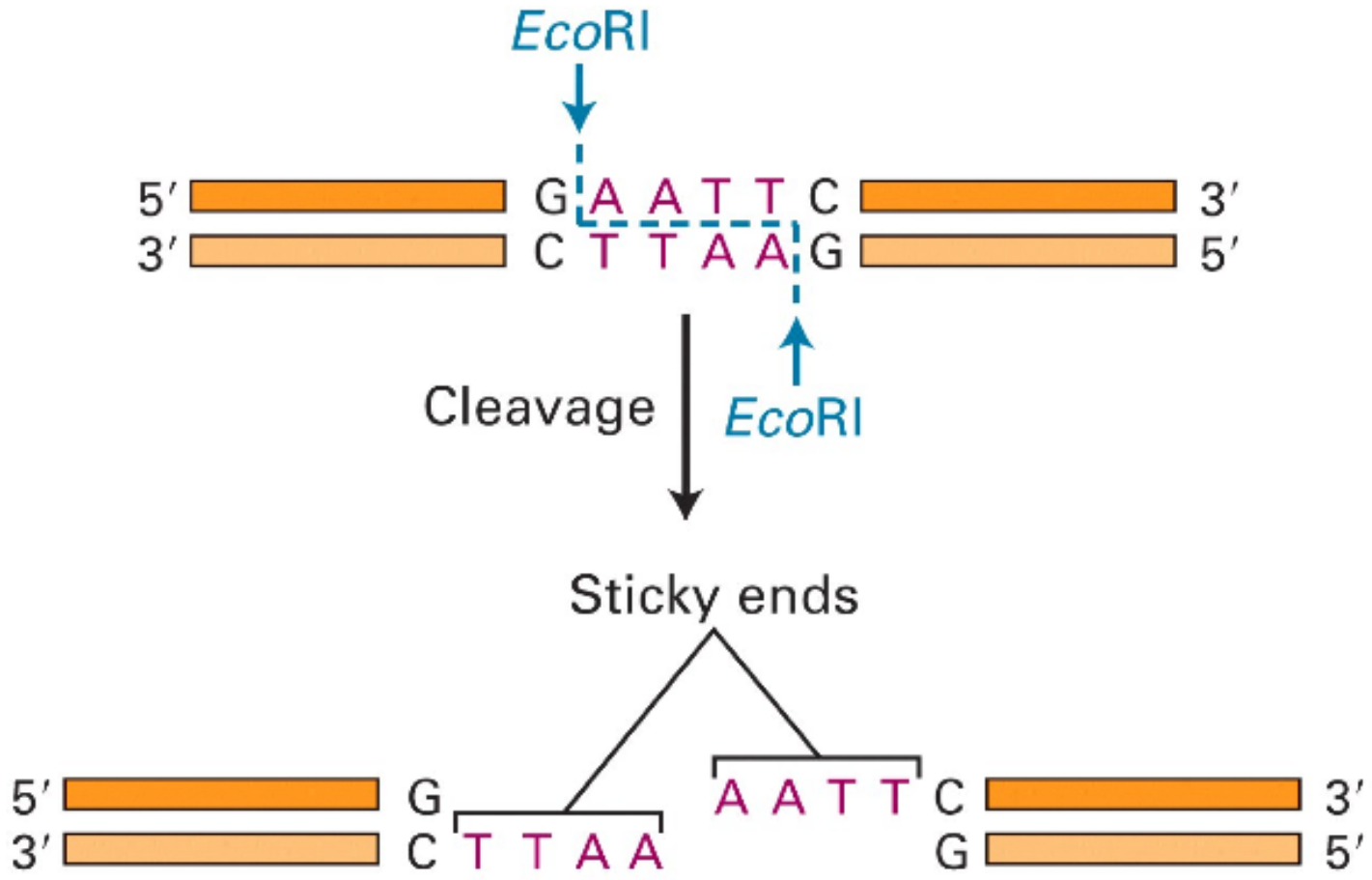
CS481: Bioinformatics Algorithms

Can Alkan
EA224
calkan@cs.bilkent.edu.tr

http://www.cs.bilkent.edu.tr/~calkan/teaching/cs481/

DNA MAPPING

Molecular Scissors



Recognition Sites of Restriction Enzymes

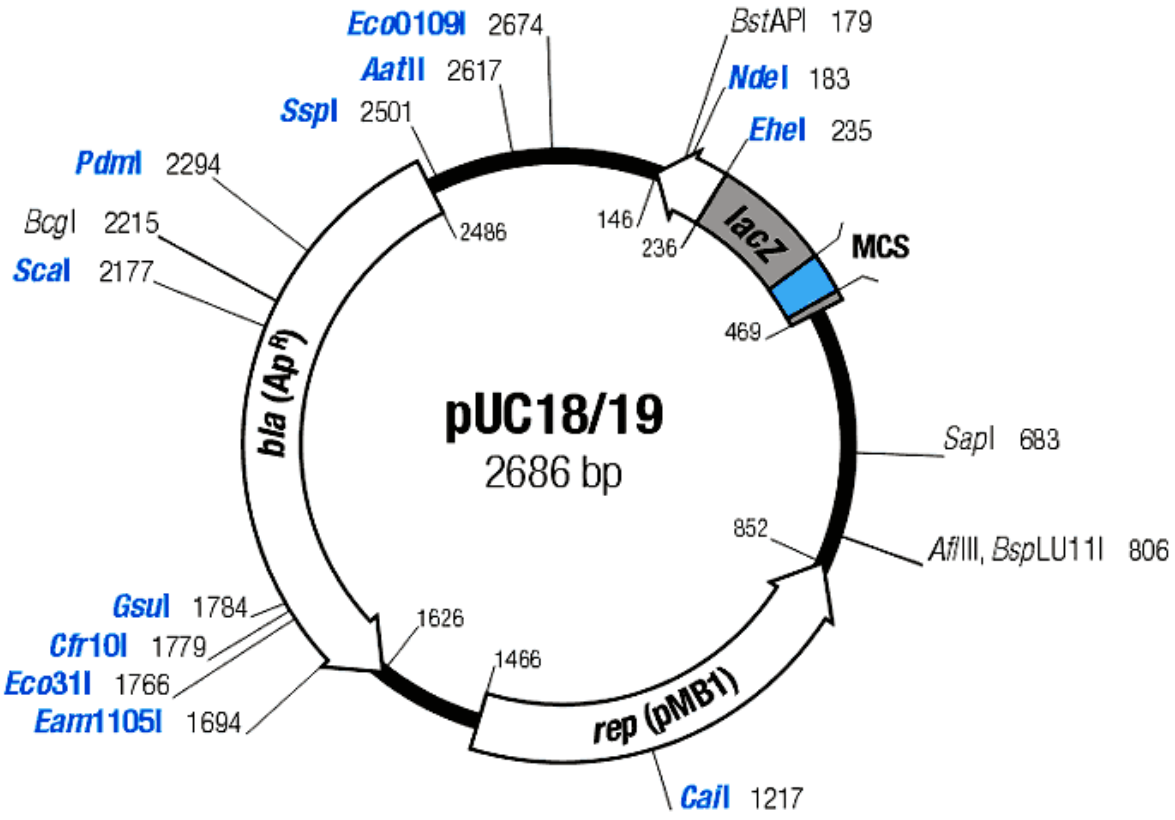
Enzyme	Source Microorganism	Recognition Site*	Ends Produced
BamHI	*Bacillus amyloliquefaciens*	↓ -G-G-A-T-C-C- -C-C-T-A-G-G- ↑	Sticky
*Eco*RI	*Escherichia coli*	↓ G A A T T C C T T A A G ↑	Sticky
*Hin*dIII	*Haemophilus influenzae*	↓ -A-A-G-C-T-T- -T-T-C-G-A-A- ↑	Sticky
*Kpn*I	*Klebsiella pneumonia*	↓ -G-G-T-A-C-C- -C-C-A-T-G-G- ↑	Sticky

Uses of Restriction Enzymes

- Recombinant DNA technology
- Cloning
- cDNA/genomic library construction
- DNA mapping

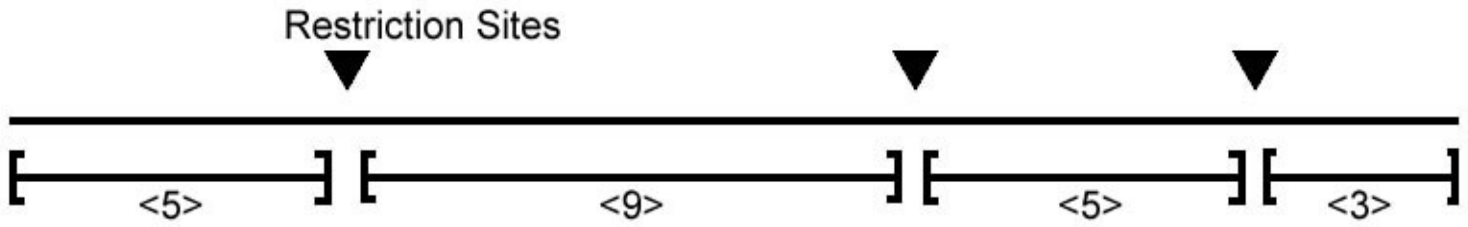
Restriction Maps

- A map showing positions of restriction sites in a DNA sequence
- If DNA sequence is known then construction of restriction map is a trivial exercise
- In early days of molecular biology DNA sequences were often unknown
- Biologists had to solve the problem of constructing restriction maps without knowing DNA sequences



Full Restriction Digest

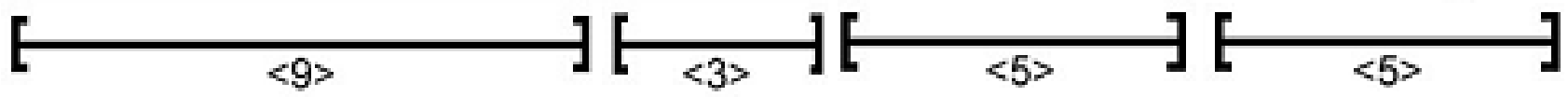
- Cutting DNA at each restriction site creates multiple restriction fragments:



- Is it possible to reconstruct the order of the fragments from the sizes of the fragments {3,5,5,9} ?

Full Restriction Digest: Multiple Solutions

- Alternative ordering of restriction fragments:



vs

Measuring Length of Restriction Fragments

- Restriction enzymes break DNA into restriction fragments.
- Gel electrophoresis is a process for separating DNA by size and measuring sizes of restriction fragments
- Can separate DNA fragments that differ in length in only 1 nucleotide for fragments up to 500 nucleotides long

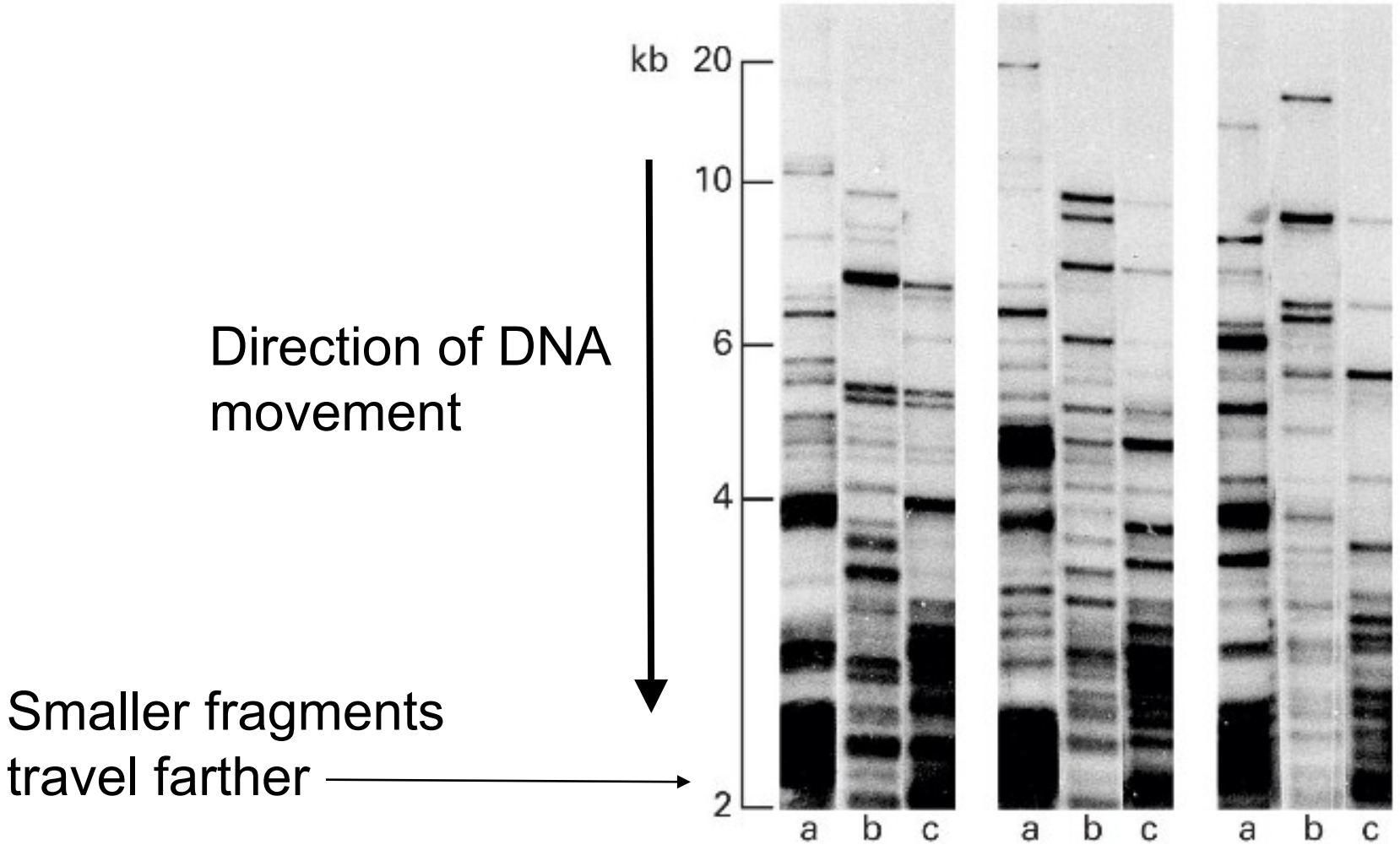
Gel Electrophoresis

- DNA fragments are injected into a gel positioned in an electric field
- DNA are negatively charged near neutral pH
  - The ribose phosphate backbone of each nucleotide is acidic; DNA has an overall negative charge
- DNA molecules move towards the positive electrode

Gel Electrophoresis (cont'd)

- DNA fragments of different lengths are separated according to size
  - Smaller molecules move through the gel matrix more readily than larger molecules
- The gel matrix restricts random diffusion so molecules of different lengths separate into different bands

Gel Electrophoresis: Example

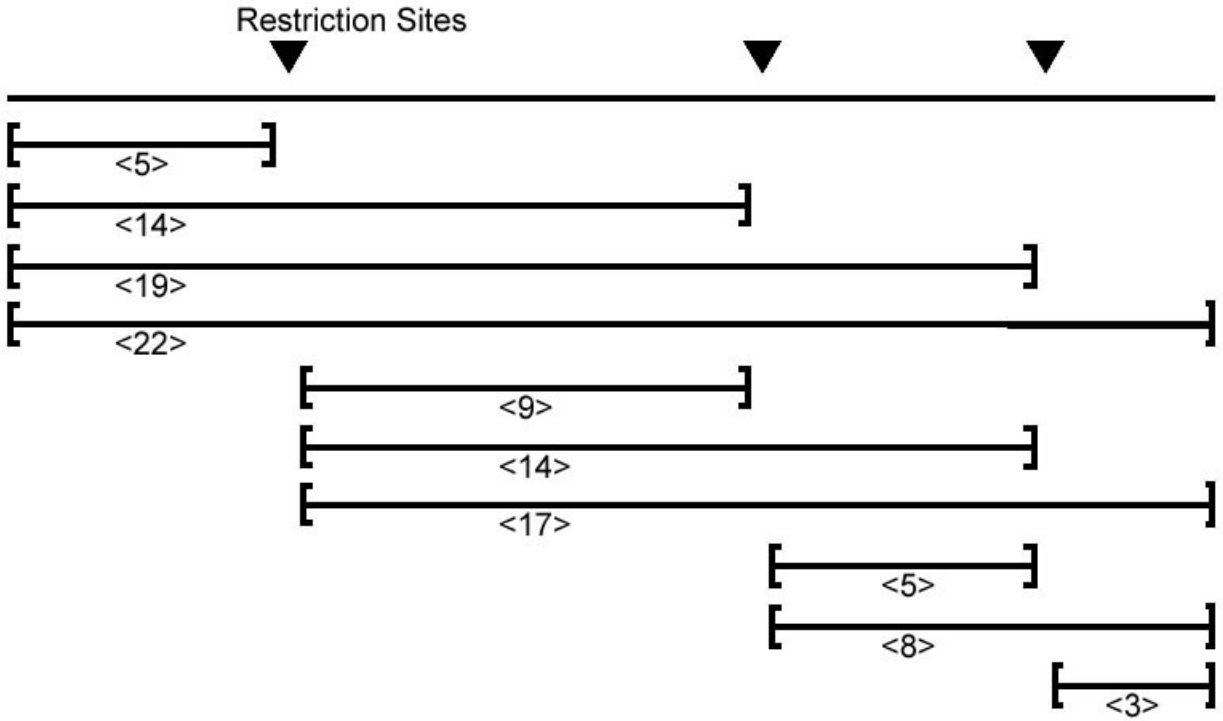


Partial Restriction Digest

- The sample of DNA is exposed to the restriction enzyme for only a limited amount of time to prevent it from being cut at all restriction sites
- This experiment generates the set of all possible restriction fragments between every two (not necessarily consecutive) cuts
- This set of fragment sizes is used to determine the positions of the restriction sites in the DNA sequence

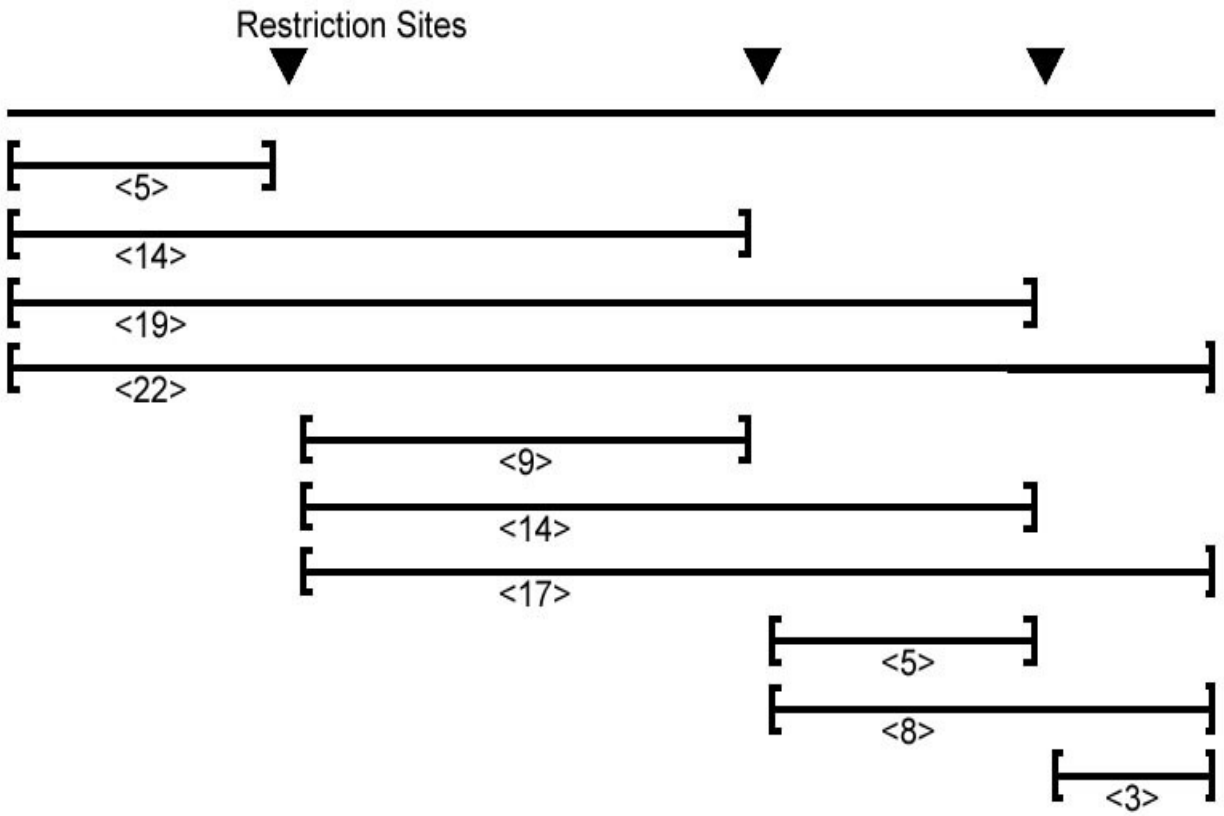
Partial Digest Example

- Partial Digest results in the following 10 restriction fragments:



Multiset of Restriction Fragments

- We assume that multiplicity of a fragment can be detected, i.e., the number of restriction fragments of the same length can be determined (e.g., by observing twice as much fluorescence intensity for a double fragment than for a single fragment)



Multiset: {3, 5, 5, 8, 9, 14, 14, 17, 19, 22}

Partial Digest Fundamentals

X: the set of ***n*** integers representing the location of all cuts in the restriction map, including the start and end

n: the total number of cuts

Δ***X***: the multiset of integers representing lengths of each of the C(n, 2) fragments produced from a partial digest

One More Partial Digest Example

X	0	2	4	7	10
0		2	4	7	10
2			2	5	8
4				3	6
7					3
10					

Representation of Δ $\boldsymbol{X}$ = {2, 2, 3, 3, 4, 5, 6, 7, 8, 10} as a two dimensional table, with elements of

$$\boldsymbol{X} = \{0, 2, 4, 7, 10\}$$

along both the top and left side. The elements at (i, j) in the table is $x_j - x_i$ for $1 \leq i < j \leq n$.

Partial Digest Problem: Formulation

Goal*:* Given all pairwise distances between points on a line, reconstruct the positions of those points

- Input: The multiset of pairwise distances L, containing $n(n-1)/2$ integers
- Output: A set X, of n integers, such that $\Delta X = L$

Partial Digest: Multiple Solutions

- It is not always possible to uniquely reconstruct a set ***X*** based only on Δ ***X***.
- For example, the set

$$X = \{0, 2, 5\}$$

and

$$(X + 10) = \{10, 12, 15\}$$

both produce Δ ***X***$=\{2, 3, 5\}$ as their partial digest set.

- The sets {0,1,2,5,7,9,12} and {0,1,5,7,8,10,12} present a less trivial example of non-uniqueness. They both digest into:

{1, 1, 2, 2, 2, 3, 3, 4, 4, 5, 5, 5, 6, 7, 7, 7, 8, 9, 10, 11, 12}

Homometric Sets

	0	1	2	5	7	9	12
0		1	2	5	7	9	12
1			1	4	6	8	11
2				3	5	7	10
5					2	4	7
7						2	5
9							3
12							

	0	1	5	7	8	10	12
0		1	5	7	8	10	12
1			4	6	7	9	11
5				2	3	5	7
7					1	3	5
8						2	4
10							2
12							

Brute Force Algorithms

- Also known as exhaustive search algorithms; examine every possible variant to find a solution
- Efficient in rare cases; usually impractical

Partial Digest: Brute Force

1. Find the restriction fragment of maximum length ***M***. ***M*** is the length of the DNA sequence.
2. For every possible set

$$\boldsymbol{X}=\{0, x_2, \ldots, x_{n-1}, M\}$$

compute the corresponding $\Delta\, \boldsymbol{X}$

5. If $\Delta\, \boldsymbol{X}$ is equal to the experimental partial digest ***L***, then ***X*** is the correct restriction map

BruteForcePDP

1. BruteForcePDP(L, n):
2. M <- maximum element in L
3. for every set of $n-2$ integers $0 < x_2 < \ldots x_{n-1} < M$
4. X <- $\{0, x_2, \ldots, x_{n-1}, M\}$
5. Form ΔX from X
6. if $\Delta X = L$
7. return X
8. output “no solution”

Efficiency of BruteForcePDP

- BruteForcePDP takes $O(\boldsymbol{M}^{n-2})$ time since it must examine all possible sets of positions.

- One way to improve the algorithm is to limit the values of x_i to only those values which occur in $\boldsymbol{L}$.

AnotherBruteForcePDP

1. AnotherBruteForcePDP(L, n)
2. M <- maximum element in L
3. for every set of $n - 2$ integers $0 < x_2 < \ldots x_{n-1} < M$
4. X <- { $0, x_2, \ldots, x_{n-1}, M$ }
5. Form ΔX from X
6. if $\Delta X = L$
7. return X
8. output “no solution”

AnotherBruteForcePDP

1. AnotherBruteForcePDP(L, n)
2. M <- maximum element in L
3. for every set of $n - 2$ integers $0 < x_2 < \ldots x_{n-1} < M$ from L
4. X <- { $0, x_2, \ldots, x_{n-1}, M$ }
5. Form ΔX from X
6. if $\Delta X = L$
7. return X
8. output “no solution”

Efficiency of AnotherBruteForcePDP

- It's more efficient, but still slow
- If $\boldsymbol{L}$ = {2, 998, 1000} ($\boldsymbol{n}$ = 3, $\boldsymbol{M}$ = 1000), BruteForcePDP will be extremely slow, but AnotherBruteForcePDP will be quite fast
- Fewer sets are examined, but runtime is still exponential: $O(\boldsymbol{n}^{2n-4})$

Branch and bound algorithm for PDP

- By Steven Skiena (Stony Brook Univ.)
- We first define $\Delta(\boldsymbol{y}, \boldsymbol{X})$

  as the multiset of all distances between point $\boldsymbol{y}$ and all other points in the set $\boldsymbol{X}$

$$\Delta(y, X) = \{|y - x_1|, |y - x_2|, \ldots, |y - x_n|\}$$

for $X = \{x_1, x_2, \ldots, x_n\}$

PartialDigest Algorithm

PartialDigest(*L*):

width <- Maximum element in *L*
DELETE(*width*, *L*)
X <- {0, *width*}
PLACE(*L*, *X*)

PartialDigest Algorithm (cont'd)

1. PLACE(*L*, *X*)
2. if *L* is empty
3. output *X*
4. return
5. *y* <- maximum element in *L*
6. Delete*(y,L)*
7. if $\Delta(y, X) \in L$
8. Add *y* to *X* and remove lengths $\Delta(y, X)$ from *L*
9. PLACE(*L*,*X*)
10. Remove *y* from *X* and add lengths $\Delta(y, X)$ to *L*
11. if $\Delta(width\text{-}y, X) \in L$
12. Add *width*-*y* to *X* and remove lengths $\Delta(width\text{-}y, X)$ from *L*
13. PLACE(*L*,*X*)
14. Remove *width*-*y* from *X* and add lengths $\Delta(width\text{-}y, X)$ to *L*
15. return

An Example

L = { 2, 2, 3, 3, 4, 5, 6, 7, 8, 10 }
X = { 0 }

An Example

L = { 2, 2, 3, 3, 4, 5, 6, 7, 8, 10 }
X = { 0 }

Remove 10 from ***L*** and insert it into ***X***. We know this must be the length of the DNA sequence because it is the largest fragment.

An Example

L = { 2, 2, 3, 3, 4, 5, 6, 7, 8, 10 }
X = { 0, 10 }

An Example

L = { 2, 2, 3, 3, 4, 5, 6, 7, 8, 10 }
X = { 0, 10 }

Take 8 from ***L*** and make ***y*** = 2 or 8. But since the two cases are symmetric, we can assume ***y*** = 2.

An Example

L = { 2, 2, 3, 3, 4, 5, 6, 7, 8, 10 }
X = { 0, 10 }

We find that the distances from $y=2$ to other elements in X are $\Delta(y, \boldsymbol{X}) = \{8, 2\}$, so we remove {8, 2} from ***L*** and add 2 to ***X***.

An Example

L = { 2, 2, 3, 3, 4, 5, 6, 7, 8, 10 }
X = { 0, 2, 10 }

An Example

L = { 2, 2, 3, 3, 4, 5, 6, 7, 8, 10 }
X = { 0, 2, 10 }

Take 7 from ***L*** and make $y = 7$ or $y = 10 - 7 = 3$. We will explore ***y*** = 7 first, so $\Delta(y, X) = \{7, 5, 3\}$.

An Example

L = { 2, 2, 3, 3, 4, 5, 6, 7, 8, 10 }
X = { 0, 2, 10 }

For ***y*** = 7 first, $\Delta(y, X) = \{7, 5, 3\}$. Therefore we remove {7, 5 ,3} from ***L*** and add 7 to ***X***.

An Example

L = { 2, 2, 3, 3, 4, 5, 6, 7, 8, 10 }
X = { 0, 2, 7, 10 }

An Example

L = { 2, 2, 3, 3, 4, 5, 6, 7, 8, 10 }
X = { 0, 2, 7, 10 }

Next: take 6 from ***L*** and make $y = 6$ or $y = 10 - 6 = 4$.

$\Delta(y, X)$ = {6, 4, 1 ,4}, which is NOT a subset of ***L*** so we will NOT explore this branch

An Example

L = { 2, 2, 3, 3, 4, 5, 6, 7, 8, 10 }
X = { 0, 2, 7, 10 }

This time make *y* = 4. $\Delta(y, X)$ = {4, 2, 3 ,6}, which is a subset of ***L*** so we will explore this branch. We remove {4, 2, 3 ,6} from ***L*** and add 4 to ***X***.

An Example

L = { 2, 2, 3, 3, 4, 5, 6, 7, 8, 10 }
X = { 0, 2, 4, 7, 10 }

An Example

L = { 2, 2, 3, 3, 4, 5, 6, 7, 8, 10 }
X = { 0, 2, 4, 7, 10 }

L is now empty, so we have a solution, which is X.

An Example

L = { 2, 2, 3, 3, 4, 5, 6, 7, 8, 10 }
X = { 0, 2, 7, 10 }

To find other solutions, we backtrack.

An Example

L = { 2, 2, 3, 3, 4, 5, 6, 7, 8, 10 }
X = { 0, 2, 10 }

More backtrack.

An Example

L = { 2, 2, 3, 3, 4, 5, 6, 7, 8, 10 }
X = { 0, 2, 10 }

This time we will explore y = 3. $\Delta(y, X) = \{3, 1, 7\}$, which is not a subset of L, so we won't explore this branch.

An Example

L = { 2, 2, 3, 3, 4, 5, 6, 7, 8, 10 }
X = { 0, 10 }

We backtracked back to the root. Therefore we have found all the solutions.

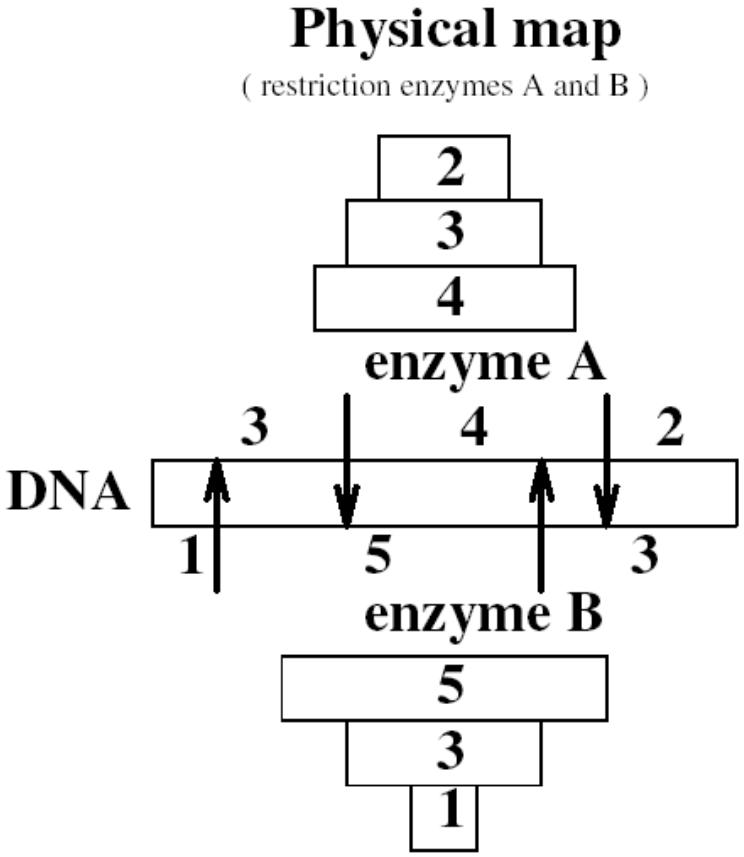
Analyzing PartialDigest Algorithm

- Still exponential in worst case, but is very fast on average
- Informally, let $T(\mathbf{n})$ be time PartialDigest takes to place n cuts
  - No branching case: $T(n) < T(n-1) + O(n)$
    - Quadratic
  - Branching case: $T(n) < 2T(n-1) + O(n)$
    - Exponential

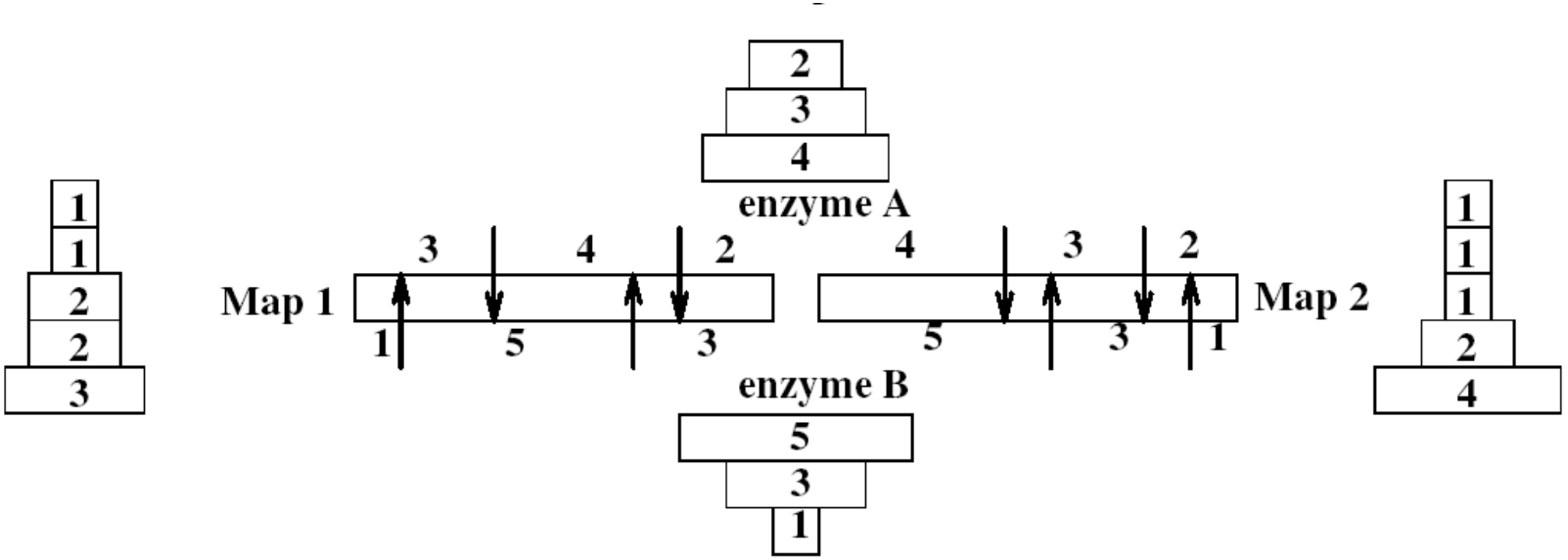
Double Digest Mapping

- Double Digest is yet another experimentally method to construct restriction maps
  - Use two restriction enzymes; three full digests:
    - One with only first enzyme
    - One with only second enzyme
    - One with both enzymes
- Computationally, Double Digest problem is more complex than Partial Digest problem

Double Digest: Example



Double Digest: Example



Without the information about ***X*** (i.e. ***A+B***), it is impossible to solve the double digest problem as this diagram illustrates

Double Digest Problem

Input: ***dA*** – fragment lengths from the digest with enzyme ***A***.

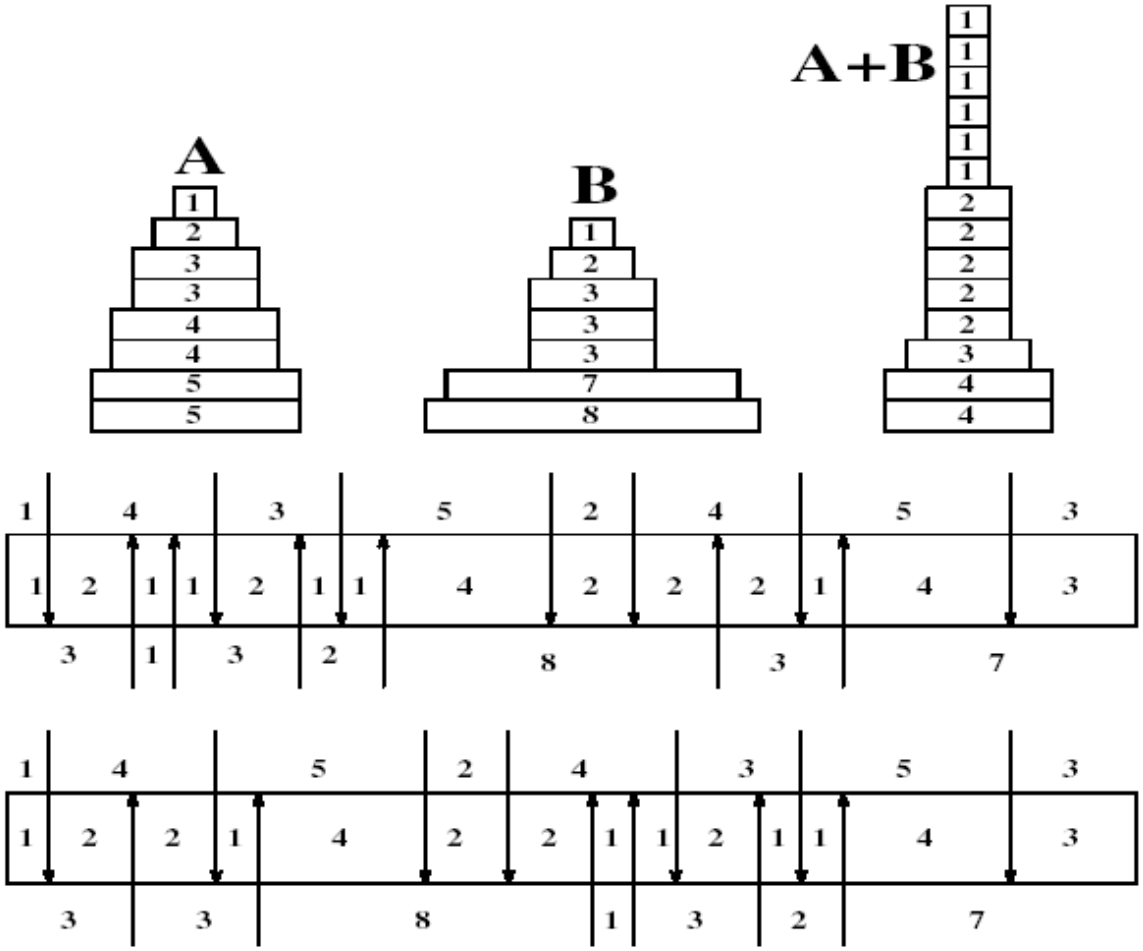
dB – fragment lengths from the digest with enzyme ***B***.

dX – fragment lengths from the digest with *both* ***A*** and ***B***.

Output: ***A*** – location of the cuts in the restriction map for the enzyme ***A***.

B – location of the cuts in the restriction map for the enzyme ***B***.

Double Digest: Multiple Solutions



MOTIFS

Random Sample

atgaccgggatactgataccgtatttggcctaggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg

acccctatttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaacttttccgaatactgggcataaggtaca

tgagtatccctgggatgacttttgggaacactatagtgctctcccgatttttgaatatgtaggatcattcgccagggtccga

gctgagaattggatgaccttgtaagtgttttccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga

tccctttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaatggcccacttagtccacttatag

gtcaatcatgttcttgtgaatggatttttaactgagggcatagaccgcttggcgcacccaaattcagtgtgggcgagcgcaa

cggttttggcccttgttagaggcccccgtactgatggaaactttcaattatgagagagctaatctatcgcgtgcgtgttcat

aacttgagttggtttcgaaaatgctctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta

ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcatttcaacgtatgccgaaccgaaagggaag

ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttctgggtactgatagca

Implanting Motif **AAAAAAAGGGGGGG**

atgaccgggatactgat**AAAAAAAGGGGGGG**ggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg

acccctatttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaacttttccgaata**AAAAAAAGGGGGGG**a

tgagtatccctgggatgactt**AAAAAAAGGGGGGG**tgctctcccgattttgaatatgtaggatcattcgccagggtccga

gctgagaattggatg**AAAAAAAGGGGGGG**tccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga

tccctttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaat**AAAAAAAGGGGGGG**cttatag

gtcaatcatgttcttgtgaatggattt**AAAAAAAGGGGGGG**gaccgcttggcgcacccaaattcagtgtgggcgagcgcaa

cggttttggcccttgttagaggcccccgt**AAAAAAAGGGGGGG**caattatgagagagctaatctatcgcgtgcgtgttcat

aacttgagtt**AAAAAAAGGGGGGG**ctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta

ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcat**AAAAAAAGGGGGGG**accgaaagggaag

ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagctt**AAAAAAAGGGGGGG**a

Where is the Implanted Motif?

atgaccgggatactgataaaaaaaaggggggggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg

acccctatttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaacttttccgaataaaaaaaaggggggga

tgagtatccctgggatgacttaaaaaaaaggggggtgctctcccgatttttgaatatgtaggatcattcgccagggtccga

gctgagaattggatgaaaaaaaaggggggtccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga

tccctttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaataaaaaaaaggggggcttatag

gtcaatcatgttcttgtgaatggatttaaaaaaaaggggggggaccgcttggcgcacccaaattcagtgtgggcgagcgcaa

cggttttggcccttgttagaggcccccgtaaaaaaaaggggggcaattatgagagagctaatctatcgcgtgcgtgttcat

aacttgagttaaaaaaaaggggggctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta

ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcataaaaaaaaggggggggaccgaaagggaag

ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttaaaaaaaaggggggga

Implanting Motif **AAAAAAGGGGGGG** with Four Mutations

atgaccgggatactgatAgAAgAAAGGttGGGggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg

acccctatttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaacttttccgaatacAAtAAAAcGGcGGGa

tgagtatccctgggatgacttAAAAtAAtGGaGtGGtgctctcccgattttgaatatgtaggatcattcgccagggtccga

gctgagaattggatgcAAAAAAAGGGattGtccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga

tccctttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaatAtAAtAAAGGaaGGGcttatag

gtcaatcatgttcttgtgaatggatttAAcAAtAAGGGctGGgaccgcttggcgcacccaaattcagtgtgggcgagcgcaa

cggttttggcccttgttagaggcccccgtAtAAAcAAGGaGGGccaattatgagagagctaatctatcgcgtgcgtgttcat

aacttgagttAAAAAAtAGGGaGccctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta

ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcatActAAAAAGGaGcGGaccgaaagggaag

ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttActAAAAAGGaGcGGa

Where is the Motif???

atgaccgggatactgatagaagaaaggttgggggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg

acccctatttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaacttttccgaatacaataaaacggcggga

tgagtatccctgggatgacttaaaataatggagtggtgctctcccgatttttgaatatgtaggatcattcgccagggtccga

gctgagaattggatgcaaaaaaagggattgtccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga

tccctttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaatataataaaggaagggcttatag

gtcaatcatgttcttgtgaatggatttaacaataagggctgggaccgcttggcgcacccaaattcagtgtgggcgagcgcaa

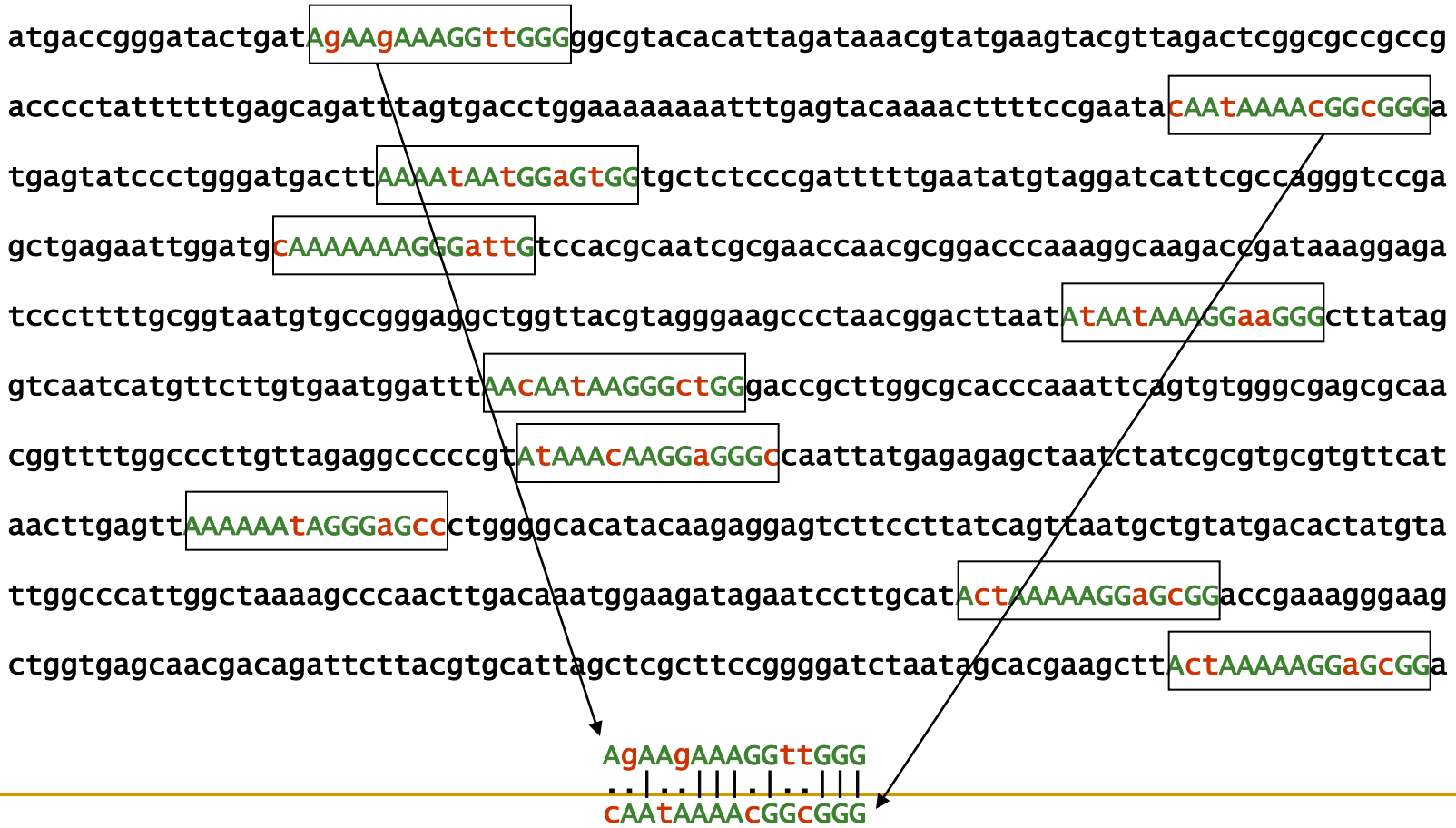
cggttttggcccttgttagaggcccccgtataaacaaggagggccaattatgagagagctaatctatcgcgtgcgtgttcat

aacttgagttaaaaaatagggagccctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta

ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcatactaaaaaggagcggaccgaaagggaag

ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttactaaaaaggagcgga

Finding (15,4) Motif



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

- An 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.

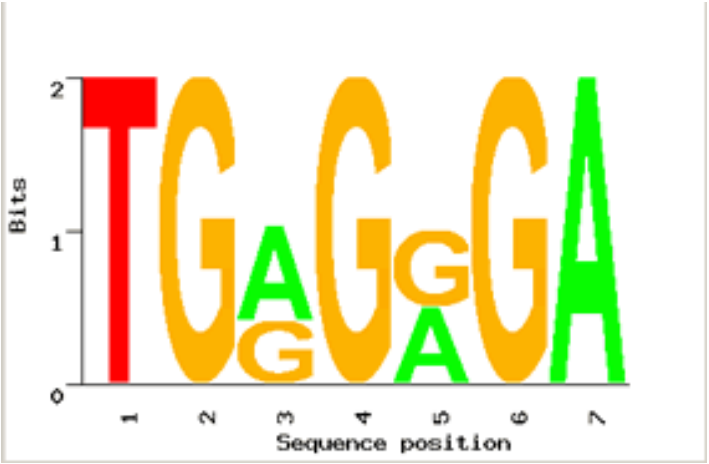
- TFBS may vary slightly across different regulatory regions since non-essential bases could mutate

Motifs and Transcriptional Start Sites

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



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?

The Motif Finding Problem

- Given a random sample of DNA sequences:

```
cctgatagacgctatctggctatccacgtacgtaggtcctctgtgcgaatctatgcgtttccaaccat
agtactggtgtacatttgatacgtacgtacaccggcaacctgaaacaaacgctcagaaccagaagtgc
aaacgtacgtgcaccctctttcttcgtggctctggccaacgagggctgatgtataagacgaaaatttt
agcctccgatgtaagtcatagctgtaactattacctgccacccctattacatcttacgtacgtataca
ctgttatacaacgcgtcatggcggggtatgcgttttggtcgtcgtacgctcgatcgttaacgtacgtc
```

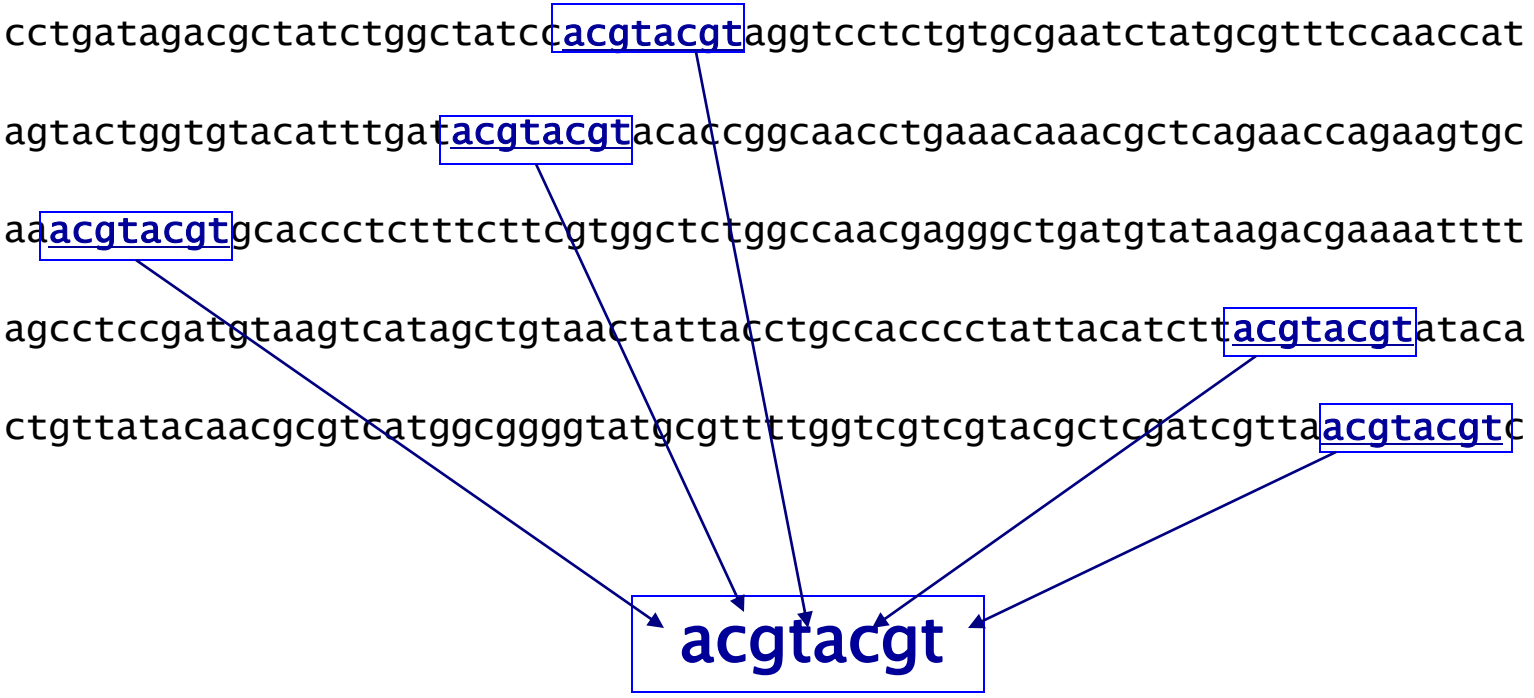
- Find the pattern that is implanted in each of the individual sequences, namely, the motif

The Motif Finding Problem (cont'd)

- Additional information:
  - The hidden sequence is 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:



The Motif Finding Problem (cont'd)

- The patterns with 2 point mutations:

cctgatagacgctatctggctatcc**aGgtacTt**aggtcctctgtgcgaatctatgcgtttccaaccat

agtactggtgtacatttgat**CcAtacgt**acaccggcaacctgaaacaaacgctcagaaccagaagtgc

aa**acgtTAgt**gcaccctctttcttcgtggctctggccaacgagggctgatgtataagacgaaaatttt

agcctccgatgtaagtcatagctgtaactattacctgccacccctattacatctt**acgtCcAt**ataca

ctgttatacaacgcgtcatggcggggtatgcgttttggtcgtcgtacgctcgatcgtta**CcgtacgG**c

The Motif Finding Problem (cont'd)

- The patterns with 2 point mutations:

cctgatagacgctatctggctatcc**aGgtacTt**aggtcctctgtgcgaatctatgcgtttccaaccat

agtactggtgtacatttgat**CcAtacgt**acaccggcaacctgaaacaaacgctcagaaccagaagtgc

aa**acgtTAgt**gcaccctctttcttcgtggctctggccaacgagggctgatgtataagacgaaaatttt

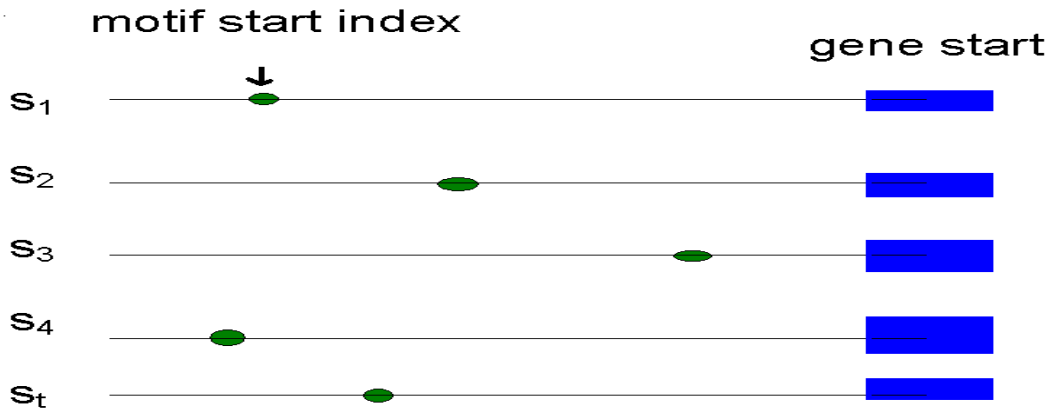
agcctccgatgtaagtcatagctgtaactattacctgccacccctattacatctt**acgtCcAt**ataca

ctgttatacaacgcgtcatggcggggtatgcgttttggtcgtcgtacgctcgatcgtta**CcgtacgG**c

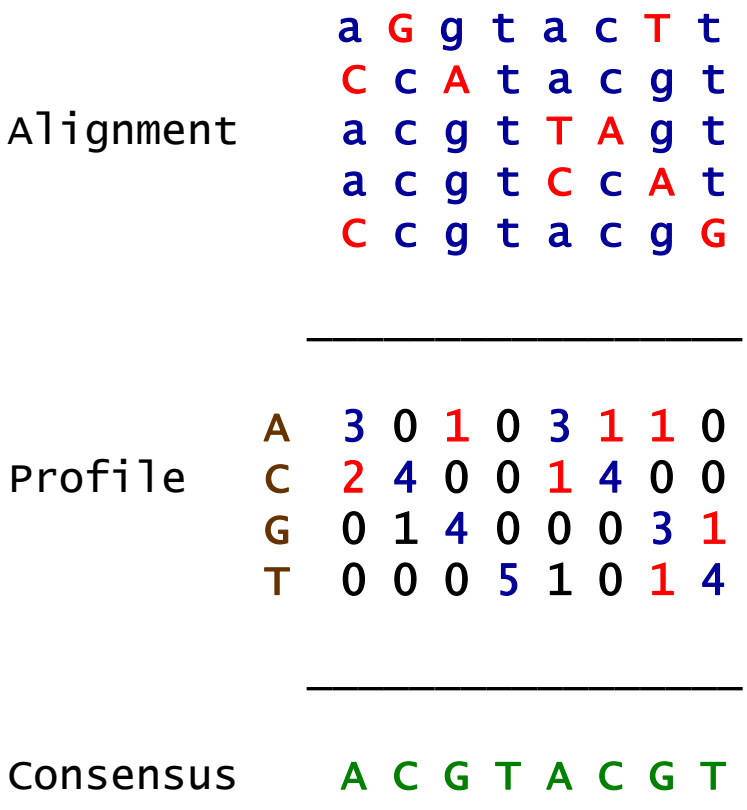
Can we still find the motif, now that we have 2 mutations?

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



- Line up the patterns by their start indexes

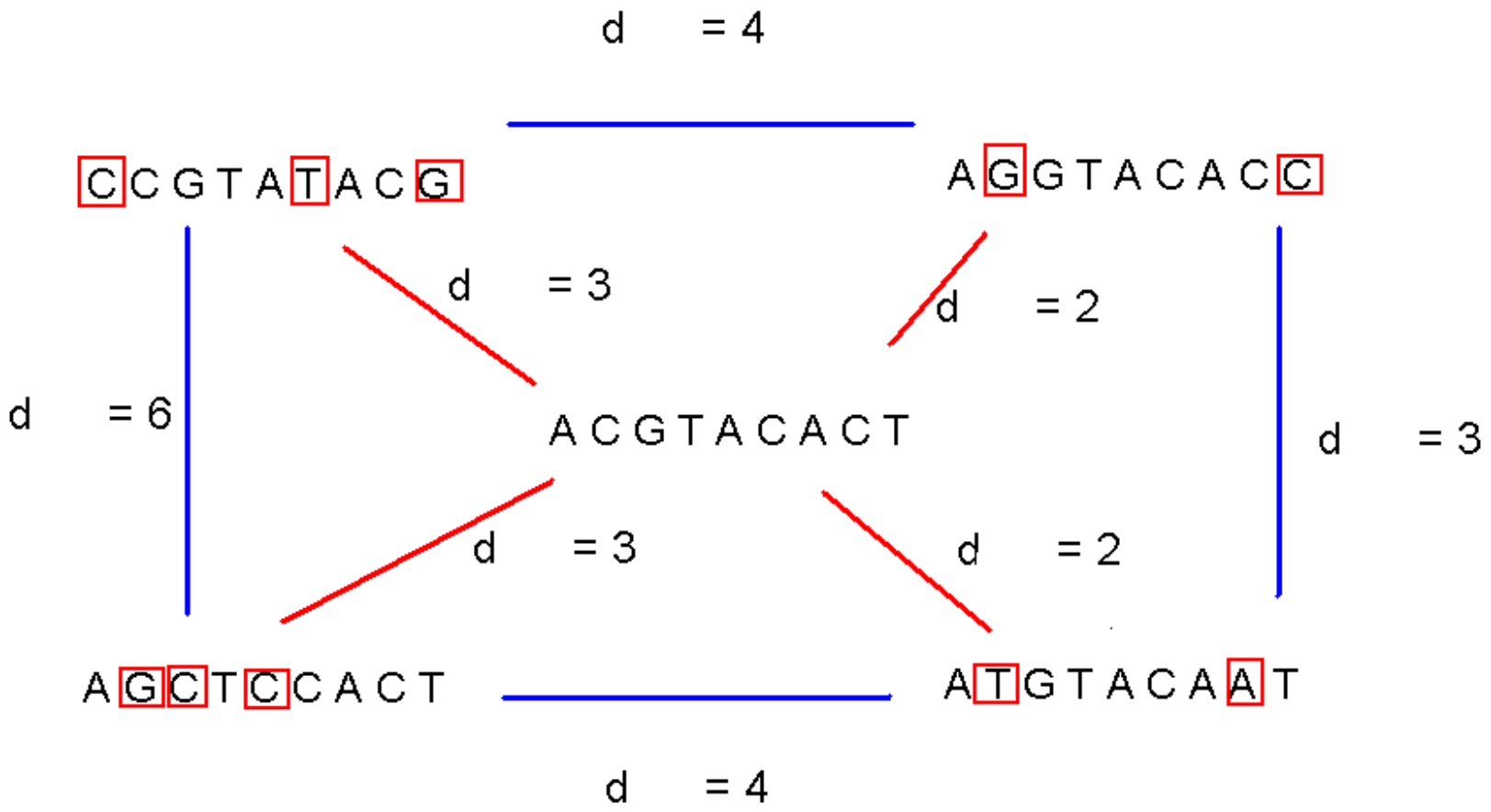
$$\mathbf{s} = (s_1, s_2, \ldots, s_t)$$

- 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

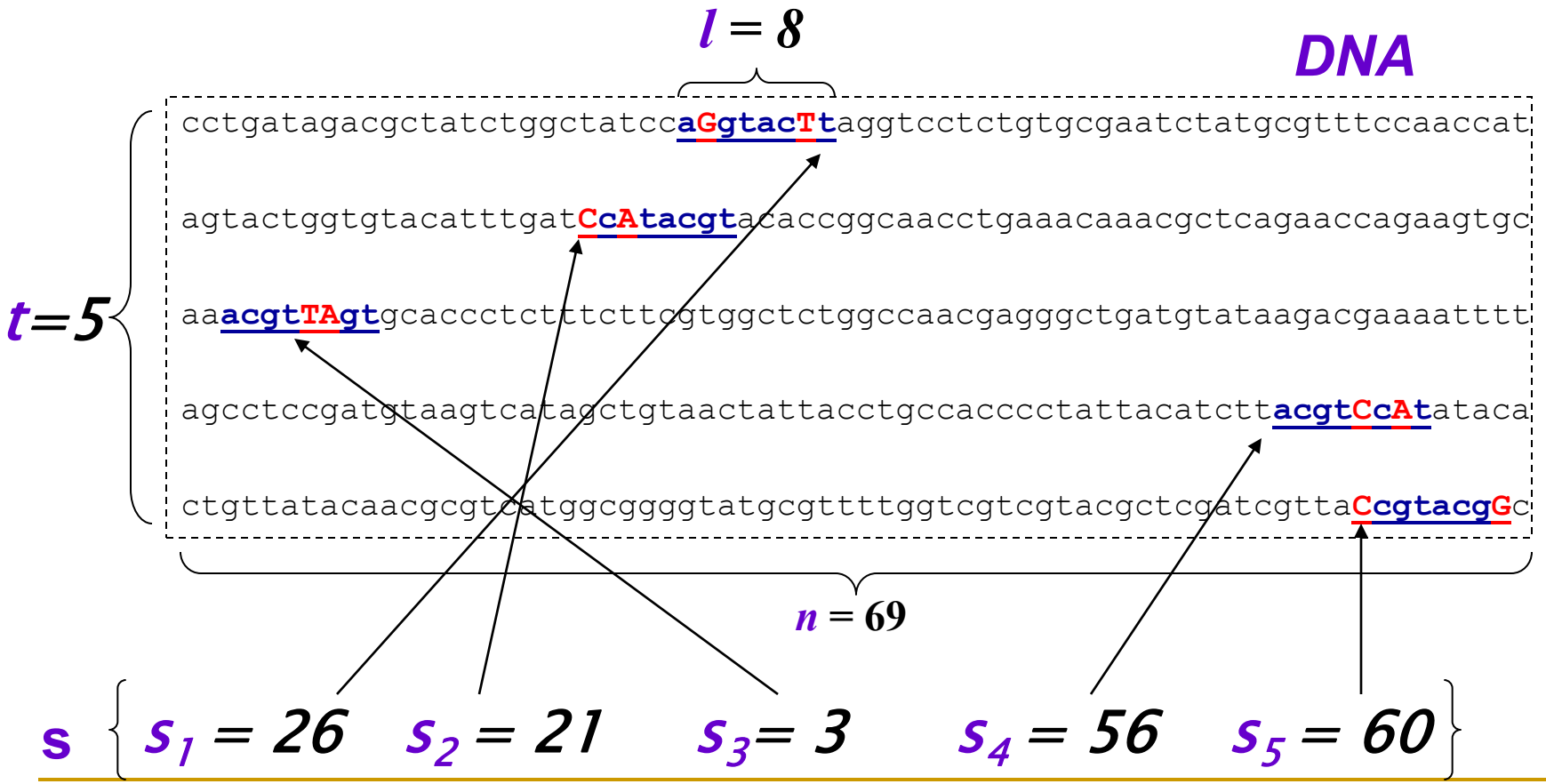
- We have a guess about the consensus sequence, but how “good” is this consensus?
- Need to introduce a scoring function to compare different guesses and choose the “best” one.

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*
- **s**=(s_1, s_2, … s_t) - array of motif's starting positions

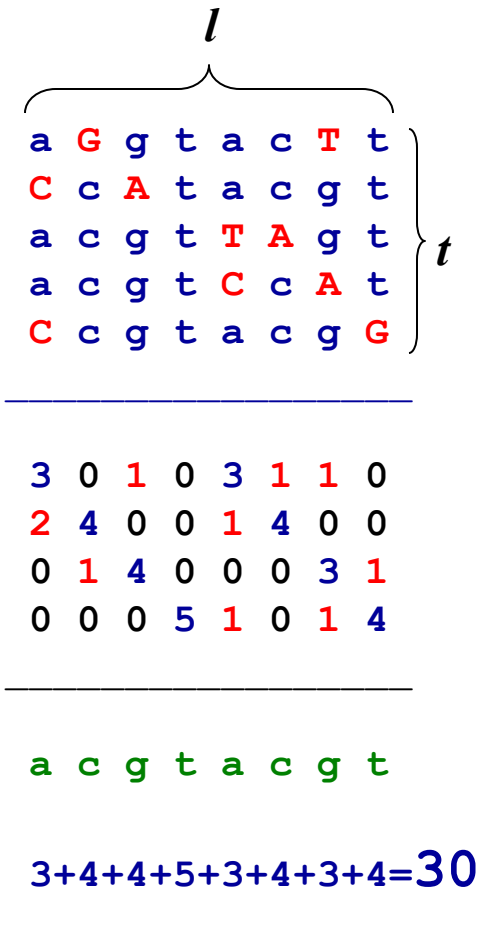
Parameters


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

Scoring Motifs

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

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



The Motif Finding Problem

- If starting positions **s**=(s_1, s_2, … 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 $\boldsymbol{t}$ x $\boldsymbol{n}$ 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***)

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_i, where:

$$s_i = [1, \ldots, n-l+1]$$
$$i = [1, \ldots, t]$$

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 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

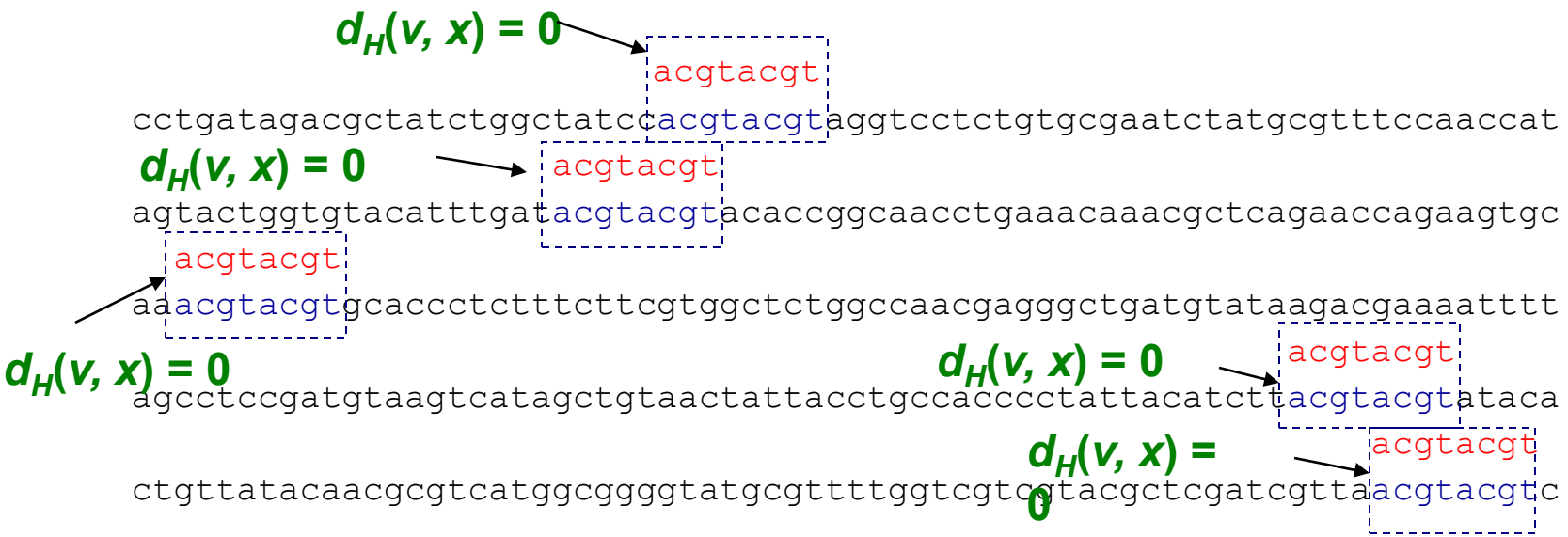
Hamming Distance

- 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: An Example

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

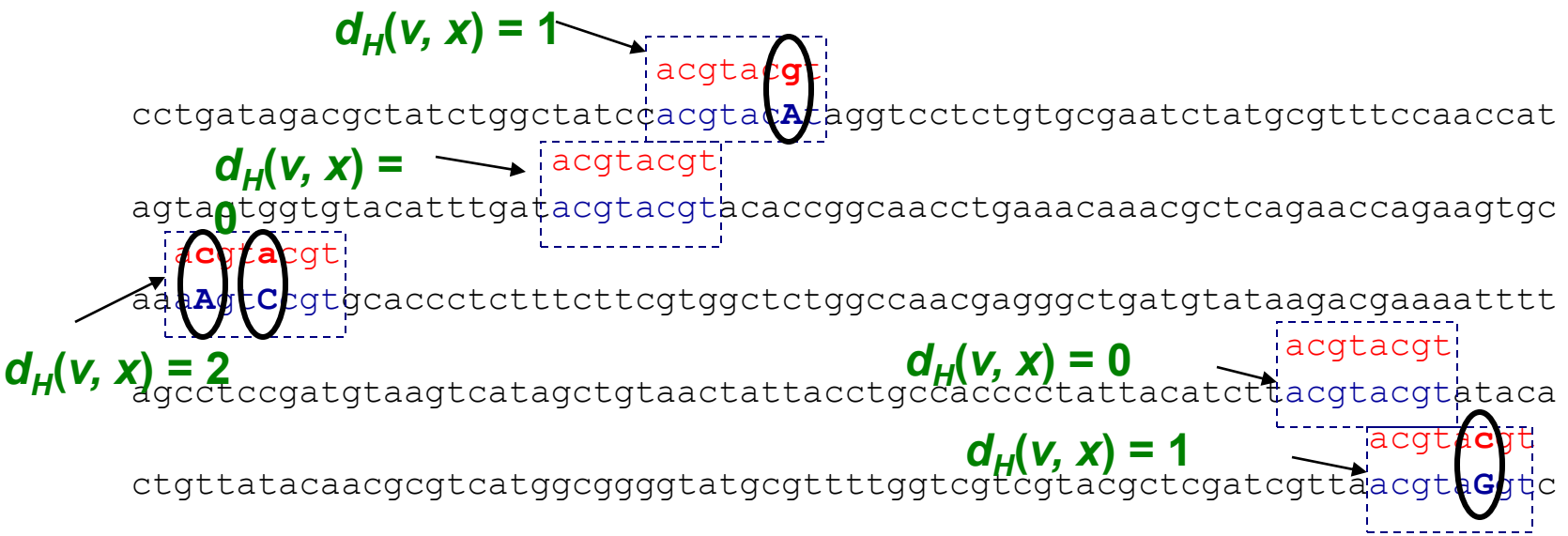


v is the sequence in red, x is the sequence in blue

- $TotalDistance(\mathbf{v}, \mathbf{DNA}) = 0$

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**)* = $min_{\mathbf{s}}\, d_H(\boldsymbol{v}, \boldsymbol{s})$, 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

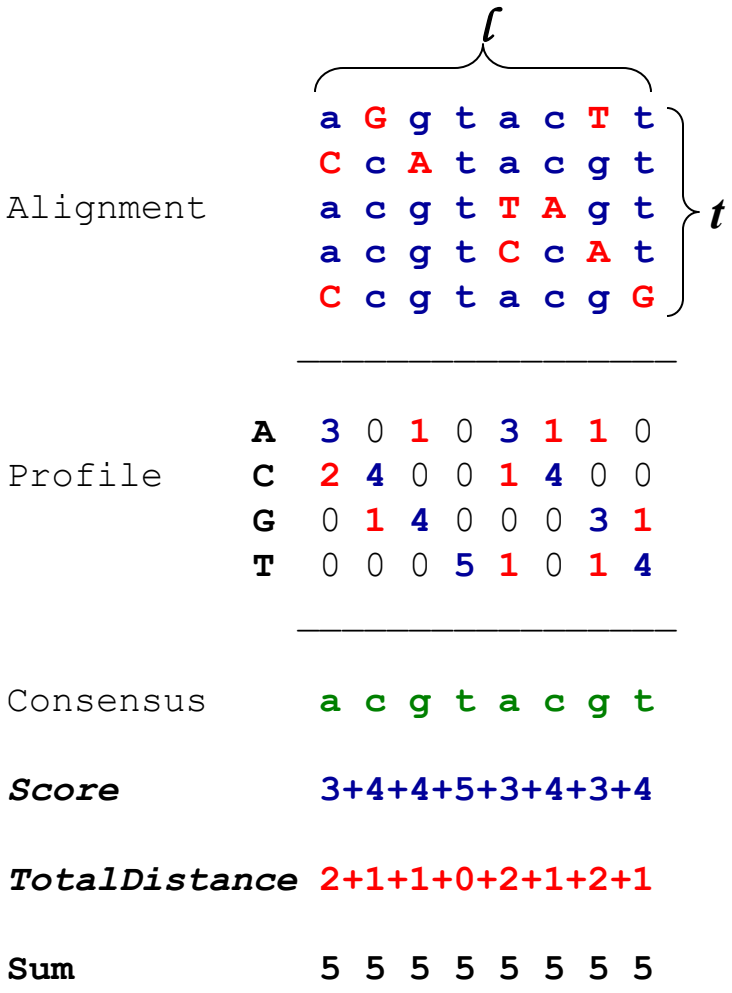
Median String Search Algorithm

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***
  $Score_i + TotalDistance_i = \boldsymbol{t}$
- 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

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 $(\boldsymbol{n} - \ell + 1)^t$ combinations!!!
  - The Median String Problem needs to examine all 4^ℓ combinations for $\boldsymbol{v}$. This number is relatively smaller

Motif Finding: Improving the Running Time

Recall the 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

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 ← 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
```

Structuring the Search

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

$\overbrace{\text{aa}\ldots\text{aa}}^{\ell}$

aa… ac

aa… ag

aa… at

.

.

tt… tt

How to organize this search?

Alternative Representation of the Search Space

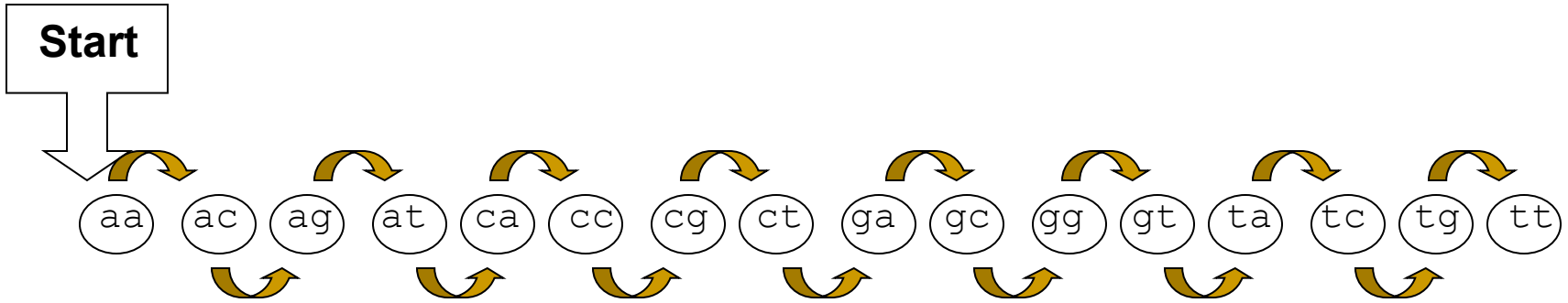
- Let **A** = 1, **C** = 2, **G** = 3, **T** = 4
- Then the sequences from AA…A to TT…T become:

$$\overbrace{11\ldots11}^{l}$$
$$11\ldots12$$
$$11\ldots13$$
$$11\ldots14$$
$$\vdots$$
$$44\ldots44$$

- 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

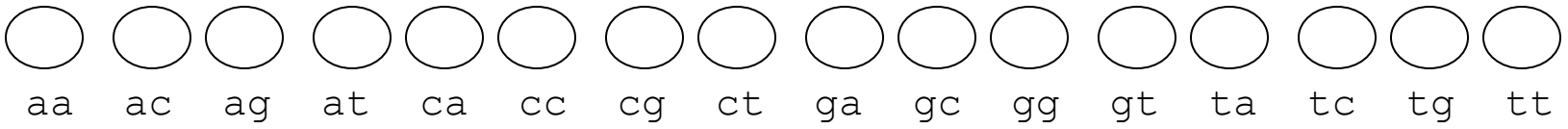
- Suppose $\ell = 2$



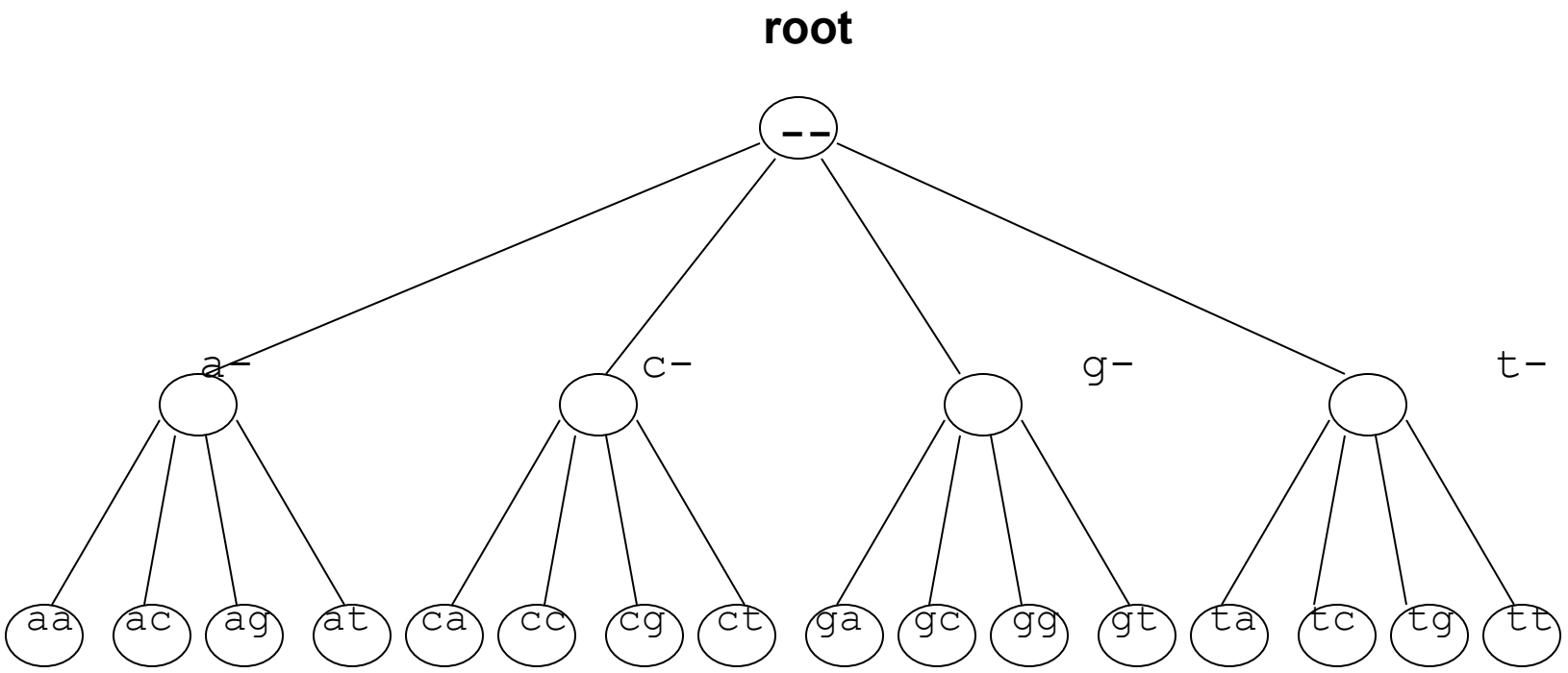
- Need to visit all the predecessors of a sequence before visiting the sequence itself

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:

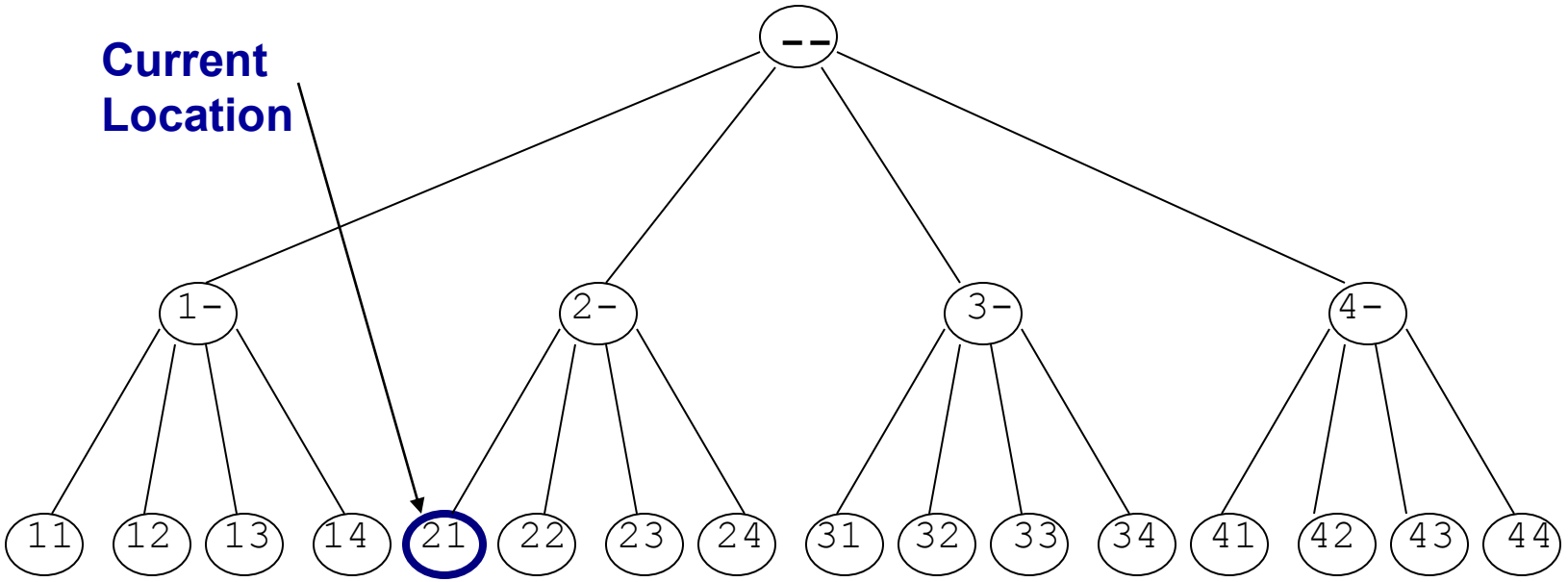
```
1. NextLeaf( a,L, k )        // a : the array of digits
2. for i ← L to 1            // L: length of the array
3.     if a_i < k            // k : max digit value
4.         a_i ← a_i + 1
5.             return a
6.         a_i ← 1
7. return a
```

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

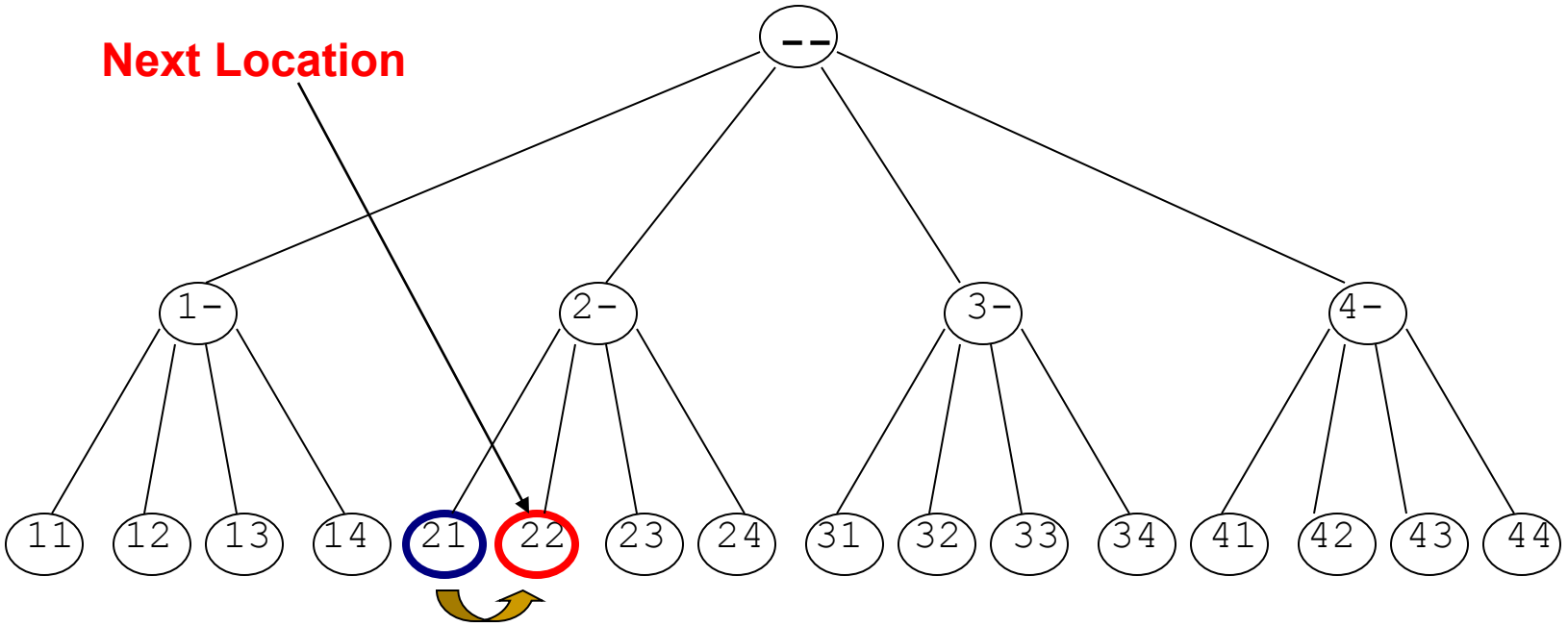
NextLeaf: Example

- Moving to the next leaf:



NextLeaf: Example (cont'd)

- Moving to the next leaf:



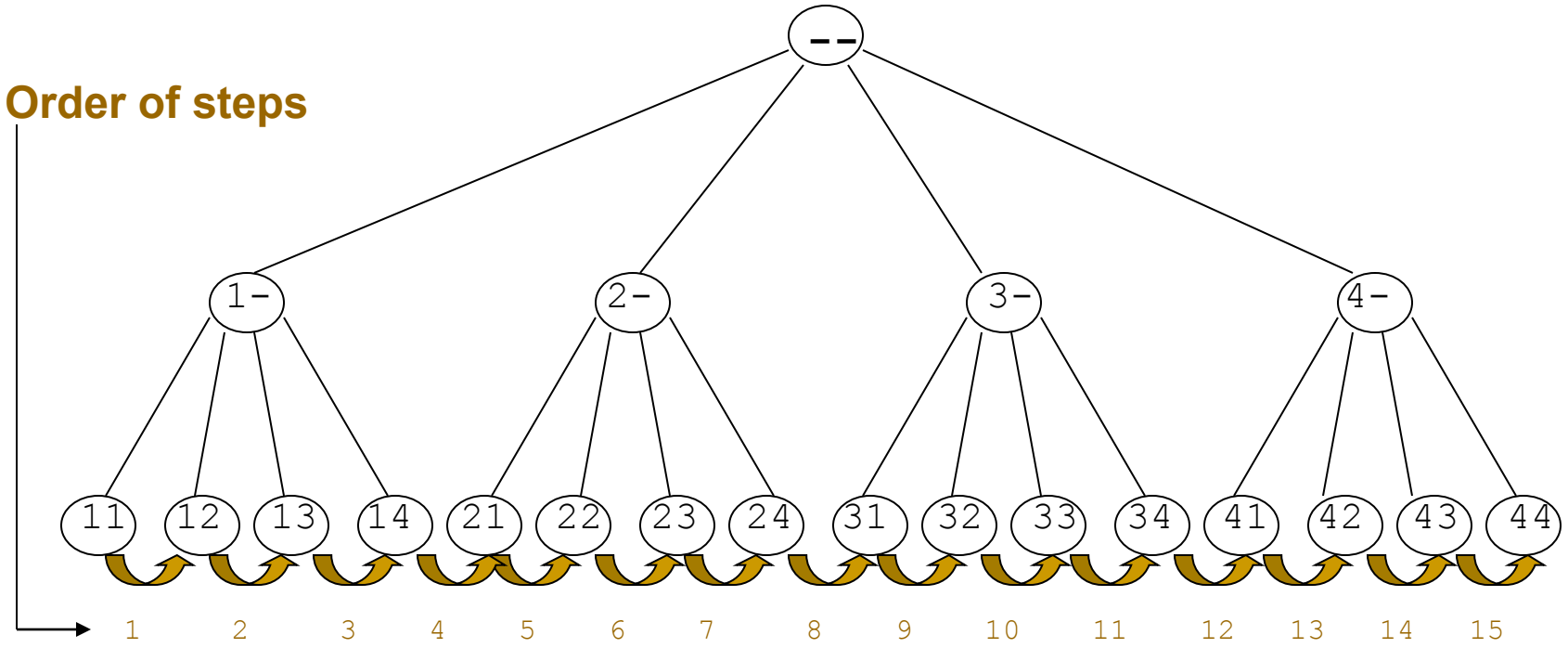
Visit All Leaves

- Printing all permutations in ascending order:

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

Visit All Leaves: Example

- Moving through all the leaves in order:



Depth First Search

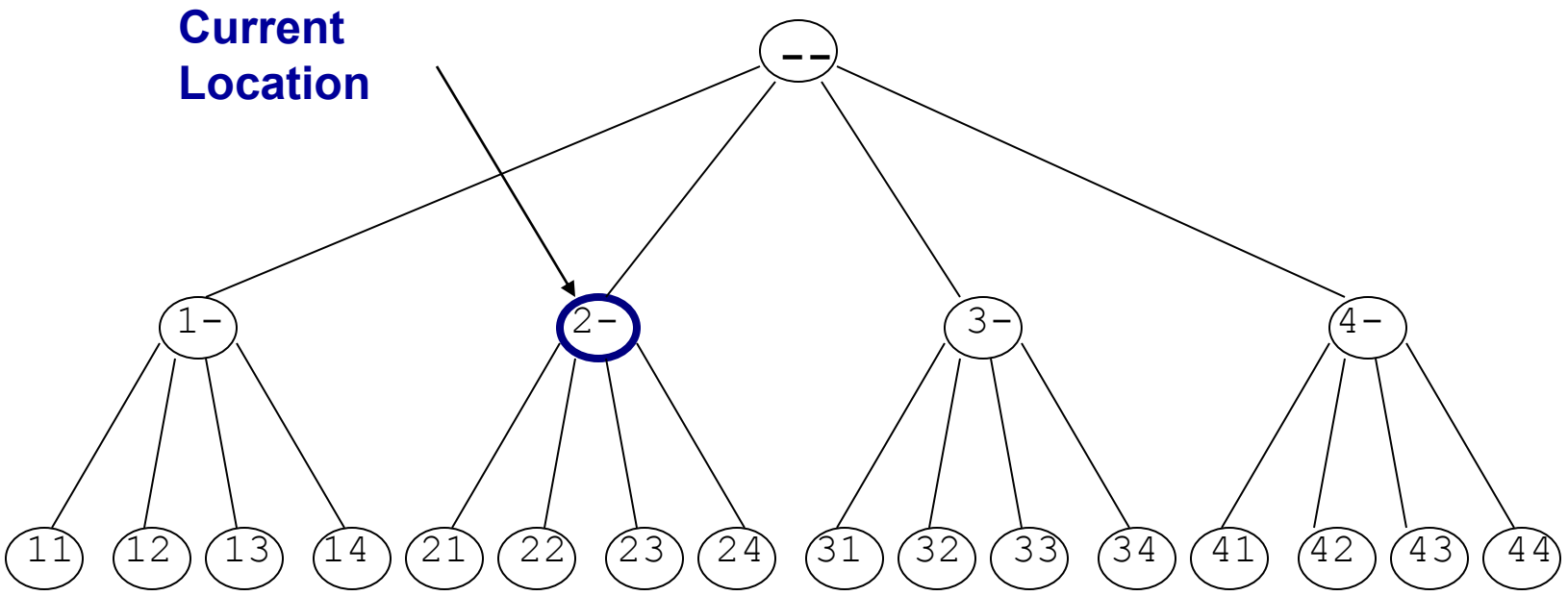
- So we can search leaves
- How about searching all vertices of the tree?
- We can do this with a *depth first* search

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} ← 1          // L: max length
4.      return ( a,i+1)      // k : max digit value
5.    else
6.      for j ← L to 1
7.        if a_j < k
8.          a_j ← a_j + 1
9.          return( a,j )
10.   return(a,0)
```

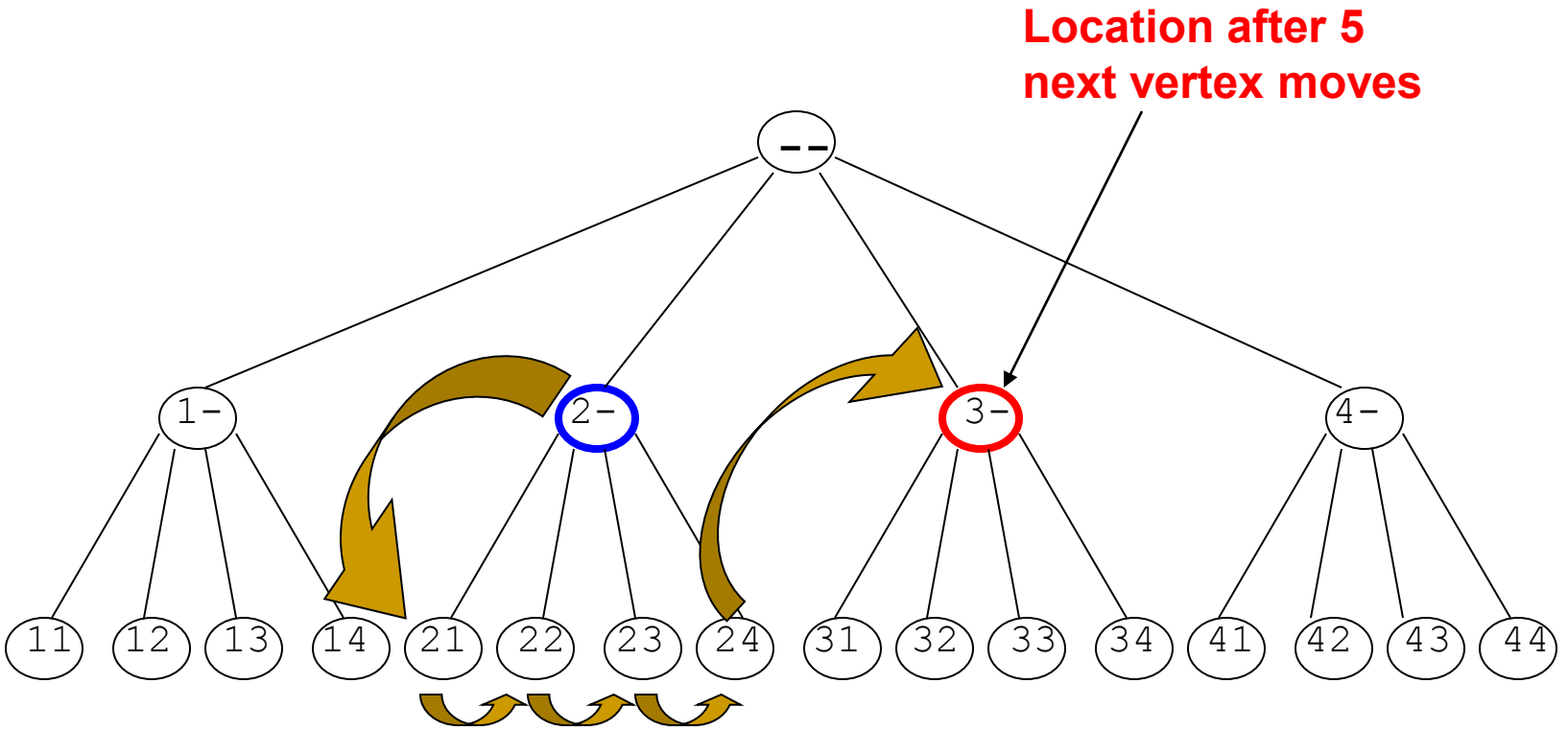
Example

- Moving to the next vertex:



Example

- Moving to the next vertices:



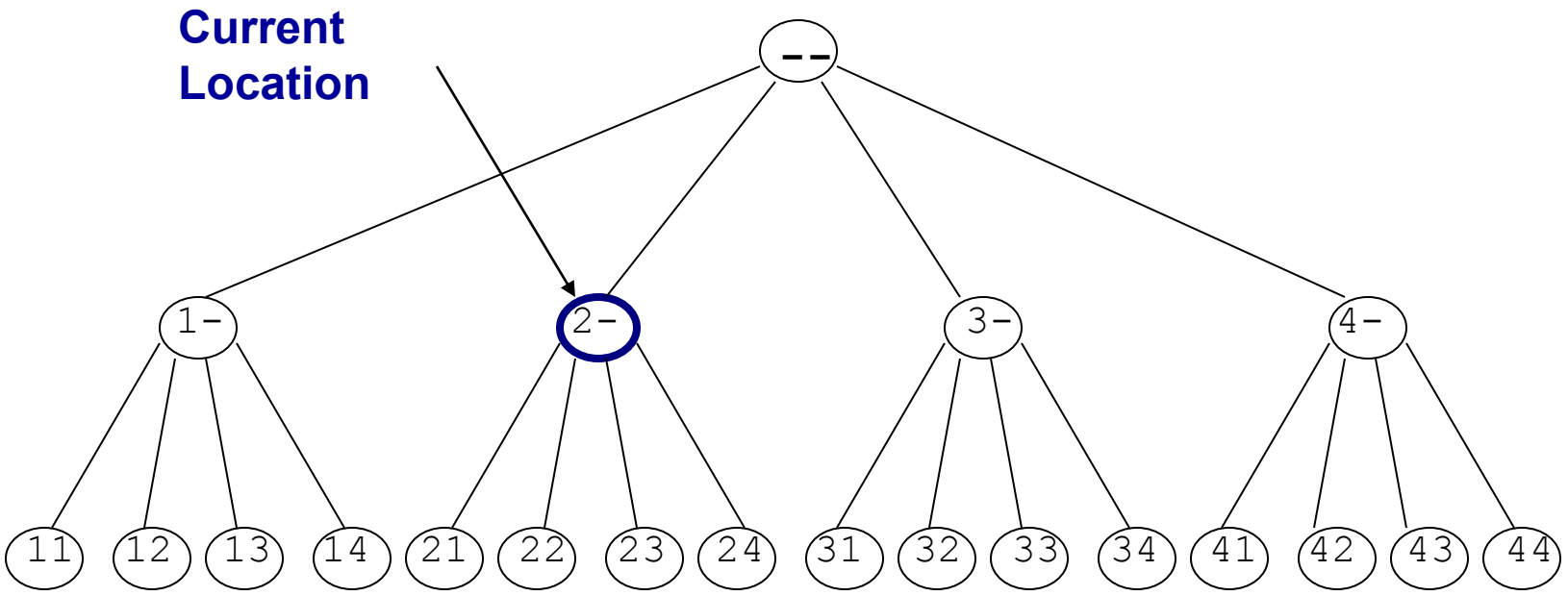
Bypass Move

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

1. Bypass(**a**,*i*,*L*,*k*) // **a**: array of digits
2. **for** *j* ← *i* to *1* // *i* : prefix length
3. **if** $a_j < k$ // *L*: maximum length
4. $a_j \leftarrow a_j + 1$ // *k* : max digit value
5. **return**(**a**,*j*)
6. **return**(**a**,0)

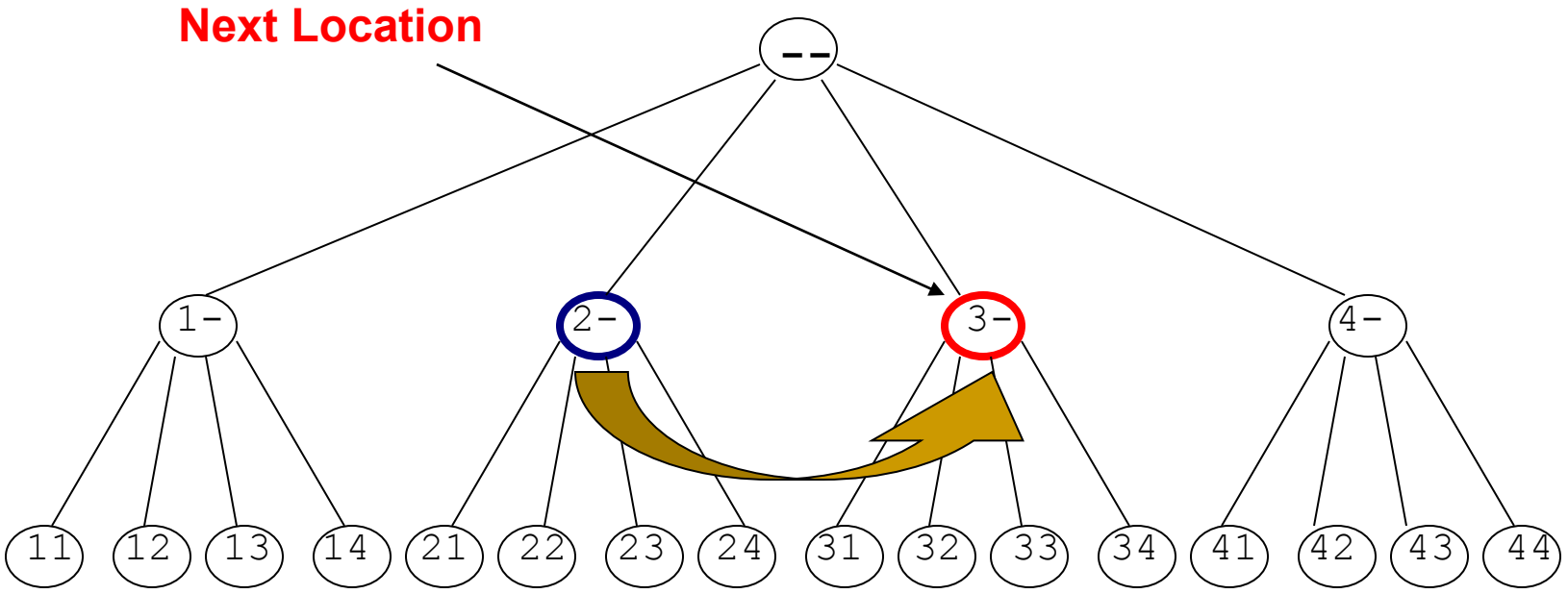
Bypass Move: Example

- Bypassing the descendants of “2-”:



Example

- Bypassing the descendants of “2-”:



Brute Force Search Again

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

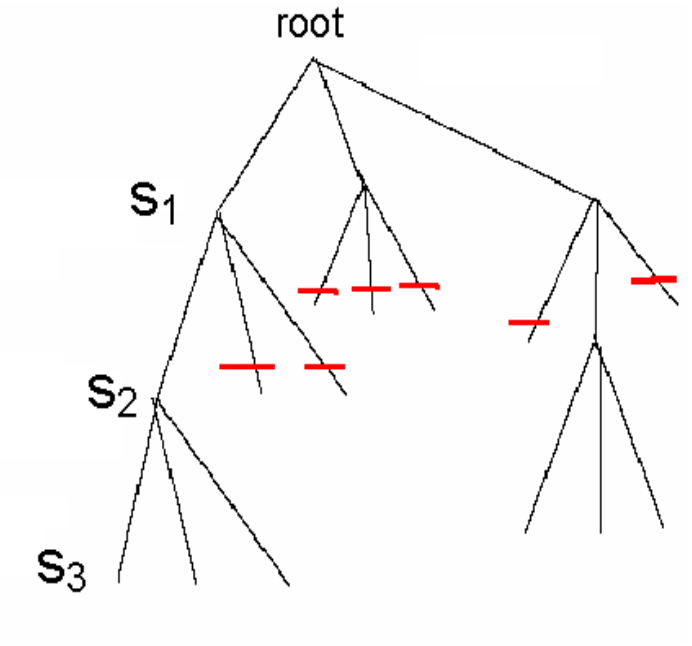
$$(t - i) * \ell \text{ to Score}(\mathbf{s}, i, DNA)$$

- If Score(**s**,*i*,***DNA***) + $(t - i) * \ell$ < **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 ← (1,…,1)
3.  bestScore ← 0
4.  i ← 1
5.  while i > 0
6.      if i < t
7.          optimisticScore ← Score(s, i, DNA) +(t – i ) * ℓ
8.          if optimisticScore < bestScore
9.              (s, i) ← Bypass(s,i, n-ℓ+1)
10.         else
11.             (s, i) ← NextVertex(s, i, n-ℓ+1)
12.     else
13.         if Score(s,DNA) > bestScore
14.             bestScore ← Score(s)
15.             bestMotif ← (s_1, s_2, s_3, …, s_t)
16.         (s,i) ← 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

```
1.  BranchAndBoundMedianStringSearch(DNA,t,n,ℓ)
2.  s ← (1,…,1)
3.  bestDistance ← ∞
4.   i ← 1
5.  while i > 0
6.    if i < ℓ
7.       prefix ←  string corresponding to the first i nucleotides of s
8.       optimisticDistance ← TotalDistance(prefix,DNA)
9.       if optimisticDistance > bestDistance
10.                  (s, i ) ← Bypass(s,i, ℓ, 4)
11.      else
12.                  (s, i ) ← NextVertex(s, i, ℓ, 4)
13.   else
14.      word ← nucleotide string corresponding to s
15.      if  TotalDistance(s,DNA) < bestDistance
16.                  bestDistance ← TotalDistance(word, DNA)
17.                  bestWord ← word
18.      (s,i ) ← NextVertex(s,i,ℓ, 4)
19. return bestWord
```

Improving the Bounds

- Given an ℓ-mer $\boldsymbol{w}$, divided into two parts at point $\boldsymbol{i}$
  - $\boldsymbol{u}$: prefix $w_1, \ldots, w_i$,
  - $\boldsymbol{v}$: suffix $w_{i+1}, \ldots, w_\ell$
- 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 $\boldsymbol{u}$

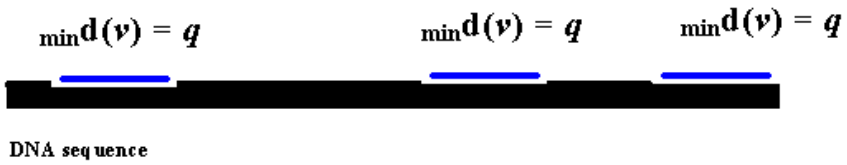
Improving the Bounds (cont'd)

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

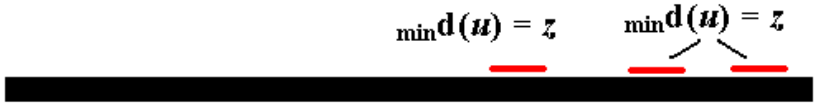
- Since ***u*** and ***v*** are two substrings of ***w***, and included in motif ***w***, we can assume that the minimum distance of ***u*** plus minimum distance of ***v*** can only be less than the minimum distance for ***w***

Better Bounds

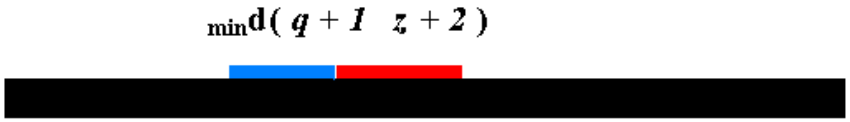
Searching for prefix *V*
We may find many instances of prefix *V* with a minimum distance *q*



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
```