DNA MAPPING
Molecular Scissors

EcoRI

Cleavage

Sticky ends

Molecular Cell Biology, 4th edition
## Recognition Sites of Restriction Enzymes

<table>
<thead>
<tr>
<th>Enzyme</th>
<th>Source Microorganism</th>
<th>Recognition Site</th>
<th>Ends Produced</th>
</tr>
</thead>
<tbody>
<tr>
<td>BamHI</td>
<td><em>Bacillus arylophilus</em></td>
<td>-G-G-A-T-C-C-</td>
<td>Sticky</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-C-C-T-A-G-G-</td>
<td></td>
</tr>
<tr>
<td>EcoRI</td>
<td><em>Escherichia coli</em></td>
<td>G A A T T C-</td>
<td>Sticky</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C T T A A G</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>-T-T-C-G-A-A-</td>
<td></td>
</tr>
<tr>
<td>KpnI</td>
<td><em>Klebsiella pneumonia</em></td>
<td>-G-G-T-A-C-C-</td>
<td>Sticky</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-C-C-A-T-G-G-</td>
<td></td>
</tr>
</tbody>
</table>
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:

  ![Diagram of restriction fragments](image)
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
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

Direction of DNA movement

Smaller fragments travel farther
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:

```
Restriction Sites

[<5>]
[<14>]
[<19>]
[<22>]
[<9>]
[<14>]
[<17>]
[<5>]
[<8>]
[<3>]
```
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

\[ \Delta X: \] the multiset of integers representing lengths of each of the \( C(n, 2) \) fragments produced from a partial digest
One More Partial Digest Example

<table>
<thead>
<tr>
<th>X</th>
<th>0</th>
<th>2</th>
<th>4</th>
<th>7</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>4</td>
<td>7</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>5</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>3</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Representation of \( \Delta \ X = \{2, 2, 3, 3, 4, 5, 6, 7, 8, 10\} \) as a two dimensional table, with elements of

\[
X = \{0, 2, 4, 7, 10\}
\]

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 \( \Delta X \).

- For example, the set

\[
X = \{0, 2, 5\}
\]

and

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

both produce \( \Delta X = \{2, 3, 5\} \) 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

![Homometric Sets Diagram]
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

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

compute the corresponding $\Delta X$

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

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

- BruteForcePDP takes $O(M^{n-2})$ 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 $L$. 
AnotherBruteForcePDP

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

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

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

- By Steven Skiena (Stony Brook Univ.)
- We first define $\Delta(y, X)$ as the multiset of all distances between point $y$ and all other points in the set $X$

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

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 \( \Delta(y, X) \) \( \not\in \) 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 \( \Delta(width-y, X) \) \( \not\in \) L
12. Add \( width-y \) to X and remove lengths \( \Delta(width-y, X) \) from L
13. PLACE(L,X )
14. Remove \( width-y \) from X and add lengths \( \Delta(width-y, X) \) 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
$\Delta(y, X) = \{8, 2\}$, 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 $X$. 

![Number line diagram]
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 $T(n)$ 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

Physical map

(restriction enzymes A and B)

DNA

enzyme A

enzyme B

1 5 3

2 3 4

5 3 1
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

atgaccgggatactgataccgtatattttgcctagggtacacaccacataagataatacgttagactcggcgccg
accctatattttgagcagatatttagtgcctgaaatggagatcataaaacccccgatatctgcctggcataaggtaca
tgagtatttggagatgcactttttggaacactatagtgctctccccgatatttttgaatatgtaggtaggtcatttcggcgccg
acccctattttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaacttttccgataatcgcctgacgggtccga
gctgagaattggatgacctttgtaagtttttccacgcaatccgcgaacccacaacacgccgccaaaaagcaacgcagaccgataaaggaga
tcccttttgcggtaatgtgcctgggagggctgtttacgtaggaaagcccttaacggaacattgaatctgtgccacttttagtcctcttagag
gtcaattcatgttttctttgtaatggatttttaactgagggcatagaccgctttggccgcacccaaattcaagttgtggttgccgacgcgaacggtttggcccttgttagagecaccctgactgatggaaactttcaattatgagagagcctaatctatcgcgtgcgtttcat
aacttgagttggtttcggaaatgtgccttcgggacacatacacaagggagtctttctctatcagtttatgctgtatgacactatgta
ttgccctattggctaaagcccaacttgacaaatggaagatagaatccttgcatttcaacgtatgccgaaccgaaagggaagtgggtgagcaacgacagattcttacggtgcattagctcctccgsgggtactaatagcagaaagcttccttttgggtactgtagca
Implanting Motif AAAAAAAAGGGGGGGG

atgaccgggatactgataAAAAAAAGGGGGGGggcgtacacattagataaaacgtatgaagtactggactgcgccgcccg
acccctatatatgagcagatattaagtgcacctggaaaaaaatggagtaacaacttttccgaataAAAAAAAGGGGGGGA
tgagtataccctgggatgacctAAAAAAAGGGGGGgtgctctccccgattttgatatatgtagagctattcgccagggtccga
gctgagaattggatgAAAAAAAGGGGGGGtccacgcatacgcaacgcaaacgcggacccaaaggcaagaccgataaaggaga
tccctttttcggtaatgtgcagggaggtcgtttacgtagggagccctaacgacttaataAAAAAAAGGGGGGGGcttatag
gtcaatcatgttttctgtgaatggatggAAAAAAAGGGGGGGgaccgctttggccacccaaactcatgtgtggtggcgagcgcaacggtttttgccctttgtaggctagccgaggtcctcttacctgtagggagccccaatcttcagttggtggcagcgcaactcgtacgtgcattagctcgcttccgggatatcaataagcgaagctt
ctggtgagcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagattcatcttttcgctgacaagcgcaacgacagat...
Where is the Implanted Motif?

atgaccgggatactgataaaaaaaaaaggggggggggtacacattagataaaacgtatgaagttacgtttagactcggcgccgcg
cacctatatatatgtgacctggaaaaaaaaatattgtaccaaaaacttttccgaataaaaaaaaaaggggggga
tgatatatcttgggatgacttaaaaaaaagggggggtgcctctccgatattttgaatatgttaggtcatattgcaccaggttccga
gctgagaattgtgatggaaaggggggtcaccgaatccgaacccagcggagccaaaggcaagaccgataaaggaga
tcccccttgcgtaatgtgcggggaggctgtttacgtaggaaagcccttaacggacttaaataaaaaaaaagggggggccattatag
gtcaatcatgtttcttgtgaatggatttaaaaaaaaggggggggaccgcttggcgcacccaaattcagtgggcgagcgcaa
cggtttttgcccccttttagaggccccccgtaaaaaaaagggggggcaattatgagagagcataatctctatcgcggtgctttcat
aacttgagtttaaaaaaagggggggctggggcacaatacaagaggagttttccttatcagttatgtgtatgacactatgta
tggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgacataaaaaaaagggggggccggagagggaga
ctggtgagcaacgacagatttcttactggcatatgctgccttcggggcatctaaatagcagcagaagctttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttt
Implanting Motif AAAAAAGGGGGGGG with Four Mutations

atgaccgggatctgat\textcolor{red}{A}g\textcolor{blue}{A}g\textcolor{red}{A}g\textcolor{blue}{A}gGtt\textcolor{red}{G}GG ggctacacattagataaaacgtatgaagtacgttagactcggcgcgcgccg
caccctatntttttgagcagatttagtgacctggaaaaaaattttgagtacaaaaacttcctccgaata\textcolor{green}{c}AA\textcolor{red}{A}A\textcolor{red}{A}A\textcolor{blue}{G}GGcGGGg
tgagtatccctgggatgactt\textcolor{green}{AAA}A\textcolor{green}{A}A\textcolor{green}{A}T\textcolor{green}{G}G\textcolor{green}{a}Gt\textcolor{green}{G}G\textcolor{green}{G}tgctctccccgattttttgaatatgtgaggatcattcgcaccaggtccga
gctgagaattgag\textcolor{green}{g}t\textcolor{green}{c}AAAAAAGGGGG\textcolor{green}{a}ttGtccacgcaatcgcgaaccaacgcggaccacaaggcaagaccgataaaggaga
tcccctttgccggtaatgtgcgcggaggctggtttacgtgaggggaaacctaatccgaacttaat\textcolor{red}{A}\textcolor{red}{T}\textcolor{red}{A}\textcolor{red}{A}AA\textcolor{red}{A}GG\textcolor{red}{a}GGGccttag
gtcaatcatgtctttgtgaatggattt\textcolor{red}{A}Ac\textcolor{red}{A}A\textcolor{red}{A}A\textcolor{red}{A}G\textcolor{red}{G}\textcolor{red}{G}\textcolor{blue}{t}\textcolor{red}{G}G\textcolor{red}{G}gaccgcttggggcacccaaattcatgtggtgggcagcgcaac
cggttttgcccccttttagaggccccctg\textcolor{red}{A}t\textcolor{red}{A}A\textcolor{red}{A}A\textcolor{red}{A}G\textcolor{red}{G}\textcolor{red}{G}\textcolor{red}{a}GGGc\textcolor{red}{c}aattatgagagagctaatctcatcgcgctgcgttcat
aaccttgag\textcolor{red}{t}tt\textcolor{red}{A}A\textcolor{red}{A}A\textcolor{red}{A}T\textcolor{red}{A}G\textcolor{red}{G}G\gcccctgggcacatacaagaggagtctttcttcatcagttatagctgtatgacacatatgtactcctttgcattgag\textcolor{red}{a}\textcolor{red}{c}\textcolor{red}{t}\textcolor{red}{A}A\textcolor{red}{A}\textcolor{red}{A}\textcolor{red}{A}G\textcolor{red}{G}\textcolor{red}{G}\gaccgaaagggag
ttgcccattgctaaaagcccaacttgacaaatggagagatgaatcctttgcat\textcolor{red}{A}c\textcolor{red}{t}\textcolor{red}{A}A\textcolor{red}{A}\textcolor{red}{A}\textcolor{red}{A}G\textcolor{red}{G}\textcolor{red}{a}\textcolor{red}{G}c\textcolor{red}{G}Ggaccgaaagggag
tctgtgagcaacgacatcctttacgtgacattagctcgcctccccggggtactaatagcaggaagct\textcolor{red}{a}\textcolor{red}{c}\textcolor{red}{t}\textcolor{red}{A}A\textcolor{red}{A}\textcolor{red}{A}\textcolor{red}{A}G\textcolor{red}{G}\textcolor{red}{a}\textcolor{red}{G}c\textcolor{red}{G}Ga
Where is the Motif???

atgaccgggtactgatagagaagaaggttgggggcgtacacattagataaagctatgaagtacgttagactcggccgcggag
accctattttttagcagatttagtgacctggaaaaaaatatgagtacaaaaacttttccgaatacaataaaacgccggga
tgatctccctgggatgacttaaataatggagtgtggtctctccccgatttttgaatatgtaggatcattccgccaggggtccga
gctgagaattggatgcavaaaaagggattgtctacgtaacccacgtcgacggacccaaagggctacggattataaaagggctatatag
tcttttttgccgtaatgtgcccggaggctgttacgttagggaagcccttaacgacattataataaatagaaagggctatatag
gtcaatcatgtttcttgtgaatggatttaacaataagggcttgggaccggctttgggcacccaaattcaggtggggcgagcg
ccgttttgcgccccgttaggggcccaccctaggggcaattttagagagctaatctctctggtggttccgctttcat
aacttgagttaaaaatatagggagcccctggggacacatcaaaaggggtaggaggaggctttctttatcagttatgtgatagtgcactatgtac
ctgggccttcctggctaaccctgacccaatggaggataactttgcataactaaaaaggagcgaccggaaagggaaag
Finding (15,4) Motif

atgaccgggatactgatAgAAGAAAGGttGGGgtccgtacacattagataaagctagacagttagacttcggcgc
acccctattttttgacgagattttagtgacactggaaaaaaattttgagtacaaaaacttttccgaatc
AtAAAAAAGGcGGGAgctggtatccctgggatgcctAAAAtAAATGGAatGGGgtctctcccgatttttaaatgtagtgattcattcgccagggtccga
gctgagaatttgattcAAAAAAAGGGattGtccacgcgaatcgcgaaccacacgggacccaaagggcaagacgcggataaaggaga
tcccttttgcggtaatgtgccgggaaggtggtttacgtagggaagcctctaaccgacttaaAtAAAtAAAGGaGGccttatag
gtcaatcatgtttctttgtgaatggatttAACAAaAAGGGctGGgaccgctttggccacccaaattcagttggtggcagcgcaca
cggttttcgcccctttagaggcccctgctAAAATAGGGAAGGccaattatgagagagcataaatctctcgctgctgtttcat
aactttagttAAAAAAATAGGGAAGccctgggccacatacaagaggagtctctctttactcgaattatgtgtgatagcactatgtacccttgagttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttt
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.
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
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

ATCCCG gene

TTCCGG gene

ATCCCG gene

ATGCCG gene

ATGCC gene

ATGCC gene
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

```
TGAGGGA
TGAGAGA
TGAGGGA
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:

  cctgatagacgctatctggctatccacgtacgtacgtaggtcctctgtgcgaatctatgcgtttctcaaccat
  agtactgggtgtacatattgatacgtacgtacaccggcaacctgaaacaaacgctcagaaccagaagtgc
  aaacgtacgtgcaccctcttttcttcgttgctctggcccaacgcagggtgtatgtatagctataagacgaatatttt
  agcctccgatgtaagtcataagctgtaactattaccctgcccacccttattacacatctttacgtacgtatcaca
  cttgtttatacaacgcgtcatggcggggtatgcgtttttgggtcggtcgtacgtcgtcgacgtcgttaacgtagtc

- 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:

  cctgatagacgctatctggctatcc\textcolor{blue}{acgtacgt}taggtcctctgtgcgaatctatgctgttttcaaccat
  agtactggtgtacattttgat\textcolor{blue}{acgtacgt}acacccggaactgtaaaaaaaccctacaacgaggtgtatgtataagcgaatatttt
  aacgacccctctttcttcgttggtcttgcccaaccgagggctgatgtataagacgaaatatttt
  agcctccgatgtaagtcataagctgttaactattaacctgcccaccctattacatctt\textcolor{blue}{acgtacgt}atatcaca
  ctgtttataacaacgcgctcatggcggggtatgctttttggtcgtcgatcgcgtc\textcolor{blue}{acgtacgt}cgtctgccctgtttatgctgcgtcgtta

\textbf{Consensus String}
The Motif Finding Problem (cont’d)

The patterns with 2 point mutations:

cctgatagacgctatctggctatccGgtacTtaggtcctctgtgcgaatctatgcgttttcaccat
agtactggtgtacattttgatCcAtacgtacacccggcaacccctgaaaacacgtcagagaccagaaagtgc
aaacgtTAggtgacccctttttcttctgtgctctgctggccaacgagggctgatgtataagacgaaaatttt
agcctccgatgtaagtgcatagctgtaactattaacctggccaccctcattagctatctttacgtCcAtataca
ttgtatcacaacgcgtcatggcggtgtatgcgttttgggtcgctgaacgctcgatcgttaCgtacgGc
The Motif Finding Problem (cont’d)

- The patterns with 2 point mutations:

  cctgatagacgctatctggctatccagttacTtagtcctctgtgcgaatctatgctgttttcaaccat
  agtactggtgtacattttgatCtCAtacgtagacccgaacctgaacacacgctcagaaccagaagtgcc
  aaagctTAgatgcacccctctttttctgtggtcttcggcacaacgagggctgatgtatatgacgaaatttt
  agccctccgatgtaagtcatagtgctgtaactattacctgccacccctattacatctttacgtCtCAtataca
  cttgttatacaacgctcatggcgggggtatgcgtttggtcgtcgtcgcgtctcgctacgctgatgttaCgt tacgGc

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

To define a motif, let's say we know where the motif starts in the sequence.

The motif start positions in their sequences can be represented as $s = (s_1, s_2, s_3, \ldots, s_t)$.
Motifs: Profiles and Consensus

- Line up the patterns by their start indexes
  \[ s = (s_1, s_2, \ldots, s_t) \]

- 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 (\( t \times n \) array)
- \( \ell \) - length of the motif (\( \ell \)-mer)
- \( s_i \) - starting position of an \( \ell \)-mer in sequence \( i \)
- \( s = (s_1, s_2, \ldots, s_t) \) - array of motif’s starting positions
Parameters

\[ l = 8 \]

DNA

\[ n = 69 \]

\[ s_1 = 26 \quad s_2 = 21 \quad s_3 = 3 \quad s_4 = 56 \quad s_5 = 60 \]

\[ t = 5 \]
**Scoring Motifs**

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

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

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>3</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>3</th>
<th>1</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

Consensus: a c g t a c g t

Score: $3+4+4+5+3+4+3+4 = 30$
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 \( \ell \) mers, one from each sequence, that maximizes the consensus score

- **Input**: A \( t \times n \) matrix of DNA, and \( \ell \), the length of the pattern to find

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

$$s_i = [1, ..., n-\ell+1]$$

$$i = [1, ..., t]$$
BruteForceMotifSearch

1. BruteForceMotifSearch(DNA, t, n, ℓ)
2. bestScore ← 0
3. for each \( s = (s_1, s_2, \ldots, s_t) \) from \((1,1 \ldots 1)\) to \((n-\ell+1, \ldots, n-\ell+1)\)
4.     if \( \text{Score}(s, DNA) > \text{bestScore} \)
5.     \hspace{1cm} bestScore ← score(s, DNA)
6.     \hspace{1cm} bestMotif ← (s_1, s_2, \ldots, 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 \(\ell\) 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:
  - \( d_H(v, w) \) is the number of nucleotide pairs that do not match when \( v \) and \( w \) are aligned. For example:

\[
d_H(AAAAAA, ACAAACC) = 2
\]
Total Distance: An Example

- Given \( \mathbf{v} = "\text{acgtacgt}" \) and \( \mathbf{s} \)

\[ d_H(\mathbf{v}, \mathbf{x}) = 0 \]

\( \mathbf{v} \) is the sequence in red, \( \mathbf{x} \) is the sequence in blue

- TotalDistance(\( \mathbf{v}, \text{DNA} \)) = 0
Total Distance: Example

- Given \( v = \text{"acgtacgt"} \) and \( s \)
  
  \[ d_H(v, x) = 1 \]

  \[ d_H(v, x) = 2 \]

  \[ d_H(v, x) = 0 \]

  \[ d_H(v, x) = 1 \]

  \( v \) is the sequence in red, \( x \) is the sequence in blue

- \( \text{TotalDistance}(v, \text{DNA}) = 1 + 0 + 2 + 0 + 1 = 4 \)
Total Distance: Definition

- For each DNA sequence $i$, compute all $d_H(v, x)$, where $x$ is an $\ell$-mer with starting position $s_i$ ($1 \leq s_i \leq n - \ell + 1$)
- Find minimum of $d_H(v, x)$ among all $\ell$-mers in sequence $i$
- $TotalDistance(v, DNA)$ is the sum of the minimum Hamming distances for each DNA sequence $i$
- $TotalDistance(v, DNA) = \min_{s} d_H(v, s)$, 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 $t \times n$ matrix DNA, and $\ell$, the length of the pattern to find
- **Output**: A string $v$ of $\ell$ nucleotides that minimizes $TotalDistance(v, DNA)$ over all strings of that length
Median String Search Algorithm

1. MedianStringSearch (DNA, t, n, l)
2. bestWord $\leftarrow$ AAA…A
3. bestDistance $\leftarrow$ $\infty$
4. for each l-mer $s$ from AAA…A to TTT…T
   if TotalDistance($s$, DNA) < bestDistance
   bestDistance $\leftarrow$ TotalDistance($s$, DNA)
5. bestWord $\leftarrow$ $s$
6. 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 $i$

\[ \text{Score}_i + \text{TotalDistance}_i = t \]

Because there are $\ell$ columns

\[ \text{Score} + \text{TotalDistance} = \ell \times t \]

Rearranging:

\[ \text{Score} = \ell \times t - \text{TotalDistance} \]

\( \ell \times t \) 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 \( 4^\ell \) combinations for \( v \). This number is relatively smaller
Recall the BruteForceMotifSearch:

1. \text{BruteForceMotifSearch}(\text{DNA}, t, n, \ell)
2. \text{bestScore} \leftarrow 0
3. \text{for each } s=(s_1, s_2, \ldots, s_t) \text{ from } (1,1 \ldots 1) \text{ to } (n-\ell+1, \ldots, n-\ell+1)
4. \text{if } (\text{Score}(s, \text{DNA}) > \text{bestScore})
5. \quad \text{bestScore} \leftarrow \text{Score}(s, \text{DNA})
6. \quad \text{bestMotif} \leftarrow (s_1, s_2, \ldots, s_t)
7. \text{return } \text{bestMotif}
Structuring the Search

- How can we perform the line

\[ \text{for each } s = (s_1, s_2, \ldots, s_t) \text{ from } (1,1 \ldots 1) \text{ to } (n-\ell+1, \ldots, n-\ell+1) \? \]

- 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 $4^\ell$ possible $\ell$-mers:

  $\ell$
  \[
  \begin{array}{c}
  \text{aa} \ldots \text{aa} \\
  \text{aa} \ldots \text{ac} \\
  \text{aa} \ldots \text{ag} \\
  \text{aa} \ldots \text{at} \\
  \vdots \\
  \text{tt} \ldots \text{tt}
  \end{array}
  \]

  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:
  
  \[
  11\ldots11 \\
  11\ldots12 \\
  11\ldots13 \\
  11\ldots14 \\
  \vdots \\
  44\ldots44 
  \]

- 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 is not the most efficient data structure for motif finding.

Let's try grouping the sequences by their prefixes.
Search Tree

- Root
  - a
    - aa
    - ac
    - ag
    - at
  - c-
    - ca
    - cc
    - cg
    - ct
  - g-
    - ga
    - gc
    - gg
    - gt
  - t-
    - ta
    - tc
    - tg
    - tt
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`
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
Moving to the next leaf:
Moving to the next leaf:
Visit All Leaves

Printing all permutations in ascending order:

1. \textbf{AllLeaves}(L,k) \quad // \quad L: \text{ length of the sequence}
2. \textbf{a} \leftarrow (1,\ldots,1) \quad // \quad k: \text{ max digit value}
3. \textbf{while} forever \quad // \quad \textbf{a}: \text{ array of digits}
4. \quad \textbf{output a}
5. \quad \textbf{a} \leftarrow \text{NextLeaf}(\textbf{a}, L, k)
6. \quad \textbf{if} \ \textbf{a} = (1,\ldots,1)
7. \quad \textbf{return}
Visit All Leaves: Example

- Moving through all the leaves in order:

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

Moving to the next vertex:
Example

- Moving to the next vertices:

  Location after 5 next vertex moves
Bypass Move

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

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

- Bypassing the descendants of “2-”:
Example

- Bypassing the descendants of “2-“:
Brute Force Search Again

1. \texttt{BruteForceMotifSearchAgain(DNA, t, n, \ell)}
2. \texttt{s \leftarrow (1, 1, ... , 1)}
3. \texttt{bestScore \leftarrow Score(s, DNA)}
4. \texttt{while forever}
5. \texttt{s \leftarrow NextLeaf(s, t, n - \ell + 1)}
6. \texttt{if (Score(s, DNA) > bestScore)}
7. \texttt{bestScore \leftarrow Score(s, DNA)}
8. \texttt{bestMotif \leftarrow (s_1, s_2, ... , s_t)}
9. \texttt{return bestMotif}
Can We Do Better?

- Sets of $s = (s_1, s_2, \ldots, s_t)$ may have a weak profile for the first $i$ positions $(s_1, s_2, \ldots, s_i)$
- Every row of alignment may add at most $\ell$ to Score
- **Optimism**: if all subsequent $(t-i)$ positions $(s_{i+1}, \ldots s_t)$ add
  $$ (t - i) \times \ell \text{ to } \text{Score}(s, i, DNA) $$
- If $\text{Score}(s, i, DNA) + (t - i) \times \ell < \text{BestScore}$, 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, 0)
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 ← (s₁, s₂, s₃, ..., s₉)
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 $s$.
  - The Median String Problem needs to examine $4^\ell$ 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:
  \[
  \text{TotalDistance (prefix, DNA)} > \text{BestDistance}
  \]
  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. \textbf{BranchAndBoundMedianStringSearch}(\textit{DNA}, t, n, \ell)
2. \(s \leftarrow (1, \ldots, 1)\)
3. \(\text{bestDistance} \leftarrow \infty\)
4. \(i \leftarrow 1\)
5. \textbf{while} \(i > 0\)
6. \quad \textbf{if} \(i < \ell\)
7. \quad \quad \text{prefix} \leftarrow \text{string corresponding to the first } i \text{ nucleotides of } s
8. \quad \quad \text{optimisticDistance} \leftarrow \text{TotalDistance(} \text{prefix, DNA} \text{)}
9. \quad \quad \textbf{if} \ \text{optimisticDistance} > \text{bestDistance}
10. \quad \quad \quad (s, i) \leftarrow \text{Bypass}(s, i, \ell, 4)
11. \quad \quad \textbf{else}
12. \quad \quad \quad (s, i) \leftarrow \text{NextVertex}(s, i, \ell, 4)
13. \quad \textbf{else}
14. \quad \quad \text{word} \leftarrow \text{nucleotide string corresponding to } s
15. \quad \quad \textbf{if} \ \text{TotalDistance}(s, \text{DNA}) < \text{bestDistance}
16. \quad \quad \quad \text{bestDistance} \leftarrow \text{TotalDistance(} \text{word, DNA} \text{)}
17. \quad \quad \quad \text{bestWord} \leftarrow \text{word}
18. \quad \quad \quad (s, i) \leftarrow \text{NextVertex}(s, i, \ell, 4)
19. \textbf{return} \text{bestWord}
Improving the Bounds

- Given an $\ell$-mer $w$, divided into two parts at point $i$
  - $u$: prefix $w_1, \ldots, w_i$
  - $v$: suffix $w_{i+1}, \ldots, w_\ell$
- Find minimum distance for $u$ in a sequence
- No instances of $u$ in the sequence have distance less than the minimum distance
- Note this doesn’t tell us anything about whether $u$ is part of any motif. We only get a minimum distance for prefix $u$
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 $d(\text{prefix}) + d(\text{suffix}) \geq \text{bestDistance}$:
  - Motif $w$ ($\text{prefix}.\text{suffix}$) cannot give a better (lower) score than $d(\text{prefix}) + d(\text{suffix})$
  - In this case, we can ByPass()
Better Bounded Median String Search

1. ImprovedBranchAndBoundMedianString(DNA, t, n, l)
2. \( s = (1, 1, \ldots, 1) \)
3. bestdistance = \( \infty \)
4. \( i = 1 \)
5. while \( i > 0 \)
6. if \( i < l \)
7. \( \text{prefix} = \text{nucleotide string corresponding to} \ (s_1, s_2, s_3, \ldots, s_i) \)
8. optimisticPrefixDistance = TotalDistance(\( \text{prefix}, \ DNA \))
9. if \( (\text{optimisticPrefixDistance} < \text{bestsubstring}[i]) \)
10. \( \text{bestsubstring}[i] = \text{optimisticPrefixDistance} \)
11. if \( (l - i < i) \)
12. optimisticSufxDistance = \( \text{bestsubstring}[l-i] \)
13. else
14. optimisticSufxDistance = 0;
15. if \( \text{optimisticPrefixDistance} + \text{optimisticSufxDistance} \geq \text{bestDistance} \)
16. \( (s, i) = \text{Bypass}(s, i, l, 4) \)
17. else
18. \( (s, i) = \text{NextVertex}(s, i, l, 4) \)
19. else
20. word = \text{nucleotide string corresponding to} \ (s_1, s_2, s_3, \ldots, s_l) \)
21. if TotalDistance(\( \text{word}, \ DNA \)) < \text{bestDistance} \)
22. bestDistance = TotalDistance(\( \text{word}, \ DNA \))
23. bestWord = \( \text{word} \)
24. \( (s, i) = \text{NextVertex}(s, i, l, 4) \)
25. return bestWord