Bound Median String Search
- Consensus and Pattern Branching: Greedy Motif Search
- PMS: Exhaustive Motif Search

Random Sample

atgaccgggatactgataccgtatttggcctaggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg

acccctatttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaacttttccgaatactgggcataaggtaca

tgagtatccctgggatgacttttgggaacactatagtgctctcccgatttttgaatatgtaggatcattcgccagggtccga

gctgagaattggatgaccttgtaagtgttttccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga

tccctttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaatggcccacttagtccacttatag

gtcaatcatgttcttgtgaatggatttttaactgagggcatagaccgcttggcgcacccaaattcagtgtgggcgagcgcaa

cggttttggcccttgttagaggcccccgtactgatggaaactttcaattatgagagagctaatctatcgcgtgcgtgttcat

aacttgagttggtttcgaaaatgctctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta

ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcatttcaacgtatgccgaaccgaaagggaag

ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttctgggtactgatagca

Implanting the Motif: AAAAAAAGGGGGGG

But where is it now?

atgaccgggatactgataaaaaaaaggggggggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg

acccctatttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaacttttccgaataaaaaaaaggggggga

tgagtatccctgggatgacttaaaaaaaaggggggtgctctcccgattttgaatatgtaggatcattcgccagggtccga

gctgagaattggatgaaaaaaaaggggggtccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga

tccctttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaataaaaaaaaggggggcttatag

gtcaatcatgttcttgtgaatggatttaaaaaaaaggggggggaccgcttggcgcacccaaattcagtgtgggcgagcgcaa

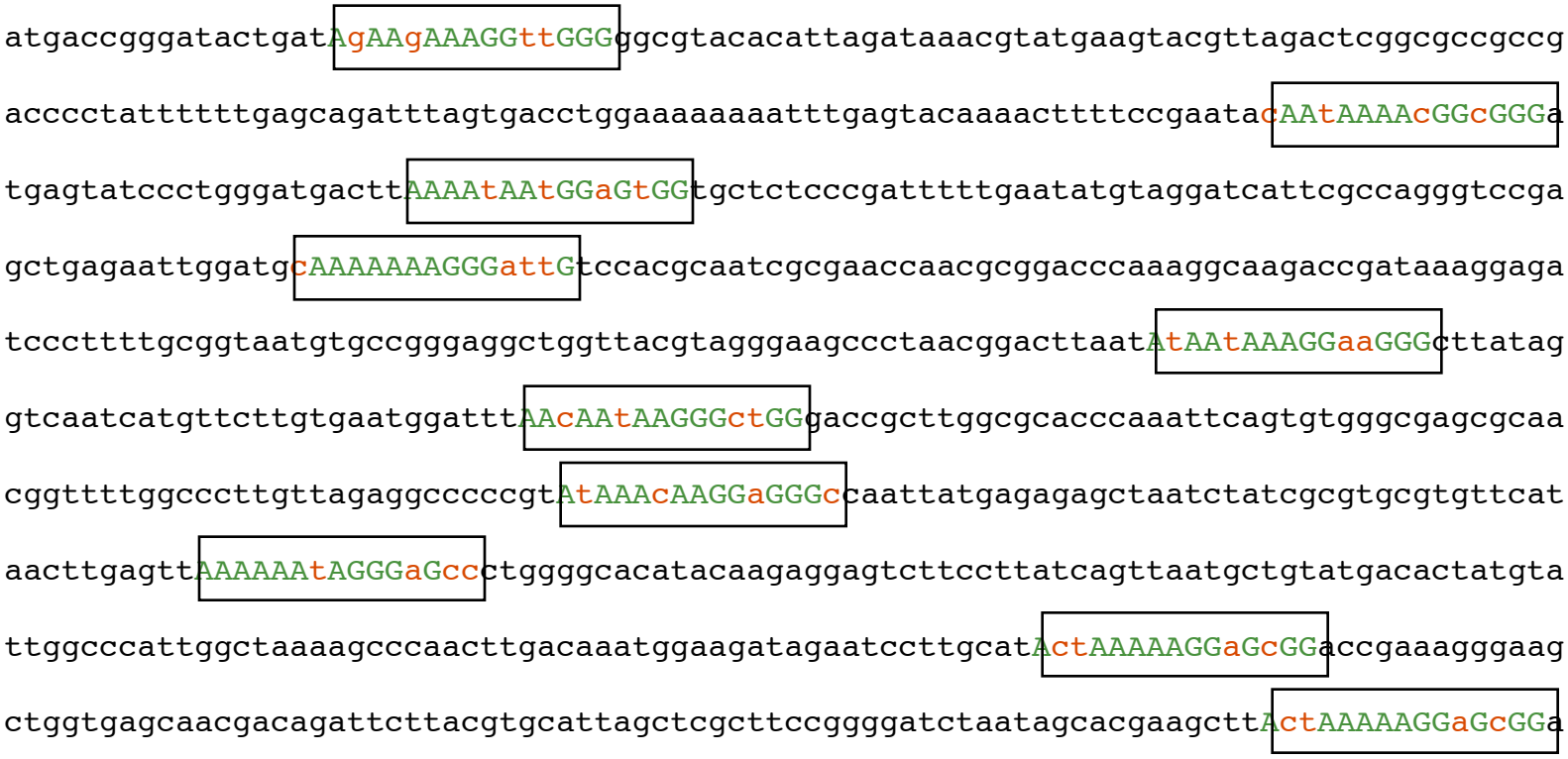
cggttttggcccttgttagaggcccccgtaaaaaaaaggggggcaattatgagagagctaatctatcgcgtgcgtgttcat

aacttgagttaaaaaaaaggggggctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta

ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcataaaaaaaaggggggggaccgaaagggaag

ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttaaaaaaaaggggggga

Implanting the Motif with Four Mutations



Oh, geez. Where is it now?!

atgaccgggatactgatagaagaaaggttgggggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg

acccctatttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaacttttccgaatacaataaaacggcggga

tgagtatccctgggatgacttaaaataatggagtggtgctctcccgatttttgaatatgtaggatcattcgccagggtccga

gctgagaattggatgcaaaaaaagggattgtccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga

tccctttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaatataataaaggaagggcttatag

gtcaatcatgttcttgtgaatggatttaacaataagggctgggaccgcttggcgcacccaaattcagtgtgggcgagcgcaa

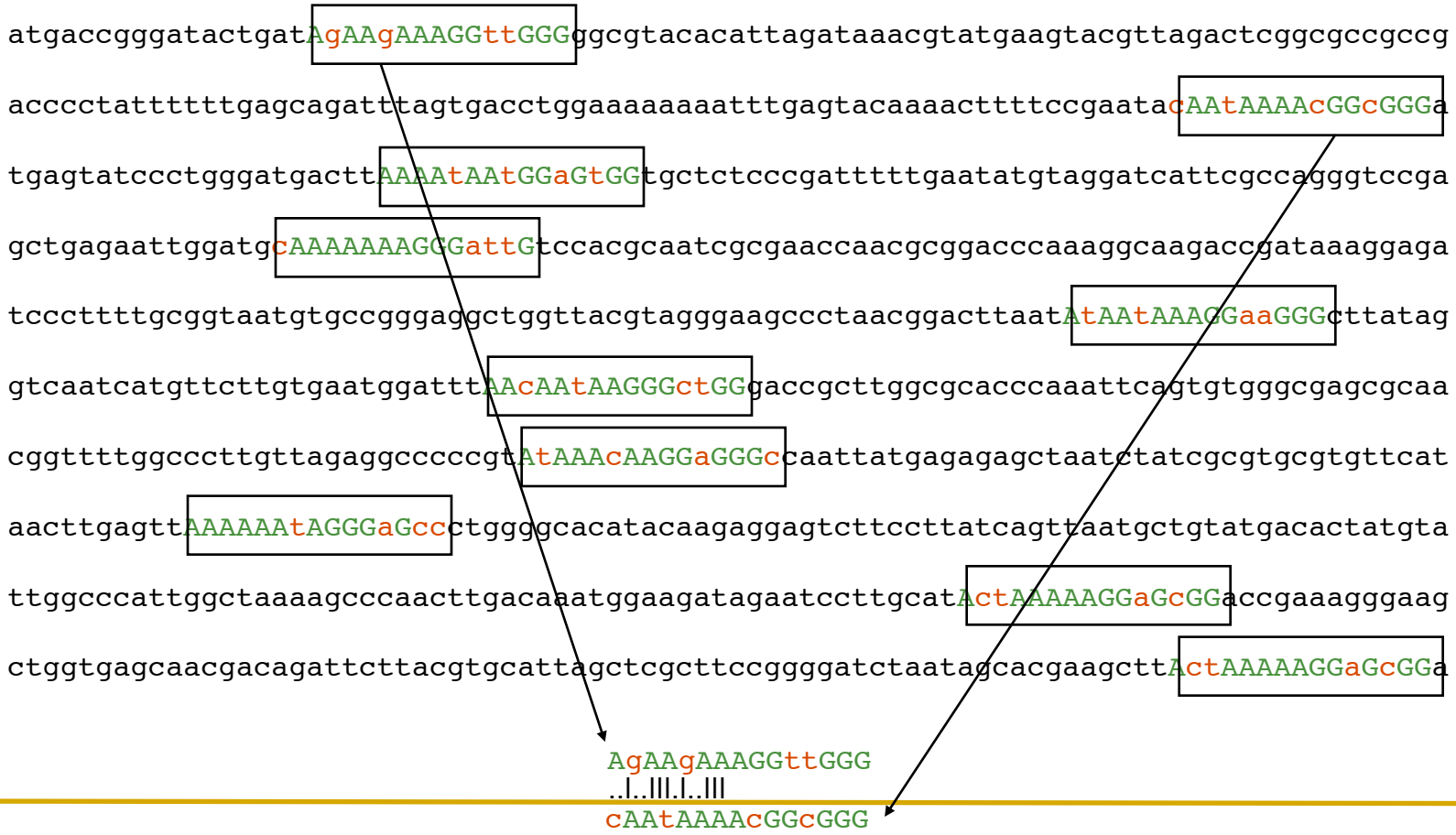
cggttttggcccttgttagaggcccccgtataaacaaggagggccaattatgagagagctaatctatcgcgtgcgtgttcat

aacttgagttaaaaaatagggagccctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta

ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcatactaaaaaggagcggaccgaaagggaag

ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttactaaaaaggagcgga

Why is Finding a (15,4) Motif Hard?



Challenge Problem

- Find a motif in a sample of
  - 20 “random” sequences (e.g. 600 nt long)
  - each sequence containing an implanted pattern of length 15,
  - each pattern appearing with 4 mismatches as (15,4)-motif.

Combinatorial Gene Regulation

- A microarray experiment showed that when gene X is knocked out, 20 other genes are not expressed

  - **How can one gene have such drastic effects?**

Regulatory Proteins

- Gene X encodes regulatory protein, a.k.a. a ***transcription factor*** (TF)
- The 20 unexpressed genes rely on gene X's TF to induce transcription
- A single TF may regulate multiple genes

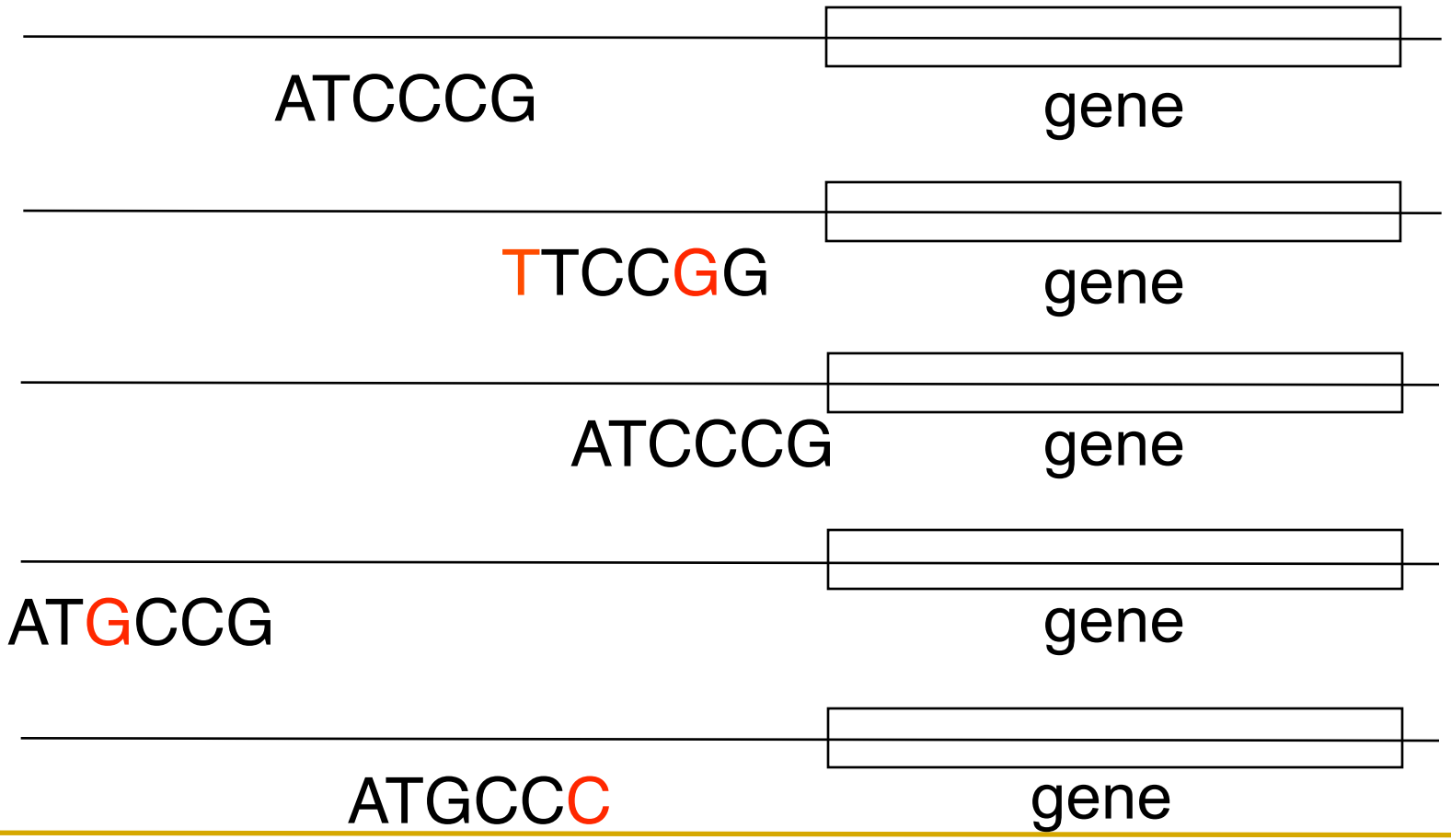
Regulatory Regions

- Every gene contains a regulatory region (RR) typically stretching 100-1000 bp upstream of the transcriptional start site
- Located within the RR are the ***Transcription Factor Binding Sites*** (TFBS), also known as ***motifs***, specific for a given transcription factor
- TFs influence gene expression by binding to a specific location in the respective gene's regulatory region - TFBS

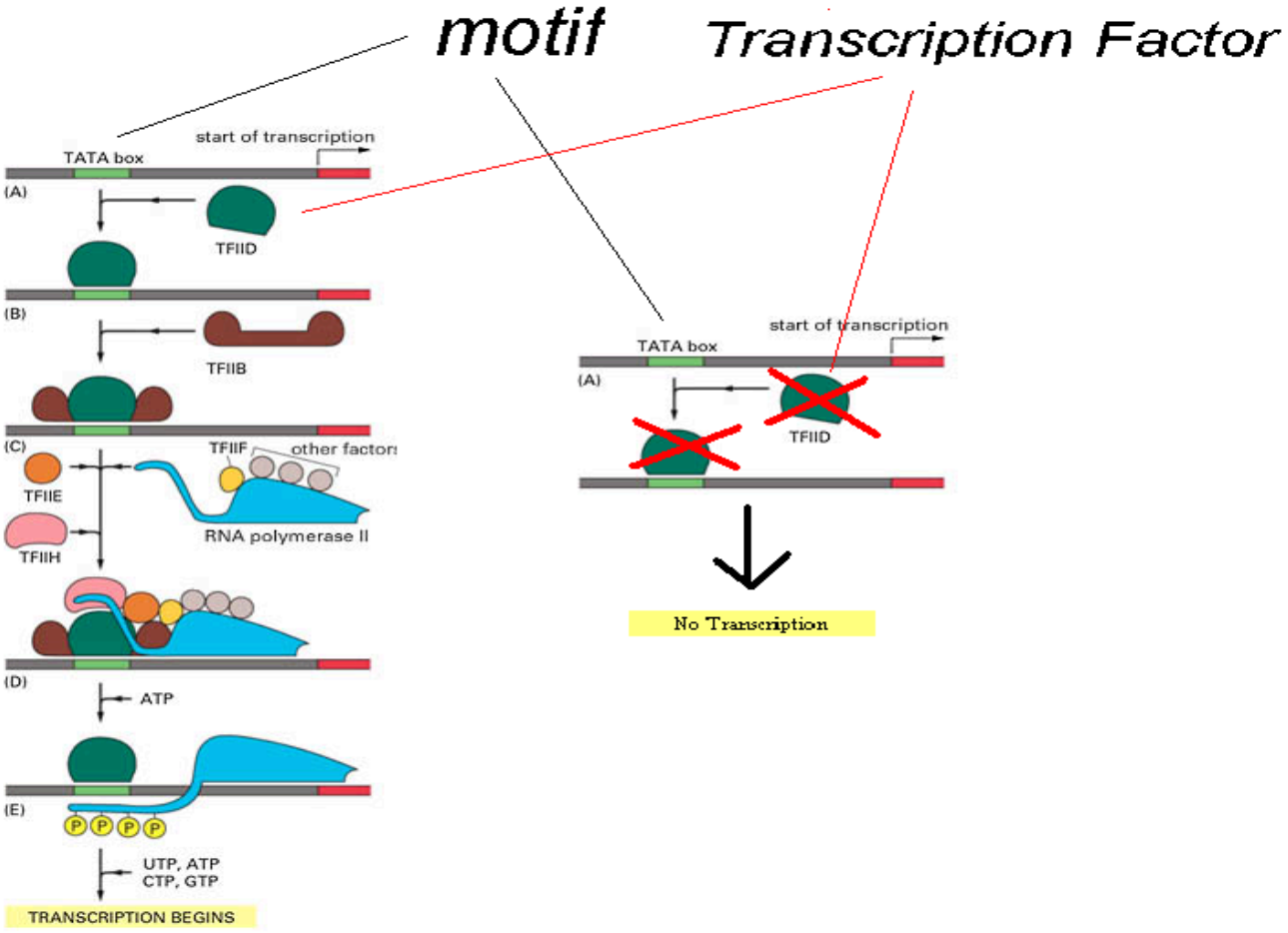
Transcription Factor Binding Sites

- A TFBS can be located anywhere within the Regulatory Region.
- TFBS may vary slightly across different regulatory regions since non-essential bases could mutate

Motifs and Transcriptional Start Sites



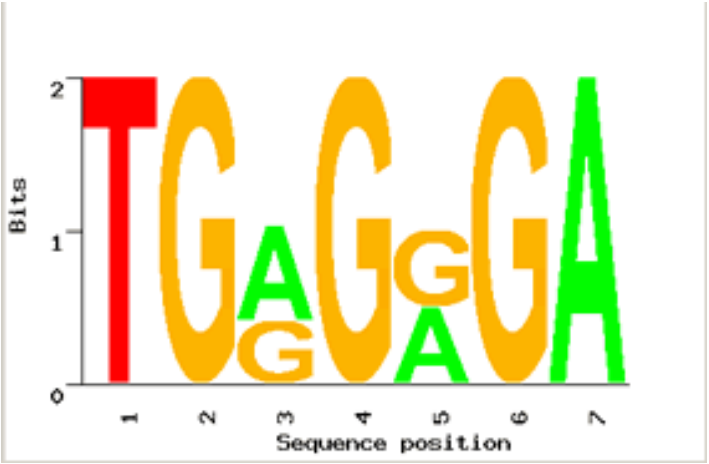
Transcription Factors and Motifs



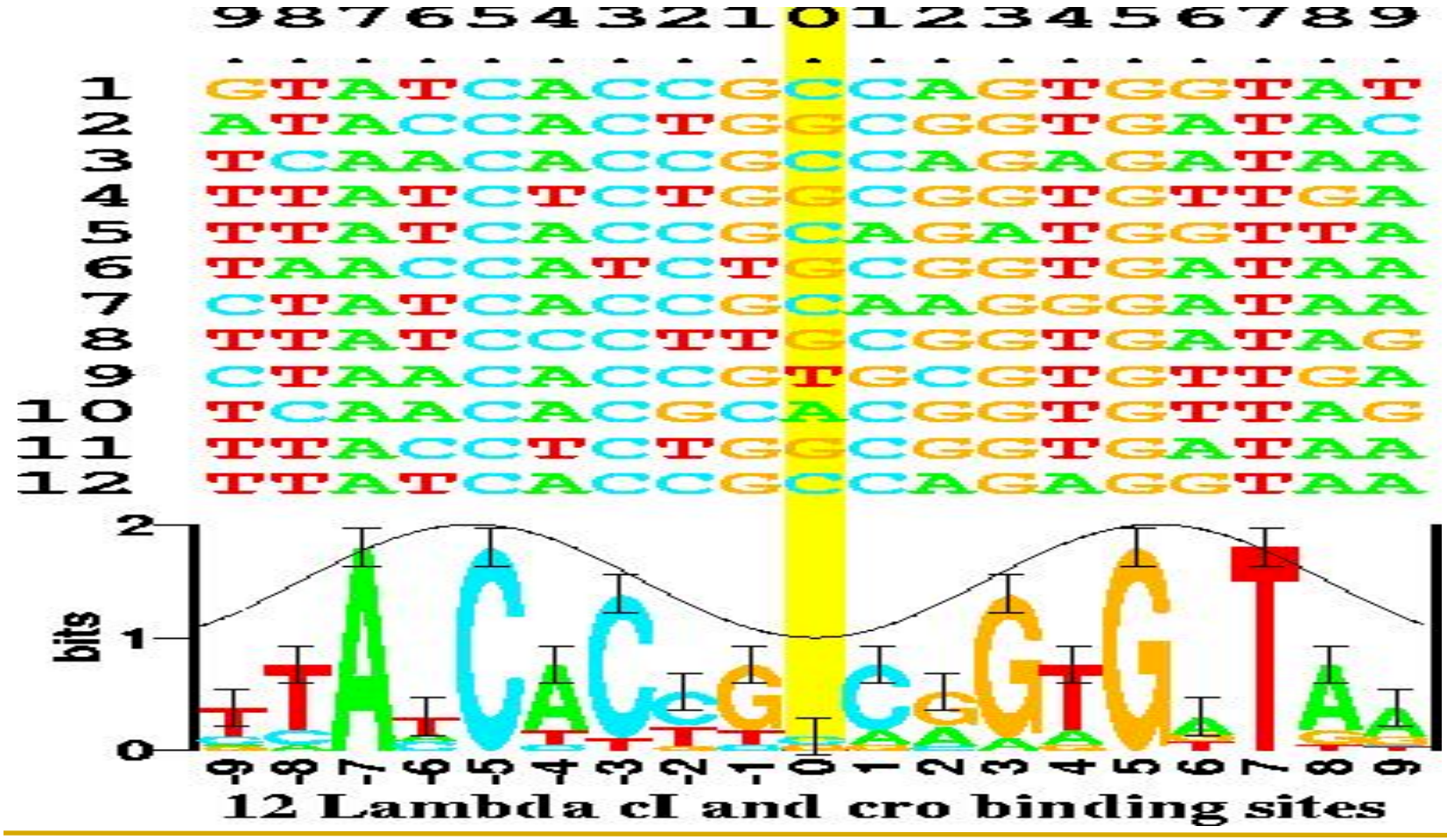
Motif Logo

- Motifs can mutate on non important bases
- The five motifs in five different genes have mutations in position 3 and 5
- Representations called *motif logos* illustrate the conserved and variable regions of a motif

TGGGGGA
TGAGAGA
TGGGGGA
TGAGAGA
TGAGGGA



Motif Logos: An Example



(http://www-lmmb.ncifcrf.gov/~toms/sequencelogo.html)

Identifying Motifs

- Genes are turned on or off by regulatory proteins
- These proteins bind to upstream regulatory regions of genes to either attract or block an RNA polymerase
- Regulatory protein (TF) binds to a short DNA sequence called a motif (TFBS)
- So finding the same motif in multiple genes' regulatory regions suggests a regulatory relationship amongst those genes

Identifying Motifs: Complications

- We do not know the motif sequence
- We do not know where it is located relative to the genes start
- Motifs can differ slightly from one gene to the next
- How to discern it from “random” motifs?

A Motif Finding Analogy

- The Motif Finding Problem is similar to the problem posed by Edgar Allan Poe (1809 – 1849) in his *Gold Bug* story

The Gold Bug Problem

- Given a secret message:

```
53++!305))6*;4826)4+.)4+);806*;48!8`60))85;]8*:+*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(+?3
4;48)4+;161;:188;+?;
```

- Decipher the message encrypted in the fragment

Hints for The Gold Bug Problem

- Additional hints:
  - The encrypted message is in English
  - Each symbol correspond to one letter in the English alphabet
  - No punctuation marks are encoded

The Gold Bug Problem: Symbol Counts

- Naive approach to solving the problem:
  - Count the frequency of each symbol in the encrypted message
  - Find the frequency of each letter in the alphabet in the English language
  - Compare the frequencies of the previous steps, try to find a correlation and map the symbols to a letter in the alphabet

Symbol Frequencies in the Gold Bug Message

- **Gold Bug Message**:

Symbol	8	;	4	)	+	*	5	6	(	!	1	0	2	9	3	:	?	`	-	]	.
Frequency	34	25	19	16	15	14	12	11	9	8	7	6	5	5	4	4	3	2	1	1	1

- **English Language**:

e t a o i n s r h l d c u m f p g w y b v k x j q z

Most frequent  Least frequent

The Gold Bug Message Decoding: First Attempt

- By simply mapping the most frequent symbols to the most frequent letters of the alphabet:

```
sfiilfcsoorntaeuroaikoaiotecrntaeleyrcooestvenpinelefheeosnlt
arhteenmrnwteonihtaesotsnlupnihtamsrnuhsnbaoeyentacrmuesotorl
eoaiitdhimtaecedtepeidtaelestaoaeslsueecrnedhimtaetheetahiwfa
taeoaitdrdtpdeetiwt
```

- The result does not make sense

The Gold Bug Problem: *l*-tuple count

- A better approach:
  - Examine frequencies of *l*-tuples, combinations of 2 symbols, 3 symbols, etc.
  - "The" is the most frequent 3-tuple in English and ";48" is the most frequent 3-tuple in the encrypted text
  - Make inferences of unknown symbols by examining other frequent *l*-tuples

The Gold Bug Problem: the ;48 clue

- Mapping "the" to ";48" and substituting all occurrences of the symbols:

```
53++!305))6*the26)h+.)h+)te06*the!e`60))e5t]e*:+*e!e3(ee)5*!t
h6(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(+?3ht
he)h+t161t:1eet+?t
```

The Gold Bug Message Decoding: Second Attempt

- Make inferences:

```
53++!305))6*the26)h+.)h+)te06*the!e`60))e5t]e*:+*e!e3(ee)5*!t
h6(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(+?3ht
he)h+t161t:1eet+?t
```

- "thet(ee" most likely means "the tree"
  - Infer "(" = "r"
- "th(+?3h" becomes "thr+?3h"
  - Can we guess "+" and "?"?

The Gold Bug Problem: The Solution

- After figuring out all the mappings, the final message is:

```
AGOODGLASSINTHEBISHOPSHOSTELINTHEDEVILSSEATWENYONEDEGRE
ESANDTHIRTEENMINUTESNORTHEASTANDBYNORTHMAINBRANCHSEVENT
HLIMBEASTSIDESHOOTFROMTHELEFTEYEOFTHEDEATHSHEADABEELINE
FROMTHETREETHROUGHTHESHOTFIFTYFEETOUT
```

The Solution (cont'd)

- Punctuation is important:

```
A GOOD GLASS IN THE BISHOP'S HOSTEL IN THE DEVIL'S SEA,
TWENY ONE DEGREES AND THIRTEEN MINUTES NORTHEAST AND BY NORTH,
MAIN BRANCH SEVENTH LIMB, EAST SIDE, SHOOT FROM THE LEFT EYE OF
THE DEATH'S HEAD A BEE LINE FROM THE TREE THROUGH THE SHOT,
FIFTY FEET OUT.
```

Solving the Gold Bug Problem

- Prerequisites to solve the problem:
  - Need to know the relative frequencies of single letters, and combinations of two and three letters in English
  - Knowledge of all the words in the English dictionary is highly desired to make accurate inferences

Motif Finding and The Gold Bug Problem: Similarities

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

  the Gold Bug problem.

Similarities (cont'd)

- **Motif Finding**:
  - In order to solve the problem, we analyze the frequencies of patterns in the nucleotide sequences
  - In order to solve the problem, we analyze the frequencies of patterns in the nucleotide sequences

- **Gold Bug Problem**:
  - In order to solve the problem, we analyze the frequencies of patterns in the text written in English

Similarities (cont'd)

- **Motif Finding**:
  - Knowledge of established motifs reduces the complexity of the problem

- **Gold Bug Problem**:
  - Knowledge of the words in the dictionary is highly desirable

Motif Finding and The Gold Bug Problem: Differences

Motif Finding is harder than **Gold Bug problem**:

- We don't have the complete dictionary of motifs
- The "genetic" language does not have a standard "grammar"
- Only a small fraction of nucleotide sequences encode for motifs; the size of data is enormous

The Motif Finding Problem

- Given a random sample of DNA sequences:

cctgatagacgctatctggctatccacgtacgtaggtcctctgtgcgaatctatgcgtttccaaccat

agtactggtgtacatttgatacgtacgtacaccggcaacctgaaacaaacgctcagaaccagaagtgc

aaacgtacgtgcaccctctttcttcgtggctctggccaacgagggctgatgtataagacgaaaatttt

agcctccgatgtaagtcatagctgtaactattacctgccacccctattacatcttacgtacgtataca

ctgttatacaacgcgtcatggcggggtatgcgttttggtcgtcgtacgctcgatcgttaacgtacgtc

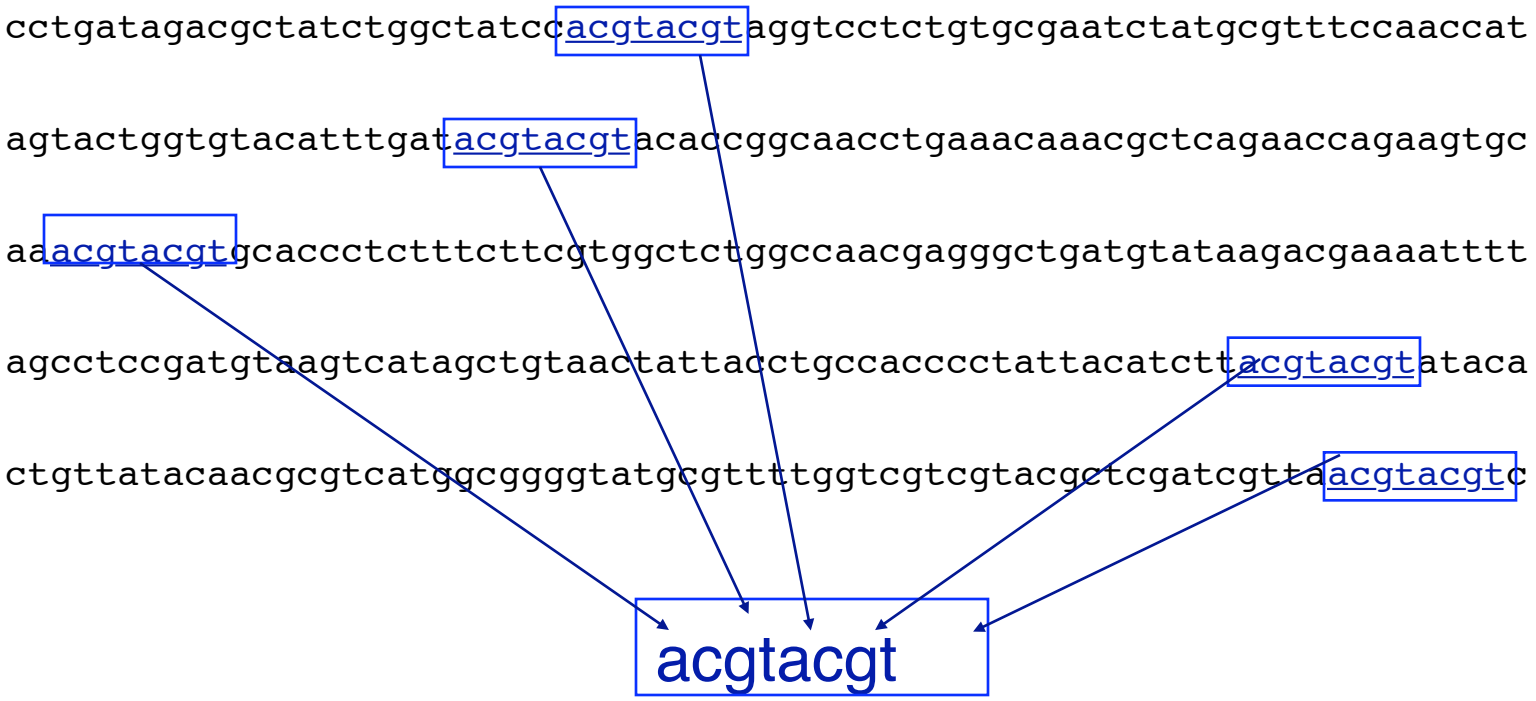
- Find the pattern that is “implanted” in each of the individual sequences, namely, the motif

The Motif Finding Problem (cont'd)

- Additional information:
  - The hidden sequence is (for example) of length 8
  - The pattern is not exactly the same in each array because random point mutations may occur in the sequences

The Motif Finding Problem (cont'd)

- The patterns revealed with no mutations:



Consensus String

The Motif Finding Problem (cont'd)

- The patterns with 2 point mutations:

cctgatagacgctatctggctatccaGgtacTtaggtcctctgtgcgaatctatgcgtttccaaccat

agtactggtgtacatttgatCcAtacgtacaccggcaacctgaaacaaacgctcagaaccagaagtgc

aaacgtTAgtgcaccctctttcttcgtggctctggccaacgagggctgatgtataagacgaaaatttt

agcctccgatgtaagtcatagctgtaactattacctgccacccctattacatcttacgtCcAtataca

ctgttatacaacgcgtcatggcggggtatgcgttttggtcgtcgtacgctcgatcgttaCcgtacgGc

The Motif Finding Problem (cont'd)

- The patterns with 2 point mutations:

cctgatagacgctatctggctatccaGgtacTtaggtcctctgtgcgaatctatgcgtttccaaccat

agtactggtgtacatttgatCcAtacgtacaccggcaacctgaaacaaacgctcagaaccagaagtgc

aaacgtTAgtgcaccctctttcttcgtggctctggccaacgagggctgatgtataagacgaaaatttt

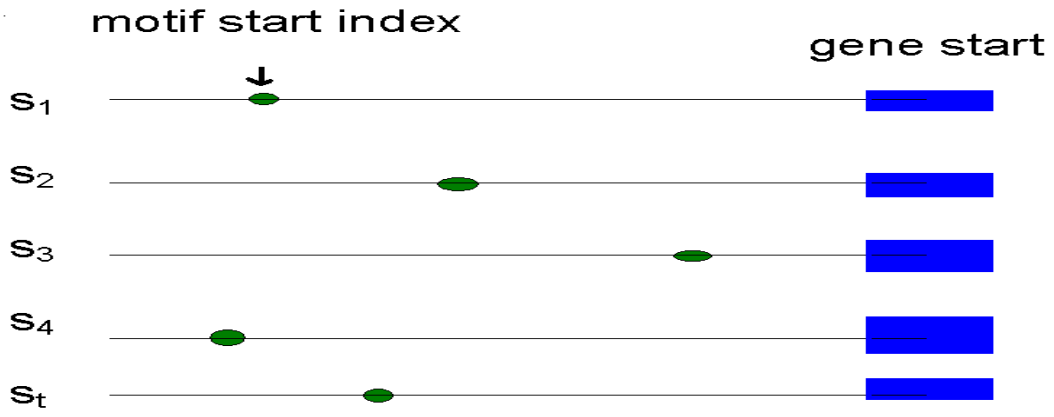
agcctccgatgtaagtcatagctgtaactattacctgccacccctattacatcttacgtCcAtataca

ctgttatacaacgcgtcatggcggggtatgcgttttggtcgtcgtacgctcgatcgttaCcgtacgGc

Can we still find the motif, now that we have 2 mutations?

Defining Motifs

- To define a motif, lets say we know where the motif starts in the sequence
- The motif start positions in their sequences 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

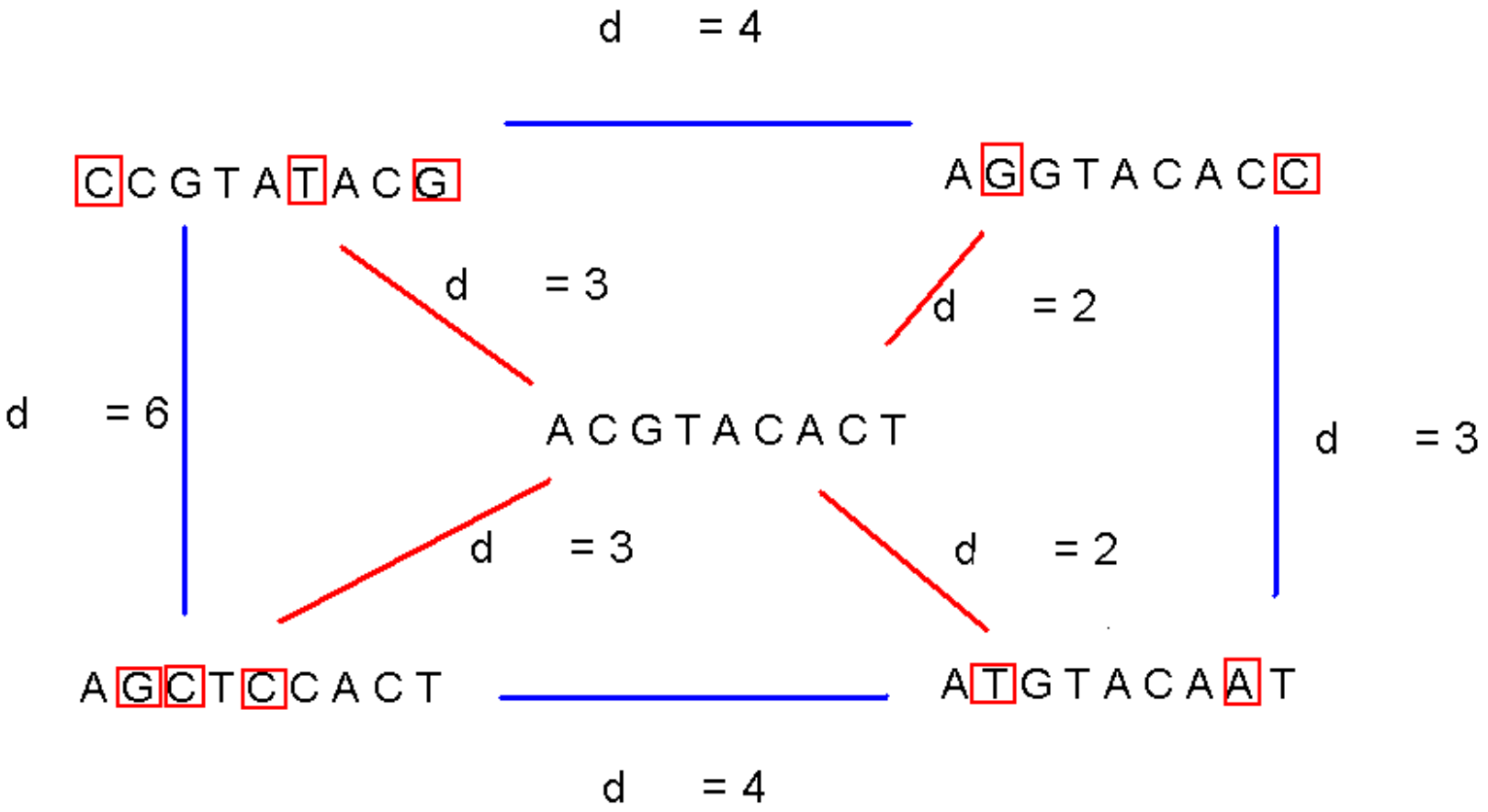
$$\mathbf{s} = (s_1, s_2, \ldots, s_t)$$

- Construct matrix profile with frequencies of each nucleotide in columns
- Consensus nucleotide in each position has the highest score in column

Consensus

- Think of consensus as an “ancestor” motif, from which mutated motifs emerged
- The *distance* between a real motif and the consensus sequence is generally less than that for two real motifs

Consensus (cont'd)



Evaluating Motifs

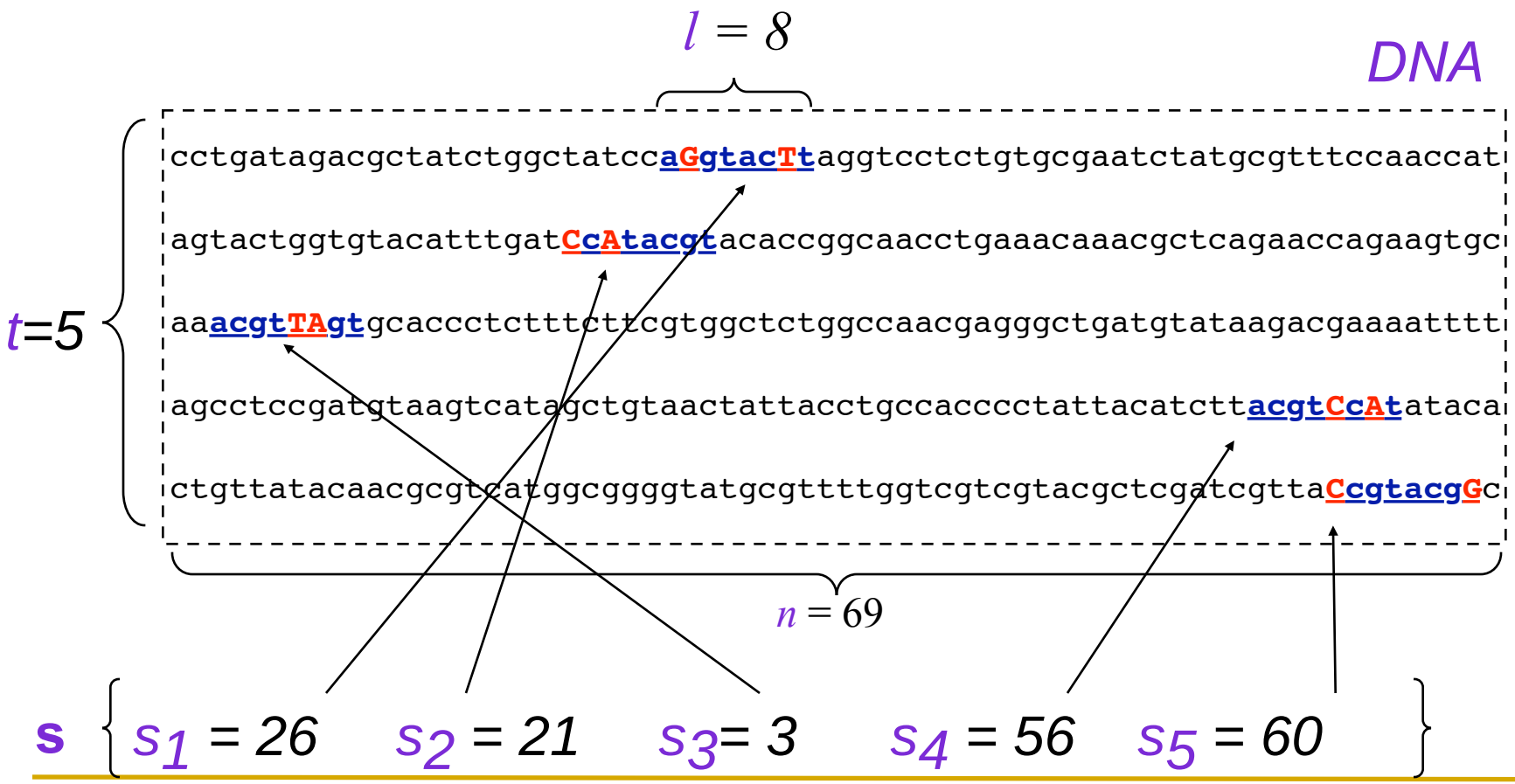
- We have a guess about the consensus sequence, but how “good” is this consensus?
- Need to introduce a scoring function to compare different guesses and choose the “best” one.

Defining Some Terms

- ***t*** - number of sample DNA sequences
- ***n*** - length of each DNA sequence
- ***DNA*** - sample of DNA sequences (*t* x *n* array)

- l - length of the motif (l-mer)
- 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

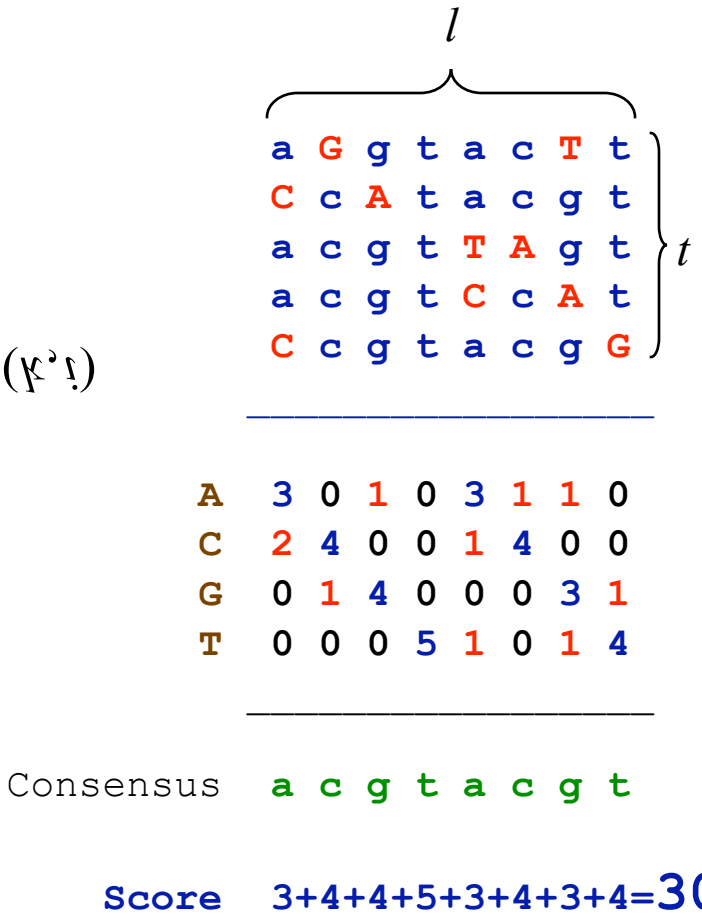
Parameters


l = 8
DNA
t=5
cctgatagacgctatctggctatccaGgtacTtaggtcctctgtgcgaatctatgcgtttccaaccat
agtactggtgtacatttgatCcAtacgtacaccggcaacctgaaacaaacgctcagaaccagaagtgc
aaacgtTAgtgcaccctctttcttcgtggctctggccaacgagggctgatgtataagacgaaaatttt
agcctccgatgtaagtcatagctgtaactattacctgccacccctattacatcttacgtCcAtataca
ctgttatacaacgcgtcatggcggggtatgcgttttggtcgtcgtacgctcgatcgttaCcgtacgGc
n = 69
s
s1 = 26
s2 = 21
s3= 3
s4 = 56
s5 = 60

Scoring Motifs

- Given **s** = $(s_1, \ldots s_t)$ and ***DNA***:

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



The Motif Finding Problem

- If starting positions **s**=($s_1, s_2, \ldots s_t$) are given, finding consensus is easy even with mutations in the sequences because we can simply construct the profile to find the motif (consensus)

- But… the starting positions **s** are usually not given. How can we find the "best" profile matrix?

The Motif Finding Problem: Formulation

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

- Input: A t x n matrix of ***DNA***, and ℓ, the length of the pattern to find

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

The Motif Finding Problem: Brute Force Solution

- Compute the scores for each possible combination of starting positions **s**
- The best score will determine the best profile and the consensus pattern in ***DNA***
- The goal is to maximize *Score*(**s**,***DNA***) by varying the starting positions s_j, where:

$$s_j = [1, \ldots, n-\ell+1]$$

$$j = [1, \ldots, t]$$

BruteForceMotifSearch

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)
4. if (*Score*(s,*DNA*) > *bestScore*)
5. *bestScore* <- *score*(s, *DNA*)
6. bestMotif <- (s_1,s_2 , . . . , s_t)
7. return bestMotif

Running Time of BruteForceMotifSearch

- Varying $(n - \ell + 1)$ positions in each of t sequences, we're looking at $(n - \ell + 1)^t$ sets of starting positions
- For each set of starting positions, the scoring function makes ℓ operations, so complexity is $\ell(n - \ell + 1)^t = O(\ell n^t)$
- That means that for $t = 8$, $n = 1000$, $\ell = 10$ we must perform approximately 10^{20} computations – it will take billions 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

Hamming Distance

- Hamming distance:
  - $d_H(\boldsymbol{v},\boldsymbol{w})$ is the number of nucleotide pairs that do not match when ***v*** and ***w*** are aligned. For example:

$$d_H(\text{AAAAAA},\text{ACAAAC}) = 2$$

Total Distance: Example

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

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 ***i***, compute all $d_H(\boldsymbol{v}, \boldsymbol{x})$, where ***x*** is an ℓ-mer with starting position s_i

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

The Median String Problem: Formulation

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

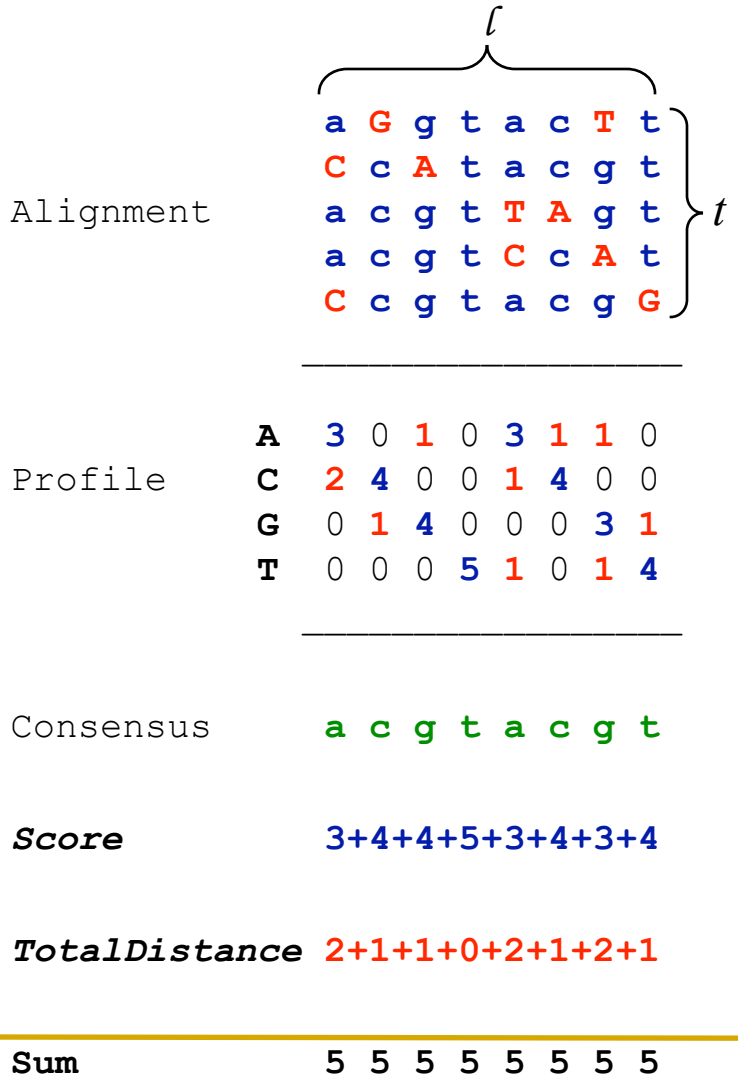
Median String Search Algorithm

```
1.  MedianStringSearch (DNA, t, n, l)
2.  bestWord <- AAA…A
3.  bestDistance <- ∞
4.      for each l-mer s from AAA…A to TTT…T
        if TotalDistance(s,DNA) < bestDistance
5.          bestDistance<-TotalDistance(s,DNA)
6.          bestWord <- s
7.      return bestWord
```

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
- Need to show that minimizing *TotalDistance* is equivalent to maximizing *Score*

We are looking for the same thing



- At any column ***i***

$Score_i + TotalDistance_i = t$

- Because there are ℓ columns

$Score + TotalDistance = \ell * t$

- Rearranging:

$Score = \ell * t - TotalDistance$

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

Motif Finding Problem vs. Median String Problem

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

Motif Finding: Improving the Running Time

Recall the BruteForceMotifSearch:

```
3.  BruteForceMotifSearch(DNA, t, n, ℓ)
4.  bestScore ß 0
5.  for each s=(s1,s2 , . . ., st) from (1,1 . . . 1) to (n-ℓ+1, . . ., n-ℓ+1)
6.      if (Score(s,DNA) > bestScore)
7.          bestScore ß Score(s, DNA)
8.          bestMotif ß (s1,s2 , . . . , st)
9.  return bestMotif
```

Structuring the Search

- How can we perform the line

for each s=(s_1,s_2 , . . ., s_t) from (1,1 . . . 1) to (n-l+1, . . ., n-l+1) ?

- We need a method for efficiently structuring and navigating the many possible motifs
- This is not very different than exploring all t-digit numbers

Median String: Improving the Running Time

```
1.  MedianStringSearch (DNA, t, n, ℓ)
2.  bestWord ß AAA…A
3.  bestDistance ß ∞
4.      for each ℓ-mer s from AAA…A to TTT…T
        if TotalDistance(s,DNA) < bestDistance
5.          bestDistanceßTotalDistance(s,DNA)
6.          bestWord ß s
7.      return bestWord
```

Structuring the Search

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

$\overbrace{\text{aa… aa}}^{l}$

aa… ac

aa… ag

aa… at

.

.

tt… tt

How to organize this search?

Alternative Representation of the Search Space

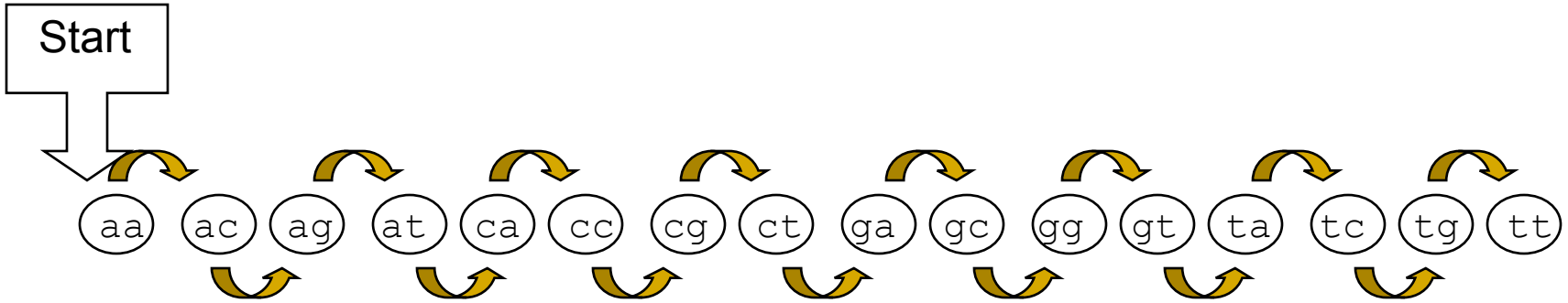
- Let **A** = 1, **C** = 2, **G** = 3, **T** = 4
- Then the sequences from AA…A to TT…T become:

$$\underbrace{11\ldots11}_{l}$$
$$11\ldots12$$
$$11\ldots13$$
$$11\ldots14$$
$$\vdots$$
$$44\ldots44$$

- Notice that the sequences above simply list all numbers as if we were counting on base 4 without using 0 as a digit

Linked List

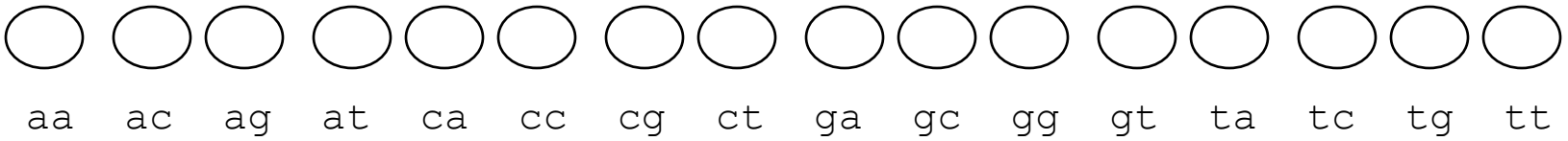
- Suppose $\ell = 2$



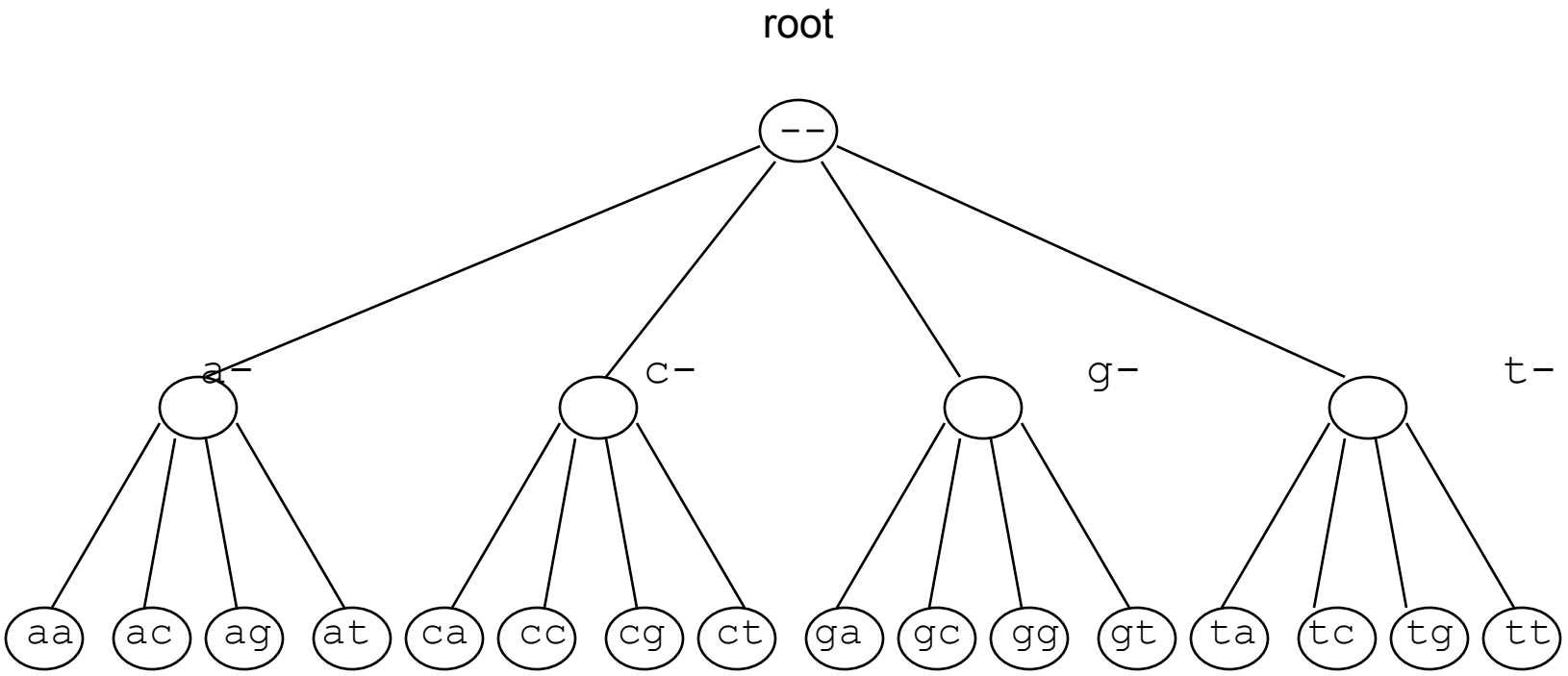
- Need to visit all the predecessors of a sequence before visiting the sequence itself

Linked List (cont'd)

- Linked list is not the most efficient data structure for motif finding
- Let's try grouping the sequences by their prefixes



Search Tree



Analyzing Search Trees

- Characteristics of the search trees:
  - The sequences are contained in its leaves
  - The parent of a node is the prefix of its children
- How can we move through the tree?

Moving through the Search Trees

- Four common moves in a search tree that we are about to explore:
  - Move to the next leaf
  - Visit all the leaves
  - Visit the next node
  - Bypass the children of a node

Visit the Next Leaf

Given a current leaf **a** , we need to compute the "next" leaf:

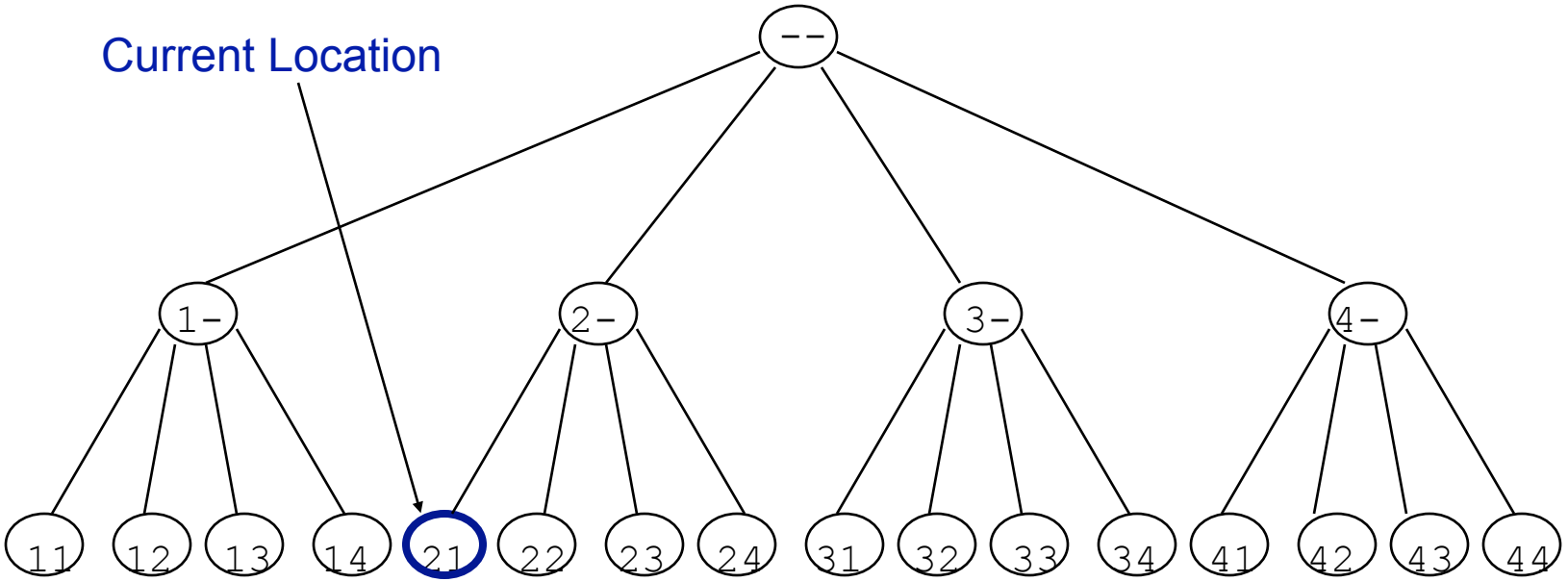
```
2.  NextLeaf( a,L, k )        // a : the array of digits
3.  for i ß L to 1            // L: length of the array
4.      if a_i < k            // k : max digit value
5.          a_i ß a_i + 1
6.              return a
7.          a_i ß 1
8.  return a
```

NextLeaf (cont'd)

- The algorithm is common addition in radix ***k:***
- Increment the least significant digit
- "Carry the one" to the next digit position when the digit is at maximal value

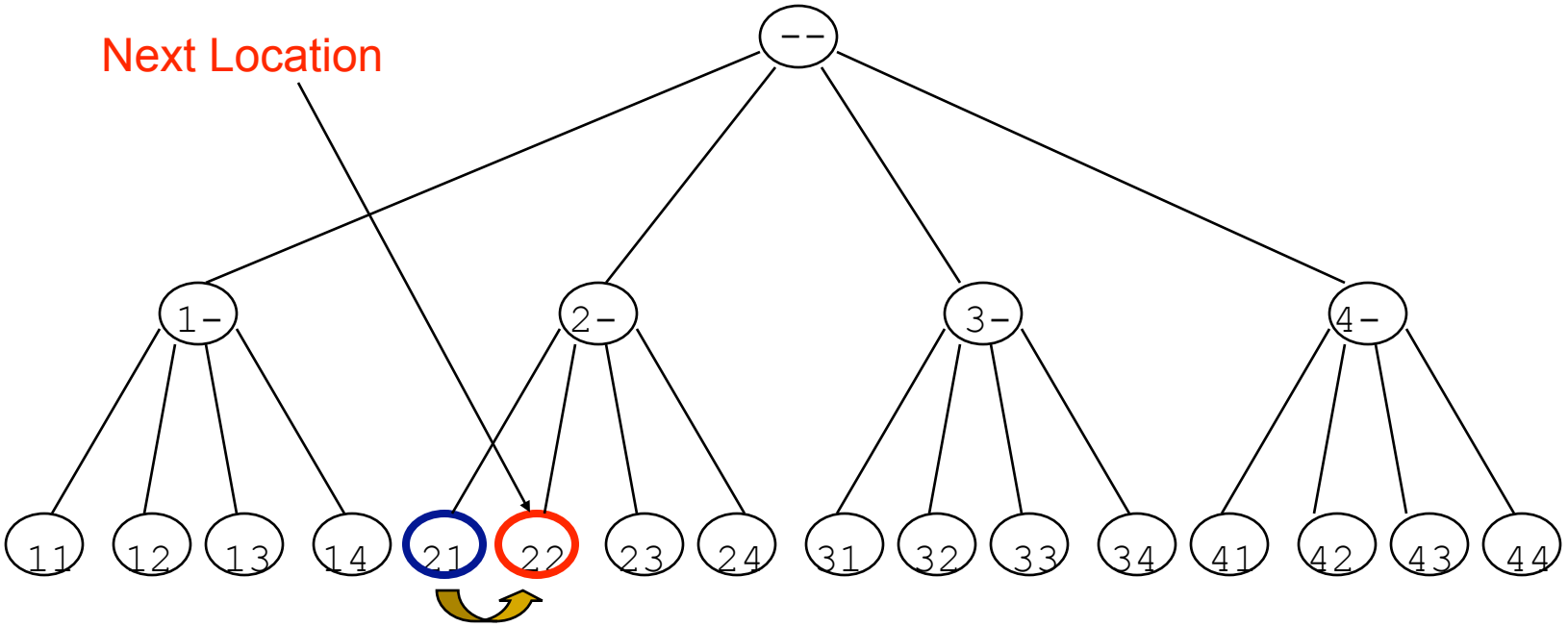
NextLeaf: Example

- Moving to the next leaf:



NextLeaf: Example (cont'd)

- Moving to the next leaf:



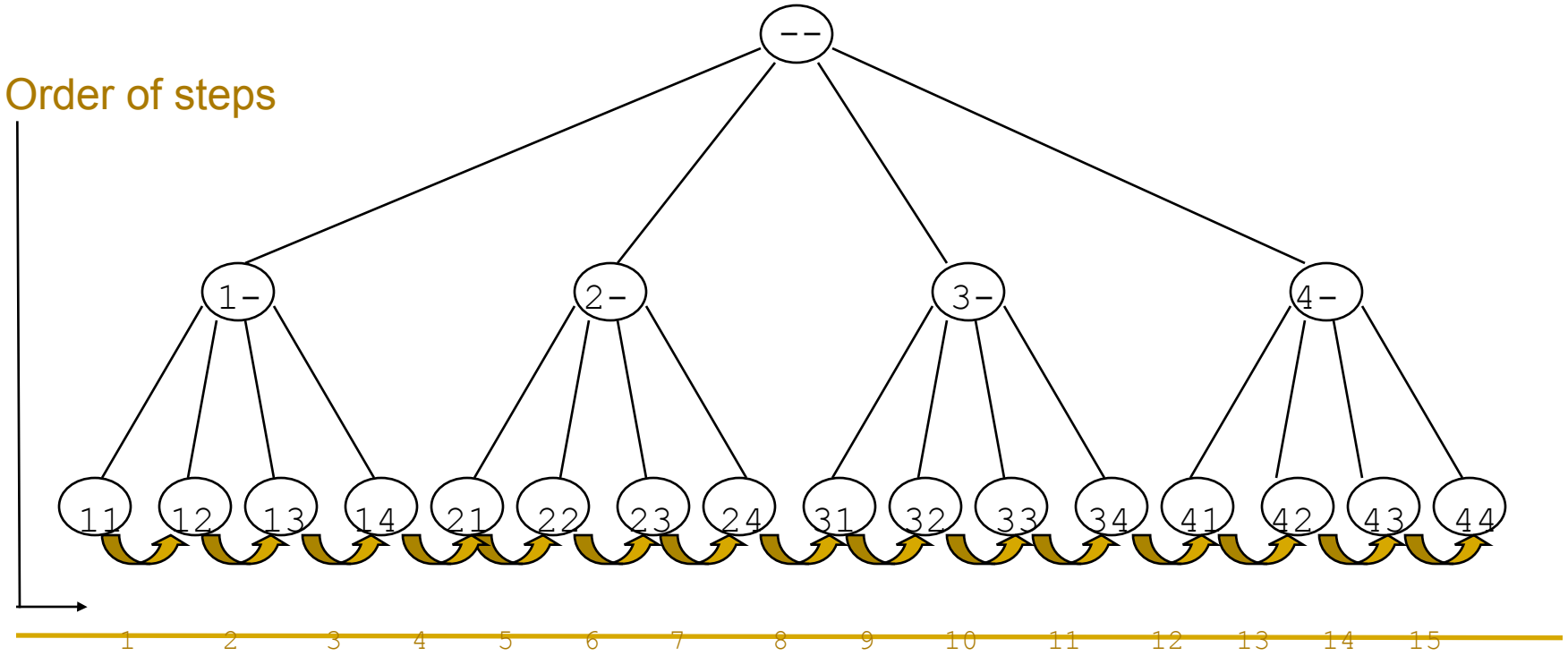
Visit All Leaves

- Printing all permutations in ascending order:

```
3.  AllLeaves(L,k)   // L: length of the sequence
4.  a ß (1,...,1)    // k : max digit value
5.   while forever   // a : array of digits
6.       output a
7.       a ß NextLeaf(a,L,k)
8.       if a = (1,...,1)
9.           return
```

Visit All Leaves: Example

- Moving through all the leaves in order:



Depth First Search

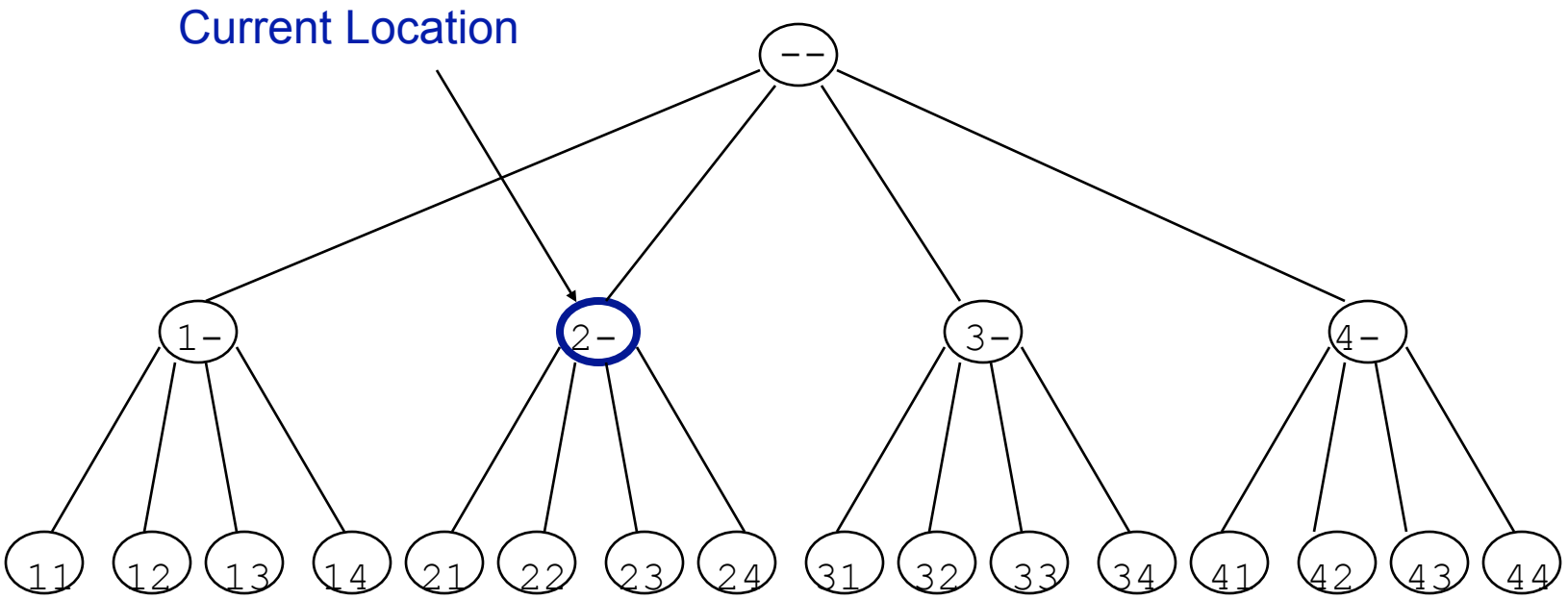
- So we can search leaves
- How about searching all vertices of the tree?
- We can do this with a *depth first* search

Visit the Next Vertex

```
1.  NextVertex(a,i,L,k)      // a : the array of digits
2.    if i < L               // i  : prefix length
3.      a i+1 ß 1            // L: max length
4.      return ( a,i+1)      // k : max digit value
5.    else
6.      for j ß ℓ to 1
7.        if aj < k
8.          aj ß aj +1
9.          return( a,j )
10.   return(a,0)
```

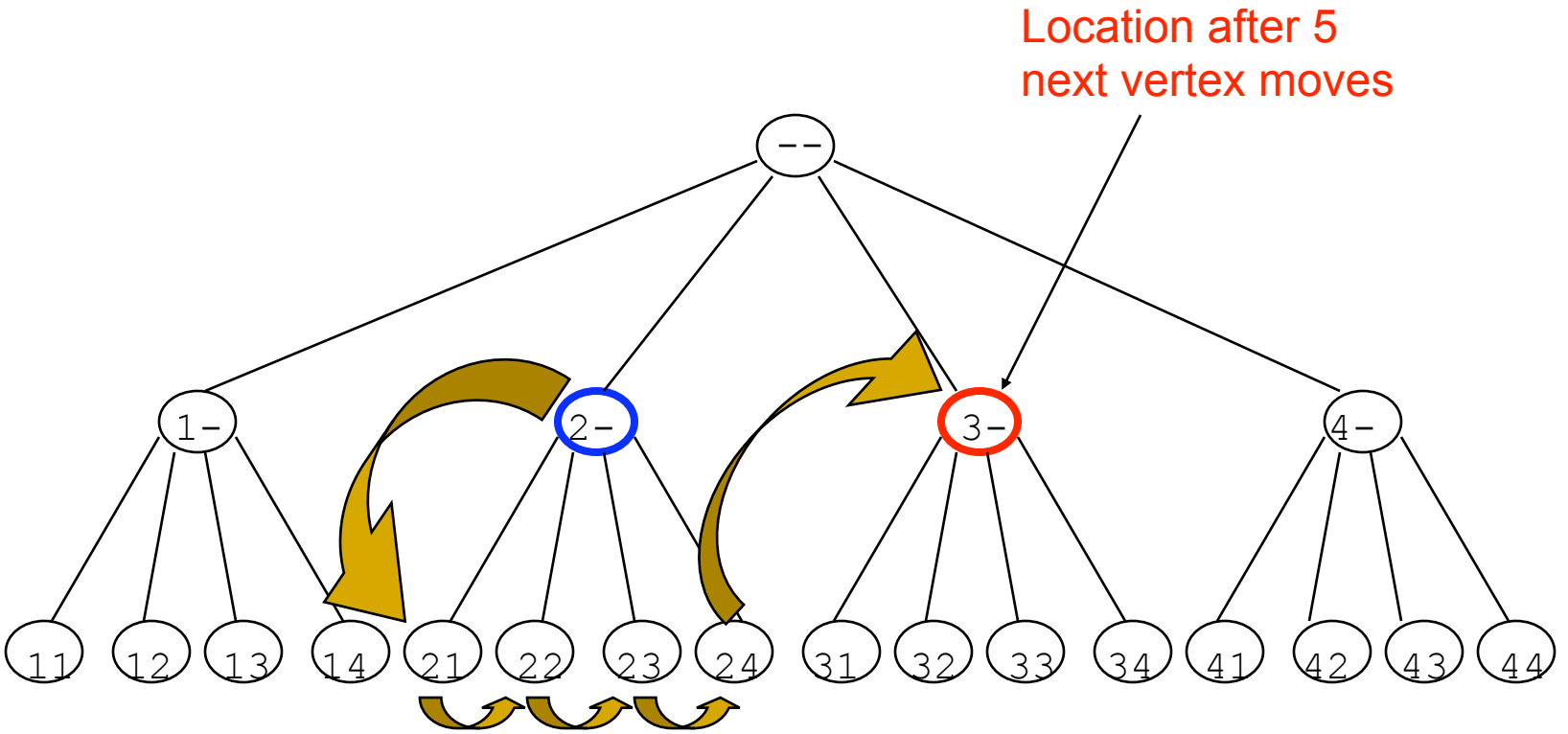
Example

- Moving to the next vertex:



Example

- Moving to the next vertices:



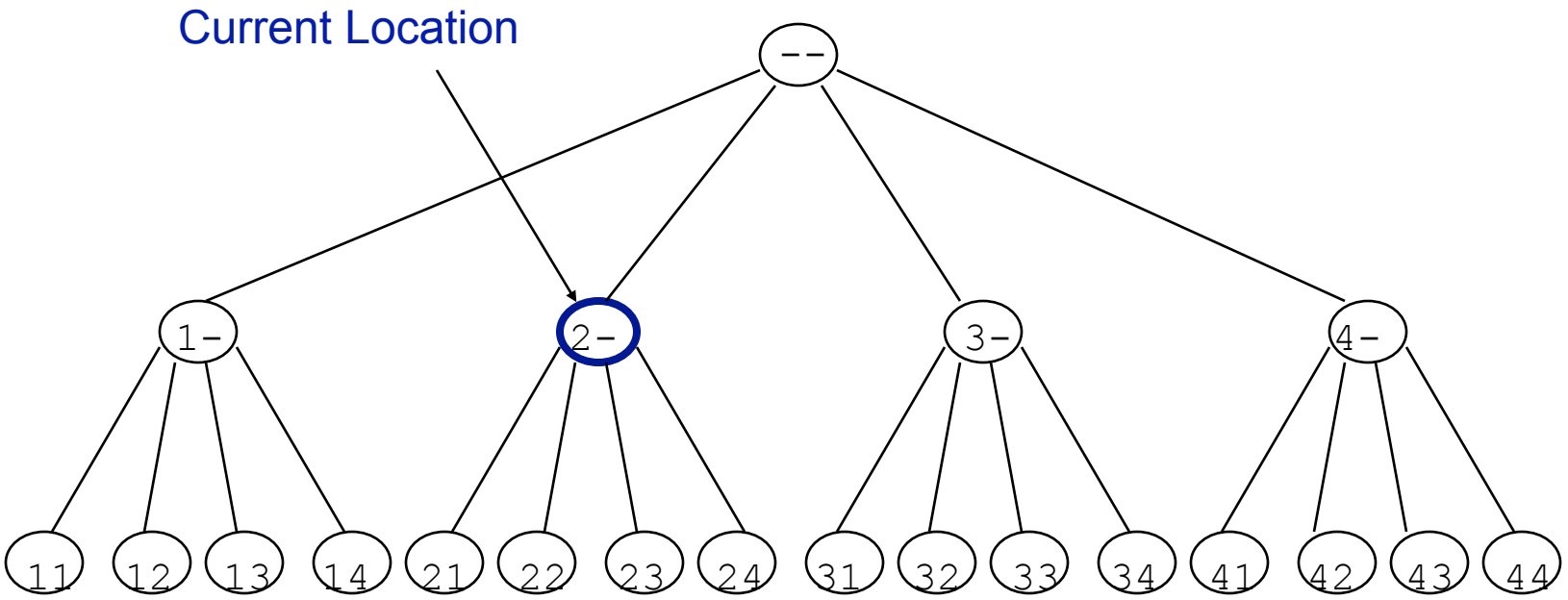
Bypass Move

- Given a prefix (internal vertex), find next vertex after skipping all its children

```
3.  Bypass(a,i,L,k)      // a: array of digits
4.  for j ß i to 1       // i : prefix length
5.      if a_j < k       // L: maximum length
6.          a_j ß a_j +1 // k : max digit value
7.          return(a,j)
8.  return(a,0)
```

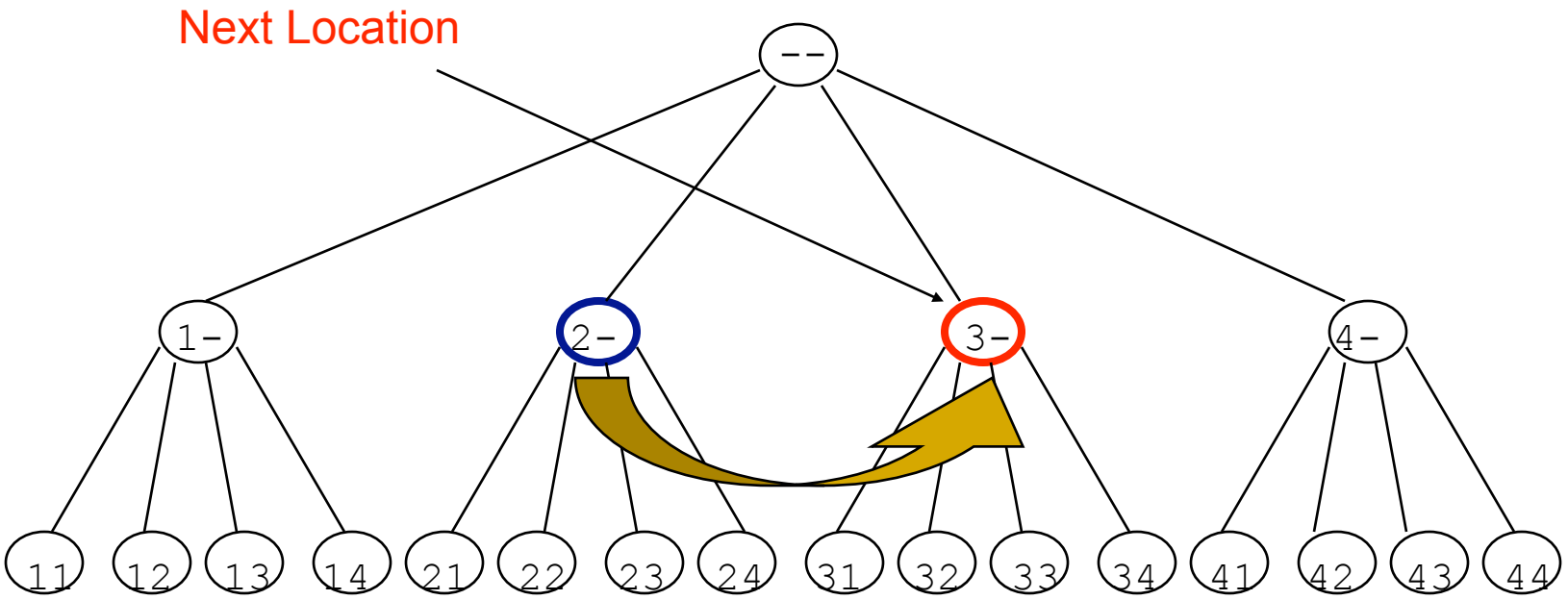
Bypass Move: Example

- Bypassing the descendants of “2-”:



Example

- Bypassing the descendants of “2-”:



Revisiting Brute Force Search

- Now that we have method for navigating the tree, lets look again at BruteForceMotifSearch

Brute Force Search Again

```
2.  BruteForceMotifSearchAgain(DNA, t, n, ℓ)
3.  s ß (1,1,…, 1)
4.  bestScore ß Score(s,DNA)
5.  while forever
6.      s ß  NextLeaf (s, t, n- ℓ+1)
7.      if (Score(s,DNA) > bestScore)
8.          bestScore ß Score(s, DNA)
9.          bestMotif ß (s1,s2 , . . . , st)
10. return bestMotif
```

Can We Do Better?

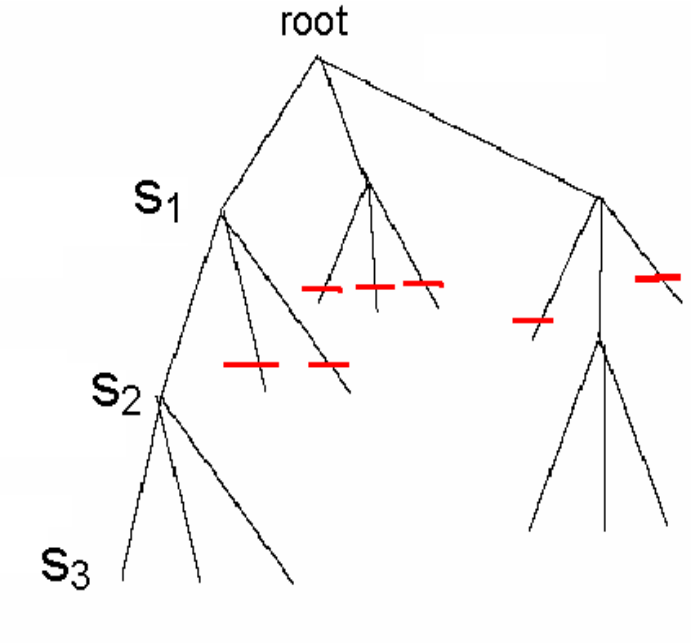
- Sets of **s=**(s_1, s_2, …,s_t) may have a weak profile for the first *i* positions (s_1, s_2, …,s_i)
- Every row of alignment may add at most ℓ to Score
- Optimism: if all subsequent *(t-i)* positions (s_{i+1}, … s_t) add

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

- If Score(**s**,***i***,***DNA***) + (***t*** – *i*) * ℓ < BestScore, it makes no sense to search in vertices of the current subtree
  - Use **ByPass()**

Branch and Bound Algorithm for 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 $(n - \ell + 1)^{t-i}$ leaves
  - Use **NextVertex()** and **ByPass()** to navigate the tree

Pseudocode for Branch and Bound Motif Search

```
1.   BranchAndBoundMotifSearch(DNA,t,n,ℓ)
2.   s ß (1,…,1)
3.   bestScore ß 0
4.   i ß 1
5.   while i > 0
6.    if i < t
7.        optimisticScore ß Score(s, i, DNA) +(t – i ) * ℓ
8.        if optimisticScore < bestScore
9.           (s, i) ß Bypass(s,i, n-ℓ+1)
10.     else
11.       (s, i) ß NextVertex(s, i, n-ℓ+1)
12.  else
13.    if Score(s,DNA) > bestScore
14.       bestScore ß Score(s)
15.       bestMotif ß (s1, s2, s3, …, st)
16.             (s,i) ß NextVertex(s,i,t,n-ℓ+ 1)
17.  return bestMotif
```

Median String Search Improvements

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

Branch and Bound Applied to Median String Search

- Note that if 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

- BranchAndBoundMedianStringSearch(*DNA,t,n,ℓ*)
- s ß (1,…,1)
- *bestDistance* ß ∞
- *i* ß 1
- while *i* > 0
- if *i* < *ℓ*
- *prefix* ß string corresponding to the first *i* nucleotides of s
- *optimisticDistance* ß TotalDistance(*prefix,DNA*)
- if *optimisticDistance* > *bestDistance*
- (s, *i*) ß Bypass(s,*i, ℓ, 4*)
- else
- (s, *i*) ß NextVertex(s, *i, ℓ, 4*)
- else
- *word* ß nucleotide string corresponding to s
- if TotalDistance(s,*DNA*) < *bestDistance*
- *bestDistance* ß TotalDistance(*word*, *DNA*)
- *bestWord* ß *word*
- (s,*i*) ß NextVertex(s,*i,ℓ, 4*)
- return *bestWord*

Improving the Bounds

- Given an ℓ-mer $\boldsymbol{w}$, divided into two parts at point $\boldsymbol{i}$
  - $\boldsymbol{u}$: prefix $w_1, \ldots, w_{\boldsymbol{i}}$,
  - $\boldsymbol{v}$: suffix $w_{\boldsymbol{i}+1}, \ldots, w_{\ell}$
- Find minimum distance for $\boldsymbol{u}$ in a sequence
- No instances of $\boldsymbol{u}$ in the sequence have distance less than the minimum distance
- Note this doesn't tell us anything about whether $\boldsymbol{u}$ is part of any motif. We only get a minimum distance for prefix $\boldsymbol{u}$

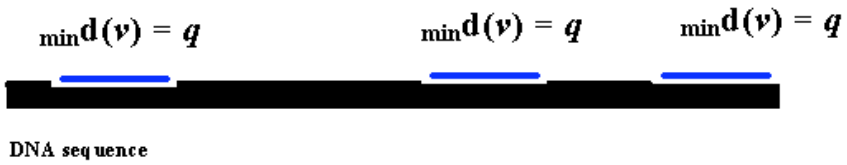
Improving the Bounds (cont'd)

- Repeating the process for the suffix ***v*** gives us a minimum distance for ***v***

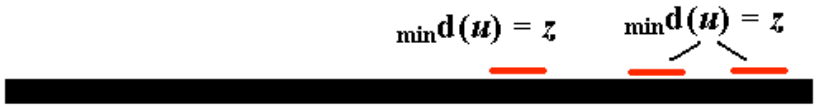
- Since ***u*** and ***v*** are two substrings of ***w***, and included in motif ***w***, we can assume that the minimum distance of ***u*** plus minimum distance of ***v*** can only be less than the minimum distance for ***w***

Better Bounds

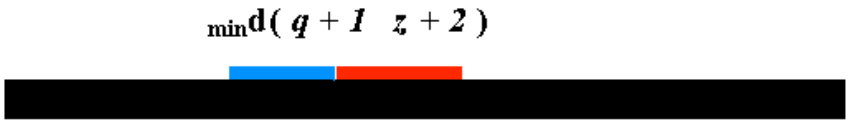
Searching for prefix V
We may find many instances of prefix V with a minimum distance q



Likewise for U



But for U and V combined, U is not at its minimum distance location, neither is V



But at least we know w (prefix u suffix v) cannot have distance *less* than $_{min}d(v) + {}_{min}d(u)$

Better Bounds (cont'd)

- If d(***prefix***) + d(***suffix***) ≥ ***bestDistance***:
  - Motif ***w*** (*prefix.suffix)* cannot give a better (lower) score than d(***prefix***) + d(***suffix***)
  - In this case, we can **ByPass()**

Better Bounded Median String Search

```
1.  ImprovedBranchAndBoundMedianString(DNA, t, n, ℓ)
2.    s = (1, 1, …, 1)
3.    bestdistance = ∞
4.    i = 1
5.    while i > 0
6.    if i < ℓ
7.      prefix = nucleotide string corresponding to (s1, s2, s3, …, si )
8.      optimisticPrefixDistance = TotalDistance (prefix, DNA)
9.        if (optimisticPrefixDistance < bestsubstring[ i ])
10.           bestsubstring[ i ] = optimisticPrefixDistance
11.           if (ℓ - i < i )
12.                 optimisticSufxDistance = bestsubstring[ℓ -i ]
13.           else
14.             optimisticSufxDistance = 0;
15.           if optimisticPrefixDistance + optimisticSufxDistance ≥ bestDistance
16.             (s, i ) = Bypass(s, i, ℓ, 4)
17.           else
18.             (s, i ) = NextVertex(s, i, ℓ,4)
19.     else
20.       word = nucleotide string corresponding to (s1,s2, s3, …, st)
21.       if TotalDistance( word, DNA) < bestDistance
22.         bestDistance = TotalDistance(word, DNA)
23.         bestWord = word
24.         (s,i) = NextVertex(s, i,ℓ, 4)
25.   return bestWord
```

More on the Motif Problem

- Exhaustive Search and Median String are both exact algorithms
- They always find the optimal solution, though they may be too slow to perform practical tasks
- Many algorithms sacrifice optimal solution for speed

CONSENSUS: Greedy Motif Search

- Find two closest l-mers in sequences 1 and 2 and forms *2 x l alignment matrix* with *Score(***s***,2,DNA)*
- At each of the following *t-2* iterations CONSENSUS finds a "best" *l*-mer in sequence *i* from the perspective of the already constructed *(i-1) x l* alignment matrix for the first *(i-1)* sequences
- In other words, it finds an *l*-mer in sequence *i* maximizing

*Score(***s***,i,DNA)*

under the assumption that the first *(i-1) l*-mers have been already chosen

- CONSENSUS sacrifices optimal solution for speed: in fact the bulk of the time is actually spent locating the first 2 *l*-mers

Some Motif Finding Programs

- **CONSENSUS**
  Hertz, Stromo (1989)
- **GibbsDNA**
  Lawrence et al (1993)
- **MEME**
  Bailey, Elkan (1995)
- **RandomProjections**
  Buhler, Tompa (2002)
- **MULTIPROFILER**
  Keich, Pevzner (2002)
- **MITRA**
  Eskin, Pevzner (2002)
- **Pattern Branching**
  Price, Pevzner (2003)

Planted Motif Challenge

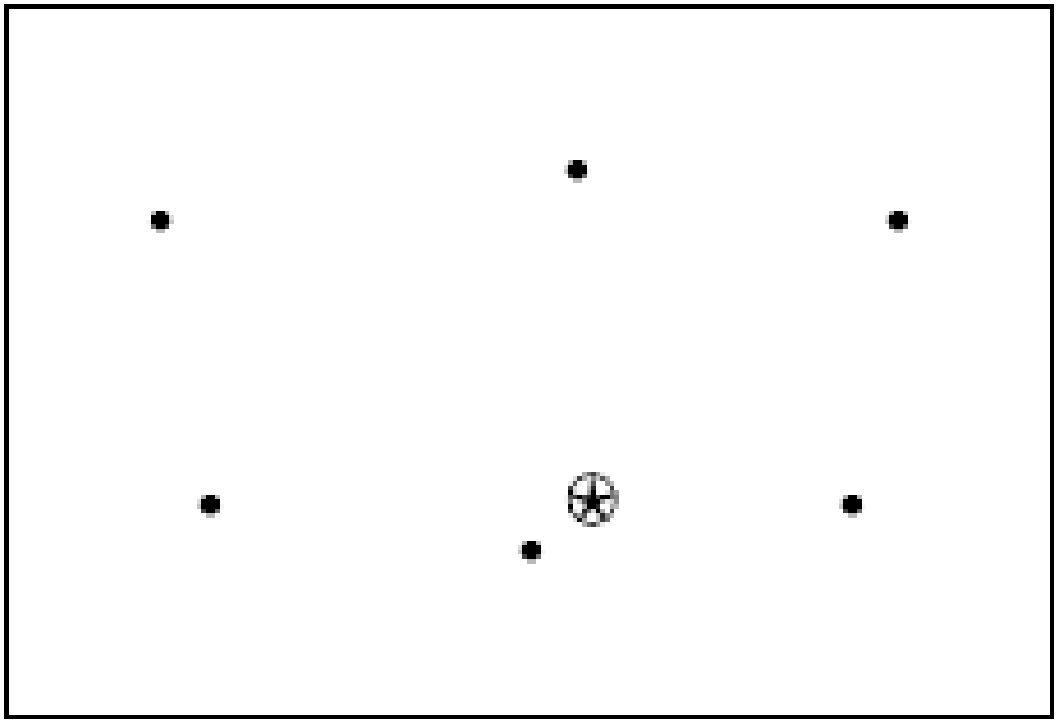
- Input:
  - *n* sequences of length *m* each.
- Output:
  - Motif *M*, of length *l*
  - Variants of interest have a hamming distance of *d* from *M*

How to proceed?

- Exhaustive search?

- Run time is high

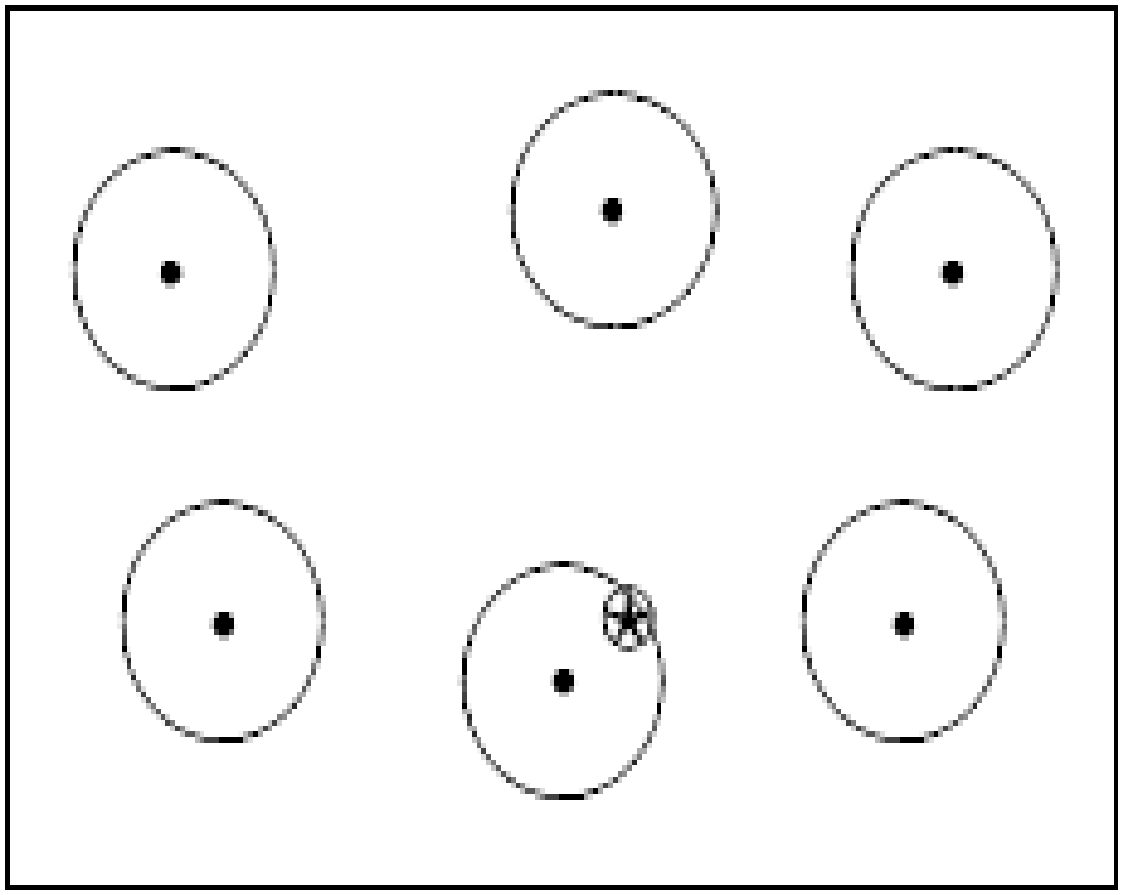
How to search motif space?



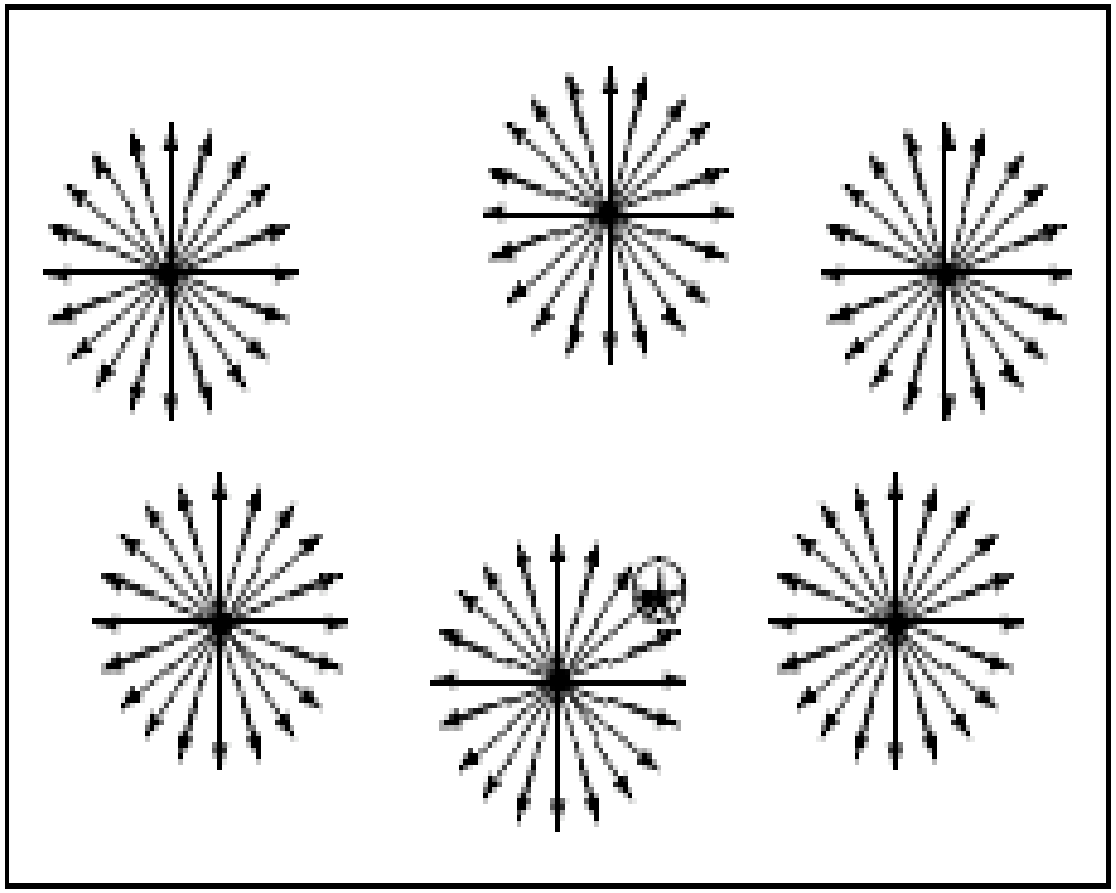
Start from random sample strings

Search motif space for the star

Search small neighborhoods

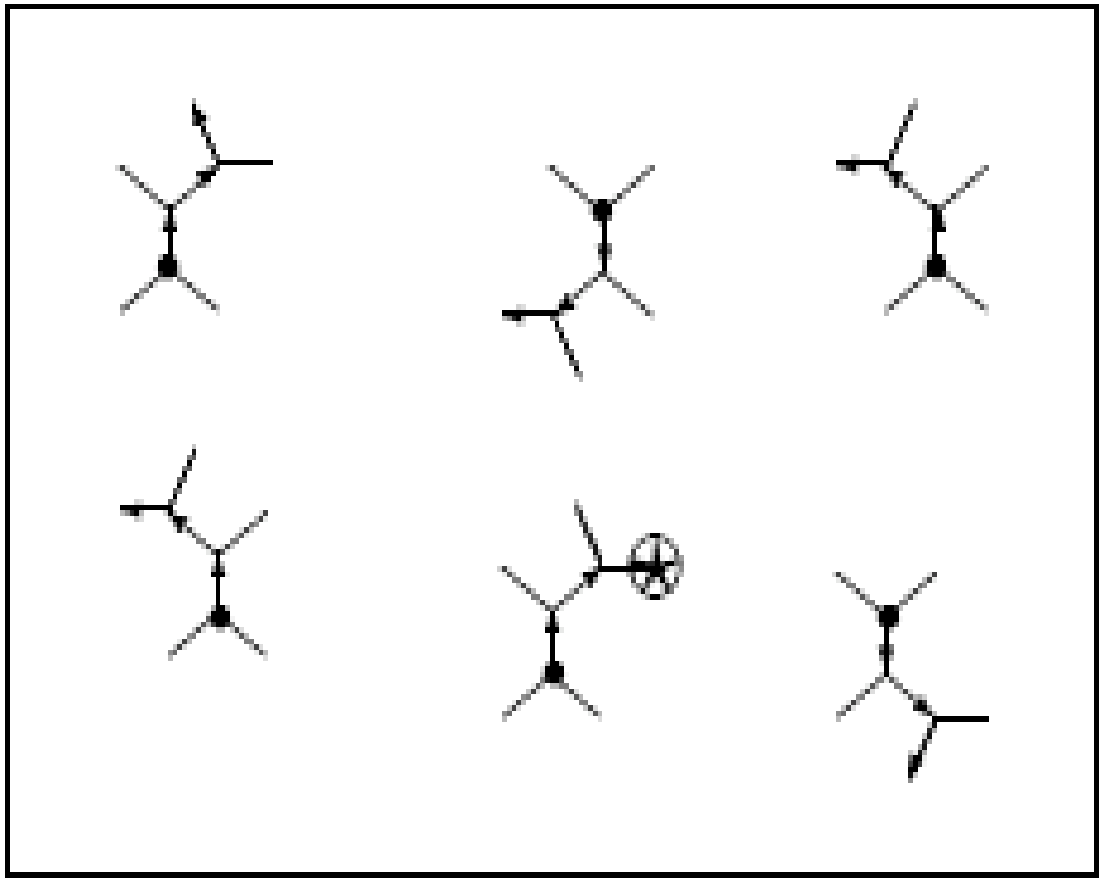


Exhaustive local search



A lot of work, most of it unecessary

Best Neighbor



Branch from the seed strings

Find best neighbor - highest score

Don't consider branches where the upper bound is not as good as best score so far

Scoring

- PatternBranching use total distance score:
- For each sequence S_i in the sample $S = \{S_1, \ldots, S_n\}$, let

  $d(A, S_i) = \min\{d(A, P) \mid P \in S_i\}$.
- Then the total distance of A from the sample is

  $d(A, S) = \sum_{S_i \in S} d(A, S_i)$.

- For a pattern A, let $D_{=Neighbor}(A)$ be the set of patterns which differ from A in exactly 1 position.
- We define BestNeighbor(A) as the pattern $B \in D_{=Neighbor}(A)$ with lowest total distance d(B, S).

PatternBranching Algorithm

```
PatternBranching(S, l, k)
d ← ∞
For each l-mer A_0 in S
    For j ← 0 to k
        If d(A_j, S) < d
            Motif ← A_j
            d ← d(A_j, S)
        A_{j+1} ← BestNeighbor(A_j)
Output Motif
```

PatternBranching Performance

- PatternBranching is faster than other pattern-based algorithms
- Motif Challenge Problem:
  - sample of n = 20 sequences
  - N = 600 nucleotides long
  - implanted pattern of length l = 15
  - k = 4 mutations

Algorithm	Success Rate	Running Time
PROJECTION	about 100%	2 minutes
MITRA	100%	5 minutes
MULTIPROFILER	99.7%	1 minute
PatternBranching	99.7%	3 seconds

PMS (Planted Motif Search)

- Generate all possible *l*-mers from out of the input sequence S_i. Let C_i be the collection of these *l*-mers.
- Example:

  AAGTCAGGAGT

  C_i = 3-mers:

  AAG AGT GTC TCA CAG AGG GGA GAG AGT

All patterns at Hamming distance $d = 1$

AAG AGT GTC TCA CAG AGG GGA
GAG AGT

CAG CGT ATC ACA AAG CGG AGA
AAG CGT

GAG GGT CTC CCA GAG TGG CGA
CAG GGT

TAG TGT TTC GCA TAG GGG TGA
TAG TGT

ACG ACT GAC TAA CCG ACG GAA
GCG ACT

AGG ATT GCC TGA CGG ATG GCA
GGG ATT

ATG AAT GGC TTA CTG AAG GTA
GTG AAT

AAC AGA GTA TCC CAA AGA GGC
GAA AGA

Sort the lists

AAG AGT GTC TCA CAG AGG GGA
GAG AGT

AAA AAT ATC ACA AAG AAG AGA
AAG AAT

AAC ACT CTC CCA CAA ACG CGA
CAG ACT

AAT AGA GAC GCA CAC AGA GAA
GAA AGA

ACG AGC GCC TAA CAT AGC GCA
GAC AGC

AGG AGG GGC TCC CCG AGT GGC
GAT AGG

ATG ATT GTA TCG CGG ATG GGG
GCG ATT

CAG CGT GTG TCT CTG CGG GGT

Eliminate duplicates

AAG AGT GTC TCA CAG AGG GGA

GAG AGT

AAA AAT ATC ACA AAG AAG AGA

AAG AAT

AAC ACT CTC CCA CAA ACG CGA

CAG ACT

AAT AGA GAC GCA CAC AGA GAA

GAA AGA

ACG AGC GCC TAA CAT AGC GCA

GAC AGC

AGG AGG GGC TCC CCG AGT GGC

GAT AGG

ATG ATT GTA TCG CGG ATG GGG

GCG ATT

CAG CGT GTG TCT CTG CGG GGT

Find motif common to all lists

- Follow this procedure for all sequences
- Find the motif common all L_i (once duplicates have been eliminated)
- This is the planted motif

PMS Running Time

- It takes time to
  - Generate variants
  - Sort lists $O\left(m\binom{l}{d}3^d\right)$
  - Find and eliminate duplicates
- Running time of this algorithm:

$$O\left(nm\binom{l}{d}3^d\frac{l}{w}\right)$$

w is the word length of the computer