Study, Analysis, and Implementation of Different Techniques to Find Tandem Repeats.

Sridhar Reddy Maddireddy 02151989, Meghana Madineni 91978425, Muneer Kattubadi 63053851
srimadd@ufl.edu, meghaname@ufl.edu, mkatt@ufl.edu
Bioinformatics, Department of Computer and Information Science and Engineering,
University of Florida, Gainesville, Florida.

***Abstract* — Tandem repeats are simple sequence repeats that are extremely common throughout the genomes of a wide range of species. Tandem repeats are helpful to determine the inherited traits of a gene from its parent and in determining the ancestor relationship. Tandem repeats are proven to be biologically significant as they can help in the discovery of dynamic mutations for genetic diseases. In this paper, we implement different methods like dynamic programming, suffix array induced sorting, to find exact tandem repeats and compare the performance of the methods.**

***Index Terms*: Tandem Repeats, Suffix Array Induced Sorting, Suffix Arrays.**

I. INTRODUCTION

Tandem repeats occur in DNA when a pattern of one or more nucleotides is repeated and the repetitions are directly adjacent to each other. An example would be ATCGTATCGTATCGT in which ATCGT is repeated 3 times. Various algorithms have been proposed to determine tandem repeats in sequences. In terms of Computer Science, this problem reduces to finding the longest repeating substring in each sequence.

We implemented the dynamic programming approach and suffix array – induced sorting methods to find tandem repeats. We compared the results of each method and found some pros and cons of using each of these methods. Further, we do a performance evaluation for both methods.

II. DYNAMIC PROGRAMMING METHOD

To find all tandem repeats in a sequence we perform slight modifications to the Smith-Waterman algorithm for local alignments. The time complexity of the method will be O (n*n) as we will be filling a two-dimensional matrix. We then align the sequence with itself to find tandem repeats.

A detailed description of the algorithm is present in [4]. The algorithm described in the reference gives us a solution to find tandem repeats with consideration of match/mismatch/gaps. Since we are looking for exact tandem repeats (to compare with Suffix Array-Induced Sorting), we perform some modifications to the dynamic programming approach to give exact tandem repeats. Then we do performance evaluation with Suffix Array-Induced Sorting.

The modifications we perform to the algorithm are:

1) Align the string with itself by placing the string on the top and on the left.

2) Set all diagonal elements to zero so we don't align a character by itself.

3) Since both strings are same, we only need to calculate one half of the matrix.

4) When calculating the next cell of the matrix, we assign a score of zero for mismatch or gap. For match, we just increment the score by one.

5) The scores in this matrix represent the length of the tandem repeat. We repeat the longest tandem

repeat in the sequence. If we have multiple repeats of the same length, we repeat all of them.

All the steps except step 4 are the same as described in [1]. The additional step we perform is step 4 to give the exact tandem repeat.

For example, consider a string "babc". From the below matrix constructed from dynamic programming method, "b" is the only tandem repeat.

		b	**a**	**b**	**c**
	0	0	0	0	0
b	0	0	0	1	0
a	0	0	0	0	0
b	0	0	0	0	0
c	0	0	0	0	0

A. Pseudo Code

- Let n be the length of the sequence.
- Let Score[n][n] be the scoring matrix.
- Let match_score (a, b) be a function which returns 1 if there is match and 0 if it is not a match.

```
for (i=0 to n)
{
  Score [0, i] = 0;     // Set top row to zero.
  Score [i, i] = 0;     //Set diagonal elements to be 0
}
 for (i= 1 to n)
 {
     for (j=i+1 to n)
     {
       if (match_score (charAt(i-1),charAt(j-1)) == 1)
               Score [i, j] = Score [i-1, j-1] + 1;
       else
               Score [i, j] =0;
     }
}
 match_score (a, b)
     if (a == b)    return 1
     else    return 0;
```

match_score returns 1 if both the characters are the same, it returns 0 for a mismatch or gap.

The algorithm runs in this way. It first initializes the top row and the diagonal elements to zero. Before calculating the next cell in the matrix, the algorithm checks if it is a match. In case of a match the algorithm increments the score by 1. Else the value of the cell is set to zero.

Once the matrix is filled, we find the maximum value in the matrix (length of the tandem repeat). Then we backtrack until we hit a zero to get the actual tandem repeat. This is repeated for all the matrix values that matched the criteria. This is the way the dynamic programming approach works.

III. SUFFIX ARRAY - INDUCED SORTING METHOD (SA-IS METHOD)

A substring of a string ending at the end of the string is called as suffix. Suffix array is lexicographically sorted array of indices of all possible suffixes of a string. For a string of length k, there are k suffixes possible.

For example, consider a string "*babc*". The following are the suffixes.

0 babc
1 abc
2 bc
3 c

To sort a suffix array using traditional sorting method, it needs O(klogk) time. In suffix array – induced sorting method, we construct a suffix array using almost pure induced sorting with linear time complexity O(k).

The sorted suffixes for the above example is as below.

1 abc
0 babc
2 bc
3 c

A. Concepts of Induced Sorting

Before understanding the Suffix Array – Induced Sorting, let us look at few concepts that constitute the method.

1) *S-type and L-type suffixes*

The SA-IS method divides the array of suffixes into S-type suffixes and L-type suffixes. S-type suffixes appear near the start of the suffix array and the L-type suffixes appear closer to the end of the suffix array. For example, in a string “cabbage”, “abbage” is a suffix of S-type and “ge” is a suffix of L-type.

2) *LMS Characters*

In a string, we say a character is an S-character if an S-type suffix starts at this position and similarly, we say a character is an L-character if an L-type suffix starts at that position. A Left-most S character (LMS character) is an S-character that has L-character to its immediate left.

C	A	B	B	A	G	E	
L	**S**	S	L	**S**	L	L	**S**

In the above example the characters above highlighted ‘**S**’ are LMS characters.

3) *LMS Substrings*

Substring starting at an LMS character till the next LMS character are called LMS Substrings. SA-IS algorithm works by sorting these LMS substrings using a special method. Two LMS substrings are equal if they have same length and same characters in same order. The follow example explains about comparing LMS substrings. Consider a string “rikki-tikki-tikka”.

LMS substring at offset 1 and that at offset 7 are equal as they have same content “ikki” and type “SLLL”.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
r	i	k	k	i	-	t	i	k	k	i	-	T	i	k	k	a	
L	**S**	L	L	L	**S**	L	**S**	L	L	L	S	L	S	L	L	L	S

4) *Bucket Sorting*

Bucket sorting works by distributing elements of an array into bins/buckets. Each bucket is again sorted recursively using bucket sort algorithm or any other sorting algorithm. Bucket sorting algorithm runs with a time complexity of O(n) where n is the size of the array that is to be sorted. In the SA-IS algorithm we used as many buckets as the number of characters present in the given string.

B. The SA-IS Algorithm

In SA-IS algorithm, Induced sorting produces suffix array. Induced sorting is to sort by using the information about the order of T_i to induce the order of the suffix T_{i-1}. It involves the following steps.

- *Guessing the LMS sort*
- *Induced sort of L-type suffixes*
- Induced sort of S-type suffixes
- Summarizing the suffix array
- Recursion (if necessary)

C. Pseudo Code Of SA-IS Algorithm

S	:The input string
SA	:The output suffix array of S;
t	:Boolean array of size n
S1	:Integer array of size n1
P1	:Integer array of size n1
B	:Integer array $[0 \ldots ||\sum|S|| - 1]$

```
SAIS (S, SA)

  1) Scan S to classify all the characters into L or S
     type and store this information in t.

  2) Scan t to find all LMS substrings in S into P1

  3) Induced sort all the LMS substrings using P1
     and B

  4) Name each LMS substring in S by its bucket
     index to get a new shortened string S1

  5) if each character in S1 is unique
        then
             Compute SA1 from S1;
        else
           SA-IS (S1, SA1)
```

Induce SA from SA1
return

D. Finding Tandem repeats using Suffix Array from SA-IS method

To find the tandem repeats from the suffix array we need to follow the below steps.

1) We need to build the suffixes from the suffix array.
2) Iterate through the suffixes in the order of suffix array and record the common substring for every two consecutive strings in the suffixes.
3) From the common substrings we return all those with maximum length as exact tandem repeats.

IV. DATASET

We perform our analysis on the FASTA data set. The FASTA data set contains nucleotide sequences in which nucleotides are represented by letter codes. Example: ACGTAAACGGTT is a nucleotide sequence.

A sequence in FASTA format begins with a single-line description, followed by lines of sequence data. The description line is distinguished from the sequence data by a greater-than (">") symbol at the beginning. An example sequence in FASTA format is:

>hsa:100130899 no KO assigned | (RefSeq) uncharacterized LOC100130899 (N)
caggacttctggatggatctgggagcctgccccttgggatttggcaaaa
…………………………………………………

We extract the sequence from the above format using a parser we have developed. We performed our analysis on about 1000 sequences of different lengths and recorded the results for analyzing the performance of the algorithms.

V. RESULTS

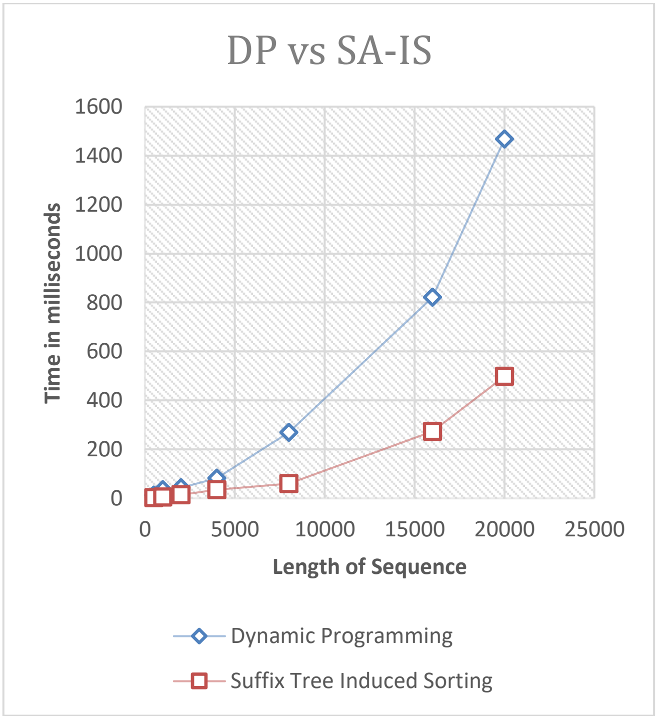
A. Implementation and Testing

We have implemented our code in Java. The dynamic programming approach and the suffix array induced sorting methods give us all the longest exact tandem repeats in the sequence. The exact tandem repeats returned by both the methods are the same. To run the code, kindly check our repository at https://github.com/sridharreddy7/TandemRepeats

For sequences of smaller length (100 – 200 characters), there is no much difference in the time taken to identify the tandem repeats. As the length of the sequences increase, there is a drastic difference in the time required to process the input sequence and identify the tandem repeats.

Bucket sorting algorithm used for induced sorting in Suffix Array - Induced Sorting method works efficiently only when the input suffixes are uniformly distributed. Because it is induced sort in nature, there may be cases in which the algorithm displays inefficiency due to poor distribution.

B. Graphical Representation of the result



We measured the time taken for each of these methods as a function of the length of the sequence and plotted a graph. From the graph presented, we can say that as the length of the input sequence increases, the time difference between Suffix Array Induced Sorting and dynamic programming methods also increases. For smaller length sequences Suffix Array Induced Sorting method and dynamic programming method runs in almost same time but as length increases Suffix Array Induced Sorting method runs much faster than dynamic programming method.

C. *Time Complexity*

The reasons for the presented graph can be clearly inferred from the time complexity analysis presented below.

	Suffix Array – Induced Sorting	**Dynamic Programming**
Construction	O(n)	O(n^2)
Tandem repeats discovery	O(n * m)	O(n^2)
Total time	O(n + n*m)	O(n^2)
Space complexity	O(n)	O(n^2)

*here **n** is the length of the sequence and **m** is the average length of the suffix.*

The time required to identify tandem repeats mainly depends on construction of the required data structure. For Dynamic programming method, a square matrix of size of the sequence is constructed and values are calculated which takes the time and space complexity of O(n^2). In SA-IS method, the time and space needed for construction is O(n). As the sequence length increases, the time taken by both the methods varies significantly.

VI. CONCLUSION

Suffix array induced sort based method to find exact tandem repeats works well and shows a significant difference in space and time taken when compared to that of dynamic programming method. SA-IS method to find tandem repeats is superior to dynamic programing method both in terms of space and time complexity.

VII. WORKLOAD DISTRIBUTION

All of us went through the algorithms/papers and got an understanding first. The implementation was split among us. Muneer worked on the Dynamic Programming approach. Sridhar and Meghana worked on the Suffix Array Induced Sorting method. Extracting tandem repeats from generated Suffix Array was done by Sridhar. Meghana researched on the dataset and built a parser for the data set. The performance analysis was done by Muneer. The final report was a combined effort with contributions from everyone.

REFERENCES

[1] Simon J. Puglisi , W. F. Smyth , Andrew H. Turpin, A taxonomy of suffix array construction algorithms, ACM Computing Surveys (CSUR), v.39 n.2, p.4-es, 2007

[2] G. Nong, S. Zhang, W. H. Chan, "Linear suffix array construction by almost pure induced-sorting", Proc. Data Compression Conf., pp. 193-202, 2009.

[3] M. Farach. Optimal Suffix Tree Construction with Large Alphabets. In Proc. 38th IEEE Symp. On Foundations of Computer Science, pages 137-143, 1997.

[4] http://dimacs.rutgers.edu/Publications/Modules/Module09-2/dimacs09-2.pdf

[5] P. Ko, S. AluruSpace efficient linear time construction of suffix arrays
J. Discrete Algorithms, 3 (2) (2005), pp. 143-156