Use of Suffix Arrays in Genome Assembly

ABHINANDHU A
Dept of Computer Science Engineering (AI)
Amrita School of Engineering ,
Amrita Vishwa Vidyapeetham,
Amritapuri ,
Kollam,Kerala,India
amenu4aie20002@am.students.amrita.edu

HARIPRASAD S
Dept of Computer Science Engineering (AI)
Amrita School of Engineering ,
Amrita Vishwa Vidyapeetham,
Amritapuri ,
Kollam,Kerala,India
amenu4aie20035@am.students.amrita.edu

M MAHADEV
Dept of Computer Science Engineering (AI)
Amrita School of Engineering ,
Amrita Vishwa Vidyapeetham,
Amritapuri ,
Kollam,Kerala,India
amenu4aie20045@am.students.amrita.edu

MARASANI JAYASURYA
Dept of Computer Science Engineering (AI)
Amrita School of Engineering ,
Amrita Vishwa Vidyapeetham,
Amritapuri ,
Kollam,Kerala,India
amenu4aie20048@am.students.amrita.edu

***Abstract*—In this project we are mainly concerned with the uses of suffix array where it could be useful in Genome assembly. Some of the major application of suffix array are mentioned and they could pave way for its use in genome assembly. The applications mentioned are pattern searching, finding the longest repeated substring, finding the longest common substring, finding the longest palindrome in a string. Both the implementation as well as the algorithm is done and a study is made on these algorithms. Suffix array with its algirithm and importance is enfacised to its maximum effect. Some of the other uses of Suffix array are full-text indices, data compression algorithms, and the field of bibliometrics.**

***Index Terms*—Suffix Array , LCP , Inverse suffix array,Long arm gapped palindrome ,Length constrained gapped palindrome**

I. Introduction

Genome assembly refers to the process of putting nucleotide sequence into the correct order.Genome assembly is the computational process of deciphering the sequence composition of the genetic material (DNA) within the cell of an organism, using numerous short sequences called reads derived from different portions of the target DNA as input.

The term genome is a collective reference to all the DNA molecules in the cell of an organism. Sequencing generally refers to the experimental (wetlab) process of determining the sequence composition of biomolecules such as DNA, RNA, and protein. Assembly is required, because sequence read lengths are much shorter than most genomes or even most genes. Although bacterial genomes are much smaller, genes are not necessarily in the same location and multiple copies of the same gene may appear in different locations on the genome.

II. Literature Review

1.Paper 1:

"Suffix Arrays and Genome Assembly"
Peter F. StadlerGuilherme P. Telles, Cristina D. A. Ciferri

This paper contains a detailed explanation of suffix array and its application in genome assembly. The main advantage of the proposed algorithm is that no string comparison is done. Then, beyond the characters comparison avoiding, once constructed the generalized enhanced suffix array construct for R, all reads in R can be removed from the main memory

2.Paper 2:

"Searching Gapped Palindromes Using Inverted Suffix Array"
Shivika Gupta; Rajesh Prasad; Sunita Yadav

Gapped palindrome is an interesting version of the palindrome which is defined as the one having a space between left and right palindromic arms of the string. A palindrome is a string that reads the same forward and backward. In this paper,they have developed an efficient algorithms to detect two different classes of gapped palindromes: long armed and length constrained in a biological sequences by using inverted suffix array. The algorithms perform the computation in O(n) time. Alsod determined palindromic weights (number and size of gapped palindromes) in the input biological string.

In this work, the steps followed are Reverse Complement, Concatenation of # and $, Construct Inverted Suffix Array, Computation of Longest Common Prefix (LCP) Length , Verify constrains. In this paper, it has presented constant time algorithm to detect long armed version of gapped palindromes in DNA sequences by using inverted suffix array and longest common prefix techniques.

III. WORK AND METHODOLOGY

A. *Suffix Array*

A suffix array is an array of all suffixes of a given string in a sorted way. It is a data structure used in searching algorithms,, full text indices, data compression algorithms, and the field of bibliometrics.Suffix arrays are very commonly used for representing genomes in search applications.

A suffix array can be developed by doing a DFS traversal of the suffix tree. Indeed Suffix tree and suffix array both can be developed from one another in linear time. A straightforward strategy to develop a suffix array is to make a variety of all postfixes ,then store it in an array and afterward sort the array..Following is the algorithm of this method.

```
Construct_SuffixArray(Input_str,len_str):
    for i in range(len_str){
        permutation =  stores each suffixes from the input in each iteration
        Suffix_Dict[i]=permutation
         Suffix_DictReverse[permutation]=i
        }
    orderedList = sorted(Suffix_Dict.values)
   for i in orderedList{
         Suffix_array.append(Suffix_DictReverse[i])
        }
    return Suffix_array
```

B. *Applications of Suffix Array*

1.Pattern Searching

Pattern matching is applied as a significant activity in various areas of the computational pipelines. Pattern matching helps to find the locations of certain DNA substrings in a DNA sequence or biological database.

The objective of example matching is to observe every one of the places of a motif M of size m in a text T of size n. To search a given pattern in a text, we first preprocess the text and constructs a suffix array of the text. Since the suffixes are sorted during suffix array construction ,Binary Search can be used to search the subsequence.The following is the algorithm for pattern matching using suffix array.

Algorithm

```
Search_pattern(pattern ,Input_str,Suffix_array):{
  l=0 , r = len(Sufiix_array)
    while l < r:
        #setting the middle for binary search
        mid_value = (l+r) //2
        if  (substring lesrr than  pattern):
            l = mid_value+1 # sets left pointer as mid_val + 1
        else:
            r = mid_value  # sets right pointer as mid_val

    def match_at(i):{
   if(match_at[Suffix_array[l] == false)
        No match is found
    first = l
    while first greater than 0 and match_at(Suffix_array[first - 1]):
        first -= 1
    last = l
    while match_at(Suffix_array[last]):
        last += 1
    return Suffix_array[first:last]
 }
```

2.Finding the longest repeated substring

The basic idea is to find the longest repeated substring in one string. For example, the longest repeated substring in “ABRACADABRA” is “ABRA.”

The brute force method requires O(n2) time and lot’s of space. By using suffix array and LCP(Longest common prefix) array improvement can be made in the naive approach.

Algorithm

```
longest_repeated_substring(input_string):
        Suffix_array, suffix_dict = Construct_SuffixArray(Input_str, len_str)
        lcp = LCP(Suffix_array, Input_str)
        idx = max(lcp)
        idx_suffix = Suffix_array[idx]
        res = suffix_dict[idx_suffix]
        return result[0:max(lcp)]
```

3.Finding the longest common substring

The longest common substring problem is to find the longest string that is a substring of two or more strings.This problem can have multiple solutions and algorithms. Their applications involve data deduplication and plagiarism detection.

A longest common substring of a collection of strings is a common substring (i.e., a shared substring) of maximum

length. For example, ”CG” is a common substring of ”ACGTACGT” and ”AACCGGTATA”, but it is not as long as possible; in this case, ”GTA” is a longest common substring of ”ACGTACGT” and ”AACCGTATA”. The longest common substring of a collection is not necessarily unique; for a simple example, note that ”AA” and ”CC” are both longest common substrings of ”AACC” and ”CCAA”.

Algorithm

```
Def longest_common_substring(text):
        sa,rsa  = suffix array,  rank of suffix array
        lcp= longest common prefix
        max_len , result = max value of lcp, {}
        for i  from 1 to len(text):
                if lcp[i]       == maxlen:
                        j1,  j1 = sa[i-1], sa[1]
                        h  = lcp[i]
                assert text[j1:j1 + h] == text[j2:j2 + h]
                eqauting substring to the asserted one
                if substring not in result:
                        result[substring]=[j1]
                appending j2  to that part of result
        return dictionary (sorted v for v in the result dictionary)
```

4.Finding Longest Palindromic Substring using suffix array

Given a string, for finding longest palindromic substring steps that include are:
Step 1: Enter the input of string S.
Step 2: Reverse the string s to get S’
Step 3: Concatenate S + # + S’ to get String Str ,where # is an alphabet not present in original String.
Step 4: Construct the suffix array of Str.
Step 5: Find the Longest Common Prefixes using adjacent suffixes from suffix array
Step 6: Find the length of longest palindrome using the following conditions,

Algorithm

```
Let the length of the Longest Palindrome ,longestlength:=0 (Initially)
    Let Position:=0.
    for(int i=1;i<Len;++i)
    {
       //Note that Len=Length of Original String +"#"+ Reverse String
       if((LCP[i]>longestlength))
       {
          //Note Actual Len=Length of original Input string .
          if((suffixArray[i-1]<actuallen && suffixArray[i]>actuallen)||
             (suffixArray[i]<actuallen && suffixArray[i-1]>actuallen))
          {
              //print :Calculating Longest Prefixes b/w suffixArray[i-1] AND  suffixArray[i]


             longestlength=LCP[i];
            //print The Longest Prefix b/w them  is ..
            //print The Length is :longestlength:=LCP[i];
              Position=suffixArray[i];
          }
       }
    }
    So the length of Longest Palindrome :=longestlength;
    and the longest palindrome is:=Str[position,position+longestlength-1];
```

5.Palindrome Pattern matching using inverted suffix array

Palindrome string can be defined as if a string is the same even when reversed. In DNA sequence palindromic DNA structure has a major impact on deciding parameters of gene activities. A palindrome in DNA is a sequence of nucleotide bases that reads same as its reverse complement. Eg: Lets take GGATCC is a DNA palindrome where CCTAGG is a complimentary string.A gapped palindrome in a biological sequence is defined as a sequences vuv’ where v is a non-empty sequence of bases, u (spacer) is a sequence of bases and v’ is the reversed complement of v, the reverse of v with all bases replaced by their potential base pair partners.There are two natural classes of gapped palindromes: Long Armed and Length Constrained. A gapped palindrome vuv’ in biological sequence is called long armed if $|u|$less than or equal to $|v|$, i.e. length of spacer should be less than or equal to the length of palindrome arm where v is a non-empty sequence of bases, u (spacer) is a sequence of bases and v’ is the reversed complement of v. A gapped palindrome vuv’ in biological sequence is called length constrained if MinGap less than or equal to $|u|$than or equal to MaxGap and MinArm $|v|$ such that MinArm, MinGap and MaxGap are some predefined constants; where v is a non-empty sequence of bases, u (spacer) is a sequence of bases and vT is the reversed complement of v, the reverse of v with all bases replaced by their potential base pair partners (C replaced by G, G by C, A by T and T by A); MinArm, MinGap and MaxGap are predefined constants.

Algorithm

A.ALGORITHM TO DETECT LONG ARMED GAPPED

PALINDROMES

Step 1: Input the DNA sequence
Step 2: Find the Reverse Complement of the DNA Sequence
Step 3: Concatenate # DNA sequence and $ to Reverse Complement.
Step 4: Find the inverted Suffix Array for both the strings.
Step 5: Find the Least Common Prefix for the inverted suffix array
Step 6: Find the Maximum LCP Length and note down its corresponding indexes.
Step 7: Verify Palindrome Arm constraints
Step 8: Verify Spacer length Constraints.

B.ALGORITHM TO DETECT LENGTH CONSTRAINED GAPPED PALINDROMES

Step 1: Input the DNA sequence
Step 2: Find the Reverse Complement of the DNA Sequence
Step 3: Concatenate # DNA sequence and $ to Reverse Complement.
Step 4: Find the inverted Suffix Array for both the strings.
Step 5: Find the Least Common Prefix for the inverted suffix array
Step 6: Find the LCP length which is equal to MinArm value.
Step 7: Verify Length Constraints
Step 8: Verify Palindrome Arm Constraints.

IV. CONCLUSION

In this paper, an emphasis is given on the use of suffix arrays in various streams of bioinformatics. Various uses with the corresponding algorithm are mentioned.The applications mentioned are pattern searching, finding the longest repeated substring, finding the longest common substring, finding the longest palindrome in a string, Palindrome Pattern matching using inverted suffix array. We have presented an algorithm to detect length constrained versions of gapped palindromes in DNA sequences by using inverted suffix arrays and longest common prefix techniques.

REFERENCES

[1].https://louisabraham.github.io/notebooks/suffix$_arrays.html$

[2].https://www.youtube.com/watch?v=53VIWj8ksyI

[3].https://www.youtube.com/watch?v=OptoHwC3D-Y

[4].S. Gupta, R. Prasad and S. Yadav, "Searching Gapped Palindromes Using Inverted Suffix Array," 2015 IEEE International Conference on Computational Intelligence Communication Technology, 2015, pp. 186-191, doi: 10.1109/CICT.2015.72.

[5].https://stackoverflow.com/questions/7043778/longest-palindrome-in-a-string-using-suffix-tree

[6].S. Gupta, R. Prasad and S. Yadav, "Searching Gapped Palindromes in DNA Sequences using Dynamic Suffix Array," Indian Journal of Science and Technology, vol. 8, no. 23, pp. 1-9, 2015