Finding Tandem repeats in a DNA sequence

Prithvi Monangi (1685 8394), Mohan Konyala (5005 3565), Deepak Addepalli (0620 0926)

Bio Informatics

University of Florida

Gainesville

[pmonangi@ufl.edu](mailto:pmonangi@ufl.edu), [konyala@ufl.edu](mailto:konyala@ufl.edu), [deepak246@ufl.edu](mailto:deepak246@ufl.edu)

*Abstract* — Tandem repeats are segments of short DNA repeated multiple times consecutively. They are believed to play a prominent role in regulating gene expression. Tandem repeats also have a much higher rate of variation than the rest of the genome (in terms of the number of copies), and this makes them ideal markers to distinguish one individual from another. It is therefore very important to have an efficient algorithm to identify if the given pattern is a tandem repeat in a given DNA sequence. In this paper, we discuss three different paradigms to solve this problem.

*Keywords:-Tandem repeats, Suffix trees, Suffix Arrays, Dynamic Programming.*

# 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. Several protein domains also form tandem repeats within their amino acid primary structure. An example would be, In ATTCG ATTCG ATTCG, ATTCG is repeated three times. Tandem repeats describe a pattern that helps determine an individual's inherited traits. In the field of Computer Science, tandem repeats in strings (e.g., DNA sequences) can be efficiently detected using suffix trees or suffix arrays or using Dynamic Programming technique.

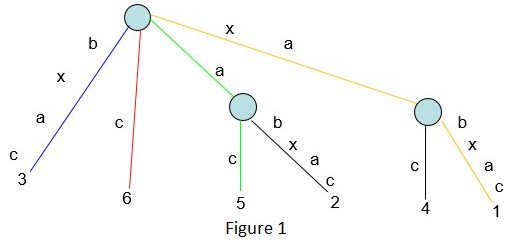
In this paper, we will discuss three different algorithms based on Suffix trees, Suffix arrays and Dynamic programming to check if the pattern is a tandem repeat in a given DNA sequence or not. For suffix tree based method, we first discuss a linear time construction algorithm for Suffix tree and then discuss about checking if the pattern is a tandem repeat or not also in linear time. Then we proceed to discuss the construction of suffix arrays and finding the tandem repeats using it. Next, we talk about a dynamic programming technique to solve the same problem. Finally, we do a comparison of all the discussed algorithms and identify the pros and cons of each of them.

# Suffix Tree Based Algorithm

Suffix Tree is very useful in numerous string processing and computational biology problems. We will discuss Ukkonen’s Suffix Tree, a linear time construction algorithm. A suffix tree **T** for m-character string S is a rooted directed tree with exactly **m** leaves numbered 1 to **m** will have the following properties.

* Root can have zero, one or more children.
* Each internal node, other than the root, has at least two children.
* Each edge is labelled with a nonempty substring of S.
* No two edges coming out of same node can have edge-labels beginning with the same character.

Concatenation of the edge-labels on the path from the root to leaf i gives the suffix of S that starts at position i, i.e. S[i…m]. For string S = xabxac with m = 6, suffix tree will look like following:



A naïve algorithm to construct a suffix tree takes O (n^2) time. We shall discuss Ukkonen’s algorithm which constructs the tree in linear time.

## High Level Description of Ukkonen’s algorithm

Ukkonen’s algorithm constructs an implicit suffix tree Ti for each prefix S [l...i] of S (Sequence of length m).It first builds T1 using 1st character, then T2 using 2nd character, then T3 using 3rd character, Tm using mth character. Implicit suffix tree Ti+1 is built on top of implicit suffix tree Ti. The true suffix tree for S is built from Tm by adding $ at the end.  
At any time, Ukkonen’s algorithm builds the suffix tree for the characters seen so far and so it has **on-line** property. Time taken is O (m).

Ukkonen’s algorithm is divided into m phases (one phase for each character in the string with length m). In phase i+1, tree Ti+1 is built from tree Ti. Each phase i+1 is further divided into i+1 extensions, one for each of the i+1 suffixes of S [1...i+1]. In extension j of phase i+1, the algorithm first finds the end of the path from the root labelled with substring S [j...i]. It then extends the substring by adding the character S (i+1) to its end if it is not there already. The pseudo code below expresses the above steps in a concise way.

* *Construct the tree T1*
* *For i from 1 to m-1 do*
* *begin {phase i+1}*
  + *For j from 1 to i+1*
    - *begin {extension j}*
    - *Find the end of the path from the root labelled S[j..i] in the current tree.*
    - *Extend that path by adding character S[i+l] if it is not there already*
  + *end;*

*end;*

Suffix extension is all about adding the next character into the suffix tree built so far. In extension j of phase i+1, algorithm finds the end of S[j..i] (which is already in the tree due to previous phase i) and then it extends S[j..i] to be sure the suffix S[j..i+1] is in the tree. Apart from this, the algorithm uses suffix links, active points and few other tricks to keep track of existing suffixes and add a new node only if necessary. Suffix links essentially provides a shortcut to add new characters into the tree.

## Tandem repeat search in suffix trees:

To search for tandem repeats in a suffix tree given a pattern of size ‘m’, we follow the following 3 steps:

### First of all, we check if the given pattern really exists in string or not. For this, traverse the suffix tree against the pattern which takes O (m) time.

### If you find the pattern in suffix tree (don’t fall off the tree), then traverse the subtree below that point and find all suffix indices on leaf nodes. All those suffix indices will be pattern indices in string

### Now, check the indices if they differ by a count of size of the pattern, it is considered as a tandem repeat. Have a count of all such consequitive repeats which is the tandem repeat count of the given pattern against the given sequence.

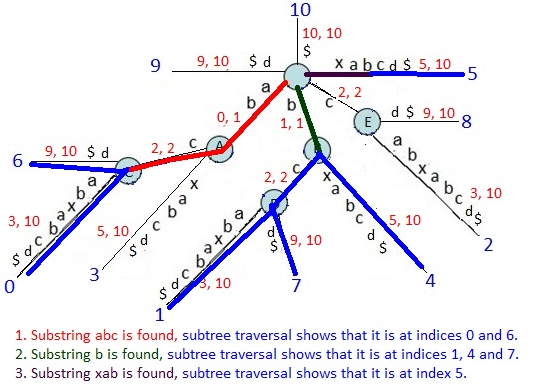
The figure 2 below shows the suffix tree structure which helps us find all the occurrences of a given pattern. 

Figure 2

# Suffix Array Based Algorithm

**Suffix trees** and **suffix arrays** are data structures for representing texts that allow substring queries like "where does this pattern appear in the text" or "how many times does this pattern occur in the text" to be answered quickly. Both work by storing all suffixes of a text, where a *suffix* is a substring that runs to the end of the text. Of course, storing actual copies of all suffixes of an n-character text would take O(n2) space, so instead each suffix is represented by a pointer to its first character in case of Suffix arrays.

A suffix array stores all the suffixes sorted in dictionary order. For example, the suffix array of the string abracadabra is shown below. The actual contents of the array are the indices in the left-hand column; the right-hand shows the corresponding suffixes.

Having constructed a Suffix tree using Ukkonen’s algorithm, we can construct a Suffix Array in linear time by doing a lexicographic order Depth First Search traversal and storing all the suffix indices in resultant suffix array, except the very 1st suffix index. Because a suffix tree of string of length N will have at most N-1 internal nodes and N leaves. Traversal of these nodes can be done in O(N) for a string of length N.

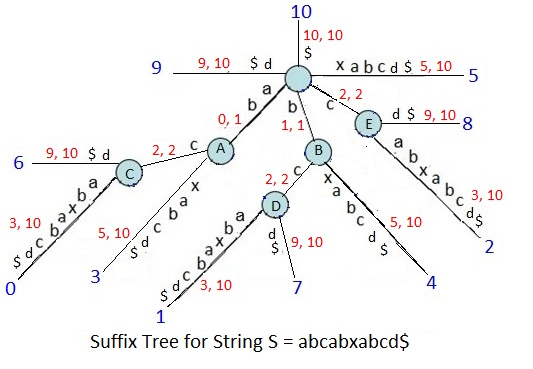


Figure 3

Consider the string abcabxabcd, its suffix array would be 0 6 3 1 7 4 2 8 9 5.

## Tandem repeat search in suffix arrays:

Face is one of the most vital visual features in a human being which has a crucial role in representing identity of the individual. It commonly includes feature extraction, reduction and classification or prediction of new images. Feature extraction is basically to find the most common features across many facial images which makes them distinguishable from others. Feature reduction is an important step because images will have thousands of pixels and not all of them represent noteworthy information as some do. We use PCA to reduce the dimensionality of the features and thus use the data in reduced dimensions to predict new images. Dimensionality reduction is done not only to improve running time but also not to miss the features with highest information. In this paper, PCA is used for both dimensionality reduction and also to classify the new image set. The new image set is projected into a lower dimension based on the Eigen values and Eigen faces and Euclidian distance is used to identify of the test images are in our original image set which is used to get the Eigen faces.

# Overview of PCA Algorithm

Principal component analysis (PCA) is one of the most popular methods for reducing the number of variables in face recognition. In PCA, faces are represented as a linear combination of weighted eigenvectors called as Eigen faces. These eigenvectors are obtained from covariance matrix of a training image set called as basis function. The number of Eigen faces that obtained would be equal to the number of images in the training set. Eigen faces take advantage of the similarity between the pixels among images in a dataset by means of their covariance matrix. These eigenvectors defined a new face space where the images are represented.

# Overview Of K means Clustering

The process of dividing or grouping a given set of data points into disjoint clusters is called Clustering. It is done in a way such that all the points in a cluster will have similar patterns and patterns of points belonging to different clusters are different. This algorithms has wide range of applications in the fields of Artificial Intelligence, Neural Networks, Machine Learning, Deep learning and Statistics.

K-means algorithm does the clustering based on the distance metric like Euclidian distance. It starts off with random initial centroids based on the number of clusters to be done. Then it assigns every point to closest centroid (cluster) which essentially forms a cluster. Now, it updates the centroids to the mean of all the points in a particular cluster. Repeat the above steps until there’s no change in the centroid locations which indicates that the clusters are well separated.

It has been shown that k – means is effective in producing decent clustering for practical applications. The k-means method has been shown to be effective in producing good clustering results for many practical applications. However, a direct algorithm of k-means method requires time proportional to the product of number of patterns and number of clusters per iteration. This is computationally very expensive especially for large datasets.

In this paper, we consider a gallery set which has 100 images of 50 x 50 pixels as the training data. Our objective is to recognize the matches for the images in Probe set (200 images) using PCA. We vary the count of Principal components (Eigen faces) and see how our recognition performance varies with that.

As a next step, we perform K means clustering technique on the whole data (gallery + probe) to do a soft biometric classification i.e gender classification in this case. We also observe the variation in clustering with respect to the number of Eigen faces considered and establish a relation between PCA and K means clustering. Finally, we analyze the clustering performance using an internal and external criteria. The next section explains the whole implementation procedure and results in detail.

# Implementation Procedure And Results

## PCA

As a first step, transform the input images into feature vectors as described in the figure 1.

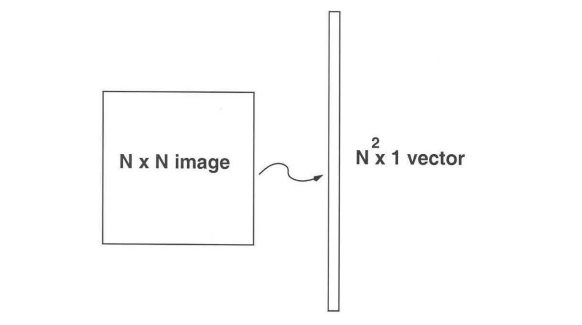
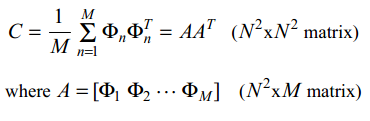


Figure 1

In our case, this transformation results in a matrix of dimensions (100 x 2500) to represent the training data (gallery set). Next, we subtract the mean image from this matrix. Now, compute the co variance matrix which gives us the relation among all the features in the data. We name the co variance matrix as C which can be calculated using the formula below.



We’ll use C to compute the Eigen values and Eigen vectors i.e Eigen faces. As we have 100 images in the training set, we’ll get a matrix of dimension 100 x 2500 representing first 100 principal components based on the highest 100 eigen values.



Figure 2

Figure 2 shows the eigen faces corresponding to the first 3 eigen vectors i.e these are the principal components along which there is highest variance. We can observe that the first 3 eigen faces have done a good job in identifying the visual features like eyes, nose, mouth and lips. It means these are the directions of features which have highest variance across all images in our training data set. It also represents the overall template of a human face.



Figure 3

Figure 3 represents the mean image of the training data set. It represents an average image of all the faces in the data. It is observable from this picture that the training data has a good mix of people belonging to both the genders. That’s why we see some features of females and males in this mean image. This a good way to understand our input data.

## Face Recognition using PCA

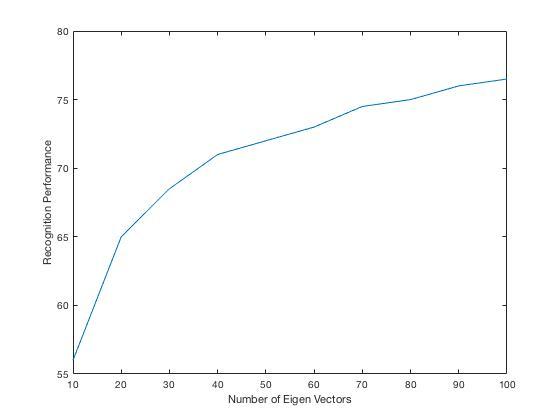
Now that we have the Eigen vectors corresponding to the highest variance, we make use of these to recognize the images in Probe data set with 200 images of 50 x 50 dimensions. First of all, project both the training and probe data into the lower dimensions using the Eigen vectors matrix varying the number of principal components considered in each iteration. In this project, we vary the number from 10 to 100 with an incrimination of 10 in each iteration. In each iteration we find the closest image in training data for each of the images in probe data using Euclidian distance as the distance metric. At this point, we need to identify if the recognition is correct or not. To do this, we check the image file names and assign a value of 1 in case of a match and 0 otherwise.

In this way we identify the correct matches (count of 1s) corresponding to a number of principal components considered. We observe the trend shown in figure 4 in the recognition performance.



Figure 4

From the figure 4, we can observe a monotonically increasing trend in the recognition performance as the number of principal components increase. This trend can be attributed to the fact that, more number of principal components represent more variance and thus more information about faces in the training data. The higher the information we have about features of faces, the better our recognition is. This is exactly what we see in figure 4. We get a recognition performance of 76.5% for 100 components as shown in the below graph.



As we increase the number of principal components further, the trend would not remain the same and would remain constant at 153 instead. It is because the majority of the variance has been captured by the 100 components as our input has only 100 images in it. So, we cannot expect to improve our recognition performance further.

## Face Recognition without PCA

The idea of recognizing the faces without PCA is to use the complete information of the pixels of both training and probe data and find the closest pairs. To achieve this, we repeat the same procedure we did in the previous section but without projecting the data into a lower dimension. That means we just find the closest pair in training set for all the images in probe set. Strikingly, we get a match count of 153 which is exactly same as with that of 100 principal components. So the observation we made at the end of the previous section is correct. The required variance for face recognition is indeed captured by the 100 components.

Figure 5 shows the variance of Proportion of Variance explained with respect to the number of principal components. PVE is the ratio of the variance explained by n\_th component and the variance explained by total number of principal components.

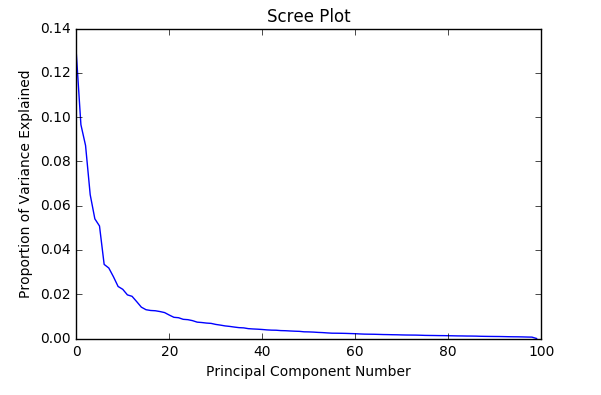


Figure 5

Figure 5 shows the scree plot which represents that almost all of the variance in the data is captured by the 100 principal components. This is the reason why we get the similar recognition performance for the data projected to 100 components and for the data without projection into lower dimensions.

# Soft Biometric Classification

When performing large scale facial recognition, a way to improve performance is to reduce the search space for matches by first by first performing soft biometric classification like gender, age or ethnicity classification etc. We use the K means clustering algorithm to cluster our data (Gallery + Probe). We do the clustering for different values of components considered and observe the relation between PCA and K means.

On performing K means on the combined data, we get the recognition rate as below for a particular number of principal components.

|  |  |
| --- | --- |
| Principal components | Cluster count of one of the clusters |
| 10 | 124 |
| 20 | 125 |
| 30 | 145 |
| 40 | 125 |
| 50 | 172 |
| 60 | 175 |
| 70 | 175 |
| 80 | 125 |
| 90 | 125 |
| 100 | 147 |

Figure 6

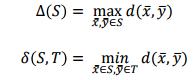
The figure 6 shows the table with number of principal components in first column, count of males in the data in second column and F1 score, an external validity metric for cluster evaluation in third column. We can observe that one of the clusters get a count of either 125 or 175 no matter how many number of principal components are considered in the range 10 to 100. This suggests that even 10 components is good enough to cluster the data into two different groups. This indicates that out of whole 300 images in the input data, two clusters are formed with approximately 175 and 125 images respectively. This is in good accordance with our data as Gender.txt file says that there are 65\*3 = 195 male individuals in the whole data set i.e. 65 in GallerySet and two versions of the same person in Probe Set.

The clustering performance very well depends on how we initialize the centroids in k means algorithm. We use two different criteria to determine the cluster validity of the data. The external criteria chosen is NMI Measure and internal criteria chosen is Dunn’s index. The idea of these indices is briefly mentioned in the next section.

## Dunn Index – Internal Cluster validity index

The Dunn index (DI) is an evaluation metric for clustering algorithms. It is an internal evaluation scheme which means the result is based on the clustered data itself. The aim is to identify groups of clusters which are compact with small variance between members of the cluster and the means of all the clusters are separated from the other sufficiently apart. For a given cluster assignment, higher the Dunn Index indicates the better clustering.

Let S and T be two non-empty subsets of R^N, then the diameter delta of S and set distance between S and T are given as follows.



Where d(x,y) is the distance between points x and y. For any partition, Dunn defined the following index:



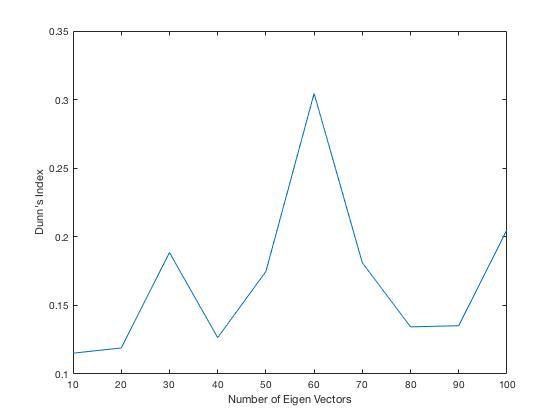
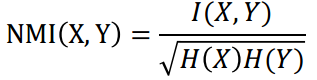


Figure 7

## The figure 7 shows the variation of Dunn index as the number of number principal components increase in the range 10 to 100.

## NMI Measure – External Cluster Validity index

Normalized Mutual Information (NMI) is a normalization of the Mutual Information (MI) score to scale the results between 0 (no mutual information) and 1 (perfect correlation). In this function, mutual information is normalized by the following formula.



Here I(X,Y) denotes the mutual information between the two random variables X and Y and H(X) denotes the entropy of of X, X will be consensus clustering while Y will be true labels.

NMI index for our clustering problem is calculated to be 0.

# Conclusion

We observe from recognition results of PCA that beyond 30 components, there is minimal gain in recognition accuracy i.e. lesser than 10%. We also note that even if only 30 Eigen vectors are used, there is lesser than 10% loss in the recognition performance. However, the computation complexity of this is 70% lesser than using purely Euclidean distance, which justifies the use of Principal Component Analysis.

The cluster count of K-Means gender classification shown in figure 6 for the gallery + probe set images was observed to be inconsistent across multiple runs. Hence, an optimal number of Eigen Vectors for efficient classification could not be determined.

It was observed from figure 7 that Dunn’s index varied widely across multiple runs of the algorithm. We thus cannot conclude on the number of principal components to be considered. This suggests that the clustering performance may not be accurate. This is also verified when examining the values of the external validation criterion, NMI. The NMI values are consistently zero which suggests that the clusters formed and target clusters have no mutual dependence. This happens because the input data set for K-Means clustering was the data obtained from PCA. PCA gives us the directions along which there is maximum variance but it does not contain any information about the gender of the subjects which results in inconsistent clustering performance.

To conclude, PCA was found to be an efficient algorithm for face recognition. PCA helped reduce the computation complexity while not degrading the recognition performance significantly. However, it was found that the Eigen faces did not contain any useful information about the gender of the subjects hence it is not a useful tool in gender recognition.

# References

1. <http://www.cs.yale.edu/homes/aspnes/pinewiki/SuffixArrays.html>
2. <http://www.geeksforgeeks.org/suffix-tree-application-4-build-linear-time-suffix-array/>
3. <http://www.geeksforgeeks.org/ukkonens-suffix-tree-construction-part-1/>
4. An improved face recognition technique based on modular PCA approach. Elsevier, Pattern Recognition Letters 25 (2004) 429–436 by Rajkiran Gottumukkal, Vijayan K.Asari
5. Principal Component Analysis for face recognition. Saurabh P.Bahurupi, D.S.Chaudhari, International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249 – 8958, Volume-1, Issue-5, June 2012
6. An Efficient K-Means Clustering Algorithm, Snajay Ranka, UFL.