Lecture 5:
Finding Regulatory Motifs Within DNA Sequences

Study Chapter 4.4-4.9
How Do You Find Genes?

• Genomes contain billions of bases ($10^9$)
• Within these there are 10s of 1000s genes ($10^4$)
• Genes are 1000s of bases long on average ($10^3$)
• So only 1% of DNA actually codes for the production of proteins ($10^9/(10^3\cdot10^4)$)
• How do we find them?
  – Look for start codons?  
    Too frequent (1/64 in 3 alignments)
  – Long strings without stop codons?  
    Lots of false positives due to replication of DNA sequences
• How does nature do it?
Regulatory Regions

• RNA polymerases seek out regulatory regions or promoting regions that are located 100-1000 bp upstream from the coding region.

• They work in conjunction with special proteins called transcription factors whose presence enables the expression of one or more genes.

• Located within the regulatory regions are the Transcription Factor Binding Sites (TFBS), also known as motifs, which are sequences specific to a given transcription factor.
Transcription Factor Binding Sites

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

```
5'-ccatttagg-3'
cggggctatgcaaccattttaggtgggtcgtcacacattcccctttcgata
cgaggctatgcaacatatttaggtgggtcgtcacacattcccctttcgata
```
Motifs and Transcriptional Start Sites

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

- ATCCCG
- TTCCGG
- ATCCCG
- ATGCCG
- ATGCCG
Identifying Motifs: Complications

- We do not know every motif sequence
- We do not know where it is located relative to the gene’s start
- Motifs can differ slightly from one gene to the next
- We only know that it occurs frequently
- How to discern a Motif’s frequent “similar” pattern from “random” patterns?
An Aside: Solving Cryptograms

• A popular form of word puzzle

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

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

• Try solving it using

http://www.ics.uci.edu/~eppstein/cryptogram/
How’s a Motif Like a Cryptogram?

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

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

• Given a random sample of DNA sequences:
  
  cctgatagacgtatctggctatccacgtacgtaggtcctctgtgcgaatctatgcgttttccaaaccat
  agtactgtgtacatatttgatacgtacgtacaccggcaacctgiaaaacgcgcagaaccagaagtgc
  aaacgtacgtgcacccttcctttttcttggtgtgcttgccaaacgggtgatgtgatatagacgaatat
  ttagcctccgatgtgactatagctgtaactattacgcacccctatattacatcttacgtacgtataca
  Ctttatacaacgcgtcattgccccgtatgcttttgtgtctcgctgtacgtacgtacgtc

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

• Additional information:
  – The hidden sequence is of length 8
  – The pattern is not exactly the same in each sequence because random point mutations have been introduced
The pattern can be found if no mutations

Probability of an 8-mer is $1/4^8$

There are 5 $(68 - 8) = 300$ possible 8-mers

Probability of any one 8-mer is 0.004

So any repeat is rare
The Problem Becomes Harder

• Introduce 2 point mutations into each pattern:

cctgatagacgctatctggctatcc\textcolor{red}{aGgtacTT}aggtcctctgtgcgaatcttatgctgtttccaaccat
agtacttgtgtcattttgat\textcolor{red}{CcAtacgt}acaccggcaacctaacaacgctcagaaccagaagttgca
\textcolor{red}{aaaacgtTAgt}gcaccctcttttctttcgtggctcttgccaacgagggctgatgtatagatgaagacgaaatlattt
agcctccgtatgtaaggtcagctgtgtaactattacctgcaccctattaattcatactt\textcolor{red}{acgtCcAtataca}
ctgttatataacgcgtcatggccgggtatgcgttttggtcgtcgtacgctcgtcgtcgt\textcolor{red}{CcgtaacgGc}

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

Can we still find the motif?
Defining a Motif

- To define a motif, let’s assume that we know where the motif starts in each sequence.
- The start positions can be represented as $s = [s_1, s_2, s_3, \ldots, s_t]$. 

```
<table>
<thead>
<tr>
<th>motif start index</th>
<th>gene start</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td></td>
</tr>
<tr>
<td>$s_2$</td>
<td></td>
</tr>
<tr>
<td>$s_3$</td>
<td></td>
</tr>
<tr>
<td>$s_4$</td>
<td></td>
</tr>
<tr>
<td>$s_t$</td>
<td></td>
</tr>
</tbody>
</table>
```
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 an actual motif and the consensus sequence is generally less than that for any two actual motifs
- *Hamming distance* is number of positions that differ between two strings

\[
\begin{array}{cccccccc}
G & A & G & A & C & T & C & A & T \\
X & X & & & & & & \\
T & A & G & A & C & G & C & A & T \\
\end{array}
\]

A Hamming distance of 2
Consensus Properties

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

\[ d = 4 \]
Defining Some Terms

- **DNA** – array of sequence fragments
- **t** - number of sample DNA sequences
- **n** - length of each DNA sequence
- **l** - length of the motif (l-mer)
- **s_i** - starting position of an l-mer in sequence i
- **s=(s_1, s_2, ..., s_t)** - array of motif’s starting positions
Illustration of Terms

\[
\begin{align*}
\text{DNA} & : l = 8 \\
& : t = 5 \\
& : n = 69
\end{align*}
\]

\[
\begin{align*}
s & \quad s_1 = 26 & s_2 = 21 & s_3 = 3 & s_4 = 56 & s_5 = 60
\end{align*}
\]
• Given \( s = (s_1, \ldots, s_t) \) and DNA:

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

Consensus \( \text{acgtacgt} \)

Score \( 3+4+4+5+3+4+3+4 = 30 \)
The Motif Finding Problem

- **Goal:** Given a set of DNA sequences, find a set of \( \ell \)-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 \( \text{Score}(s, \text{DNA}) \).
Brute Force Solution

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

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

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

- Varying \((n - L + 1)\) positions in each of \(t\) sequences, we’re looking at \((n - L + 1)^t\) sets of starting positions.
- For each set of starting positions, the scoring function makes \(L\) operations, so complexity is \(L(n - L + 1)^t = O(Ln^t)\).
- That means that for \(t = 8\), \(n = 1000\), \(L = 10\) we must perform approximately \(10^{20}\) computations.
- It will take millions of years.
The Median String Problem

• Given a set of $t$ DNA sequences find a pattern that appears in all $t$ sequences with the minimum number of mutations

• This pattern will be the motif

• Rather than finding the maximal consensus string, this approach attempts to the minimal distance string
Total Distance: An Example

- Given \( \nu = \text{“acgtacgt”} \) and \( x \)

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

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

\[
\nu \text{ is the sequence in red, } x \text{ is the sequence in blue}
\]

- **TotalDistance(\( \nu \),DNA) = 0**
Total Distance: An Example

- Given $v = \text{"acgtacgt"}$ and $s$

  $d_H(v, x) = 1$

  \[
  \begin{align*}
  \text{cctgatagacgctatctggctatccacgtac} & \rightarrow \text{acgtacgt} \\
  \text{agtaactgtgtaacttgtgaatctatgcggttctgcgaatctatgcggtttcaccat} & \rightarrow \text{acgtacgt}
  \end{align*}
  \]

  $d_H(v, x) = 0$

  \[
  \begin{align*}
  \text{acgtacgt} & \rightarrow \text{acgtacgt} \\
  \text{acgctcgtacacccgtaacttttaccgggtcgtgatgatgataagacgaaata} & \rightarrow \text{acgtacgt}
  \end{align*}
  \]

  $d_H(v, x) = 2$

  \[
  \begin{align*}
  \text{acgtacgt} & \rightarrow \text{acgtacgt} \\
  \text{acgctcgtacacccgtaacttttaccgggtcgtgatgatgataagacgaaata} & \rightarrow \text{acgtacgt}
  \end{align*}
  \]

  $d_H(v, x) = 0$

  \[
  \begin{align*}
  \text{acgtacgt} & \rightarrow \text{acgtacgt} \\
  \text{acgctcgtacacccgtaacttttaccgggtcgtgatgatgataagacgaaata} & \rightarrow \text{acgtacgt}
  \end{align*}
  \]

  $d_H(v, x) = 1$

  \[
  \begin{align*}
  \text{acgtacgt} & \rightarrow \text{acgtacgt} \\
  \text{acgctcgtacacccgtaacttttaccgggtcgtgatgatgataagacgaaata} & \rightarrow \text{acgtacgt}
  \end{align*}
  \]

$v$ 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 $d_H(v, x)$, 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 $i$
- $\text{TotalDistance}(v, \text{DNA})$ is the sum of the minimum Hamming distances for each DNA sequence $i$
- $\text{TotalDistance}(v, \text{DNA}) = \min_s d_H(v, s)$, where $s$ is the set of starting positions $s_1, s_2, \ldots s_t$
The Median String Problem

- **Goal**: Given a set of DNA sequences, find a median string
- **Input**: A $t \times n$ 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 ← t × l
4. for each l-mer, s, from “aaa…a” to “ttt…t”
5. if TotalDistance(s, DNA) < bestDistance
6. bestDistance ← TotalDistance(s, DNA)
7. bestWord ← s
8. return bestWord
Equivalent Problems

- Motif Finding Problem $\equiv$ 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’re looking for the same thing

\[
\begin{align*}
\text{Alignment} & \quad \text{Score}_i + \text{TotalDistance}_i = t \\
\begin{array}{l}
\text{A} & 3 & 0 & 1 & 0 & 3 & 1 & 1 & 0 \\
\text{C} & 2 & 4 & 0 & 0 & 1 & 4 & 0 & 0 \\
\text{G} & 0 & 1 & 4 & 0 & 0 & 0 & 3 & 1 \\
\text{T} & 0 & 0 & 0 & 5 & 1 & 0 & 1 & 4 \\
\end{array}
\end{align*}
\]

\[
\text{Consensus} \quad \begin{array}{l}
\text{a c g t a c g t} \\
\text{Score} \quad 3+4+4+5+3+4+3+4 \\
\text{TotalDistance} \quad 2+1+1+0+2+1+2+1 \\
\text{Sum} \quad 5 \quad 5 \quad 5 \quad 5 \quad 5 \quad 5 \quad 5 \quad 5
\end{array}
\]

- At any column \( i \)
  \[
  \text{Score}_i + \text{TotalDistance}_i = t
  \]

- Because there are \( l \) columns
  \[
  \text{Score} + \text{TotalDistance} = l^* t
  \]

- Rearranging:
  \[
  \text{Score} = l^* t - \text{TotalDistance}
  \]

- \( l^* t \) is constant the minimization of the right side is equivalent to the maximization of the left side
Why Bother?

• What is the point of reformulating the Motif Finding problem 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
Improving Motif Finding

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

• How can we perform the line

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

• We need a method for efficiently structuring and navigating the many possible motifs
• This is not very different than exploring all \( "(n-l+1)\)-digit" numbers
Improving Median String

1. MedianStringSearch(DNA, t, n, l)
2. bestWord ← “aaa…a”
3. bestDistance ← 8
4. for each l−mer, s, from “aaa…a” to “ttt…t”
5. if TotalDistance(s, DNA) < bestDistance
6. bestDistance ← TotalDistance(s, DNA)
7. bestWord ← s
8. return bestWord
How to Best Explore Permutations?

- For the Median String Problem we need to consider all $4^\ell$ possible $\ell$-mers:

  \[
  \begin{align*}
  &aa\ldots aa \\
  &aa\ldots ac \\
  &aa\ldots ag \\
  &aa\ldots at \\
  &aa\ldots ca \\
  &\vdots \\
  &tt\ldots tt
  \end{align*}
  \]

  How to organize this search?
Simple Code

- We’ve used variations of this idea before

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

- Each call generates a new permutation
```
This is the basic loop structure that we have used for many examples thus far (e.g. BruteForceChange)

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

Is there another way to search permutations?
Our standard method for enumerating permutations just traverses the leaf nodes.

Suppose after checking the first or second letter we already know the solution could not be the one we are looking for?
Analyzing Search Trees

• Characteristics of the search trees:
  – The unique permutations reside at leaves
  – A parent node is a common prefix of its children

• How can we traverse the tree?

• Things we’d like to do:
  – Visit all the nodes (interior and leaves)
  – Visit the next node (in an ordered way)
  – Bypass the children of a node
Depth First Search

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

Initial Location

Location after 10 moves
Visiting the Next Vertex

- Uses 0s to encode unspecified part of interior nodes

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

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

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

- Bypassing descendents of nodes “12—” and “211—”
Revisiting Brute Force Search

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

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

- Sets of \( s = (s_1, s_2, \ldots, s_t) \) 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 \( \text{Score} \)
- **Optimism**: if all subsequent \( (t-i) \) positions \( (s_{i+1}, \ldots, s_t) \) add

\[
(t - i) \times \ell \quad \text{to} \quad \text{Score}(s, i, \text{DNA})
\]

- If \( \text{Score}(s, i, \text{DNA}) + (t - i) \times \ell < \text{BestScore} \), it makes no sense to search in vertices of the current subtree
  - Use **ByPass**()
Before we apply a branch-and-bound strategy let’s rewrite the brute-force algorithm using a search tree

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

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

• This saves us from looking at \((n - l + 1)^{t-i}\) leaves
  - Use NextVertex() and ByPass() to navigate the tree
def BranchAndBoundMotifSearch(DNA,t,n,l):
    s = [0 for i in xrange(t)]
    bestScore = 0
    i = 0
    while (True):
        if (i < t):
            optimisticScore = Score(s, DNA, l) + (t-i)*l
            if (optimisticScore < bestScore):
                s, i = Bypass(s,i,t,n-l+1)
            else:
                s, i = NextVertex(s,i,t,n-l+1)
        else:
            score = Score(s, DNA, l)
            if (score > bestScore):
                bestScore = score
                bestMotif = [x for x in s]
                s, i = NextVertex(s,i,t,n-l+1)
            if (sum(s) == 0):
                break
    return bestMotif
A Bad Example

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

• The target motif has a consensus score of 30
• But \([2, 5, 46, 4, 1] = 31\) and \([2, 5, 46, 6, 1] = 34\)
• Recall the computational differences between motif search and median string search
  – The Motif Finding Problem needs to examine all \((n-l+1)^t\) combinations for \(s\).
  – The Median String Problem needs to examine \(4^t\) combinations of \(v\). This number is relatively small
• We want to use median string algorithm with the Branch and Bound trick!
Insight for Improving Median Search

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

\[
\text{TotalDistance (prefix, DNA)} > \text{BestDistance}
\]

there is no use exploring the remaining part of the word

- We can eliminate that branch and BYPASS exploring that branch further
def BranchAndBoundMedianSearch(DNA, t, n, l):
    s = [1 for i in xrange(t)]
    bestDistance, bestWord = l*t, ''
    i = 1
    while (i > 0):
        if (i < l):
            prefix = NucleotideString(s, i)
            optimisticDistance = TotalDistance(prefix, DNA)
            if (optimisticDistance > bestDistance):
                s, i = Bypass(s, i, l, t)
            else:
                s, i = NextVertex(s, i, l, t)
        else:
            word = NucleotideString(s, l)
            if (TotalDistance(word, DNA) < bestDistance):
                bestDistance = TotalDistance(word, DNA)
                bestWord = word
                s, i = NextVertex(s, i, l, t)
    return bestWord
Further Improvements

• More improvements to Motif searching
  – Why just prune based on prefixes?
    Can you consider suffixes too?
  – Consider multiple letters at a time?

• How do you really find a gene?
  – Co-occurrence of a likely promoter
  – A series of intron/exon boundaries
  – Plausible open-read frame
  – Likely codon usage
  – Still makes only about 50% accurate predictions

• Next Time
  – We look into greedy algorithms