Identify glk-ci-patterns by short gappped k-mers

We create the whole procedure in order to identify those glk-ci-patterns that appear multiple times in the input sequences (especially when g_0 and k_0 are relatively large).

Heuristic ideas:

1. For a glk-ci-pattern, its g_0 gaps and k_0 characters are divided into two parts, the head region and the tail region, which are both gapped k-mers.

2. If a glk-ci-pattern is commonly observed in a sequence, its head region and tail region should also be commonly observed. On the other hand, if we can locate the positions where the head region or the tail region appears, we are able to find the corresponding glk-ci-pattern.

3. $\left\lceil \frac{g_0+k_0}{2} \right\rceil$ characters in a glk-ci-pattern will always appear in either the head region or the tail region.

Procedure:

1. For a sequence s and the given parameters g_0, l_0, k_0, take $r = \left\lceil \frac{g_0+k_0}{2} \right\rceil$

2. For each gapped k-mer with length r, calculate the number of times it appears in s.

3. Set a boundary that separates the gapped k-mers into two groups: "Commonly Observed "and "Seldom Observed".

4. For each gapped k-mer in the "Commonly Observed "group, find its locations. Every time we locate a specific gapped k-mer in the sequence, we extract the $\left\lfloor \frac{g_0+k_0}{2} \right\rfloor + l_0$ characters that are in front of it and execute step 5. We also extract the $\left\lfloor \frac{g_0+k_0}{2} \right\rfloor + l_0$ characters that are behind it and execute step 5.

5. For the characters obtained, identify the $g_2l_2k_2$-patterns they are able to generate. Here $g_2 = g_0 - g_1$, $k_2 = k_0 - k_1$, $l_2 = l_0$, where g_1 is the number of gaps and k_1 is the number of characters in the gapped k-mer we have found.

6. Find the $g_2l_2k_2$-patterns that appear multiple times and link them with the gapped k-mer to get the desired result.

Example:

Suppose we set $g_0 = 3$, $l_0 = 4$, $k_0 = 5$, and the glk-ci-pattern **ANCGN−−−−−TNC** should be found to appear multiple times in a sequence s. In order to do that, we perform the following procedure.

1. $r = \left\lceil \frac{g_0+k_0}{2} \right\rceil = 4$.

2. For each gapped k-mer with length 4, we calculate the number of times it appears in s. It is likely that we obtain **ANCG** and **NCGN** are the two gapped k-mers that appear most frequently.

3. For the k-mer **ANCG**, we find its locations and then extract the $\left\lfloor \frac{g_0+k_0}{2} \right\rfloor + l_0 = 8$ characters that are in front of it and $\left\lfloor \frac{g_0+k_0}{2} \right\rfloor + l_0$ characters that are behind it

4. When we are processing the characters that are behind **ANCG**, $g_1 = g_0 - g^{'} = 2$, $k_1 = k_0 - k^{'} = 2$, $l_1 = l_0 = 4$. Hence, we use the 8 characters to generate the 2-2-4-patterns and identify that **N−−−−TNC** are generated a lot of times.

5. Link **N−−−−TNC** with **ANCG** to get **ANCGN−−−−TNC**.

Note that in this process, we will obtain a lot of information that is not related to the desired pattern **ANCGN−−−−TNC**, for example, patterns that starts or ends with **NCGN**, also patterns that ends with **ANCG**. However, this is expected since there should be multiple patterns in a sequence instead of just one. Also, its easy for two glk-ci-patterns to share some characters, especially when the start of one pattern is exactly the same as the end of another.

Methods

Some difficult problems we need to solve in the preceding procedure is:

1. How do we calculate the frequency of every gapped k-mer in the sequence?

2. How do we record the frequencies so that it would be easy for us to know what the gapped k-mer is once we find its frequency is high?

3. How to find the $g_2l_2k_2$-patterns that appear multiple times before/after the gapped k-mer?

For the first problem and the second problem, we come up with a method that can count and store the frequency of each gapped k-mer easily. In order to do so, we first need to count and store the frequency of each k-mer in the sequence, and then calculate the frequency of gapped k-mers following some rules (which are mentioned in **Explanations for Algorithm 2**).

The way we count and store the frequency of each k-mer is to number every k-mer lexicographically. Intuitively, if $r = 5$, we need **AAAAA**=1, **AAAAC**=2, **AAAAG**=3, **AAAAT**=4, **AAACA**=5...Therefore, it's natural for us to change each nucleotide into a quaternary number and then transfer the whole number into its decimal form. The underlying rule should be:

$$A \to 0, C \to 1, G \to 2, T \to 3$$

For example, **AAAAG** will be changed into 00002, and this number is 2 in decimal form. **ATAAC** will be changed into 03001, and it is 193 in decimal form.

After we get the number of times each k-mer appears in the sequence, we will be able to use some loops to calculate the frequency of each gapped k-mer. However, we still need to set up some rules so that the frequency of gapped k-mers will be calculated and stored in order. The following rules are simple and useful at the same time.

Rule 1. For two gapped k-mers with a length of r , the one with less gaps has higher priority than the other.(Higher priority means that its frequency is calculated and stored before the other)

For example,

$$\begin{bmatrix} A & & N & & G & & N \\ C & & G & & T & & T \\ G & > & T & > & N & > & N \\ C & & C & & N & & N \end{bmatrix}$$

Rule 2. If two gapped k-mers have the same number of gaps, then we compare their positions of gaps in the gapped k-mers from the first gap to the last one. The one with larger position index will have lower priority.

For example,

$$\begin{bmatrix} A & & C & & G & & N \\ C & & N & & N & & T \\ N & > & T & > & N & > & N \\ N & & N & & T & & A \end{bmatrix}$$

Rule 3. If two gapped k-mers have the same number (g) of gaps, and the gaps are in the same positions, then we compare the remaining $r - g$ nucleotides lexicographically from the first nucleotide to the last one.

For example,

$$\begin{bmatrix} A & & A & & G & & T \\ C & & T & & C & & T \\ N & > & N & > & N & > & N \\ N & & N & & N & & N \end{bmatrix}$$

For example, when we want to calculate the frequency of **AANCN**, we need to add up the frequency of **AAACA, AAACC, ..., AATCT**. Therefore, we need a nested loop which states that:

Loop 1: Calculate the frequency of AANCN
For **N=A** to **T**
  For **M=A** to **T** ▷ This **M** is used to avoid confusion
    Add up the frequency of **AANCM**
  end For
end For

In order to do so, we first need to calculate the number corresponding to the first k-mer that matches **AANCN**, which is **AAACA** and its number is 4 in decimal form. Then, the numbers corresponding to **AAACC, AAACG, AAACT** are $4+1=5, 5+1=6$, and $6+1=7$. The next k-mer we want is **AACCA**, of which the corresponding number is $4+4^2=20$. For **AAGCA** and **AATCA**, the numbers are $20+4^2=36$ and $36+4^2=52$. We can see that in the inner loop, there is a small number (1) that we add 4 times (The loop ends after we add it the last time), we call this number the "*step*" in the inner loop. There is a large number (16) which we add in order to proceed to the next inner loop, we call this number the "*distance*".

Therefore, the loop we generated just now will be rewritten into:

Loop 2: Calculate the frequency of AANCN
Define $step := 1$
Define $distance := 16$
Define $t := 5$
For $i = 1$ to 4
  For $j = 1$ to 4
    Add up the frequency of t
    $t = t + step$
  end For
  $t = t - 4 * step + distance$
end For

It should be observed that this loop can also be applied when we want to calculate the frequency of **AANGN**. The only difference is that the third nucleotide is replaced by **G**, which means that only difference is that initial value of t is different. Hence, we should generate the loop in such a way that it can be utilized to calculate the frequency of **AANAN, AANCN, ..., ACNAN, ACNCN, ..., TTNTN**. According to the priority rules, we should construct loops as follows:

Loop 3: Calculate the frequency of different gapped k-mers
For **X=A** to **T**
  For **Y=A** to **T** ▷ **Y** is used to avoid confusion
    For **Z=A** to **T** ▷ **Z** is used to avoid confusion
      Calculate the frequency of **XYNZN** ▷ Utilize **Loop 2**
    end For
    New space to store the frequency of next gapped k-mer
  end For
  New space to store the frequency of next gapped k-mer
end For
New space to store the frequency of next gapped k-mer

Note that **AAAAA** is the first k-mer which matches the pattern **AANAN**, and the number corresponding to it is 0. **AAACA** is the first k-mer which matches the pattern **AANCN**, and its number is $0+4^1=4$. Similarly, we have $4+4^1=8$ for **AAAGA** and $8+4^1=12$ for **AAATA**. The next gapped k-mer whose frequency we want to calculate is **ACNAN**. The first k-mer that matches it is **ACAAA** and the number corresponding to it is $0+4^3=64$. Also, the number is $64+4^3=128$ for **AGAAA** and $128+4^3=196$ for **ATAAA**. Hence, if we define *step* and *distance* in a similar way as before, the loop we constructed will turn into:

Loop 4
Define $t := 1$
For $i = 1$ to 4
  For $j = 1$ to 4
    For $k = 1$ to 4
      For $l = 1$ to 4 ▷Utilize **Loop 2**
        For $m = 1$ to 4
          Add up the frequency of t
          $t = t + 4^0$ ▷ $step = 1$
        end For
        $t = t - 4 * 4^0 + 4^2$ ▷ $step = 1, distance = 16$
      end For ▷ This is the end of **Loop 2**
      $t = t - 4 * 4^2 + 4^1$ ▷ $step = 16, distance = 4$
      New space to store the frequency of next gapped k-mer
    end For
    $t = t - 4 * 4^1 + 4^3$ ▷ $step = 4, distance = 64$
    New space to store the frequency of next gapped k-mer
  end For
  $t = t - 4 * 4^3 + 4^4$ ▷ $step = 64, distance = 256$
  New space to store the frequency of next gapped k-mer
end For
New space to store the frequency of next gapped k-mer
▷ Here, we can let $step = 256, distance = 0$ for consistency

Explanations for Algorithms

In **Algorithm 1**, we transfer each short k-mer into a number in order to record its frequency in the sequence lexicographically. We use a queue q of length r to represent the k-mer and "translate" each letter into a quaternary number.

$$A \to 0, C \to 1, G \to 2, T \to 3$$

Every time we push one number into the queue, we "translate" the queue back to its corresponding decimal number and record it in the array num. In this process, we utilize the array $Coef$ and calculate the $index$ of the k-mer in num by:

$$index = (\sum_{k=1}^{r} q[k] * Coef[k]) + 1$$

For example, a short k-mer **ATCG** found in the sequence will be firstly translated into {0312} in the queue. Then, {0312} will be changed into its decimal form by

$$0 * 4^3 + 3 * 4^2 + 1 * 4^1 + 2 * 4^0 = 54$$

Hence,

$$num[54 + 1] = num[54 + 1] + 1$$

It's worth noticing that the array num is used to record the frequency of all the gapped k-mers. Therefore its size should be

$$size(num) = \begin{cases} 4^r + 4^{r-LB}\binom{r}{LB} + 4^{r-LB-1}\binom{r}{LB+1} + ... + 4^{r-UB}\binom{r}{UB} = 4^r + \sum_{k=LB}^{UB} 4^{r-k} * \binom{r}{k} & if(LB \neq 0) \\ 4^r + 4^{r-1}\binom{r}{1} + 4^{r-2}\binom{r}{2} + ... + 4^{r-UB}\binom{r}{UB} = \sum_{k=0}^{UB} 4^{r-k} * \binom{r}{k} & if(LB == 0) \end{cases}$$

where UB is the maximum number of gaps in the k-mer and LB is the minimum number of gaps. Also, before we apply the algorithms, all entries of num should be initialized to 0.

If $UB == r, LB == 0$ or 1, then

$$size(num) = \sum_{k=0}^{r} 4^{r-k}\binom{r}{k} = (1+4)^r = 5^r$$

In **Algorithm 2**, we calculate the number of times each gapped k-mer appears in the sequence by a series of functions. The priority rules that we set up in the **Method** section should be utilized now.

Therefore, in the calculation process, we need to parametrize three things in order before we do direct calculation.

1. The number of gaps.
2. The positions of the gaps in the gapped k-mer.
3. The nucleotides.

Functions in Algorithm 2

1. The function GAPBOUND calculates the upper bound (*UpperBound*) and the lower bound (*LowerBound*) of the number of gaps in the gapped k-mer. The *UpperBound* and the *LowerBound* are utilized in the next function GAPPEDKMER. When the number of gaps (g_0) is larger than or equal to the length of the k-mer (r), then the *UpperBound* should be r, and the *LowerBound* will be the number that makes the other half of the *glk*-pattern become $g + k - r$ gaps. Otherwise, the *UpperBound* should be the total number of gaps (g_0) and the *LowerBound* would be 0 because it's impossible to have $g_0 < r$ and $g_0 > g_0 + k_0 - r$ at the same time.

Proof. Suppose it were true that

$$\begin{cases} g_0 < r \\ g_0 > g_0 + k_0 - r \end{cases}$$

Then

$$2 * r > g_0 + k_0$$

Since

$$r = \left\lceil \frac{g_0 + k_0}{2} \right\rceil$$

Therefore

$$g_0 + k_0 = 2 * r - 1$$

Hence

$$r = g_0 + k_0 - r + 1$$

However g_0 is an integer between r and $g_0 + k_0 - r$, which is impossible

2. In the function GAPPEDKMER , *new* is the array index in *num* for recording the frequency of gapped k-mers. It is initialized to be $1 + 4^r$ since there are 4^r k-mers originally. Then, for each possible number of gaps , we use the function INDEXto generate the positions of gaps and nucleotides and store them in *gapindex* and *nucindex* respectively. Afterwards, the function NUCLOOP starts generating the recursive loops. When the recursive loops finish adding up all the numbers, either the g gaps change their positions, or we start over with a new value of g.

3. The functions NUCLOOP and GAPLOOP generate recursive loops using *nucindex* and *gapindex*. There are a lot of variables in the two functions, but the key ones are t, *new*, *nuc/gap*, *step* and *distance*. The others are constant variables and can be passed by reference.

t is the position variable that is used to sum up the frequency of k-mers.
new is the array index in *num* for new gapped k-mers
nuc/gap is the remaining number of nucleotides/gaps
step is a number that t will add four times in one loop in order to locate the next k-mer
distance is a number that t will add only one time in one loop in order to locate the next k-mer

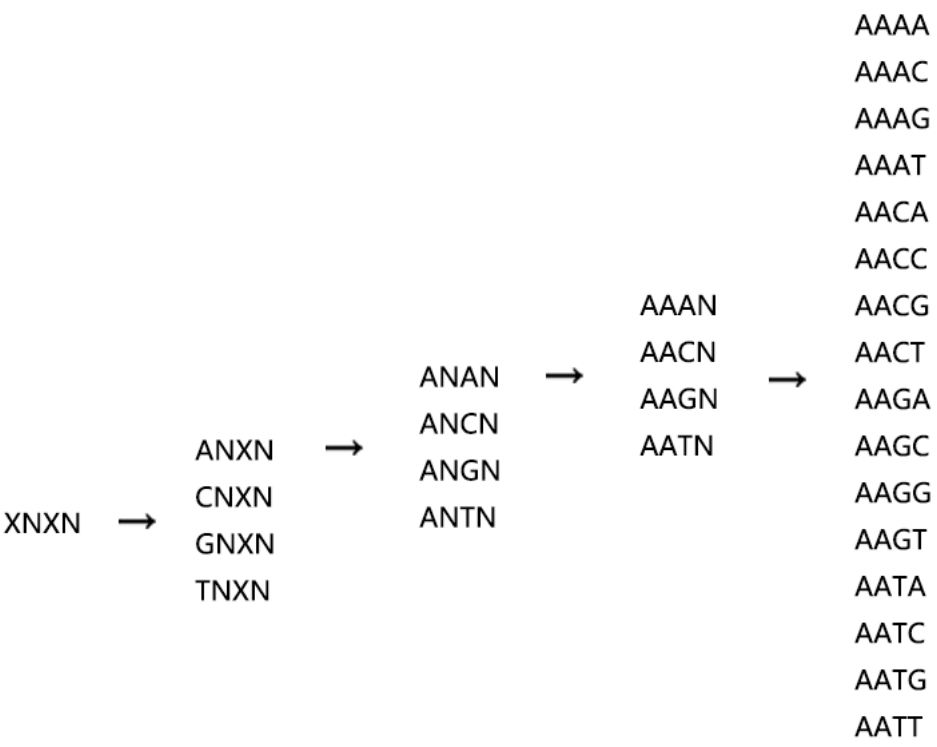

Figure 1: gapped k-mers

For example, if we are going to calculate the number of gapped k-mers in the form **XNXN** where **X**'s are nucleotides, then we need to consider $4^2 = 16$ different gapped k-mers. Therefore, the following procedure is executed.

Define t:=1
for $i = 1$ to 4 **do**
  for $j = 1$ to 4 **do**
    for $k = 1$ to 4 **do**
      for $l = 1$ to 4 **do**
        $num[new] = num[new] + num[t]$
        $t = t + 4^{1-1}$ ▷ $step = 4^0$
      end for
      $t = t - 4 * 4^{1-1} + 4^{3-1}$ ▷ $step = 4^0, distance = 4^2$
    end for
    $t = t - 4 * 4^{3-1} + 4^{2-1}$ ▷ $step = 4^2, distance = 4^1$
    $new = new + 1$
  end for
  $t = t - 4 * 4^{2-1} + 4^{4-1}$ ▷ $step = 4^1, distance = 4^3$
  $new = new + 1$
end for
$t = t - 4 * 4^{4-1} + 0$ ▷ $step = 4^3, distance = 0$
$new = new + 1$ ▷ This $new++$ actually prepares for the next loop with different gaps positions

4. The function INDEX generates the *gapindex* array by consecutively looking for an index (i) whose value can be added by one, and then set back the indices before i to their original positions.

For example,

If the *gapindex* array holds 4 numbers {2345} and we calls INDEX

Then the function INDEX tries to find a number which can be added by 1 and fails.

Therefore $i == 4$

Hence, we add 5 by 1 and get 6.

Afterwards, the three numbers before 6 are reset to be {123}.

Eventually, *gapindex* holds {1236} and a new loop starts.

In **Algorithm 3**, we use the *num* array to find the gapped k-mers that belong to the "Commonly Observed" group. After we know the upper bound(UB) and the lower bound (LB) of gaps in the sequence, we are able to find an g_x, such that

$$LB \leq g_x \leq UB \tag{1}$$

$$\begin{cases} 4^r + \sum_{k=LB}^{g_x-1} 4^{r-k} * \binom{r}{k} < index \leq 4^r + \sum_{k=LB}^{g_x} 4^{r-k} * \binom{r}{k} & if(LB \neq 0) \\ \sum_{k=0}^{g_x-1} 4^{r-k} * \binom{r}{k} < index \leq \sum_{k=0}^{g_x} 4^{r-k} * \binom{r}{k} & if(LB == 0) \end{cases} \tag{2}$$

where *index* is the array index of the gapped k-mer in *num*. Then, we can find an c_x such that

$$(c_x - 1) * 4^{r-g_x} < index - LHS \leq c_x * 4^{4-g_x}$$

where LHS is the left hand side in the inequality system (2).

Therefore, we will be able to know the number of gaps and their positions by using g_x and c_x, and afterwards it will be easy to know what the nucleotides are.

For example,

Suppose that $g_0 = 4, k_0 = 3$, then we have

$$r = \left\lceil \frac{g_0 + k_0}{2} \right\rceil = 4$$

Therefore

$$LB = 1, UB = 4$$

If we are going to find the gapped k-mer whose *index* is 581, then we notice that

$$4^4 + \sum_{k=1}^{1} 4^{r-k} * \binom{r}{k} = 512 < 581 < 608 = 4^4 + \sum_{k=1}^{2} 4^{r-k} * \binom{r}{k}$$

Hence,

$$g_x = 2$$

Since,

$$4 * 4^2 < 581 - 512 < 5 * 4^2$$

Therefore,

$$c_x = 5$$

In conclusion, we know that there are 2 gaps and they are in positions {2,4}, the gapped k-mer is in the form **NXNX**

Afterwards, we are able to know the 581 th gapped k-mer should be **NTNG**

Algorithm 1 Calculation of short k-mers in the sequence

```
1: function KMERCOUNT (k_0, g_0, s, num) ▷ k_0, g_0 are respectively the number of nucleotides and the number of gaps.
                                          ▷ s is the sequence. num is the array that stores the frequency of all gapped k-mers
2:     Define r := ⌈(g_0 + k_0)/2⌉
3:     Define q to be a queue of size r                    ▷ q is a queue used to store the current k-mer
4:     Define Coef to be an array of size r
5:     for j = 1 to r do
6:         Coef[j] = 4^(j-1)
7:     end for
8:     Define i := 1                                       ▷ The i is used in both loops, so it needs to be defined outside
9:     for i to r do
10:        q.push(TRANSLATE(s[i])                          ▷Push the first r elements into the queue
11:    end for
12:    do{
13:        for k = 1 to r do
14:            index = q[k] * Coef[k] + 1                  ▷ + 1 because array index starts at 1
15:        end for
16:        num[index]++
17:        q.pop()
18:        q.push(TRANSLATE(s[i]))
19:        i = i + 1
20:    }while i <= n;
21:end function
22:function TRANSLATE (char)                               ▷The function translates nucleotides into numbers.
23:    Define ret to be the return value
24:    switch (char)                                       ▷For A,C,G,T, we have a corresponding number
25:        case (A):
26:            ret = 0
27:            break
28:        case (C):
29:            ret = 1
30:            break
31:        case (G):
32:            ret = 2
33:            break
34:        case (T):
35:            ret = 3
36:            break
37:    Return ret
38:end function
```

Algorithm 2 Calculation of short gapped k-mers in the sequence

function GAPBOUND (k_0, g_0, num) ▷ The function calculates the bounds of gaps and calls GAPPEDKMER
  Define $r := \left\lceil \frac{g_0+k_0}{2} \right\rceil$
  if $g_0 >= r$ **then**
    Define $UpperBound := r$ ▷ When $g_0 == r, k_0 == r$
    Define $LowerBound := r - k_0$
  else
    Define $UpperBound := g_0$
    Define $LowerBound := 0$ ▷ When $g_0 < r$,it is impossible to have $g_0 >= \left\lfloor \frac{g_0+k_0}{2} \right\rfloor$
  end if
  GAPPEDKMER $(r, num, UpperBound, LowerBound)$
end function
function GAPPEDKMER (r, num, UB, LB)
  if $UB == 0$ **then**
    Return
  else
    Define $new := 1 + 4^r$
    Define $min := minimum\{1, LB\}$ ▷ min is the smaller one between 1 and LB
    for $g = min$ to UB **do** ▷ For every possible number of gaps
    Define $gapindex$ to be an array of size g ▷ The array $gapindex$ should be passed by reference
      for $i = 1$ to g **do** ▷ We initialize $gapindex$ to be $1, 2, 3, ...g$
        Initialize $gapindex[i] := i$
      end for
      for $i = 1$ to $\binom{r}{g}$ **do**
        INDEX $(g, gapindex)$ ▷ The function INDEX changes the $gapindex$ array every time it is called
        Define $nucindex$ to be the array that contains the elements in $\{1, 2, 3, ...r\} \backslash \{gapindex\}$
        NUCLOOP $(1, new, r - g, 4^{nucindex[r-g]-1}, 0, g, nucindex, gapindex)$
      end for
    end for
  end if
end function
function NUCLOOP $(t, new, nuc, step, distance, g, nucindex, gapindex)$
▷ $g, nucindex, gapindex$ are passed by reference
  if $nuc == 0$ **then**
    GAPLOOP $(t, new, g, 4^{gapindex[g]-1}, step, gapindex)$
  else
    for $i = 1$ to 4 **do**
      NUCLOOP $(t, new, nuc - 1, 4^{nucindex[nuc-1]-1}, step, g, nucindex, gapindex)$
    end for
    $t = t + distance - 4 * step$
  end if

```
    new = new + 1
end function
function GAPLOOP (t, new, gap, step, distance, gapindex)        ▷ gapindex is passed by reference
    if gap == 1 then
        for i = 1 to 4 do
            num[new] += num[t]
            t = t + step
        end for
        t = t + distance − 4 ∗ step                              ▷ t goes back and prepare for the next loop
    else
        for i = 1 to 4 do
            GAPLOOP (t, new, gap − 1, 4^(gapindex[gap−1]−1), step, gapindex)
                                  ▷ The new step is 4^(gapindex[gap−1]−1) and the new distance is the previous step
        end for
        t = t + distance − 4 ∗ step                              ▷ t goes back and prepare for the next loop
    end if
end function
function INDEX (g, gapindex)                                    ▷ gapindex is passed by reference
    for i = 1 to g − 1 do
        if  gapindex[i + 1] − gapindex[i] == 1 then
            i = i + 1
        else
            gapindex[i] = gapindex[i] + 1
            if i ≠ 1 then
                for j = 1 to i − 1 do
                    gapindex[j] = j
                end for
            end if
            break
        end if
    end for
    if i == g then
        gapindex[g] = gapindex[g] + 1
        for j = 1 to g − 1 do
            gapindex[j] = j
        end for
    end if
end function
```

Discussion

Is it possible to calculate the frequency of $(g+1)$-gap gapped k-mers from g-gap gapped k-mers directly? This process will greatly improve the calculation efficiency and save time. It was initially our idea but we failed because there were too many repeated patterns and cannot be easily summarized into one entry.

To be precise, when we utilize the number of g-gap gapped k-mers, we face the difficult problem that there will be g different patterns with different *step*, but their value need to be combined into 1 entry. A possible solution is to analyze the gapped k-mer itself, but how to get the positions of gaps in the original gapped k-mers remains unsolved.