EXERCISE 7: MOTIF SEARCH

Programming in Bioinformatics

The Motif Finding Problem

The motif finding problem is a fundamental problem in computational biology that involves finding patterns in sequences of DNA or protein. The problem is to identify a short sequence of nucleotides or amino acids of length *l* that is repeated in a larger sequence of DNA or protein.

Consider a set of *t* DNA sequences and each sequence has length *n*, thus creating a matrix $t \times n$. Select one position in each of these sequences *t* and create an array of starting indexes $s = (s_1, s_2, \ldots, s_t)$, with $1 \le s_i \le n - l + 1$. The *l*-mers starting at these positions form an $t \times l$ alignment matrix. A $4 \times l$ frequency profile can be constructed to total the number of each nucleotide base in each position of the alignment matrix. The most common letter in each column of the matrix is concatenated together to form a new string called the consensus string and the sum of the number of times that each of those letters occurs in their respective columns is the score for a particular consensus string.

Alignment matrix

$$\begin{array}{ccccccc}
A & T & C & C & G & T & A \\
G & T & G & C & A & T & A \\
A & A & G & C & G & T & A \\
A & T & G & C & G & T & G
\end{array}$$

Frequency profile

$$\begin{array}{cccccccc}
A & 3 & 1 & 0 & 0 & 1 & 0 & 3 \\
C & 0 & 0 & 1 & 4 & 0 & 0 & 0 \\
G & 1 & 0 & 3 & 0 & 3 & 0 & 1 \\
T & 0 & 3 & 0 & 0 & 0 & 4 & 0
\end{array}$$

Consensus string $\quad$ A T G C G T A

Score $\quad 3+3+3+4+3+4+3 = 23$

The method for finding the consensus string generates all possible sets of starting positions as arrays and then determines the set that produces the best consensus string according to the calculated scores. A tree is used to generate all possible sets of starting positions, and four tree traversal methods are used in the brute force and the branch and bound algorithms for solving the motif-finding problem.

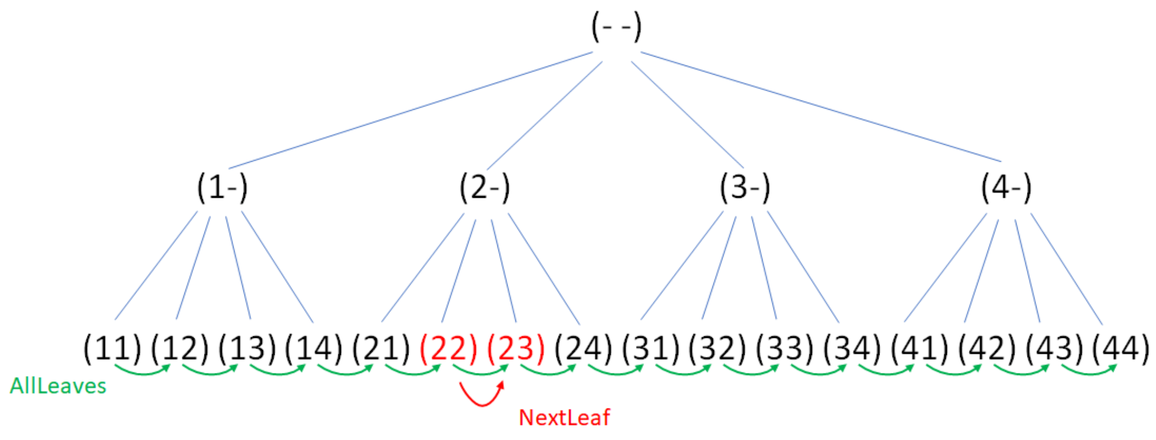

Figure 1: All 2-mers in the four-letter alphabet {1, 2, 3, 4} represented as leaves in a tree. Traversal methods NextLeaf (in red) and AllLeaves (in green) are highlighted.

NextLeaf

The NextLeaf method produces only the leaves of the tree and not the other vertices. It is similar to the natural process of counting. The leaves of this tree are used as the sets of starting positions to search in each DNA sequence for the best motif.

AllLeaves

The AllLeaves method outputs all *l*-mers in the order as is shown in Figure 1.

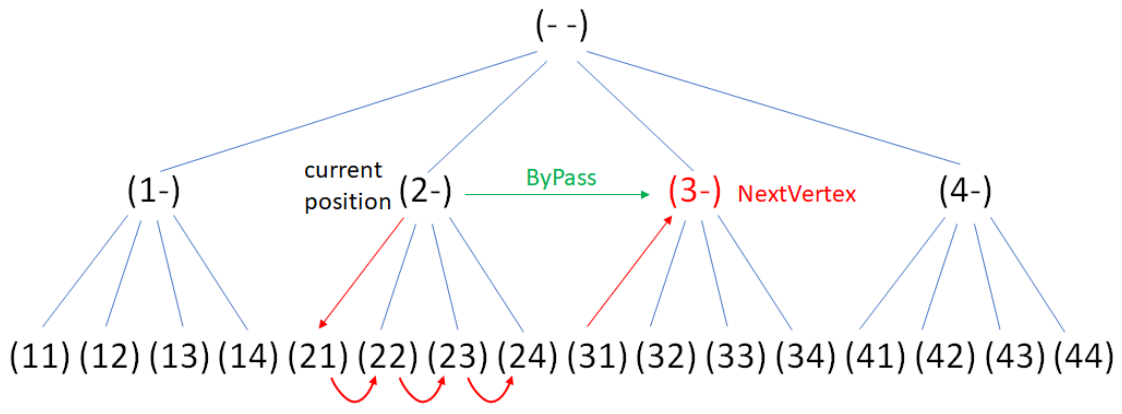

Figure 2: All 2-mers in the four-letter alphabet {1, 2, 3, 4} represented as leaves in a tree. Traversal methods NextVertex (in red) and ByPass (in green) are highlighted.

NextVertex

The tree traversal NextVertex sequentially generates every vertex of the tree from the root to the leaves.

ByPass

The ByPass method is used in conjunction with the NextVertex method in one of the approaches to solving the motif-finding problem. It is used to skip certain branches in the tree generated by the NextVertex method if it is determined that they will not produce a desired result. When we no longer wish to continue moving down a certain branch of the tree to check those vertices the Bypass method jumps up and over in the tree to the next branch to check those vertices.

Score

The DNA score totals the number of positions between two strings that are the same. Thus, a higher score means that the strings are more identical to one another. The Score method returns the total score of each consensus string. It first creates the profile matrix for the DNA, which is an integer array with length four. The profile matrix compares the motifs found at each position in the current starting position set, and totals the number of times each letter (A, C, G, and T) occurs in each column of the matrix. The letters with the most occurrences in each column are concatenated together to create the consensus string. The total DNA score for the consensus string is the sum of the number of times that each of those letters occurs in their respective columns.

Word-based algorithms

There are three word-based algorithms to solve the motif-finding problem: a brute-force algorithm, a branch-and-bound algorithm and a greedy algorithm.

The brute force algorithm determines the consensus string by determining which set of starting positions produces the best DNA score. It is an exhaustive search because it checks every possible set of starting positions produced by the NextLeaf method.

The branch and bound algorithm uses a combination of the NextVertex method and the Bypass method to skip branches of the tree if it is determined that they cannot produce a better score than the score that has already been obtained.

The greedy algorithm does not use any of the aforementioned tree traversals because it is not an exhaustive search method. However, the greedy method does do an exhaustive search on the first two strands of DNA to determine the best motif in these two strands. This motif is called the seed. The method then sequentially searches the remaining DNA strands for the motif in each strand that best matches the seed and the motifs that have already been found.

We will implement the brute-force and branch-and-bound motif search algorithms. More detailed description of the functions can be also found in An Introduction to Bioinformatics Algorithms (Book 1, chapter 4).

The Brute-Force Motif Search

Task 1

- In R, create a function `Score()`, that calculates the score for a consensus string.
- Input:
    - an array of starting indexes
    - `DNAStringSet` of sequences (for example file `seq_score.fasta`)
    - motif length
- Output:
    - the score for the consensus string

Task 2

- In R, create function `NextLeaf()` according to the following pseudocode.
- Input:
    - *L*-mer $a = (a_1\ a_2\ \dots\ a_L)$ of starting indexes
    - number of DNA sequences
    - `k = n - l + 1`, where *n* is length of DNA sequences and *l* is motif length
- Output:
    - *L*-mer of the next leaf in the tree

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

Task 3

- In R, create a function `BFMotifSearch()` according to the following pseudocode.
- Input:
    - `DNAStringSet` of DNA sequences (for example file `seq_motif.fasta`)
    - number of DNA sequences
    - length of each DNA sequence
    - motif length
- Output:
    - an array of starting positions for each DNA sequence with the best score for the consensus string

```
BFMotifSearch(DNA, t, n, l)
s ← (1, 1, ... , 1)
bestScore ← Score(s, DNA, l)
while forever
  s ← NextLeaf(s, t, n - l + 1)
  if Score(s, DNA, l) > bestScore
    bestScore ← Score(s, DNA, l)
    bestMotif ← ($s_1$, $s_2$, . . . , $s_t$)
  if s = (1, 1, . . . , 1)
    return bestMotif
```

The Branch-and-Bound Motif Search

Task 4

- In R, create a function `NextVertex()` according to the following pseudocode.
- Input:
    - *L*-mer `a = (`a_1 a_2 … a_L`)` of starting indexes
    - level of vertex
    - number of DNA sequences
    - `k = n - l + 1`, where *n* is length of DNA sequences and *l* is motif length
- Output:
    - *L*-mer of the next vertex in the tree
    - current level of vertex

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

Task 5

- In R, create a function `ByPass()` according to the following pseudocode.
- Input:
    - *L*-mer `a = (`a_1 a_2 … a_L`)` of starting indexes
    - level of vertex
    - number of DNA sequences
    - `k = n - l + 1`, where *n* is length of DNA sequences and *l* is motif length
- Output:
    - *L*-mer of the next leaf after a skip of the subtree
    - current level of vertex

```
ByPass(a, i, L, k)
 for j ← i to 1
   if $a_j < k$
     $a_j \leftarrow a_j + 1$
     return (a, j)
 return (a, 0)
```

Task 6

- In R, create a function `BBMotifSearch()` according to the following pseudocode.
- Input:
    - DNAStringSet of DNA sequences (for example file `seq_motif.fasta`)
    - number of DNA sequences
    - length of each DNA sequence
    - motif length
- Output:
    - an array of starting positions for each DNA sequence with the best score for the consensus string
- Modify function `Score()` to calculate a partial consensus score for first `i` rows of `DNA`.

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