Motif Finding

Selected Topics in Computer Intelligence - 2015

Bioinformatics Programming

Computer Engineering, Chiang Mai University

Regulatory Motifs in DNA Sequences

- Organisms are susceptible to infections from pathogens
  - Anything that can cause disease in its host (animals, plants, , …)
  - Infectious agent – Microorganisms
    - virus, bacterium, fungus, …

- Some of the diseases that are caused by viral pathogen:
  - Smallpox, influenza, measles, chickenpox, ebola
- In Fruit files, they have a small set of immunity genes
  - Usually temporarily inactive but get switch on when it is infected
  - When the genes are active, they produce proteins that destroy the pathogen – curing the infection

Regulatory Motifs in DNA Sequences

- We can design an experiment to find out which genes are switched on as an immune response
  - Determining what triggers their activation
- Many immune genes in the flies genome have strings similar to …

...TCGGGGATTTCC...

  - This **short strings** are some sort of binding sites – Regulatory Motifs
  - The transcription factor proteins bind to these motifs, encouraging transcription process to start from here on

Regulatory Motifs in DNA Sequences

- Ideally, there may be many regions from genes in the genome that contain this binding site
- We generally do not know what pattern of these short strings look like.
- We need an algorithm that…
  - Takes a set of sequences from a genome - input
  - Outputs set of short substrings that seem to occur quite often
    - What does the pattern looks like?

Regulatory Motifs in DNA Sequences

- A popular approach to motif finding is based on the assumption:
  - Frequent or rare words in genome may correspond to regulatory motifs in DNA

Profiles

- DNA sequences with implanted motifs

```
CGGGGCTGGGTCGTCACATTCCCCTTTCGATA
TTTGAGGGTGCCCAATAACCAAAGCGGACAAA
GGGATGCCGTTTGACGACCTAAATCAACGGCC
AAGGCCAGGAGCGCCTTTGCTGGTTCTACCTG
AATTTTCTAAAAAGATTATAATGTCGGTCCTC
CTGCTGTACAACTGAGATCATGCTGCTTCAAC
TACATGATCTTTTGTGGATGAGGGAATGATGC
```

```
CGGGGCTATGCAACTGGGTCGTCACATTCCCCTTTCGATA
TTTGAGGGTGCCCAATAAATGCAACTCCAAAGCGGACAAA
GGATGCAACTGATGCCGTTTGACGACCTAAATCAACGGCC
AAGGATGCAACTCCAGGAGCGCCTTTGCTGGTTCTACCTG
AATTTTCTAAAAAGATTATAATGTCGGTCCATGCAACTTC
CTGCTGTACAACTGAGATCATGCTGCATGCAACTTTCAAC
TACATGATCTTTTGATGCAACTTGGATGAGGGAATGATGC
```

Profiles

- DNA sequences with implanted motifs

```
CGGGGCTATGCAACTGGGTCGTCACATTCCCCTTTCGATA
TTTGAGGGTGCCCAATAAATGCAACTCCAAAGCGGACAAA
GGATGCAACTGATGCCGTTTGACGACCTAAATCAACGGCC
AAGGATGCAACTCCAGGAGCGCCTTTGCTGGTTCTACCTG
AATTTTCTAAAAAGATTATAATGTCGGTCCATGCAACTTC
CTGCTGTACAACTGAGATCATGCTGCATGCAACTTTCAAC
TACATGATCTTTTGATGCAACTTGGATGAGGGAATGATGC
```

```
CGGGGCTATGCAACTGGGTCGTCACATTCCCCTTTCGATA
TTTGAGGGTGCCCAATAAATGCAACTCCAAAGCGGACAAA
GGATGCAACTGATGCCGTTTGACGACCTAAATCAACGGCC
AAGGATGCAACTCCAGGAGCGCCTTTGCTGGTTCTACCTG
AATTTTCTAAAAAGATTATAATGTCGGTCCATGCAACTTC
CTGCTGTACAACTGAGATCATGCTGCATGCAACTTTCAAC
TACATGATCTTTTGATGCAACTTGGATGAGGGAATGATGC
```

Profiles

- DNA sequences with implanted motifs

```
CGGGGCTATGCAACTGGGTCGTCACATTCCCCTTTCGATA
TTTGAGGGTGCCCAATAAATGCAACTCCAAAGCGGACAAA
GGATGCAACTGATGCCGTTTGACGACCTAAATCAACGGCC
AAGGATGCAACTCCAGGAGCGCCTTTGCTGGTTCTACCTG
AATTTTCTAAAAAGATTATAATGTCGGTCCATGCAACTTC
CTGCTGTACAACTGAGATCATGCTGCATGCAACTTTCAAC
TACATGATCTTTTGATGCAACTTGGATGAGGGAATGATGC
```

```
CGGGGCTATcCAgCTGGGTCGTCACATTCCCCTTTCGATA
TTTGAGGGTGCCCAATAAggGCAACTCCAAAGCGGACAAA
GGATGgAtCTGATGCCGTTTGACGACCTAAATCAACGGCC
AAGGAaGCAACcCCAGGAGCGCCTTTGCTGGTTCTACCTG
AATTTTCTAAAAAGATTATAATGTCGGTCCtTGgAACTTC
CTGCTGTACAACTGAGATCATGCTGCATGCcAtTTTCAAC
TACATGATCTTTTGATGgcACTTGGATGAGGGAATGATGC
```

The pattern ATGCAACT is randomly mutated in 2 positions;
no two patterns are the same.

Profiles

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

- A small collection of NF-kB binding sites

TCGGGGATTTCC

Profiles

- Can we reconstruct the **pattern P** by analyzing the **DNA sequences**?
- To formulate the motif finding problem, we need to define precisely what we mean by "motif"
  - We CANNOT rely on a single string!!!
- A more flexible representation of a motif is required
  - Profile matrix!!!

Profiles

- Profile Matrix
  - A set of ***t*** DNA sequences
  - Each sequence has ***n*** nucleotides
  - ***s = (s1, s2, s3, …, st)***
    - An array contains set of positions, ***si*** ,in each DNA sequence
    - The pattern ***l***-mers starting at these positions
    - These positions can be compiles into alignment matrix (***t*** x ***l***)
  - Based on the alignment matrix, the profile matrix (***4*** x ***l***) is computed – aka profile
  - We can form a consensus string from the most popular element in each column of the alignment matrix

Profiles

- Superposition of the seven highlight 8-mer

```
                    CGGGGCTATcCAgCTGGGTCGTCACATTCCCCTT...
TTTGAGGGTGCCCAATAAggGCAACTCCAAAGCGGACAAA
                      GGATGgAtCTGATGCCGTTTGACGACCTA...
                    AAGGAaGCAACcCCAGGAGCGCCTTTGCTGG...
AATTTTCTAAAAAGATTATAATGTCGGTCCtTGgAACTTC
    CTGCTGTACAACTGAGATCATGCTGCATGCcAtTTTCAAC
                TACATGATCTTTTGATGgcACTTGGATGAGGGAATGATGC
```

s = (8, 19, 3, 5, 32, 27, 15)

Profiles

- The alignment matrix, profile matrix and consensus string when s = (8, 19, 3, 5, 32, 27, 15)

Alignment		A	T	C	C	A	G	C	T
		G	G	G	C	A	A	C	T
		A	T	G	G	A	T	C	T
		A	A	G	C	A	A	C	C
		T	T	G	G	A	A	C	T
		A	T	G	C	C	A	T	T
		A	T	G	G	C	A	C	T
Profile	**A**	5	1	0	0	5	5	0	0
	T	1	5	0	0	0	1	1	6
	G	1	1	6	3	0	1	0	0
	C	0	0	1	4	2	0	6	1
Consensus		A	T	G	C	A	A	C	T

Profiles

- By varying the starting positions in ***s***, we construct a large number of different profile matrices
- It is necessary to have some way of grading these profiles
  - High conservation of pattern?
  - No conservation at all?
- We need to find the starting positions ***s*** corresponding to the most conserved profile!!!

The Motif Finding Problem

- $P(s)$ – the profile matrix given starting position s
- $M_{P(s)}(j)$ – the largest count in column j of $P(s)$
- $Score(s, DNA) = \sum_{j=1}^{l} M_{P(s)}(j)$ – consensus score
  - Measure the strength of a profile corresponding to the starting position s
  - $Score = l.t$ - the best possible alignment
  - $Score = l.t / 4$ - the worst possible alignment

The Motif Finding Problem

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

Input: A $t \times n$ matrix of DNA, and l, the length of the pattern to find.

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

The Motif Finding Problem

- Consensus score

Alignment		A	T	C	C	A	G	C	T
		G	G	G	C	A	A	C	T
		A	T	G	G	A	T	C	T
		A	A	G	C	A	A	C	C
		T	T	G	G	A	A	C	T
		A	T	G	C	C	A	T	T
		A	T	G	G	C	A	C	T
Profile	**A**	5	1	0	0	5	5	0	0
	T	1	5	0	0	0	1	1	6
	G	1	1	6	3	0	1	0	0
	C	0	0	1	4	2	0	6	1
Consensus		A	T	G	C	A	A	C	T

$$Score(s, DNA) = 5 + 5 + 6 + 4 + 5 + 5 + 6 + 1 = 42$$

Median String

- Median string
  - Another view of Motif finding problem
  - Using the Hamming distance to compare two strings, v and w,

    $d_H(v, w)$ - the number of positions that differ in the two strings
- e.g., $d_H(ATTGTC, ACTCTC) = 2$

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

Median String

- Total Hamming Distance

$$d_H(v,s) = \sum_{i=1}^{t} d_H(v,s_i)$$

- The minimum possible total hamming distance

$$totalDistance(v,DNA) = \min_s(d_H(v,s))$$

Median String Problem:
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

- Total Hamming Distance for the consensus string **ATGCAACT**

A	**T**	C	**C**	**A**	G	**C**	**T**
G	G	**G**	**C**	**A**	**A**	**C**	**T**
A	**T**	**G**	G	**A**	T	**C**	**T**
A	A	**G**	**C**	**A**	**A**	**C**	C
T	**T**	**G**	G	**A**	**A**	**C**	**T**
A	**T**	**G**	**C**	C	**A**	T	**T**
A	**T**	**G**	G	C	**A**	**C**	**T**

- Total Hamming distance = # nonbold letters

Search Trees

- Both Motif Finding and Median String problems have a large number of alternatives to find the best one
  - Number of starting positions : $(n - l + 1)^t$
  - Number of possible l-mers, with k-letter alphabet : k^l

```
NextLeaf(a, L, k)
for i ← L to 1
    if a_i < k
        a_i ← a_i + 1
        return a
    a_i ← 1
return a
```

```
AllLeaves(L, k)
a ← (1, ..., 1)
while forever
    output a
    a ← NextLeaf(a, L, k)
    if a = (1, 1, ..., 1)
        return
```

  - Similar to counting decimal numbers

Search Tree

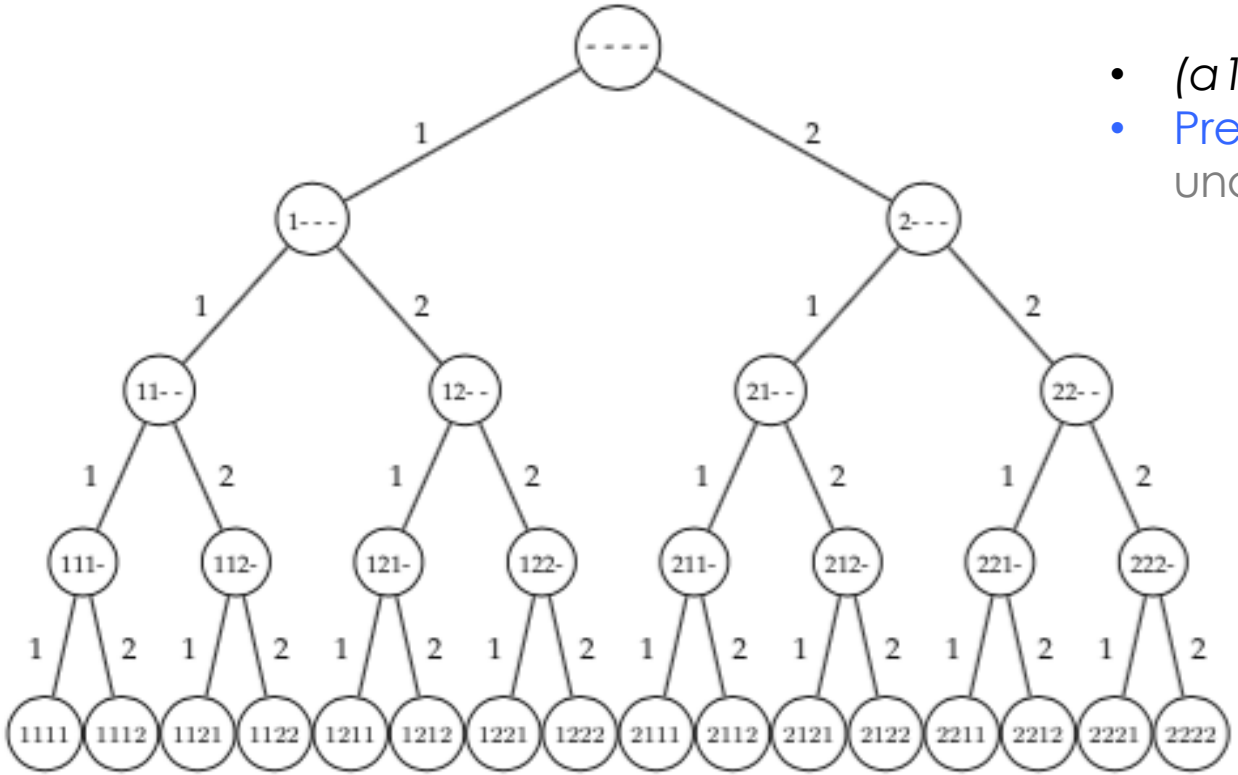
- Possible alternatives

$$
\begin{array}{cccccc}
(& 1, & 1, & \ldots, & 1, & 1 &) \\
(& 1, & 1, & \ldots, & 1, & 2 &) \\
(& 1, & 1, & \ldots, & 1, & 3 &) \\
 & & & \vdots & & & \\
(& 1, & 1, & \ldots, & 1, & n-l+1 &) \\
(& 1, & 1, & \ldots, & 2, & 1 &) \\
(& 1, & 1, & \ldots, & 2, & 2 &) \\
(& 1, & 1, & \ldots, & 2, & 3 &) \\
 & & & \vdots & & & \\
(& 1, & 1, & \ldots, & 2, & n-l+1 &) \\
 & & & \vdots & & & \\
(& n-l+1, & n-l+1, & \ldots, & n-l+1, & 1 &) \\
(& n-l+1, & n-l+1, & \ldots, & n-l+1, & 2 &) \\
(& n-l+1, & n-l+1, & \ldots, & n-l+1, & 3 &) \\
 & & & \vdots & & & \\
(& n-l+1, & n-l+1, & \ldots, & n-l+1, & n-l+1 &)
\end{array}
$$

AA$\cdots$AA
AA$\cdots$AT
AA$\cdots$AG
AA$\cdots$AC
AA$\cdots$TA
AA$\cdots$TT
AA$\cdots$TG
AA$\cdots$TC
$\vdots$
CC$\cdots$GG
CC$\cdots$GC
CC$\cdots$CA
CC$\cdots$CT
CC$\cdots$CG
CC$\cdots$CC

Search Tree

- We often represent all l-mers as leaves in a tree
  - L levels (excluding the root level)
  - Each vertex has k children



- *(a1,a2,…,ai)* is a vertex at level *i*
- Prefix to all of the leaves underneath it

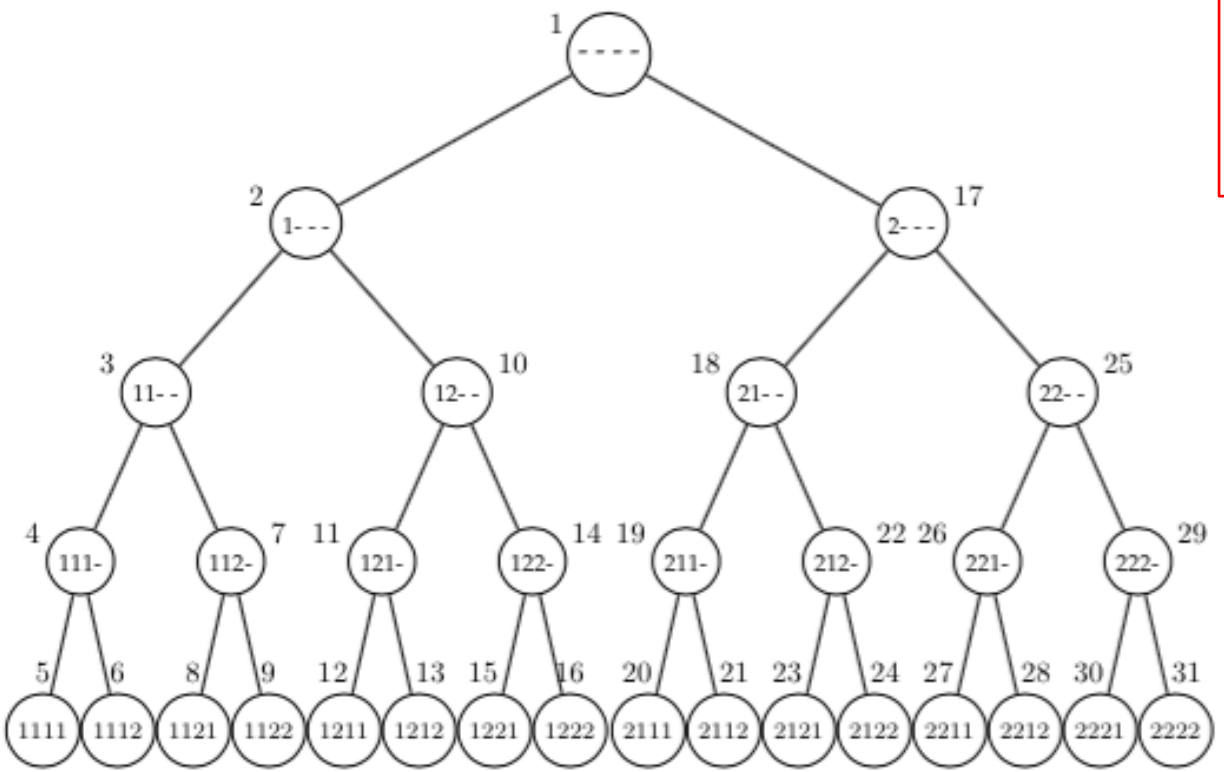
4-mers, k=2 (1, 2)

Search Tree

- Internal vertices do not represent a sensible choice
  - Since we only want to find an L-mers profile
  - We have to find a way to scan only the leaves of a tree
    - But now we can use the branch-and-bound technique
- Depth first traversing
  - Starting from the root and consider each of its k children in order
  - For each child, again consider each of its k children and so on

Search Tree

- Order of traversing vertices for a tree with $L = 4$, and $k = 2$
  - A recursive algorithm can be used




```
PREORDER(v)
output v
if v has children
    PREORDER( left child of v )
    PREORDER( right child of v )
```

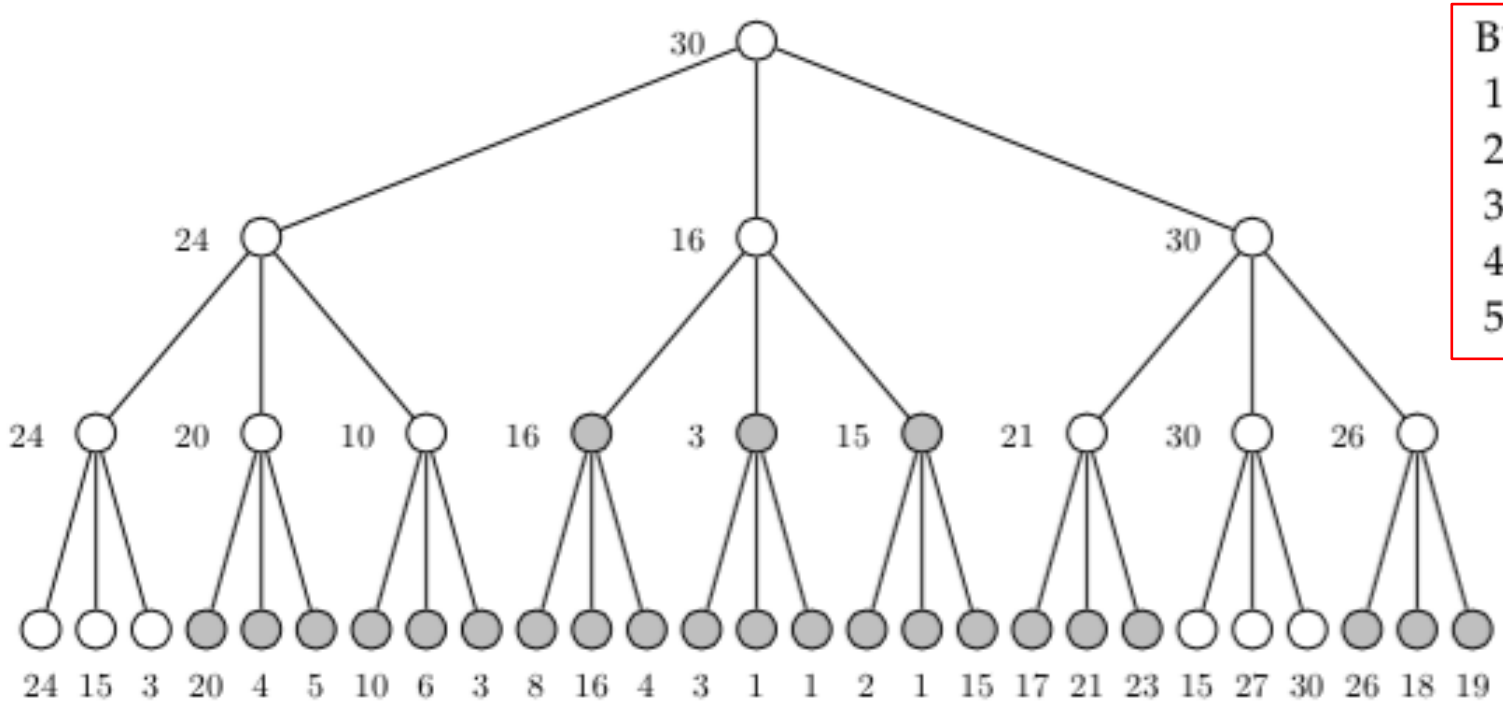
```
NEXTVERTEX(a, i, L, k)
if i < L
    a_{i+1} ← 1
    return (a, i + 1)
else
    for j ← L to 1
        if a_j < k
            a_j ← a_j + 1
            return (a, j)
return (a, 0)
```

Search Tree

- Branch-and-Bound approach
  - Ignore any descendents of a vertex if it is not interesting!!!
    - None of descendents could possibly have a better score than the best leaf that has already been explored
  - At each vertex we calculate a bound
    - The most optimistic score of any leaves in the subtree
    - Decide whether or not to consider its children
      - Skip an entire subtree rooted at the vertex!!!!

Search Tree

- Branch-and-Bound (cont.)




```
BYPASS(a, i, L, k)
for j ← i to 1
    if a_j < k
        a_j ← a_j + 1
        return (a, j)
return (a, 0)
```

- Scores at internal vertices represent the max score in the subtree rooted at that vertex

Search Motifs

- The brute force approach

```
BRUTEFORCEMOTIFSEARCH(DNA, t, n, l)
1  bestScore ← 0
2  for each (s_1, ..., s_t) from (1, ..., 1) to (n − l + 1, ..., n − l + 1)
3      if Score(s, DNA) > bestScore
4          bestScore ← Score(s, DNA)
5          bestMotif ← (s_1, s_2, ..., s_t)
6  return bestMotif
```

- $(n - l + 1)^t$ possible starting positions
- For each s, requires $O(l)$ operations
- Overall complexity : $O(ln^t)$
- How to implement line 2?

Search Motifs

- The brute force approach (cont.)
  - Using NEXTLEAF to traverse all the leaf nodes

```
BRUTEFORCEMOTIFSEARCHAGAIN(DNA, t, n, l)
s ← (1, 1, ..., 1)
bestScore ← Score(s, DNA)
while forever
    s ← NEXTLEAF(s, t, n − l + 1)
    if Score(s, DNA) > bestScore
        bestScore ← Score(s, DNA)
        bestMotif ← (s_1, s_2, ..., s_t)
    if s = (1, 1, ..., 1)
        return bestMotif
```

```
NEXTLEAF(a, L, k)
for i ← L to 1
    if a_i < k
        a_i ← a_i + 1
        return a
    a_i ← 1
return a
```

- *DNA* – *pattern to find with length* l
- t – # DNA sequences
- n – length of each sequence

Search Motifs

- The branch-and-bound strategy
  - Using NEXTVERTEX to explore each leaf

```
SimpleMotifSearch(DNA, t, n, l)
s ← (1, ..., 1)
bestScore ← 0
i ← 1
while i > 0
    if i < t
        (s, i) ← NextVertex(s, i, t, n − l + 1)
    else
        if Score(s, DNA) > bestScore
            bestScore ← Score(s, DNA)
            bestMotif ← (s_1, s_2, ..., s_t)
        (s, i) ← NextVertex(s, i, t, n − l + 1)
return bestMotif
```

```
NextVertex(a, i, L, k)
if i < L
    a_{i+1} ← 1
    return (a, i + 1)
else
    for j ← L to 1
        if a_j < k
            a_j ← a_j + 1
            return (a, j)
return (a, 0)
```

Search Motifs

- How to rule out some of uninteresting sets of positions immediately without iterating over their subtree
- $Score(\mathbf{s}, i, DNA)$ - Partial consensus score
  - Score of the $i \times l$ alignment matrix (only first i rows of DNA)
  - Corresponding to starting position $(s1, s2, s3, \ldots, si, -, -, \ldots, -)$
  - At best, the remaining $(t - i)$ rows can only improve the score by $(t - i).l$
  - Max possible consensus score = $Score(\mathbf{s}, i, DNA) + (t - i).l$
- If $Score(\mathbf{s}, i, DNA) + (t - i).l < bestScore$
  - We can skip that branch of tree
  - saving trouble of $(n - l + 1)^{t-i}$ searching

Search Motifs

- With partial consensus score

```
BranchAndBoundMotifSearch(DNA, t, n, l)
s ← (1, ..., 1)
bestScore ← 0
i ← 1
while i > 0
    if i < t
        optimisticScore ← Score(s, i, DNA) + (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
        if Score(s, DNA) > bestScore
            bestScore ← Score(s)
            bestMotif ← (s_1, s_2, ..., s_t)
        (s, i) ← NextVertex(s, i, t, n − l + 1)
return bestMotif
```

```
NextVertex(a, i, L, k)
if i < L
    a_{i+1} ← 1
    return (a, i + 1)
else
    for j ← L to 1
        if a_j < k
            a_j ← a_j + 1
            return (a, j)
return (a, 0)
```

```
Bypass(a, i, L, k)
for j ← i to 1
    if a_j < k
        a_j ← a_j + 1
        return (a, j)
return (a, 0)
```

Finding a Median String

- An alternative approach to finding motifs
- With the brute force algorithm

```
BRUTEFORCEMEDIANSEARCH(DNA, t, n, l)
bestWord ← AAA···AA
bestDistance ← ∞
for each l-mer word from AAA...A to TTT...T
    if TOTALDISTANCE(word, DNA) < bestDistance
        bestDistance ← TOTALDISTANCE(word, DNA)
        bestWord ← word
return bestWord
```

- Running time – $O(4^l.n.t)$
- Typical motif has a length (l) ranging from 8 to 15
- Analyzed regions (n) ranging from 500 to 1000 nucleotides

Finding a Median String

- If A, C, G, T are coded as numeral (1, 2, 3, 4)

```
SimpleMedianSearch(DNA, t, n, l)
s ← (1, 1, ..., 1)
bestDistance ← ∞
i ← 1
while i > 0
    if i < l
        (s, i) ← NextVertex(s, i, l, 4)
    else
        word ← nucleotide string corresponding to (s_1, s_2, ... s_l)
        if TotalDistance(word, DNA) < bestDistance
            bestDistance ← TotalDistance(word, DNA)
            bestWord ← word
        (s, i) ← NextVertex(s, i, l, 4)
return bestWord
```

```
NextVertex(a, i, L, k)
if i < L
    a_{i+1} ← 1
    return (a, i + 1)
else
    for j ← L to 1
        if a_j < k
            a_j ← a_j + 1
            return (a, j)
return (a, 0)
```

Finding a Median String

- The branch-and-bound approach
  - If the total distance between i-prefix of word and DNA is larger than the smallest seen so far
  - Again, we can skip subtrees of that corresponding vertex

Finding a Median String

- The branch-and-bound approach (cont.)

```
BranchAndBoundMedianSearch(DNA, t, n, l)
s ← (1, 1, ..., 1)
bestDistance ← ∞
i ← 1
while i > 0
    if i < l
        prefix ← nucleotide string corresponding to (s_1, s_2, ..., s_i)
        optimisticDistance ← TotalDistance(prefix, DNA)
        if optimisticDistance > bestDistance
            (s, i) ← Bypass(s, i, l, 4)
        else
            (s, i) ← NextVertex(s, i, l, 4)
    else
        word ← nucleotide string corresponding to (s_1, s_2, ... s_l)
        if TotalDistance(word, DNA) < bestDistance
            bestDistance ← TotalDistance(word, DNA)
            bestWord ← word
        (s, i) ← NextVertex(s, i, l, 4)
return bestWord
```

```
NextVertex(a, i, L, k)
if i < L
    a_{i+1} ← 1
    return (a, i + 1)
else
    for j ← L to 1
        if a_j < k
            a_j ← a_j + 1
            return (a, j)
return (a, 0)
```

```
Bypass(a, i, L, k)
for j ← i to 1
    if a_j < k
        a_j ← a_j + 1
        return (a, j)
return (a, 0)
```