JADAVPUR UNIVERSITY

Tri-clustering from Closer Property using Suffix Forest

by

SANGHAMITRA LAHIRI

CLASS ROLL NUMBER: **001711002026**

REGISTRATION NUMBER: **140982 OF 2017-18**

EXAMINATION ROLL NUMBER: **M4SWE19026**

A thesis submitted in partial fulfillment for the

degree of Master of Engineering in Software Engineering

UNDER SUPERVISION OF

Dr. Kartick Chandra Mondal

ASSISTANT PROFESSOR

in the

Faculty of Engineering Technology

Department of Information Technology

MAY, 2019

DEPARTMENT OF INFORMATION TECHNOLOGY
FACULTY OF ENGINEERING AND TECHNOLOGY

JADAVPUR UNIVERSITY

Certificate of Submission

I hereby recommend the thesis entitled, "Tri-clustering from Closer Property using Suffix Forest", prepared by Miss. Sanghamitra Lahiri (CLASS ROLL NUMBER: 001711002016; REGISTRATION NUMBER: 140982 OF 2017-18; EXAMINATION ROLL NUMBER: M4SWE19026), under my supervision, be accepted in partial fulfillment of the requirements for the degree of Master of Engineering in Software Engineering from the Department of Information Technology under Jadavpur University.

Dr. KARTICK CHANDRA MONDAL
Supervisor
Assistant Professor
Department of Information Technology
Jadavpur University

Countersigned by:

Head of the Department
Information Technology
Jadavpur University

Dean
Faculty of Engineering Technology
Jadavpur University

DEPARTMENT OF INFORMATION TECHNOLOGY
FACULTY OF ENGINEERING AND TECHNOLOGY

JADAVPUR UNIVERSITY

CERTIFICATE OF APPROVAL

(Only in case of thesis is approved)

The thesis at instance is hereby approved as creditable study of an engineering subject carried out and presented in a manner satisfactory to warrant its acceptance as a prerequisite to the degree for which it has been submitted. It is understood that by this approval the undersigned does not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn therein, but approve this thesis for the purpose for which it is submitted.

Signature of the External Examiner

Signature of the Supervisor
Dr. KARTICK CHANDRA MONDAL
Assistant Professor
Department of Information Technology
Jadavpur University

DEPARTMENT OF INFORMATION TECHNOLOGY
FACULTY OF ENGINEERING AND TECHNOLOGY

JADAVPUR UNIVERSITY

Declaration of Originality and Compliance of Academic Ethics

I, hereby declare that entitled Tri-clustering from Closer Property using Suffix Forest contains only the work completed by me as a part of the Master of Engineering in Software Engineering course, during the year 2018-2019, under the supervision of Dr. Kartick Chandra Mondal, Assistant Professor, Department of Information Technology, Jadavpur University.

All information, materials, and methods that are not original to this work have been properly referenced and cited. I am the only responsible person if found guilty of plagiarism. I am aware/warned of the plagiarism issues by our supervisor several time during the progress of our thesis.

The idea of the work has been given by my supervisor and guided by him only. The work has been done solely by me and I did not include anybody's work in it. The supervisor has full authority to use, modify, correct and distribute the work presented here.

I also declare that no part of this project has been submitted for the award for any other degree prior to this project by me. Also, I declare that I will not distribute or use the full or part of this project work for the award of any other degree by us in the future.

All information in this document has been obtained and presented in accordance with academic rules and ethical conduct

(Signature with Date)

Acknowledgements

I take this opportunity to express my deepest sense of gratitude towards my respected guide Dr. Kartick Chandra Mondal, Assistant Professor, Department of Information Technology, Jadavpur University for giving me a great opportunity to work under his excellent guidance and motivating me throughout. I would like to thank him for giving me ample liberty to conduct research on my field of interest. He has been an inspiration to me for his perfection towards work.

I would also like to thank Ph.D. Scholar in Information Technology Department at Jadavpur University, Mrs. Moumita Ghosh for supporting me technically throughout the thesis work as well as the entire Information Technology Department for giving me the liberty to work and complete the thesis on time.

I would like to express my gratitude and love towards my respected parents, my elder brother and my peers for being the main source of inspiration and motivation throughout my career and the making of this thesis.

Sanghamitra Lahiri

JADAVPUR UNIVERSIY

Abstract

Faculty of Engineering Technology

Department of Information Technology

Master of Engineering

by Sanghamitra Lahiri

One can observe a huge change among forests in various states in India as time passes. And many kinds of forests are lost day by day without any proper information which can refer year wise various forest distribution in Indian states. Nowadays machine learning techniques are used in each and every aspects of the current world. Here also, a part of machine learning called clustering can be used to keep track of Indian forest distribution. But these forest data prediction is much efficiently predicted using three-dimensional dataset, where first dimension (object) is for various states, second dimension (attribute) is for various kind of forests and finally, the third dimension (condition) represents the various year (in which year data are taken). So, it's quite clear that single dimensional clustering can't be so effective for these kinds of datasets. As well as two-dimensional clustering is also failed because a third dimension i.e. year is involved here. So, a new technique of three-dimensional clustering called Tri-clustering is used here to efficiently solve these kinds of problem where three dimensions are involved with a dataset. Here, in this thesis, I have modified some bi-clustering algorithm and make them efficient for a tri-clustered case which, leads us to a better understanding of forest distribution in India. I have applied those algorithms on Indian Forest distribution datasets and got a suitable result, by which it becomes easy to predict future aspects of Indian Forest. These algorithms can be applied to other kinds of datasets where three dimensions are involved. But apart from Indian Forest Distribution dataset, every other dataset should be pre-processed before applying these algorithms, because each and every dataset have their specific per-procession process. So, a user should be careful before applying these algorithms to any other kind of dataset otherwise it shows some error.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Tri-clustering

The basic concept of clustering is to separate similar chunk of dataset form a large dataset [1]. There are several types of clustering like single-clustering, bi-clustering, subspace clustering, co- clustering, hierarchical clustering etc [2].

In this paper a tri-clustering process is introduced which can be applied to those data which are coming from online data sources, to get better handling efficiency on those data. Here Indian forest's data are used to create primary tri-clusters and at the end, this algorithm is able to uniquely identify, an incoming data belongs to which tri-cluster. A unique feature of this paper is, here suffix tree is used to store clusters, which is more space efficient in term of the computer storage system.

It is very important to keep track of the forest area because only this part of the world shrinking day by day as civilization expands. So, there is a huge chance to lost many kinds of forests from this world permanently. From here we can say that keep proper information is very important which can tell us, which kind of forest become necessary in which part of the world. To solve this kind of query tri-clustering is the most efficient technique.

In this particular experiment, only Indian forest diversity is taken. Here database tables are prepared to contain year wise various kind of forest area (in sq.km) according to their states. Total seven tables are prepared and each table is for one year.

Before start working with any data set, pre-processing is very important, because through data pre-processing only useful information comes under consideration for an experiment.

Here, after data pre-processing FIST 2.0 [3] algorithm is used to determine bi-clusters for each and every pre-processed data set. FIST algorithm involves Apriori algorithm which requires frequently closed itemsets based on min support. This algorithm uses suffix tree data structure to predict bi-clusters, so that, this consumes very less memory compared to other bi-clustering algorithms. Next, using some rules these suffix trees are combined and a generalized suffix forest, which contains ‘condition’ (i.e. year), is prepared. From this final suffix forest, one can easily conclude the tri-clusters.
After preparing these tri-clusters it will be very easy to determine the extinction of various kinds of forests in India.

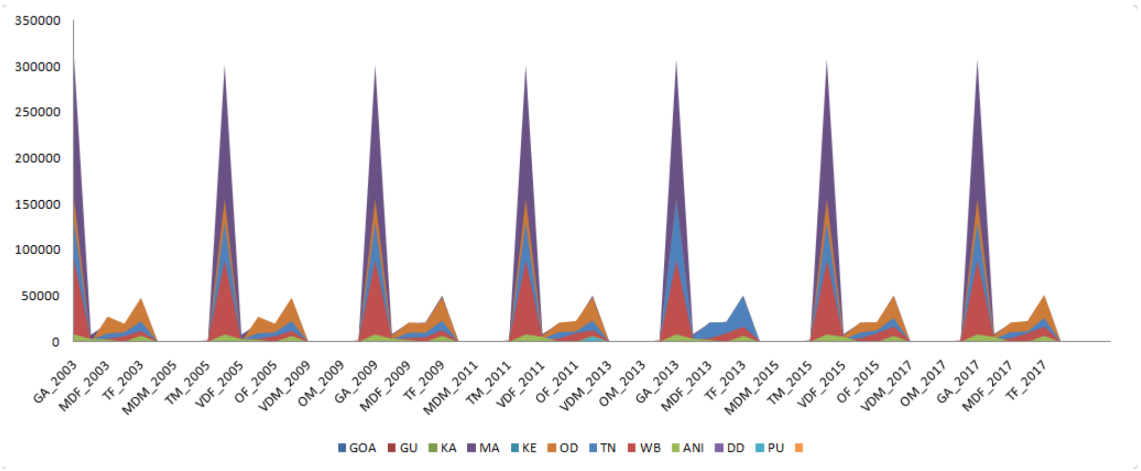


Figure 1.1: Year Wise Indian Forest Distribution

The procedure which is mentioned in this thesis can be used in various other kinds of three-dimensional predictive data sets.

1.2 Suffix Tree

A suffix tree is generally used for pattern searching [4]. It is also known as PAT Tree (Position tree). It generally used where a large number of the alphabet is involved. This kind of trees is drastically used by commercial industries, where pattern searching plays a huge role. There are lots of algorithms created using suffix tree for various purpose-

1. Suffix tree construction for large string [5].
2. Suffix tree can also be used for spelling approximation [6].
3. It is also used for parallel computing, where parallel processing of RAM is needed [7].

General suffix tree creation (using string: **abc**) is shown in below figure (Figure 1.2):

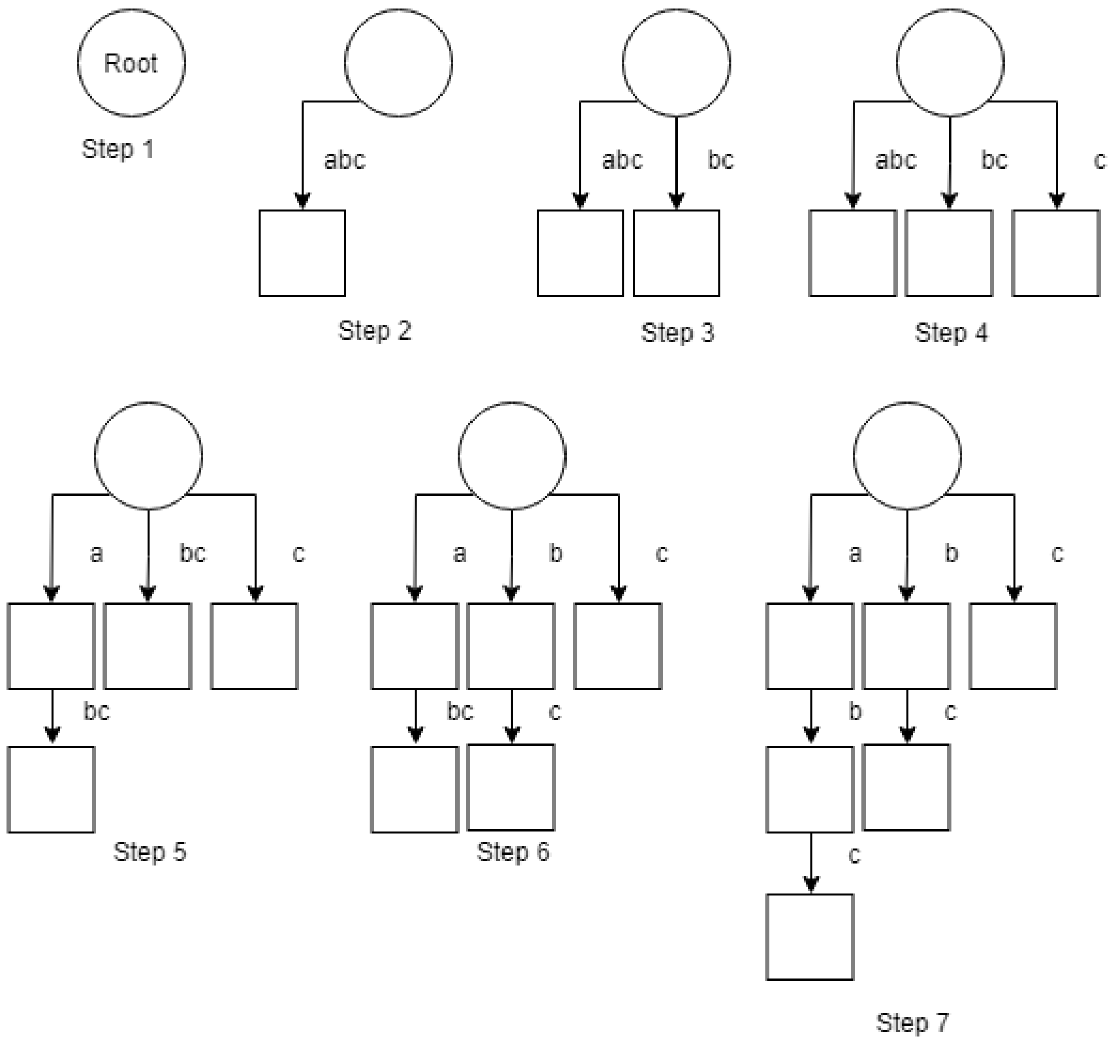


Figure 1.2: Suffix Tree Creation

The suffix tree for the string S of length n is defined as a tree such that [8]:

1. The tree has exactly n leaves numbered from 1 to n.
2. Except for the root, every internal node has at least two children.
3. Each edge is labeled with a non-empty sub-string of S.
4. No two edges starting out of a node can have string-labels beginning with the same character.
5. The string obtained by concatenating all the string-labels found on the path from the root to leaf i spells out suffix $S[i..n]$, for i from 1 to n.

Since such a tree does not exist for all strings, S is padded with a terminal symbol not seen in the string. This ensures that no suffix is a prefix of another and that there will be n leaf nodes, one for each of the n suffixes of S. Since all internal non-root nodes are branching, there can be at most $n-1$ such nodes, and $n+(n-1)+1=2n$ nodes in total (n leaves, $n-1$ internal non-root nodes, 1 root).

1.2.1 Generalised Suffix Tree

Suffix trees can represent only a single string, where the generalized suffix tree is used to express a set of strings. Each string must be terminated with a different symbol or word. In the generalized suffix tree, no two leaf node can hold the same symbol or word. If it happens then first appeared (from left to right) symbol or word will get higher priority than the latter one and the latter one will be deleted [9].
Generalized suffix tree of two string ABAB and BABA is shown in the below figure (Figure 1.3):

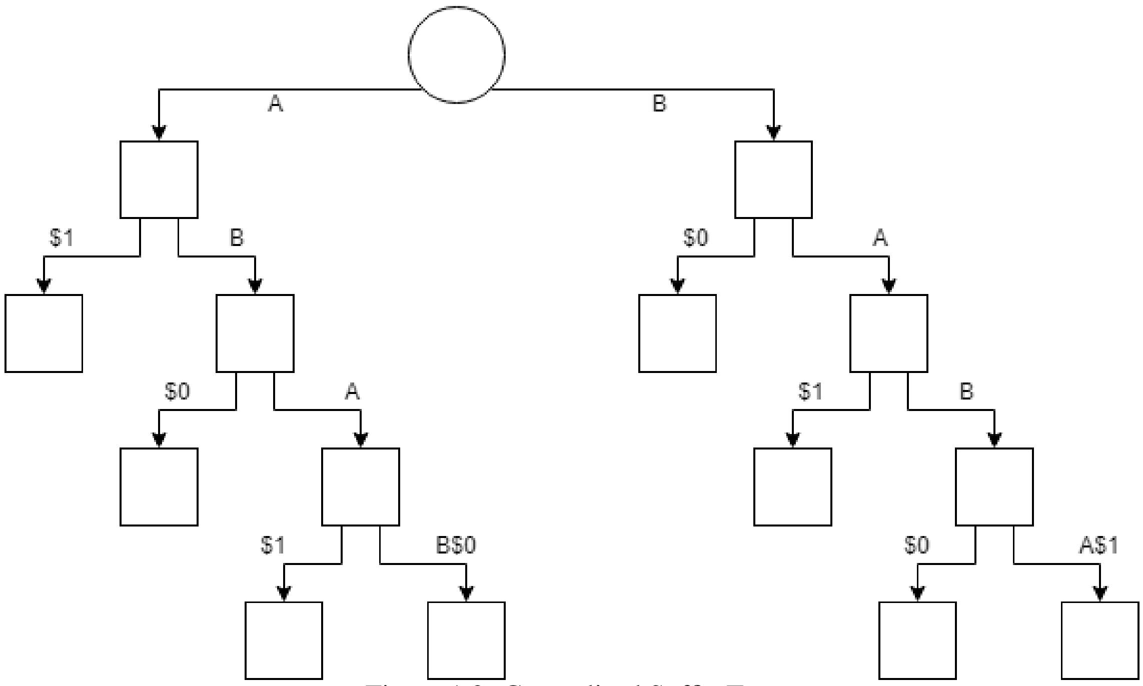


Figure 1.3: Generalised Suffix Tree

In generalized suffix tree's leaf node denoted with a \$ symbol and every leaf node holds different word or symbol. Each string starts from the root node separately. This is how generalized suffix tree created. In this project, the generalized suffix tree is used to create clusters, which takes less computer storage space [10].

1.3 Suffix Forest

Suffix forest is a newly discovered data structure for searching frequent itemsets in datastreams [11]. It uses the simplicity of the suffix tree in its basic algorithm. Suffix tree can only contain one string of data. As the data quantity grows, lots of suffix tree become created. So, to combine all this suffix tree, suffix forest is invented. It first scans the whole dataset and then and devised to support aggregation queries on demand.
Suppose there are two suffix trees for strings 'abc' and 'abcd', then their combined suffix forest is shown in below figure (Figure 1.4).

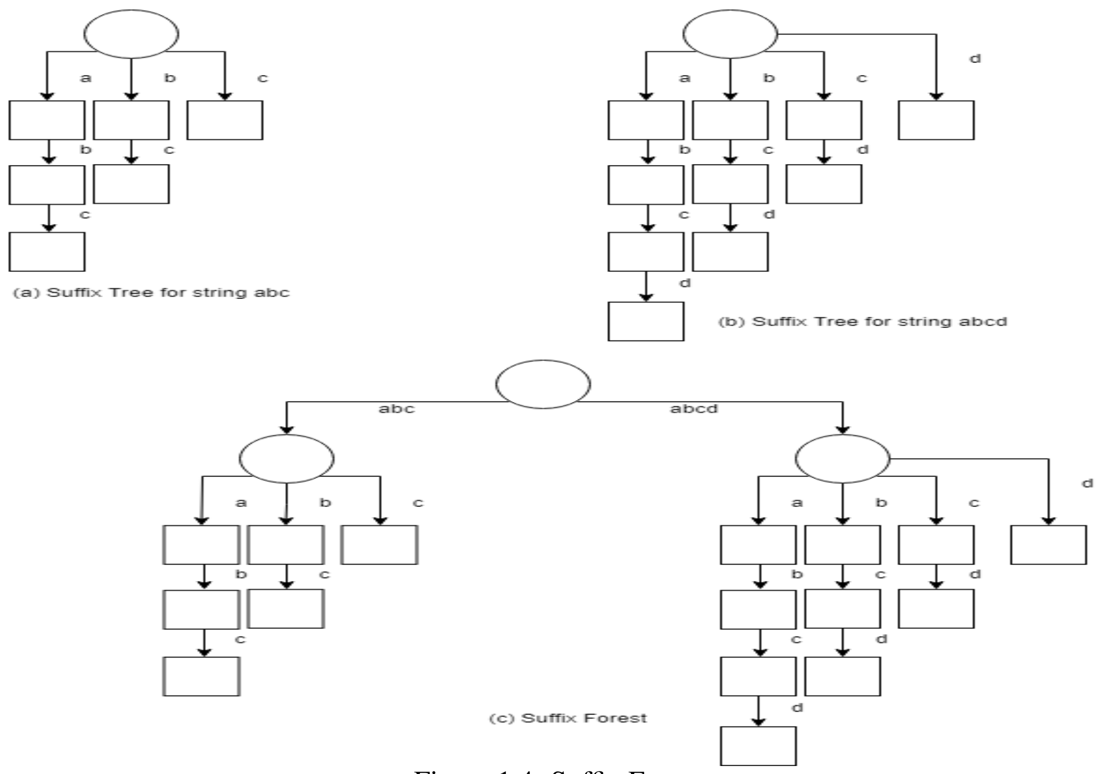

a
b
c
b
c
c
(a) Suffix Tree for string abc
a
b
c
d
b
c
d
c
d
d
(b) Suffix Tree for string abcd
abc
abcd
(c) Suffix Forest

Figure 1.4: Suffix Forest

1.3.1 Generalised Suffix Forest

In this thesis work, I have used a generalized suffix forest. In generalized suffix tree we can use many strings form only one dataset. But in generalized suffix forest, we can easily merge those generalized suffix tree, so that lots of datasets can be merged.
The main difference between suffix tree, generalized suffix tree, suffix forest and, generalized suffix forest is:-

- Suffix tree can hold only one string, but generalized suffix forest can hold lots of strings.

- Generalised suffix tree can hold lots of strings which comes under single iteration, whereas generalized suffix forest can insert lots of strings belongs to different time stamps.
- Suffix forest also can hold lots of strings of different time stamps but those strings must have stayed in one dataset. Generalized suffix forest is able to overcome this difficulty also. It can hold strings from different datasets.

So, from the above-mentioned differences between all other kinds of trees, generalized suffix tree is more suitable for those datasets where three dimensions are involved.

1.4 Motivation and Contribution

The main motivation of this comes from the urge of finding the new clustering concept, using which clustering technique may become more useful for industrial and scientific purpose. Clustering techniques become more acceptable for future data prediction purpose so, clustering related topic was the first preference for thesis topic as well as tri-clustering technique is totally new research topic in the market nowadays.
In this thesis, the generalized suffix forest concept is extended from the generalized suffix tree concept and some new algorithms are proposed to make the tri-clustering from the bi-clustering concept. As well as these algorithms are applied to some datasets to check its correctness.

1.5 Organization of the Thesis

This thesis is an outcome of theoretical and practical researches on Tri-clustering from Closer Property using Suffix Forest. In chapter 2, the discussion will be on related work or previous work on tri-clustering, Formal Context Analysis(FCA) for a dyadic case, FCA for a triadic case and some practical cases where tri-clustering was used. All the previous work related taxonomy are given in the Bibliography section at the end. The next chapter i.e chapter 3 contains the algorithm which is created for this thesis. In this chapter eight algorithms are created to create tri-clusters. The next chapter 4 is the implementation chapter where those algorithms are applied to some datasets and the final making of tri-clusters are shown. The 5_{th} chapter is on result analysis, where real-world scenario and practical research result differences and it's adaptability is concerned. The 6_{th} chapter will conclude the whole thesis work. In the next chapter i.e. Chapter 7 another way of tri-clustering is proposed, and that method in this is not practically proved, just theoretically designed. In the next section, an appendix is stated. It has four sections wherein the section A, which kind of datasets can be used for this experiment is stated, in the section B data pre-processing techniques are stated, and in the section C the user guide which states how to implement this project practically is given for further use of this project by other users, and finally in the last section D it shows the experimental setup to complete this thesis practically.

Chapter 2

Related Work

2.1 Previous Work on Bi-clustering

2.1.1 Bi-clustering for Gene expression Micro-array Data

For micro-array gene data many clustering algorithms are proposed before but they failed to give appropriate clustering output. So a new technique for clustering is proposed by Sara C. Madeira and Arlindo L. Oliveira [12]. This new clustering technique is known as bi-clustering for gene expression data, where those data produce highly correlate genes.

2.1.2 Bi-clustering for Bird Population Status

Bird population status depends upon weather, migrated place's local weather, breeding ground status etc. So, it's very difficult to calculate birds population. From these background one bird population measuring bi-clustering procedure is proposed [13]. These procedure predicts almost appropriate bird population status of current world.

2.1.3 Bi-clusters using Suffix tree

Bi-clusters are the basic building block of this thesis work. In this thesis work to make bi-clusters FIST algorithm [14] is used. This algorithm uses Apriori algorithm [15] is used. Here suffix tree is also used for a basic data structure.

2.1.4 FCA for dyadic case

The dyadic case based on binary relationships is a classic one in the FCA [16]. Let us consider it first.

$\kappa := (G, M, I)$ is called a dyadic formal context.

G = sets of objects,

M = sets of attributes,

I = binary relationship and I should be $I \subseteq G \times M$.

Then, the pair $(g, m) \in I$ can be interpreted as the "object g has the attribute m." It is convenient to write formal contexts in the form of Boolean matrices or in tabular form.

Now, let us consider the following mapping: -

$$\phi : 2^G \rightarrow 2^M \tag{2.1}$$

$$\psi : 2^M \rightarrow 2^G \tag{2.2}$$

ϕ in equation 2.1 is Galois connection between $(2^G, \subseteq)$. [17]

ψ in equation 2.2 is Galois connection between $(2^M, \subseteq)$.

Then from equation 2.1 and equation 2.2 we can say,

$$\phi(A) = m|gImforallg \in A, \tag{2.3}$$

$$\psi(B) = m|gImforallg \in B \tag{2.4}$$

From equation 2.3 and 2.4, the following is true $(A_1, A_2 \subseteq GandB_1, B_2 \subseteq M)$

$$A_1 \subseteq A_2 \Rightarrow \phi(A_2) \subseteq \phi(A_1) \tag{2.5}$$

$$B_1 \subseteq B_2 \Rightarrow \psi(B_2) \subseteq \psi(B_1) \tag{2.6}$$

$$A_1 \subseteq \psi(\phi(A_1))andB_1 \subseteq \phi(\psi(B_1)) \tag{2.7}$$

In equation 2.5, 2.6 and 2.7, instead of ϕ and ψ a single notation $(\cdot)'$ is used, which is called the Galois operator [18] or prime operator. This is acceptable, since the question of which of the two operators is used is uniquely determined by the set to which the operator is applied.

It is now possible to define a formal concept: -

A pair of sets (A, B) can be called a dyadic formal concept if

$$A \subseteq G \tag{2.8}$$

$$B \subseteq M \tag{2.9}$$

$$A' = B \tag{2.10}$$

$$B' = A \tag{2.11}$$

Set A = formal extent.

Set B = formal intent.

Based on the definition of formal concepts [19] it can be seen that if we interpret the formal context as a Boolean matrix, the formal concepts are maximal rectangles consisting of the units in the context (up to a permutation of rows and columns).

The set of all formal concepts of context form a partial order [20] i.e.: - $(A, B) \geq (C, D) \Leftrightarrow C \subseteq A(B \subseteq D)$.

For the operator $(\cdot)'$ we have the following properties $(A, A_1, A_2 \subseteq GandB \subseteq M)$

$$A_1 \subseteq A_2 \Rightarrow A_2' \subseteq A_1' \tag{2.12}$$

$$A_1 \subseteq A_2 \Rightarrow A_{1\mathrm{J}} \subseteq A_{2\mathrm{J}} \tag{2.13}$$

$$A \subseteq A'' \tag{2.14}$$

$$A''' = A'(\Rightarrow A'''' = A'') \tag{2.15}$$

$$(A_1 \cup A_2)' = A_1' \cap A_2' \tag{2.16}$$

$$A \subseteq B' \Leftrightarrow B \subseteq A' \Leftrightarrow A \times B \in I \tag{2.17}$$

Similar propositions are also true if we reverse the subsets of the objects and the subsets of the attributes.
Finally, we define the closure operator [21].
The mapping $\phi : 2^G \to 2^G$ is called a closure operator on the set G. This mapping assigns a closure $\phi X \subseteq G$ with the following properties to each subset $\phi X \subseteq G$

$$\phi\phi X = \phi X(idempotenceproperty)[22] \tag{2.18}$$

$$X \subseteq \phi X(extensivityproperty)[23] \tag{2.19}$$

$$X \subseteq Y \Rightarrow \phi X \subseteq \phi X(monotonicityproperty)[24] \tag{2.20}$$

$(\cdot)''$ operators for the context are closure operators.

2.2 Previous Work on Tri-clustering

2.2.1 Tri-Clustering of gene expression micro-array data

- Microarray Data $\rightarrow$ Microarrays allow monitoring of gene expression for tens of thousands of genes in a parallel and huge amount of valuable data is produced (Here time point is being considered) [25]. The raw microarray data are images, which have to be transferred into gene expression matrices-tables were rows represent genes, columns represent various samples such as tissue or experimental conditions, and number in each cell characterize the expression of the level of the particular gene in a particular sample.

- By tri-clustering we can group genes in two categories [26] $\rightarrow$
  1. Under particular condition
  2. Under particular time point

  Thus tri-cluster is being capable of measuring 3D data.

- Tri-clustering attacks $NP - hard$ problems, thus algorithms based on heuristic are well suited for it [27].

- In heuristic algorithm two types of work is done in two different parts $\rightarrow$
  1. Classification tasks where supervised learning is used [28]. It uses Euclidean distance and Manhattan distance function as fitness function to improve the output's efficiency of the algorithm.
  2. Optimization problem, it uses sum of a set of calculated parameters related to the problem domain as its fitness function [29].

- In generic algorithm each solution is generally represented as a string of binary numbers, known as a chromosome.

- Algorithm used:- Here TriGen algorithm (based on heuristic algorithm) is used for searching patterns of similarity for genes on a three-dimensional space (i.e. gene, conditions and time factors) [30].

- Fitness function used in TriGenalgo $\rightarrow$
  1. MSR_{3D} $\rightarrow$ This is three dimension adaptation of measure Mean Squared Residue(MSR) i.e. MSR_{3D} is used [31]. Here validation mechanism is correlation among genes, conditions and time stamp. Here two types of correlation measures are used $\rightarrow$ Pearsons and Spearman.
  2. LSL $\rightarrow$ Least Squared Lines measures the quality of tri-cluster based on the similarity among the slope of the angles formed by the LSL from each of the profiles formed by the genes, conditions and times of the tri-cluster. Here graphic validation for validation mechanism is used [32].

3. MSM $\rightarrow$ Multi Slope Measure measures the quality of a tri-cluster based on the similarity among the angles of the slopes formed by each profile formed by genes, conditions and times of the tri-cluster. Functional annotations of the genes as validation mechanism are used [33].

- Framework $\Rightarrow$ TIRQ$\rightarrow$ TRIcluster Quality is produce to introduce a single measure that globally assesses the quality of the tri-clusters generated by any tri-clustering algorithm [34].

- TRIQ has three general equations [35] $\rightarrow$

  1. Biological quality (BIOQ) or Biological quality of tri-clusters $\rightarrow$ The gene ontology (GO) project is a major bioinformatics initiative with the aim of standardizing the representation of gene and gene product attributes across species and datasets. The biological quality of a tri-cluster is calculated bases on GO analysis that identifies for a set of genes in a tri-cluster: biological process, cellular components and molecular functions.

  2. Graphical quality (GRQ) or Graphical quality for tri-clusters $\rightarrow$ The graphic quality of a tri-cluster is a quantitative representation of a qualitative measure: how homogeneous the members of the tri-cluster are. Visual Validation by means of graphically representing the tri-cluster on their three components: gene, conditions and time points.

  3. Pearson quality (PEQ) or Value for the Pearson correlation of tri-clusters and Spearman quality (SPQ) or Value for Spearman correlation of tri-clusters $\rightarrow$ Random variables for a tri-cluster TRI are defined to calculate PEQ and SPQ, based on its subset of genes, conditions and time stamps. Thus every tri-cluster will have a set of random variables 'vars' composed of the combination of each gene and each experimental condition. Each of these variables will have an expression level for each time stamp.

- These tri-cluustering techniques are applied on some data set previously, and those datasets are Yeast Cell Cycle Datasets, Mouse GD54510 Datasets, and Human GDS4472 Datasets.

2.2.2 Tri-Clustering approach for searching optimal patterns

- Triadic data can be processed in this below mentioned technique (Figure 2.1) [36] $\rightarrow$

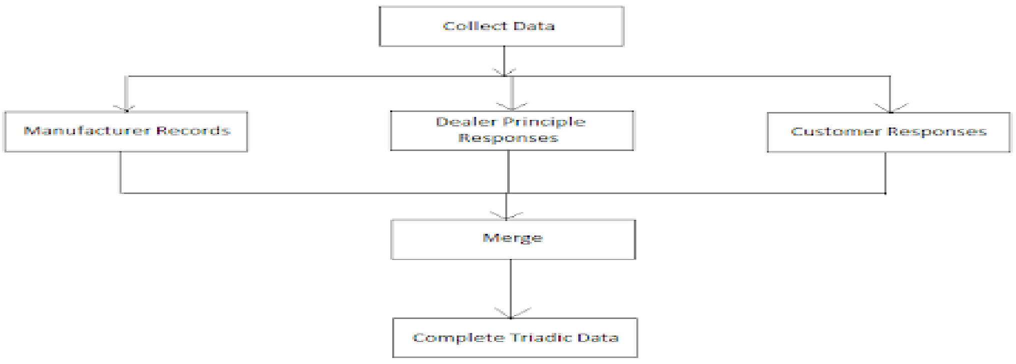


Figure 2.1: Triadic Data Processing Technique

- Synthetic Data $\rightarrow$ The creation of synthetic data is an involved process of data anonymization, that is to say, that synthetic data is a subset of anonymized data [37]. Synthetic data is used in a variety of fields as a filter for information that would otherwise compromise the confidentiality of particular aspects of data [38].
- Algorithm used : -
  1. TRIAS algorithm $\rightarrow$ Triadic Formal Concept Analysis TRIAS algorithm is a method of finding triadic formal concepts that are closed 3-sets [39]. It is used for searching optimal tri-patterns and absolutely dense tri-clustering. TRIAS is based on NextClosure algorithm [40] that enumerates all formal concepts of the dyadic concept in lectic order, the lexicographic order on bit vectors describing subsets of objects. The TRIAS algorithm was designed to mine folksonomies in resource sharing system e.g. in social bookmarking system like delicious bibsonomy. TRIAS has a precursor which is TRIPAT algorithm [41], which is used for analyzing triadic data from psychological studies.
     - Fitness functions for TRIAS algorithm [42] $\rightarrow$
       (a) FirstFreqCon
       (b) NextFreqCon
  2. OAC tri-clustering algorithm $\rightarrow$ It measures average density of the output, diversity, coverage and noise-tolerance. It produces large number of dataset

compared to TRIAS algorithm [43].

3. Box OAC tri-clustering algorithm [44] $\rightarrow$ It has two algorithms $\rightarrow$
   - TriBox[45]
   - SpecTric
4. Prime OAC tri-clustering algorithm $\rightarrow$ It is used to calculate the results of the prime operation for all the possible combinations of two elements of different sets of the context, and then enumerate all triples of the ternary relation for a context generating a prime operation based tri-cluster for each. It generated tri-cluster T was not added to the set of all tri-cluster τ on previous steps, then T is added to τ. It is possible to implement hash-functions [46] for tri-clusters in order to significantly optimize computation time by simplifying the comparison of tri-clusters. Also a minimal density threshold can be used. It is best on scalability to large real-world datasets [47].
5. Greedy OAC tri-clustering algorithm technique [48] $\rightarrow$
   - Prime operator based OAC tri-cluster τ is needed [49].
   - Need to store triples, which are generated during tri-clustering process, for each of the tri-clusters and the full set of these triples τ.
   - For improving greedy approach we have to sort the resulting set t_i, by two parameters:
     * Density of the tri-cluster
     * Volume of the tri-cluster
   - For each tri-cluster T, enumerates all the triples t_i in it that are also present in τ.
   - Check all of the tri-clusters $T(t_i)$ generated by t_i, and try to merge them with the original tri-cluster. Merge conditions are:
     * If the density of the new tri-cluster is large enough we merge the tri-clusters, otherwise we remove $T(t_i)$ from τ.
     * If the density of the new tri-cluster and the measure of the intersection size of the original tri-clusters are large enough, we merge them, otherwise we remove $T(t_i)$ from τ.

    * If the density of the new tri-cluster and the measure of the intersection size of the original tri-clusters are large enough, we merge them otherwise, if just the density is large enough we do not merge the tri-clusters, but we also do not remove $T(t_i)$. And in case both the density and the interaction size are not large enough, we remove $T(t_i)$ from τ.
  - Finally we remove t_i from τ. The algorithm halts when τ is empty or when we have enumerated all of the tri-clusters from τ.
  - It takes four parameters as input:
    (a) minimum similarity threshold
    (b) minimum sample threshold
    (c) minimum gene threshold
    (d) minimum time threshold
  - It gives one parameter as output: coherent clusters along gene-sample-time dimension.

- These kind of tri-clustering concept uses Formal Concepts Analysis (FCA) framework [50].

- Criteria for evaluation of tri-clusters [51] $\rightarrow$

  1. There are four criteria are considered to make cluster set and these criterias are Cardinality, Density, Diversity, and Coverage.
  2. Noise-tolerance, Speed, and Complexity these criteria should be maintained for any optimal pattern search tri-clustering algorithm.

- The above mentioned algorithms are performed on some real-time datasets for experiment purpose and those are Mobile Operators, Movies, and Bibsonomy.

- As well as these algorithms are performed on some synthetic datasets and those datasets are Non-overlapping noised tri-contexts, Random uniform triple generation.

2.2.3 Tri-clustering approach for time evolving graph

- Framework used to perform these kind of task is MDL[52]
- Method $\rightarrow$ Here a parameter free, group vertices whose edges are similarly distributed over clusters is used. In addition this method will create partitions into time interval to time segment, during which the edge distributions between the clusters and stationary [53].

  1. Model Definition $\rightarrow$ As the group edges are evolving through time, we replace the synthetic representation by a unique image graph by a sequence of image graphs [54] [55]. Each image graph is supposed to be a synthetic representation of the graph on a specific time segment. Now that the different components of an image graph are introduced, their parametrization must be specified. A model characterizing an image graph is defined by $\rightarrow$
     - (a) The number of source and target clusters (KS and KT)
     - (b) The number of time segments (N)
     - (c) The partition of the source vertices (resp. target vertices)into the source (resp. target) clusters of vertices
     - (d) The distribution of the temporal edges of the graph on the co-clusters of source vertices, target vertices and time (i.e the edges of the image graph). Given this specification, we can derive from the graph the frequency of the clusters and time segments. Since time is a continuous variable, we can deduce the time segments bounds from their frequency
     - (e) For each source (resp. target) cluster of vertices, the distribution of the edges whose source (resp. target) belongs to the cluster on the vertices of the cluster
  2. MODL, the citation $\rightarrow$ Given the model definition, the method we use is similar to a co-clustering with 2 features [56] $\rightarrow$
     - (a) The source and the target vertices are grouped
     - (b) Time is discretized
  3. Algorithm $\rightarrow$ The criterion is minimized using a greedy bottom up merge heuristic. It starts from the finest image graph, i.e. the one with one cluster

per vertices and one interval per timestamp. The merge of source and target clusters and the merges between adjacent time intervals are evaluated and performed so that the criterion decreases.

4. Simplifying the image graph $\rightarrow$ When huge graphs are studied, the number of clusters of vertices and of time segments may be too high for an easy interpretation. This problem has been raised in, where an agglomerative method is suggested as an exploratory analysis tool. Here the method which is used consists in merging successively the clusters and the time segments in the least costly way until the image graph is synthetic enough for an easy interpretation. From an optimal image graph according to the criterion details, clusters of source vertices, of target vertices or time segments are merged sequentially.

5. At each step, the merged clusters (or time segments) are the ones that induce the smallest increase of the value of the criterion.

- The synthetic datasets on which these algoritms are applied, are Experiments on graphs with significative patterns, Experiments on stationary graphs, and Experiments on random graphs.

- The real-life dataset where these algorithms are used is The London cycle datasets.

2.2.4 FCA for triadic case

Contexts based on three sets and a ternary relationship [57][58][51].
As a first step, let us extend the concept of the formal context. Then, the tuple $\rightarrow$
$\kappa := (G, M, B, I)$ is called a triadic formal context.
G = sets of objects
M = sets of attributes
B = sets of conditions
I = ternary relationship and I should be $I \subseteq G \times M \times B$.
Then, the three $(g, m, b) \in I$ can be interpreted as the fact that the "object g has the attribute m under condition b."
Galois operators (prime operators) [59] $(\cdot)'$ in the triadic case map either from the Cartesian product of two sets to the remainder, or vice versa mentioned in below equations 2.21

and 2.22 $\rightarrow$

$$2^G \rightarrow 2^M \times 2^B; 2^M \rightarrow 2^G \times 2^B; 2^B \rightarrow 2^G \times 2^M \tag{2.21}$$

$$2^G \times 2^M \rightarrow 2^B; 2^G \times 2^B \rightarrow 2^M; 2^M \times 2^B \rightarrow 2^G \tag{2.22}$$

A tuple (X, Y, Z) can be called triadic formal concept if $\rightarrow$

$$(X, Y)' = Z \tag{2.23}$$

$$(X, Z)' = Y \tag{2.24}$$

$$(Y, Z)' = X \tag{2.25}$$

Set X = formal extent.
Set Y = formal intent.
Set Z = formal modus.
As in the dyadic case, the triadic formal concept is the maximal cuboid in the context. Double prime operators that, however, are not closure operators $(\cdot)'$ are also defined for the triadic context operators.

Chapter 3

Proposed Algorithms

3.1 Data Pre-processing Phase:

This is the first phase. Here a data pre-processing mechanism is stated (Algorithm 1). This algorithm first takes the raw dataset as input. Then in the first step, the all non-character or numeric features are converted to floating point number. In the next step, the user has to decide which mathematical operation can be performed on the dataset to get the desired output and using those output discretization application will be easier. These kinds of operations which are decided by the user is known as domain knowledge application. In the third step, equal frequency discretization is done. In step four, column-wise mean, max, and min value is calculated. Now finally in step five, the values which are equals to the max value of its column is replaced with '5', the values which are less than max value and greater the mean value of its column is replaced with '4', the values which are equals to the mean value of its column is replaced with '3', the values which are less than mean value and greater the min value of its column is replaced with '2', the values which are equals to the min value of its column is replaced with '1'. This is how data is pre-processed and labeled dataset is prepared.

```
Algorithm 1 Data Pre-processing
Input: Raw Dataset
Output: Pre-processed Dataset
begin:
Step 1:-All non-character features are converted to floating point number.
Step 2:-Domain Knowledge applied according to their requirement like division, multiplication, addition
or subtraction can be applied with respect to other feature on another feature [60].
Step 3:-Equal Frequency Discritization Method applied on resulting numerical value columns [61].
```

Step 4:-Column wise mean[62], max and min value calculated.
Step 5:-
If a value is equals to max value of that row then that value masked by '5'.
If a value is in between max and mean value of that row then that value masked by '4'.
If a value is equals to mean value of that row then that value masked by '3'.
If a value is in between mean and min value of that row then that value masked by '2'.
If a value is equals to min value of that row then that value masked by '1'.
end

3.2 Generate Sorted Frequent Dataset:

Here the dataset which generates sorted frequent pattern is shown (Algorithm 2). The first column of the dataset which contains the character value is deleted first (line 1). As the dataset contains only labeled data so, each attribute is combined with '1', '2', '3', '4', or '5' label and this combination of attribute and its label is known as Attribute:Value pair, which is created (line 2). Now calculate the frequency of each Attribute:Value pair in the dataset (line 3). Sort the Attribute:Value pairs in decreasing order according to their frequency column (line 4). Now, if frequency of one Attribute:Value pair is greater than equals to min-Support (user given input) or if the value part equals to 4 or 5 then only, one successive Item number will be added to that Attribute:Value pair otherwise if that Attribute:Value pair failed to overcome these criteria then deleted (line 5-9). Now traverse the pre-processed dataset row-wise. In the time of traversing if it is found that one attribute and its value combination is present in Sorted Frequent Item Table then, the Item Number corresponding to that Attribute:Value pair, will be added in ascending order to the corresponding row values in Sorted Frequent Dataset (line 10-14).

Algorithm 2 Generate Sorted Frequent Dataset

```
Input: Pre-processed Dataset, min-Support
Output: Sorted Frequent Dataset
 1: Delete character value column from pre-processed data set.
 2: Create Attribute:Value pair.
 3: Calculate each Attribute:Value pair's frequency from the pre-processed dataset.
 4: Sort the Attribute:Value pairs in decreasing order according to their frequency.
 5: if $(frequency \geq min - Support) \parallel (Value == 4 \parallel Value == 5)$ then
 6:     Add successive Item Numbers to each Attribute:Value pair.{Sorted Frequent Item Table is created}
 7: else
 8:     Delete Attribute:Value pair.
 9: end if
10: for all row R in pre-processed dataset do
11:     if at least one attribute and its value in R is in Sorted Frequent Item Table then
12:         write ordered items corresponding to R values in Sorted Frequent Dataset
13:     end if
14: end for
```

3.3 Construction of Frequent Generalized Suffix Tree:

In the suffix tree there are four types of node created.

1. **HTree (Suffix Tree)** This is the root node with some node pointer, which is equals to number of rows in SFD.

2. **HNode (Internal Node)** This is an internal node which have $SFD - 2$ open next item pointer, one closed object list pointer, and one data container which contains Item Number.

3. **HNode (Internal Node with Object List** This is an internal node which has next item node pointer and object list pointer both open. It has $SFD - 2$ open next item pointer, one open object list pointer and one data part for Item Number.

4. **HNode (Leaf Node)** This is the leaf node where the next item pointers are closed and only the object list pointers are open with data part.

These are the constructing part of the suffix tree data structure. Here a little bit modification done. Previously object list only has the data part, in this case object list has two parts. One data part and another condition list pointer part. But in this phase that condition list pointer part is closed because of no need (Figure 3.1).

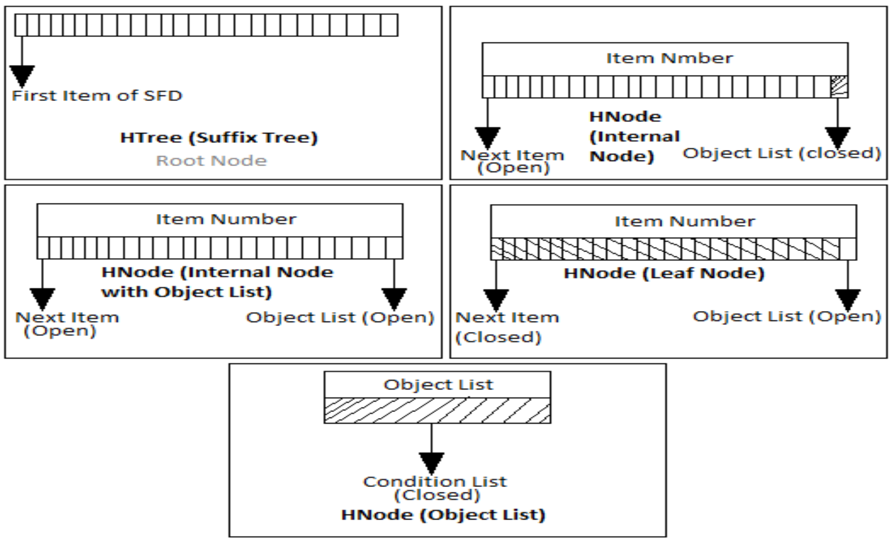


Figure 3.1: Structure of Nodes in FGIST

The algorithm for creating frequent generalized suffix tree is described below (Algorithm 3). First, this algorithm takes SFD and produce FGIST. A number is assigned to the total number of rows in SFD (say n) (line 1). Then n number of root nodes are created which points to the item numbers (line 2). Another number is taken which has n number of instances and each instance hold a number of items in each row in SFD (line 3-5). Now for each rows in SFD if the array pointer points to the first Item in item number then root node will directly pointed to that item and if the array pointer points to neither the first item, nor the last item then internal node or internal node with the object list or root node will points to that item (line 6-13). If it is the last item in the item list of a row then it will be pointed to a leaf node item which holds object list and also pointed from root node as well (line 14-18). Now there is an object associated with every row in the SFD and an object list will be created at the end of every row insertion at the end of every leaf node of the suffix tree (line 19-20). This is how the whole suffix tree is created. After this, we can easily observe that there are lots of proper subsets created for almost every object list. So, the proper subset object lists and there paths are deleted in the breadth-first left to right scanning manner (i.e. started scanning from left to right of a suffix tree in breadth-first manner, which object list scanned first will stay and it's proper subset which scanned later will be deleted along with its path from root to leaf) (line 21-31).

Algorithm 3 Construction of Frequent Generalized Suffix Tree

```
Input: Sorted Frequent Database
Output: Frequent Generalized Itemset Suffix Tree
 1: $n \leftarrow number$ of rows in $SFD$
 2: Create n root nodes.
 3: for $i = 1$ to $n$ do
 4:     $m_i \leftarrow number$ of Items in each rows in $SFD$.
 5: end for
 6: for $i = 1$ to $n$ do
 7:     for $j = 1$ to $|m_i|$ do
 8:         if $j == 1$ then
 9:             $HTree \leftarrow Item[j]$
10:         else if $j \neq 1 \parallel j \neq |m_i|$ then
11:             $HTree \leftarrow Item[j]$
12:             $HNode(InternalNode) \rightarrow next \leftarrow Item[j]$
13:             $HNode(InternalNodeWithObjectList) \rightarrow next \leftarrow Item[j]$
14:         else if $j == |m_i|$ then
15:             $HTree \leftarrow Item[j]$
16:             $HNode(LeafNode) \leftarrow Object[i]$
17:         end if
18:     end for
19:     $ObjectList \leftarrow Object[i]$
20: end for
21: for $HTree$ to $ObjectList$ do
22:     $current \leftarrow 1$
23:     $next \leftarrow current + 1$
24:     if $current.ObjectList == next.ObjectList$ then
25:         Delete HTree to ObjectList whole path.
26:         $next++$
27:     else
28:         $next++$
29:     end if
30:     $current++$
31: end for
```

3.4 Merging Two Suffix Tree Procedure:

Here the merging procedure of two suffix tree is described (Algorithm 7). This algorithm takes two suffix trees which need to be merged and their condition. The suffix tree, which has a bigger root node pointer, is considered to be the root node pointer of the resulting suffix tree (line 4-8). This algorithm starts matching node by node of two suffix trees (line 10-14).

In line 11 the algorithm call function 4 and it states $\rightarrow$ If internal node of first suffix tree and second suffix tree is same i.e. all the item number are same, then those item number will be added to the resulting suffix tree (line 2-4). Else if the internal node of first suffix tree is not same with second suffix tree i.e. different item numbers (line 5), then at first it will check if the item number of suffix tree 1 is greater then or not from item number of suffix tree 2 for a particular node. If suffix tree 1 item number is greater than suffix tree 2 item number then, suffix tree 2's item number will be added to the resulting suffix tree's next node. After that, suffix tree 1's item number will be added to the resulting suffix tree's next node (line 6-10). Again if the item number of suffix tree 2 is greater then the item number of suffix tree 1 for a particular node. Then, suffix tree 1's item number will be added to resulting suffix tree's next node. After that, suffix tree 2's item number will be added to the resulting suffix tree's next node (line 11-17).

Algorithm 4 Function for Internal Node Merging

```
1: begin:
2: if $HNode(InternalNode)_1 \rightarrow next \rightarrow Item[i] == HNode(InternalNode)_2 \rightarrow next \rightarrow Item[j]$ then
3:     $Item[k] = Item[i] = Item[j]$
4:     Add a new node with $Item[k]$ at $HNode(InternalNode) \rightarrow next$ or $HTree \rightarrow next$
5: else if $HNode(InternalNode)_1 \rightarrow next \rightarrow Item[i] \neq HNode(InternalNode)_2 \rightarrow next \rightarrow Item[j]$ then
6:     if $Item[i] > Item[j]$ then
7:         $Item[k] = Item[j]$
8:         Add a new node with $Item[k]$ at previous $HNode(InternalNode) \rightarrow next$ or $HTree \rightarrow next$
9:         $Item[k] = Item[i]$
10:        Add a new node with $Item[k]$ at previous $HNode(InternalNode) \rightarrow next$ or $HTree \rightarrow next$
11:    else if $Item[i] < Item[j]$ then
12:        $Item[k] = Item[i]$
13:        Add a new node with $Item[k]$ at previous $HNode(InternalNode) \rightarrow next$ or $HTree \rightarrow next$
14:        $Item[k] = Item[j]$
15:        Add a new node with $Item[k]$ at previous $HNode(InternalNode) \rightarrow next$ or $HTree \rightarrow next$
16:    end if
17: end if
```

In line 12 the algorithm call function 5 and it states $\rightarrow$ First, this algorithm will check the internal node with object list for suffix tree 1 is same as suffix tree 2 or not. If it is same i.e. the item number list is same, then that node will be to the resulting suffix tree (line 2-4). Else if the internal node with object list of first suffix tree is not same with second suffix tree i.e. different item numbers (line 5), then at first it will check if the item number of suffix tree 1 is greater then or not from item number of suffix tree 2 for a particular node. If suffix tree 1 item number is greater than suffix tree 2 item number then, suffix tree 2's item number will be added to the resulting suffix tree's next node. After that, suffix tree 1's item number will be added to the resulting suffix tree's next node (line 6-10). Again if the item number of suffix tree 2 is greater then the item number of suffix tree 1 for a particular node. Then, suffix tree 1's item number will be added to the resulting suffix tree's next node. After that, suffix tree 2's item number will be added to the resulting suffix tree's next node (line 11-17).

Algorithm 5 Function for Internal Node with Object List Merging

```
1: begin:
2: if $HNode(InternalNodeWithObjectList)_1 \rightarrow next \rightarrow Item[i] == HNode(InternalNodeWithObjectList)_2 \rightarrow next \rightarrow Item[j]$ then
3:     $Item[k] = Item[i] = Item[j]$
4:     Add a new node with $Item[k]$ at $HNode(InternalNode) \rightarrow next$ or $HTree \rightarrow next$ or $HNode(InternalNodeWithObjectList) \rightarrow next$
5: else if $HNode(InternalNodeWithObjectList)_1 \rightarrow next \rightarrow Item[i] \neq HNode(InternalNodeWithObjectList)_2 \rightarrow next \leftarrow Item[j]$ then
6:     if $Item[i] > Item[j]$ then
7:         $Item[k] = Item[j]$
8:         Add a new node with $Item[k]$ at previous $HNode(InternalNodeWithObjectList) \rightarrow next$ or $HNode(InternalNode) \rightarrow next$ or $HTree \rightarrow next$
9:         $Item[k] = Item[i]$
10:        Add a new node with $Item[k]$ at previous $HNode(InternalNodeWithObjectList) \rightarrow next$ or $HNode(InternalNode) \rightarrow next$ or $HTree \rightarrow next$
11:    else if $Item[i] < Item[j]$ then
12:        $Item[k] = Item[i]$
13:        Add a new node with $Item[k]$ at previous $HNode(InternalNodeWithObjectList) \rightarrow next$ or $HNode(InternalNode) \rightarrow next$ or $HTree \rightarrow next$
14:        $Item[k] = Item[j]$
15:        Add a new node with $Item[k]$ at previous $HNode(InternalNodeWithObjectList) \rightarrow next$ or $HNode(InternalNode) \rightarrow next$ or $HTree \rightarrow next$
16:    end if
17: end if
```

In line 13 the algorithm call function 6 and it states $\rightarrow$ If leaf node of suffix tree 1 is same as leaf node of suffix tree 2 or not. If it is same then that item number will be added to the resulting suffix tree. Now it will check whether the object list connected with leaf node of suffix tree 2 is a proper subset of the object list connected with leaf node of suffix tree 1 or not. If it is true then the intersection of those two object list will be added to the resulting suffix tree as well as condition list connected with those object list will be merged and placed into the resulting suffix tree's condition list; if it is false then the union of those two object list will be added to the resulting suffix tree as well as condition list connected with those object list will be merged and placed into the resulting suffix tree's condition list (line 2-11). Now if the leaf node of suffix tree 1 is not same with leaf node with suffix tree 2 (line 12), then it will check suffix tree 1's item number is greater than suffix tree 2's item number or not. If it is true then suffix tree 2's leaf node will be added to the resulting suffix tree, suffix tree 2's object list will be added to the resulting suffix tree and suffix tree 2's condition list will be added to the resulting suffix tree (line 13-16); after that suffix tree 1's leaf node will be added to the resulting suffix tree, suffix tree 1's object list will be added to the resulting suffix tree and suffix tree 1's condition list will be added to the resulting suffix tree (line 18-19). If it is false then suffix tree 1's leaf node will be added to the resulting suffix tree, suffix tree 1's object list will be added to the resulting suffix tree and suffix tree 1's condition list will be added to the resulting suffix tree (line 20-24); after that suffix tree 2's leaf node will be added to the resulting suffix tree, suffix tree 2's object list will be added to the resulting suffix tree and suffix tree 2's condition list will be added to the resulting suffix tree (line 26-30).

Algorithm 6 Function for Leaf Node Merging

```
1: begin:
2: if $HNode(LeafNode)_1 \to next \to Item[i] == HNode(LeafNode)_2 \to next \to Item[j]$ then
3:     $Item[k] = Item[i] = Item[j]$
4:     Add a new node with $Item[k]$ at $HNode(InternalNodeWithObjectList) \to next$ or $HNode(InternalNode) \to next$ or $HTree \to next$
5:     if $HNode(LeafNode)_1 \to next \to ObjectList_1 \subset HNode(LeafNode)_2 \to next \to ObjectList_2$ then
6:         $HNode(LeafNode) \to next \leftarrow ((ObjectList_1 \cap ObjectList_2) \equiv NewObjectList)$
7:         $NewObjectList \to next \leftarrow ((Condition_1 \cup Condition_2) \equiv ConditionList)$
8:     else if $HNode(LeafNode)_1 \to next \to ObjectList_1 \neq HNode(LeafNode)_2 \to next \to ObjectList_2$ then
9:         $HNode(LeafNode) \to next \leftarrow ((ObjectList_1 \cup ObjectList_2) \equiv NewObjectList)$
10:        $NewObjectList \to next \leftarrow ((Condition_1 \cup Condition_2) \equiv ConditionList)$
11:    end if
12: else if $HNode(LeafNode)_1 \to next \to Item[i] \neq HNode(LeafNode)_2 \to next \leftarrow Item[j]$ then
13:    if $Item[i] > Item[j]$ then
14:        $Item[k] = Item[j]$
15:        Add a new node with $Item[k]$ at previous $HNode(InternalNodeWithObjectList) \to next$ or $HNode(InternalNode) \to next$ or $HTree \to next$
16:        $HNode(LeafNode) \to next \leftarrow (ObjectList_2 \equiv NewObjectList)$
               $NewObjectList \to next \leftarrow (Condition_2 \equiv ConditionList)$
17:        $Item[k] = Item[i]$
18:        Add a new node with $Item[k]$ at previous $HNode(InternalNodeWithObjectList) \to next$ or $HNode(InternalNode) \to next$ or $HTree \to next$
19:        $HNode(LeafNode) \to next \leftarrow (ObjectList_1 \equiv NewObjectList)$
               $NewObjectList \to next \leftarrow (Condition_1 \equiv ConditionList)$
20:    else if $Item[i] < Item[j]$ then
21:        $Item[k] = Item[i]$
22:        Add a new node with $Item[k]$ at previous $HNode(InternalNodeWithObjectList) \to next$ or $HNode(InternalNode) \to next$ or $HTree \to next$
23:        $HNode(LeafNode) \to next \leftarrow (ObjectList_1 \equiv NewObjectList)$
24:        $NewObjectList \to next \leftarrow (Condition_1 \equiv ConditionList)$
25:        $Item[k] = Item[j]$
26:        Add a new node with $Item[k]$ at previous $HNode(InternalNodeWithObjectList) \to next$ or $HNode(InternalNode) \to next$ or $HTree \to next$
27:        $HNode(LeafNode) \to next \leftarrow (ObjectList_2 \equiv NewObjectList)$
28:        $NewObjectList \to next \leftarrow (Condition_2 \equiv ConditionList)$
29:    end if
30: end if
```

In this way the resulting suffix tree will be created. After creation of the suffix tree, we can easily observe that there are lots of proper subset created for almost every object list. So, the proper subset object lists and there paths are deleted in the breadth first left to right scanning manner (i.e. started scanning from left to right of a suffix tree in breadth first manner, which object list scanned first will stay and it's proper subset which scanned later will be deleted along with it's path from root to leaf) and the condition list associated with that deleted object list will be added to the current condition list (line 15-26).

Algorithm 7 Merging Two Suffix Tree Algorithm

```
Input: Two Suffix Tree, Conditions
Output: Merged Suffix Tree with Condition List
 1: $n_1 \leftarrow number$ of root node pointer in $Suffix_1$.
 2: $n_2 \leftarrow number$ of root node pointer in $Suffix_2$.
 3: $n \leftarrow number$ of root node pointer in $resultingSuffix$.
 4: if $n_1 > n_2$ then
 5:     $n \leftarrow n_1$
 6: else
 7:     $n \leftarrow n_2$
 8: end if
 9: Create $n$ root nodes.
10: for $i = 1\&\&j = 1\&\&k = 1$ to $n_1\&\&n_2\&\&n$ do
11:     CALL function 4
12:     CALL function 5
13:     CALL function 6
14: end for
15: for $HTree$ to $NewObjectList$ do
16:     $current \leftarrow 1$
17:     $next \leftarrow current + 1$
18:     if $current.NewObjectList == next.NewObjectList$ then
19:         Delete HTree to ObjectList whole path.
20:         Add $next.ConditionList$ to $current.ConditionList$
21:         $next++$
22:     else
23:         $next++$
24:     end if
25:     $current++$
26: end for
```

3.5 Extracting Tri-clusters from Suffix Tree:

After the creation of the merged suffix tree, we need to extract the clusters. These clusters hold three types of datatype. First one is Item Number, Second one is Object and the third data type is Year. So, one table with three columns can be created to hold these datatypes in separate columns. The extraction procedure is described below (Algorithm 8). First a tri-cluster table is created with columns $Item, ObjectList, ConditionList, Attribute : Value, Tri-cluster$ (line 1-2). Then the number of leaf nodes are counted from completely merged suffix tree (line 3). Now, traverse for first leaf node to last leaf node in breadth-first manner and add condition list of suffix tree to condition list of tri-cluster table, add object list of suffix tree to object list of tri-cluster table, add item of suffix tree to item of the tri-cluster table (line 4-9). Now, one number is taken which describes rows in the tri-cluster table (line 10). For each row number of item is counted and stored into a variable (line 11-12). The number of rows in Item number added with sorted Frequent Item Table is counted and stored into a variable (line 13-14). Now, if each item number in each row of the tri-cluster table is present in Item number added with sorted Frequent Item Table then, the corresponding Attribute:Value pair of Item number added with sorted Frequent Item Table will be added to the tri-cluster table (line 15-21). Now, for each row of tri-cluster table concatenate $Attribute : Value, ObjectList, ConditionList$ columns and add them to Tri-cluster column in tri-cluster table (line 22-24).

Algorithm 8 Construction of Tricluster Table

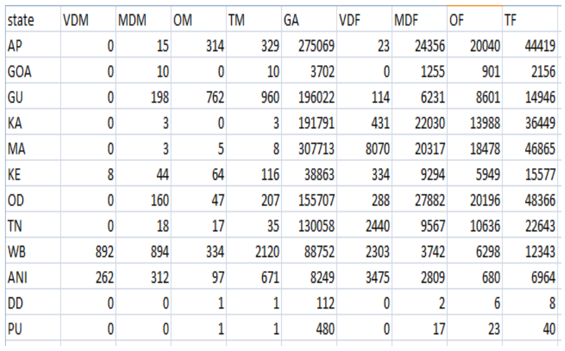
```
Input: Completely merged suffix tree, Item Number added with sorted Frequent Item Table with
    $minSupport$
Output: Tri-cluster Table
 1: Create a $Tri - clusterTable$
 2: Create columns:- $Item, ObjectList, ConditionList, Attribute : Value, Tri - cluster$
 3: $n \leftarrow number$ of $LeafNode$ in Completely merged suffix tree
 4: for $i = 1$ to $n$ do
 5:     Traverse through $HNode(LeafNode)$ to $HTree$
 6:     Add ConditionList to $ConditionList$
 7:     Add ObjectList to $ObjectList$
 8:     Add Item to $Item$
 9: end for
10: $m \leftarrow number$ of rows in Tri-cluster table
11: for $j = 1$ to $m$ do
12:     $o \leftarrow number$ of items in a row of Tri-cluster Table
13:     for $k = 1$ to $o$ do
14:         $p \leftarrow number$ of rows in Item Number added with sorted Frequent Item Table with $minSupport$

15:         for $l = 1$ to $p$ do
16:             if $Item[j][k] == ItemNumber[l]$ then
17:                 Add $Attribute : Value[l]$ into $Attribute : Value[j][k]$
18:             end if
19:         end for
20:     end for
21: end for
22: for $i = 1$ to $m$ do
23:     Concatenate $Attribute : Value, ObjectList, ConditionList$ and Add them to $Tri - cluster$
24: end for
```

Chapter 4

Explain with Dataset

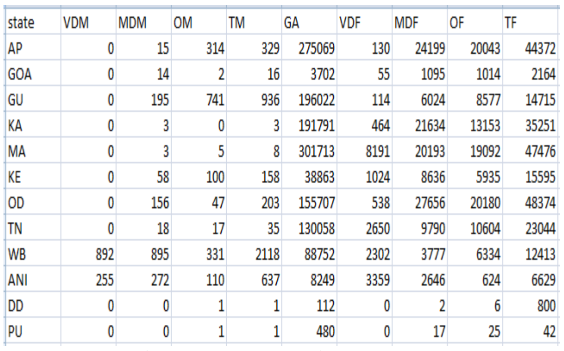
4.1 Year Wise Main Data Set Representation:

Here seven dataset are presented as seven years data for Indian state wise various forests' area measured (Figure 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7) [63]. The years are 2003, 2005, 2009, 2011, 2013, 2015 and 2017. These are used as 'condition' dimension. The 'attributes' are Very Dense Mangrove (VDM), Medium Dense Mangrove (MDM), Open Mangrove (OM), Total Mangrove (TM), Geographic Area (GA), Very Dense Forest (VDF), Medium Dense Forest (MDF), Open Forest (OF) and Total Forest (TF). The 'objects' are Andhra Pradesh (AP), Goa, Gujarat (GU), Karnataka (KA), Maharashtra (MA), Kerala (KE), Orissa (OD), Tamil Nadu (TN), West Bengal (WB), Andaman and Nicobar Iceland (ANI), Daman and Diu (DD) and Pondicherry (PU).



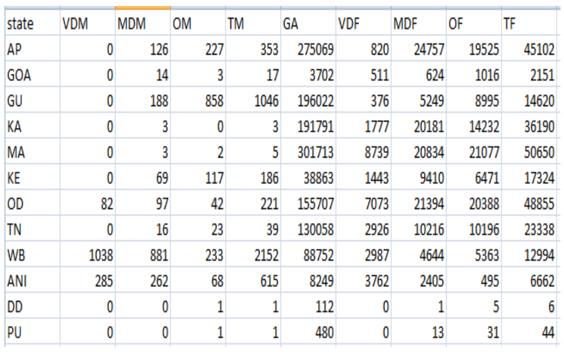
state	VDM	MDM	OM	TM	GA	VDF	MDF	OF	TF
AP	0	15	314	329	275069	23	24356	20040	44419
GOA	0	10	0	10	3702	0	1255	901	2156
GU	0	198	762	960	196022	114	6231	8601	14946
KA	0	3	0	3	191791	431	22030	13988	36449
MA	0	3	5	8	307713	8070	20317	18478	46865
KE	8	44	64	116	38863	334	9294	5949	15577
OD	0	160	47	207	155707	288	27882	20196	48366
TN	0	18	17	35	130058	2440	9567	10636	22643
WB	892	894	334	2120	88752	2303	3742	6298	12343
ANI	262	312	97	671	8249	3475	2809	680	6964
DD	0	0	1	1	112	0	2	6	8
PU	0	0	1	1	480	0	17	23	40

Figure 4.1: 2003's Dataset



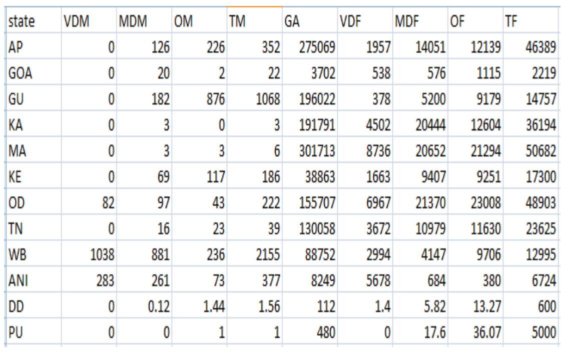
state	VDM	MDM	OM	TM	GA	VDF	MDF	OF	TF
AP	0	15	314	329	275069	130	24199	20043	44372
GOA	0	14	2	16	3702	55	1095	1014	2164
GU	0	195	741	936	196022	114	6024	8577	14715
KA	0	3	0	3	191791	464	21634	13153	35251
MA	0	3	5	8	301713	8191	20193	19092	47476
KE	0	58	100	158	38863	1024	8636	5935	15595
OD	0	156	47	203	155707	538	27656	20180	48374
TN	0	18	17	35	130058	2650	9790	10604	23044
WB	892	895	331	2118	88752	2302	3777	6334	12413
ANI	255	272	110	637	8249	3359	2646	624	6629
DD	0	0	1	1	112	0	2	6	800
PU	0	0	1	1	480	0	17	25	42

Figure 4.2: 2005's Dataset



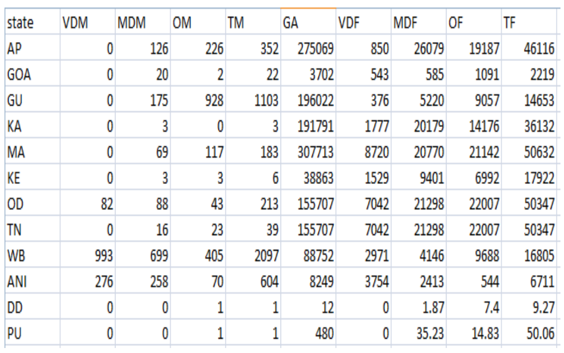
state	VDM	MDM	OM	TM	GA	VDF	MDF	OF	TF
AP	0	126	227	353	275069	820	24757	19525	45102
GOA	0	14	3	17	3702	511	624	1016	2151
GU	0	188	858	1046	196022	376	5249	8995	14620
KA	0	3	0	3	191791	1777	20181	14232	36190
MA	0	3	2	5	301713	8739	20834	21077	50650
KE	0	69	117	186	38863	1443	9410	6471	17324
OD	82	97	42	221	155707	7073	21394	20388	48855
TN	0	16	23	39	130058	2926	10216	10196	23338
WB	1038	881	233	2152	88752	2987	4644	5363	12994
ANI	285	262	68	615	8249	3762	2405	495	6662
DD	0	0	1	1	112	0	1	5	6
PU	0	0	1	1	480	0	13	31	44

Figure 4.3: 2009's Dataset



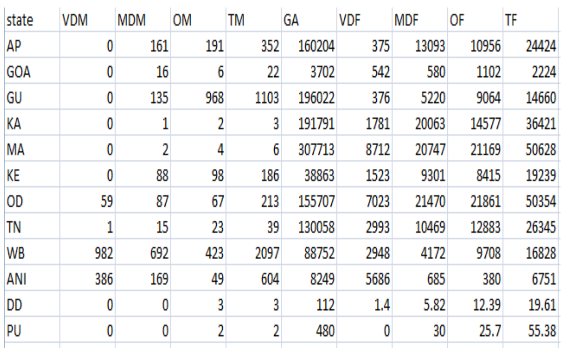
state	VDM	MDM	OM	TM	GA	VDF	MDF	OF	TF
AP	0	126	226	352	275069	1957	14051	12139	46389
GOA	0	20	2	22	3702	538	576	1115	2219
GU	0	182	876	1068	196022	378	5200	9179	14757
KA	0	3	0	3	191791	4502	20444	12604	36194
MA	0	3	3	6	301713	8736	20652	21294	50682
KE	0	69	117	186	38863	1663	9407	9251	17300
OD	82	97	43	222	155707	6967	21370	23008	48903
TN	0	16	23	39	130058	3672	10979	11630	23625
WB	1038	881	236	2155	88752	2994	4147	9706	12995
ANI	283	261	73	377	8249	5678	684	380	6724
DD	0	0.12	1.44	1.56	112	1.4	5.82	13.27	600
PU	0	0	1	1	480	0	17.6	36.07	5000

Figure 4.4: 2011's Dataset



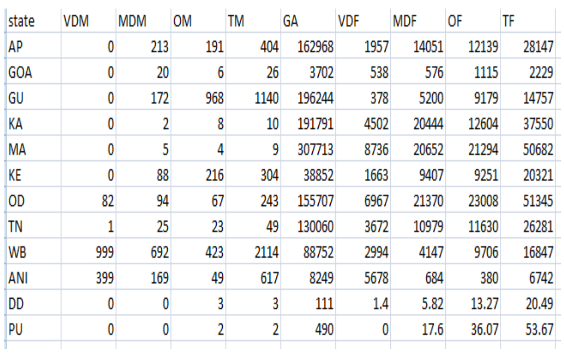
state	VDM	MDM	OM	TM	GA	VDF	MDF	OF	TF
AP	0	126	226	352	275069	850	26079	19187	46116
GOA	0	20	2	22	3702	543	585	1091	2219
GU	0	175	928	1103	196022	376	5220	9057	14653
KA	0	3	0	3	191791	1777	20179	14176	36132
MA	0	69	117	183	307713	8720	20770	21142	50632
KE	0	3	3	6	38863	1529	9401	6992	17922
OD	82	88	43	213	155707	7042	21298	22007	50347
TN	0	16	23	39	155707	7042	21298	22007	50347
WB	993	699	405	2097	88752	2971	4146	9688	16805
ANI	276	258	70	604	8249	3754	2413	544	6711
DD	0	0	1	1	12	0	1.87	7.4	9.27
PU	0	0	1	1	480	0	35.23	14.83	50.06

Figure 4.5: 2013's Dataset



state	VDM	MDM	OM	TM	GA	VDF	MDF	OF	TF
AP	0	161	191	352	160204	375	13093	10956	24424
GOA	0	16	6	22	3702	542	580	1102	2224
GU	0	135	968	1103	196022	376	5220	9064	14660
KA	0	1	2	3	191791	1781	20063	14577	36421
MA	0	2	4	6	307713	8712	20747	21169	50628
KE	0	88	98	186	38863	1523	9301	8415	19239
OD	59	87	67	213	155707	7023	21470	21861	50354
TN	1	15	23	39	130058	2993	10469	12883	26345
WB	982	692	423	2097	88752	2948	4172	9708	16828
ANI	386	169	49	604	8249	5686	685	380	6751
DD	0	0	3	3	112	1.4	5.82	12.39	19.61
PU	0	0	2	2	480	0	30	25.7	55.38

Figure 4.6: 2015's Dataset



state	VDM	MDM	OM	TM	GA	VDF	MDF	OF	TF
AP	0	213	191	404	162968	1957	14051	12139	28147
GOA	0	20	6	26	3702	538	576	1115	2229
GU	0	172	968	1140	196244	378	5200	9179	14757
KA	0	2	8	10	191791	4502	20444	12604	37550
MA	0	5	4	9	307713	8736	20652	21294	50682
KE	0	88	216	304	38852	1663	9407	9251	20321
OD	82	94	67	243	155707	6967	21370	23008	51345
TN	1	25	23	49	130060	3672	10979	11630	26281
WB	999	692	423	2114	88752	2994	4147	9706	16847
ANI	399	169	49	617	8249	5678	684	380	6742
DD	0	0	3	3	111	1.4	5.82	13.27	20.49
PU	0	0	2	2	490	0	17.6	36.07	53.67

Figure 4.7: 2017's Dataset

4.2 Data Pre-processing:

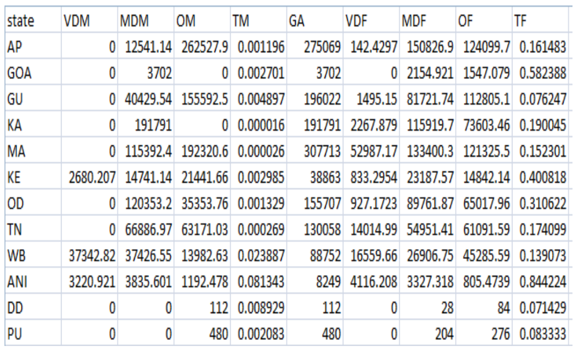
4.2.1 Domain Knowledge Application

After converting 2003's data set into floating point number we become able to apply domain knowledge upon this data set. The geographic area of one state will never be change so, if total mangrove and total forest area will be divided by geographic area then a perfect fractional overview of total forest and total mangrove area with respect to that state's geographic area can be calculated.

In the same way, the fractional overview can be achieved for very dense mangrove, medium dense mangrove and open mangrove area by dividing them with total mangrove area. In the other hand, the fractional overview can be achieved for very dense forest, medium dense forest and open forest area by dividing them with total forest area.
There are three techniques to handle categorical feature:-

1. Mean Replacement
2. One Hot Encoding[64]
3. Domain Knowledge

All this procedures are useful for extraction of important features from a data set. In this data set mean replacement is failed to produce any specific information. In the same way, one hot encoding will create sparse and large vectors, which is also not useful in this case. But there is no specific task in the procedure domain knowledge. Here the user can apply any kind of mathematical work upon the data set to extract necessary useful information (Figure 4.8).



state	VDM	MDM	OM	TM	GA	VDF	MDF	OF	TF
AP	0	12541.14	262527.9	0.001196	275069	142.4297	150826.9	124099.7	0.161483
GOA	0	3702	0	0.002701	3702	0	2154.921	1547.079	0.582388
GU	0	40429.54	155592.5	0.004897	196022	1495.15	81721.74	112805.1	0.076247
KA	0	191791	0	0.000016	191791	2267.879	115919.7	73603.46	0.190045
MA	0	115392.4	192320.6	0.000026	307713	52987.17	133400.3	121325.5	0.152301
KE	2680.207	14741.14	21441.66	0.002985	38863	833.2954	23187.57	14842.14	0.400818
OD	0	120353.2	35353.76	0.001329	155707	927.1723	89761.87	65017.96	0.310622
TN	0	66886.97	63171.03	0.000269	130058	14014.99	54951.41	61091.59	0.174099
WB	37342.82	37426.55	13982.63	0.023887	88752	16559.66	26906.75	45285.59	0.139073
ANI	3220.921	3835.601	1192.478	0.081343	8249	4116.208	3327.318	805.4739	0.844224
DD	0	0	112	0.008929	112	0	28	84	0.071429
PU	0	0	480	0.002083	480	0	204	276	0.083333

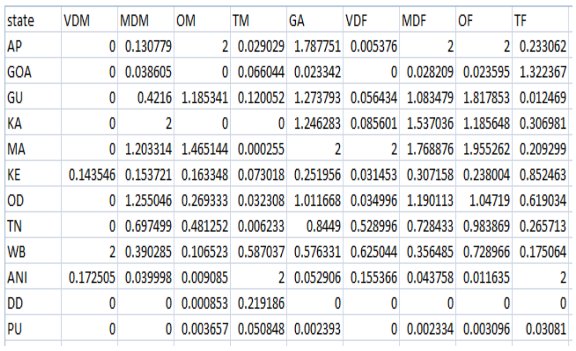
Figure 4.8: 2003's Dataset after applying Domain Knowledge

4.2.2 Equal Frequency Discretization Application

Discretization is used to divide the range of the continuous attribute into intervals (Figure 4.9). Every interval is labeled a discrete value, and then the original data will be mapped to the discrete values. This is an unsupervised learning algorithm, which put same number of values in each interval.
Every interval will have $\frac{n}{k}$ values.
n= n points in the whole range of attribute value.
k= k number of intervals.

1. Calculate split points, decide initial divide intervals.
2. Merge the initial divided intervals. That is to merge the intervals, which contains few records, into the nearest interval.
3. Map values of the continuous attribute into discrete values with respect to the divided intervals.

   - The number of the initial divided intervals k is between with $\lceil \log n \rceil$ and $\lceil \log n - \log \log n \rceil + 1$, if $\lceil \log n - \log \log n \rceil > 8$, then let $k = 9$ i.e. k is no more than 9. In latter experiment, we set:-
     $k = min\{\lceil \log n - \log \log n \rceil + 1, 9\}$
     $\therefore k = min\{\lceil \log 9 - \log \log 9 \rceil + 1, 9\}[n = 9]$
     $\Rightarrow k = min\{2, 9\}$
     $\Rightarrow k = 2$
   - When an interval contains fewer records than a fraction of the frequency $1k = MinF$, the interval need to be merged because of its less frequency. MinF can be set in the range [0.3, 0.5]. In our case, fraction of $frequency = \frac{1}{k} = \frac{1}{2} = MinF = 0.5$.
   - There are two different strategies t merge the object in low-frequency interval.
     Method 1:-This method is used to see the interval as a unit to be merged into the nearest interval.
     Method 2:-This is used to merge every object in the interval one by one into the nearest interval.

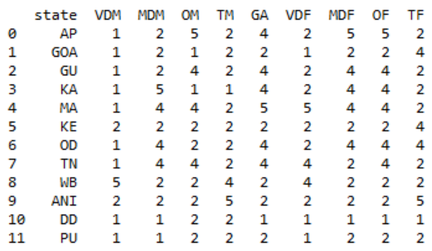


state	VDM	MDM	OM	TM	GA	VDF	MDF	OF	TF
AP	0	0.130779	2	0.029029	1.787751	0.005376	2	2	0.233062
GOA	0	0.038605	0	0.066044	0.023342	0	0.028209	0.023595	1.322367
GU	0	0.4216	1.185341	0.120052	1.273793	0.056434	1.083479	1.817853	0.012469
KA	0	2	0	0	1.246283	0.085601	1.537036	1.185648	0.306981
MA	0	1.203314	1.465144	0.000255	2	2	1.768876	1.955262	0.209299
KE	0.143546	0.153721	0.163348	0.073018	0.251956	0.031453	0.307158	0.238004	0.852463
OD	0	1.255046	0.269333	0.032308	1.011668	0.034996	1.190113	1.04719	0.619034
TN	0	0.697499	0.481252	0.006233	0.8449	0.528996	0.728433	0.983869	0.265713
WB	2	0.390285	0.106523	0.587037	0.576331	0.625044	0.356485	0.728966	0.175064
ANI	0.172505	0.039998	0.009085	2	0.052906	0.155366	0.043758	0.011635	2
DD	0	0	0.000853	0.219186	0	0	0	0	0
PU	0	0	0.003657	0.050848	0.002393	0	0.002334	0.003096	0.03081

Figure 4.9: 2003's Dataset after applying Equal Frequency Discretization

4.2.3 Mean Value Replacement Application

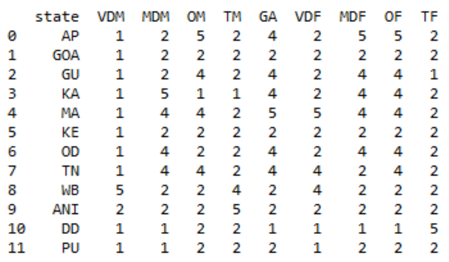
Mean value replacement is usually used to replace a specific range of values into '1' or '0's. In general the values of a column which are greater than or equals to the mean value of that column are replaced by 1's and those are less that mean value are replaced with 0's. But in this particular project some levels are given to some specific range of values. First max, min and mean values are calculated for each and every numerical value column. The values which matches with the maximum value of that column are replaced by '5'. The values which ranges between maximum and mean value of that column are replaced by '4'. The values which matches with mean value of that column are replaced by '3'. The values which ranges between mean and minimum value of that column are replaced by '2'. The values which matches with the maximum value of that column are replaced by '1'. Basically all the above mentioned data pre-processing phases are comes under feature engineering (Figure 4.10).



```
   state  VDM  MDM  OM  TM  GA  VDF  MDF  OF  TF
0     AP    1    2   5   2   4    2    5   5   2
1    GOA    1    2   1   2   2    1    2   2   4
2     GU    1    2   4   2   4    2    4   4   2
3     KA    1    5   1   1   4    2    4   4   2
4     MA    1    4   4   2   5    5    4   4   2
5     KE    2    2   2   2   2    2    2   2   4
6     OD    1    4   2   2   4    2    4   4   4
7     TN    1    4   4   2   4    4    2   4   2
8     WB    5    2   2   4   2    4    2   2   2
9    ANI    2    2   2   5   2    2    2   2   5
10    DD    1    1   2   2   1    1    1   1   1
11    PU    1    1   2   2   2    1    2   2   2
```

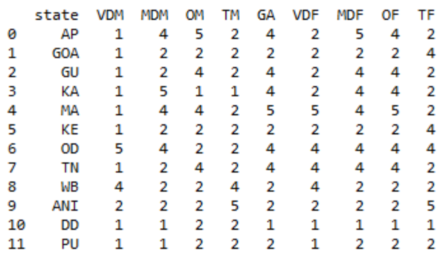
Figure 4.10: 2003's Dataset after applying Mean Value Replacement

The above mentioned data pre-processing phases are applied to 2005, 2009, 2011, 2013, 2015 and 2017 dataset also. And the resulting datasets are (Figure 4.11, 4.12, 4.13, 4.14, 4.15, 4.16):-



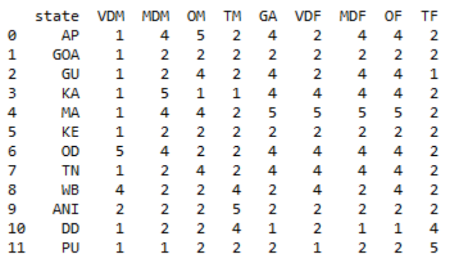
```
   state  VDM  MDM  OM  TM  GA  VDF  MDF  OF  TF
0     AP    1    2   5   2   4    2    5   5   2
1    GOA    1    2   2   2   2    2    2   2   2
2     GU    1    2   4   2   4    2    4   4   1
3     KA    1    5   1   1   4    2    4   4   2
4     MA    1    4   4   2   5    5    4   4   2
5     KE    1    2   2   2   2    2    2   2   2
6     OD    1    4   2   2   4    2    4   4   2
7     TN    1    4   4   2   4    4    2   4   2
8     WB    5    2   2   4   2    4    2   2   2
9    ANI    2    2   2   5   2    2    2   2   2
10    DD    1    1   2   2   1    1    1   1   5
11    PU    1    1   2   2   2    1    2   2   2
```

Figure 4.11: 2005's Dataset after applying Mean Value Replacement



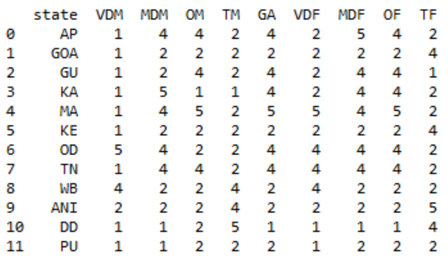
```
   state  VDM  MDM  OM  TM  GA  VDF  MDF  OF  TF
0     AP    1    4   5   2   4    2    5   4   2
1    GOA    1    2   2   2   2    2    2   2   4
2     GU    1    2   4   2   4    2    4   4   2
3     KA    1    5   1   1   4    2    4   4   2
4     MA    1    4   4   2   5    5    4   5   2
5     KE    1    2   2   2   2    2    2   2   4
6     OD    5    4   2   2   4    4    4   4   4
7     TN    1    2   4   2   4    4    4   4   2
8     WB    4    2   2   4   2    4    2   2   2
9    ANI    2    2   2   5   2    2    2   2   5
10    DD    1    1   2   2   1    1    1   1   1
11    PU    1    1   2   2   2    1    2   2   2
```

Figure 4.12: 2009's Dataset after applying Mean Value Replacement



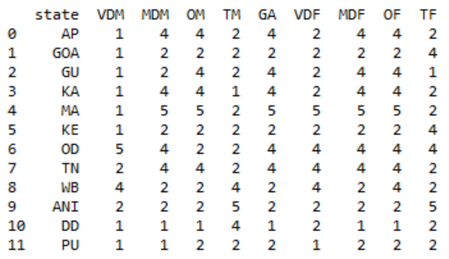
```
   state  VDM  MDM  OM  TM  GA  VDF  MDF  OF  TF
0     AP    1    4   5   2   4    2    4   4   2
1    GOA    1    2   2   2   2    2    2   2   2
2     GU    1    2   4   2   4    2    4   4   1
3     KA    1    5   1   1   4    4    4   4   2
4     MA    1    4   4   2   5    5    5   5   2
5     KE    1    2   2   2   2    2    2   2   2
6     OD    5    4   2   2   4    4    4   4   2
7     TN    1    2   4   2   4    4    4   4   2
8     WB    4    2   2   4   2    4    2   4   2
9    ANI    2    2   2   5   2    2    2   2   2
10    DD    1    2   2   4   1    2    1   1   4
11    PU    1    1   2   2   2    1    2   2   5
```

Figure 4.13: 2011's Dataset after applying Mean Value Replacement



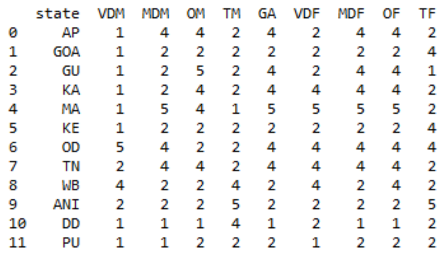
```
   state  VDM  MDM  OM  TM  GA  VDF  MDF  OF  TF
0     AP    1    4   4   2   4    2    5   4   2
1    GOA    1    2   2   2   2    2    2   2   4
2     GU    1    2   4   2   4    2    4   4   1
3     KA    1    5   1   1   4    2    4   4   2
4     MA    1    4   5   2   5    5    4   5   2
5     KE    1    2   2   2   2    2    2   2   4
6     OD    5    4   2   2   4    4    4   4   2
7     TN    1    4   4   2   4    4    4   4   2
8     WB    4    2   2   4   2    4    2   2   2
9    ANI    2    2   2   4   2    2    2   2   5
10    DD    1    1   2   5   1    1    1   1   4
11    PU    1    1   2   2   2    1    2   2   2
```

Figure 4.14: 2013's Dataset after applying Mean Value Replacement



```
   state  VDM  MDM  OM  TM  GA  VDF  MDF  OF  TF
0     AP    1    4   4   2   4    2    4   4   2
1    GOA    1    2   2   2   2    2    2   2   4
2     GU    1    2   4   2   4    2    4   4   1
3     KA    1    4   4   1   4    2    4   4   2
4     MA    1    5   5   2   5    5    5   5   2
5     KE    1    2   2   2   2    2    2   2   4
6     OD    5    4   2   2   4    4    4   4   4
7     TN    2    4   4   2   4    4    4   4   2
8     WB    4    2   2   4   2    4    2   4   2
9    ANI    2    2   2   5   2    2    2   2   5
10    DD    1    1   1   4   1    2    1   1   2
11    PU    1    1   2   2   2    1    2   2   2
```

Figure 4.15: 2015's Dataset after applying Mean Value Replacement



```
   state  VDM  MDM  OM  TM  GA  VDF  MDF  OF  TF
0     AP    1    4   4   2   4    2    4   4   2
1    GOA    1    2   2   2   2    2    2   2   4
2     GU    1    2   5   2   4    2    4   4   1
3     KA    1    2   4   2   4    4    4   4   2
4     MA    1    5   4   1   5    5    5   5   2
5     KE    1    2   2   2   2    2    2   2   4
6     OD    5    4   2   2   4    4    4   4   4
7     TN    2    4   4   2   4    4    4   4   2
8     WB    4    2   2   4   2    4    2   4   2
9    ANI    2    2   2   5   2    2    2   2   5
10    DD    1    1   1   4   1    2    1   1   2
11    PU    1    1   2   2   2    1    2   2   2
```

Figure 4.16: 2017's Dataset after applying Mean Value Replacement

4.3 Generate Sorted Frequent Dataset:

4.3.1 Item Table Creation

In Algorithm 2 first all the attribute:value pair is generated and their support is counted from the previously pre-processed data set. There are two columns in that table, first column hold Attribute:Value pair and the next column contains their support which is counted from previous pre-processed dataset is applied on 2003's pre-processsed dataset(Table 4.1).

Table 4.1: Item Table for 2003

Attribute:Value	**Support**	**Attribute:Value**	**Support**
VDM:1	9	GA:2	5
VDM:2	2	GA:5	1
VDM:5	1	GA:1	1
MDM:1	2	VDF:2	6
MDM:2	6	VDF:1	3
MDM:4	3	VDF:4	2
MDM:5	1	VDF:5	1
OM:5	1	MDF:5	1
OM:1	2	MDF:2	6
OM:2	6	MDF:4	4

Continued on next page

Table 4.1 – continued from previous page

Attribute:Value	Support	Attribute:Value	Support
OM:4	3	MDF:1	1
TM:2	9	OF:5	1
TM:1	1	OF:2	5
TM:4	1	OF:4	5
TM:5	1	OF:1	1
GA:4	5	TF:2	7
TF:4	3	TF:1	1
TF:5	1		

4.3.2 Frequent Item Table Creation with Minimum Support

A min-support threshold should be determined by the user. In this case, suppose the min-support threshold set to 33.33% that is $\frac{4}{12}$ for these datasets. After that, which attribute:value pair failed to meet the min support threshold will be deleted. Only the values '4' and '5' associated with attributes, those attribute:value pairs will not be deleted irrespective of their frequency. This is an exceptional case. These Attribute:Value need to be kept to avoid important information loss. These procedure applied on 2003's Item Table (Table 4.2).

Table 4.2: Frequent Item Table for 2003 with $min - support = \frac{4}{12}$

Attribute:Value	Support	Attribute:Value	Support
VDM:1	9	GA:5	1
VDM:5	1	VDF:2	6
MDM:2	6	VDF:4	2
MDM:4	3	VDF:5	1
MDM:5	1	MDF:5	1
OM:5	1	MDF:2	6
OM:2	6	MDF:4	4
OM:4	3	OF:5	1
TM:2	9	OF:2	5
TM:4	1	OF:4	5
TM:5	1	TF:2	7
GA:4	5	TF:4	3
GA:2	5	TF:5	1

4.3.3 Item Number Addition

All Attribute:Value pairs are sorted in decreasing order of their support and one item number is associated with them in increasing order successive manner applied on 2003's Frequent Item Table (Table 4.3).

Table 4.3: Item Number added with sorted Frequent Item Table for 2003 with $min{-}support = \frac{4}{12}$

Attribute:Value	Support	Item Number	Attribute:Value	Support	Item Number
VDM:1	9	1	OM:4	3	14
TM:2	9	2	TF:4	3	15
TF:2	7	3	VDF:4	2	16
MDM:2	6	4	VDM:5	1	17
OM:2	6	5	MDM:5	1	18
VDF:2	6	6	OM:5	1	19
MDF:2	6	7	TM:4	1	20
GA:4	5	8	TM:5	1	21
GA:2	5	9	GA:5	1	22
OF:2	5	10	VDF:5	1	23
OF:4	5	11	MDF:5	1	24
MDF:4	4	12	OF:5	1	25
MDM:4	3	13	TF:5	1	26

In this particular example all the Attribute:Value pair placed in 'Item Number added with sorted Frequent Item Table for 2003 with $min - support = \frac{4}{12}$' table, is able to cross the $minSupport$ threshold for all exampled year. Moreover another row added to this table. If the Attribute:Value pair and their corresponding Item Number will change as their support may change a little bit, then the consistency of this particular example will break. So, to maintain consistency the Support column is removed from the Table 4.3 and a generalized 'Item Number added with sorted Frequent Item Table with $min - support = \frac{4}{12}$' table made (Table 4.4), which is used for exampled years.

Table 4.4: Item Number added with sorted Frequent Item Table with $min - support = \frac{4}{12}$

Attribute:Value	Item Number	Attribute:Value	Item Number
VDM:1	1	TF:4	15
TM:2	2	VDF:4	16
TF:2	3	VDM:5	17
MDM:2	4	MDM:5	18
OM:2	5	OM:5	19
VDF:2	6	TM:4	20

Continued on next page

Table 4.4 – continued from previous page

Attribute:Value	Item Number	Attribute:Value	Item Number
MDF:2	7	TM:5	21
GA:4	8	GA:5	22
GA:2	9	VDF:5	23
OF:2	10	MDF:5	24
OF:4	11	OF:5	25
MDF:4	12	TF:5	26
MDM:4	13	VDM:4	27
OM:4	14		

4.3.4 Sorted Frequent Database Creation

Now mapping between Objects of pre-processed data set with Item Number gives SFD of 2003 with $min - support = \frac{4}{12}$ as follows (Table 4.5).

Table 4.5: Sorted Frequent Database for 2003 with $min - support = \frac{4}{12}$

Object	Item	Object	Item
AP	1 2 3 4 5 6 19 24 25	OD	1 2 5 6 8 11 12 13 15
GOA	1 2 4 7 9 10 15	TN	1 2 3 7 8 11 13 14 16
GU	1 2 3 4 6 8 11 12 14	WB	3 4 5 7 9 10 16 17 20
KA	1 3 6 8 11 12 18	ANI	4 5 6 7 9 10 21 26
MA	1 2 3 11 12 13 14 22 23	DD	1 2 5
KE	2 4 5 6 7 9 10 15	PU	1 2 3 5 7 9 10

In this way SFD table for 2005, 2009, 2011, 2013, 2015 and 2017 are prepared (Table 4.6, 4.7, 4.8, 4.9, 4.10, 4.11). Here the mapping is done between each year's SFD table and generalized 'Item Number added with sorted Frequent Item Table with $min - support = \frac{4}{12}$' table.

Table 4.6: Sorted Frequent Database for 2005 with $min - support = \frac{4}{12}$

Object	Item	Object	Item
AP	1 2 3 4 6 8 19 24 25	OD	1 2 5 6 8 11 12 13
GOA	1 2 3 4 5 6 7 9 10	TN	1 2 3 7 8 11 13 14 16
GU	1 2 4 6 8 11 12 14	WB	3 4 5 7 9 10 16 17 20
KA	1 3 6 8 11 12 18	ANI	3 4 5 6 7 9 10 21
MA	1 2 3 11 12 13 14 22 23	DD	1 2 5 26
KE	1 2 3 4 5 6 7 9 10	PU	1 2 3 5 7 9 10

Table 4.7: Sorted Frequent Database for 2009 with $min - support = \frac{4}{12}$

Object	Item	Object	Item
AP	1 2 3 6 8 11 19 24	OD	2 5 8 11 12 13 15 16 17
GOA	1 2 4 5 6 7 9 10 15	TN	1 2 3 4 8 11 12 14 16
GU	1 2 3 4 6 8 11 12 14	WB	3 4 9 10 12 14 16 20 27
KA	1 3 6 8 11 12 18	ANI	4 5 6 7 9 10 21 26
MA	1 2 3 13 14 16 22 23 25	DD	1 2 5
KE	1 2 4 5 6 7 9 10 15	PU	1 2 3 5 7 9 10

Table 4.8: Sorted Frequent Database for 2011 with $min - support = \frac{4}{12}$

Object	Item	Object	Item
AP	1 2 3 6 8 11 12 13 19	OD	2 3 5 8 11 12 13 16 17
GOA	1 2 3 4 5 6 7 9 10	TN	1 2 3 4 8 11 12 14 16
GU	1 2 4 6 8 11 12 14	WB	3 4 5 7 9 11 16 20 27
KA	1 3 8 11 12 16 18	ANI	3 4 5 6 7 9 10 21
MA	1 2 3 13 14 22 23 24 25	DD	1 4 5 6 15 20
KE	1 2 3 4 5 6 7 9 10	PU	1 2 5 7 9 10 26

Table 4.9: Sorted Frequent Database for 2013 with $min - support = \frac{4}{12}$

Object	Item	Object	Item
AP	1 2 3 6 8 11 13 14 24	OD	2 3 5 8 11 12 13 16 17
GOA	1 2 4 5 6 7 9 10 15	TN	1 2 3 8 11 12 13 14 16
GU	1 2 4 6 8 11 12 14	WB	3 4 5 7 9 10 16 20 27
KA	1 3 6 8 11 12 18	ANI	4 5 6 7 9 10 20 26
MA	1 2 3 12 13 19 22 23 25	DD	1 5 15 21
KE	1 2 4 5 6 7 9 10 15	PU	1 2 3 5 7 9 10

Table 4.10: Sorted Frequent Database for 2015 with $min - support = \frac{4}{12}$

Object	Item	Object	Item
AP	1 2 3 6 8 11 12 13 14	OD	2 5 8 11 12 13 15 16 17
GOA	1 2 4 5 6 7 9 10 15	TN	2 3 8 11 12 13 14 16
GU	1 2 4 6 8 11 12 14	WB	3 4 5 7 9 11 16 20 27
KA	1 3 6 8 11 12 13 14	ANI	4 5 6 7 9 10 21 26
MA	1 2 3 18 19 22 23 24 25	DD	1 3 6 20
KE	1 2 4 5 6 7 9 10 15	PU	1 2 3 5 7 9 10

Table 4.11: Sorted Frequent Database for 2017 with $min - support = \frac{4}{12}$

Object	Item	Object	Item
AP	1 2 3 6 8 11 12 13 14	OD	2 5 8 11 12 13 15 16 17
GOA	1 2 4 5 6 7 9 10 15	TN	2 3 8 11 12 13 14 16
GU	1 2 4 6 8 11 12 19	WB	3 4 5 7 9 11 16 20 27
KA	1 2 3 4 9 11 12 14 16	ANI	4 5 6 7 9 10 21 26
MA	1 3 14 18 22 23 24 25	DD	1 3 6 20
KE	1 2 4 5 6 7 9 10 15	PU	1 2 3 5 7 9 10

4.4 Construct Frequent Generalized Suffix Tree:

Here step by step Algorithm 3 is applied on 2003's SFD to create FGST shown in next steps.

Step 1:- Create root(Figure 4.17)

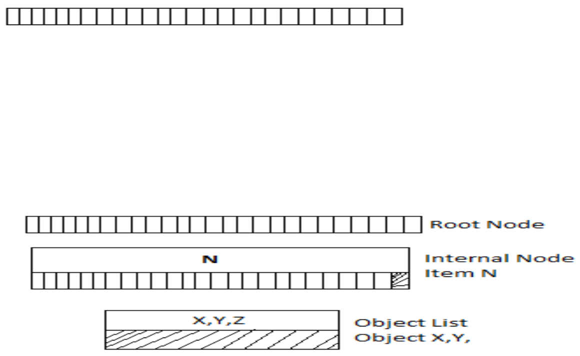


Figure 4.17: Suffix Tree Construction Step 1

Step 2:- Read Row 1 of SFD where Item Number:- 1 2 3 4 5 6 19 24 25 and Object:- AP

Step 3:- As first item number is 1, start building first branch from first root node and build suffix 1 2 3 4 5 6 19 24 25 AP(Figure 4.18)

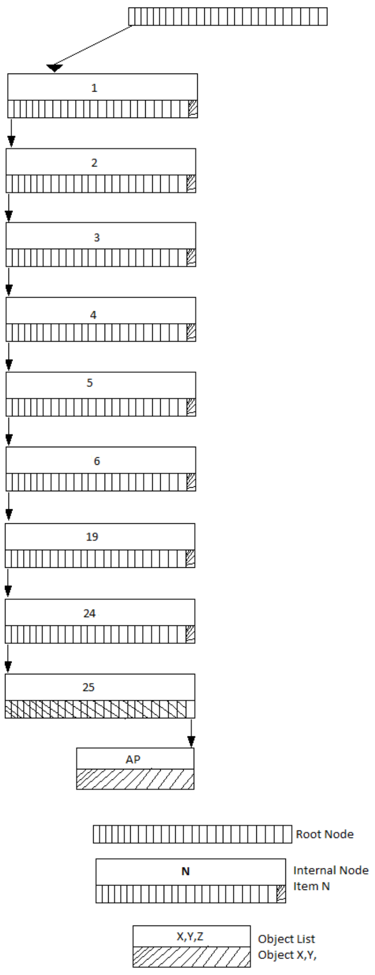


Figure 4.18: Suffix Tree Construction
Step 2 3

Step 4:- Read Row 1 of SFD where Item Number:- 1 2 3 4 5 6 19 24 25 and Object:- AP

Step 5:- As next item number is 2, start building next branch from next root node and delete previous item number and build suffix 2 3 4 5 6 19 24 25 AP (Figure 4.19)

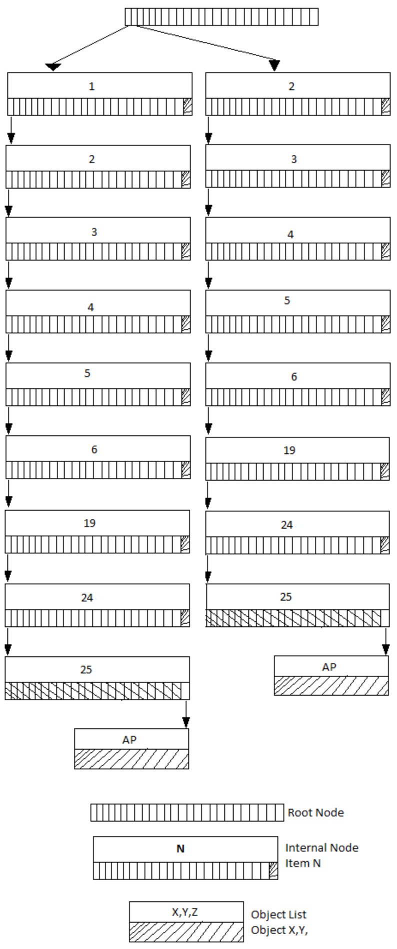


Figure 4.19: Suffix Tree Construction
Step 4 5

Step 6:- Read Row 1 of SFD where Item Number:- 1 2 3 4 5 6 19 24 25 and Object:- AP

Step 7:- As next item number is 3, start building next branch from next root node and delete previous item number and build suffix 3 4 5 6 19 24 25 AP (Figure 4.20)

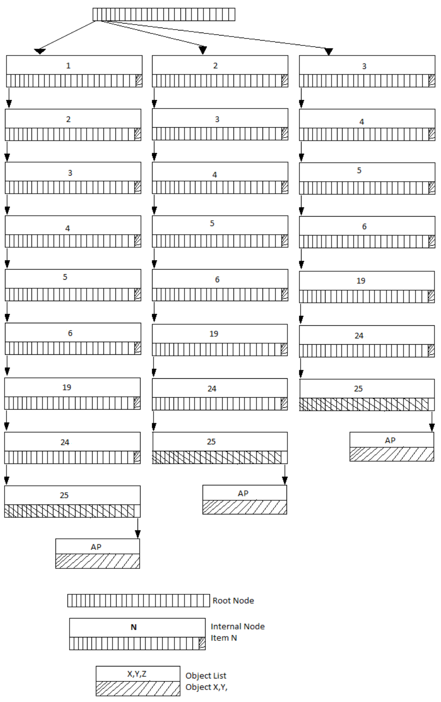


Figure 4.20: Suffix Tree Construction
Step 6 7

Step 8:- Read Row 1 of SFD where Item Number:- 1 2 3 4 5 6 19 24 25 and Object:- AP

Step 9:- As next item number is 4, start building next branch from next root node and delete previous item number and build suffix 4 5 6 19 24 25 AP(Figure 4.21)

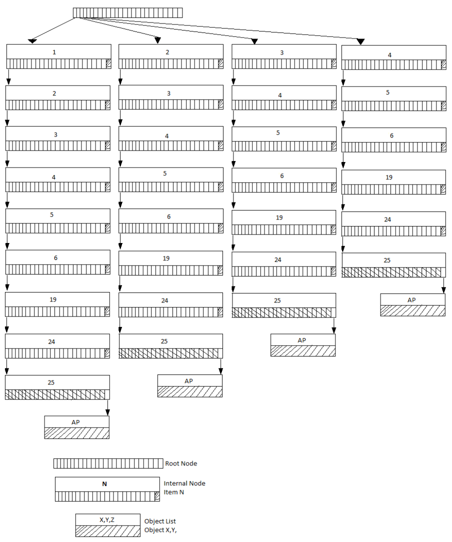


Figure 4.21: Suffix Tree Construction
Step 8 9

Step 10:- Read Row 1 of SFD where Item Number:- 1 2 3 4 5 6 19 24 25 and Object:- AP

Step 11:- As next item number is 5, start building next branch from next root node and delete previous item number and build suffix 5 6 19 24 25 AP (Figure 4.22)

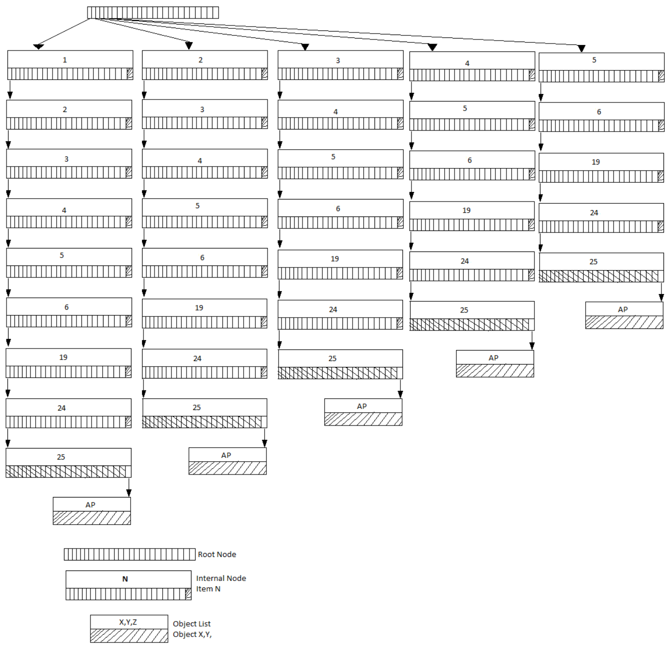


Figure 4.22: Suffix Tree Construction Step 10 11

Step 12:- Read Row 1 of SFD where Item Number:- 1 2 3 4 5 6 19 24 25 and Object:- AP

Step 13:- As next item number is 6, start building next branch from next root node and delete previous item number and build suffix 6 19 24 25 AP(Figure 4.23)

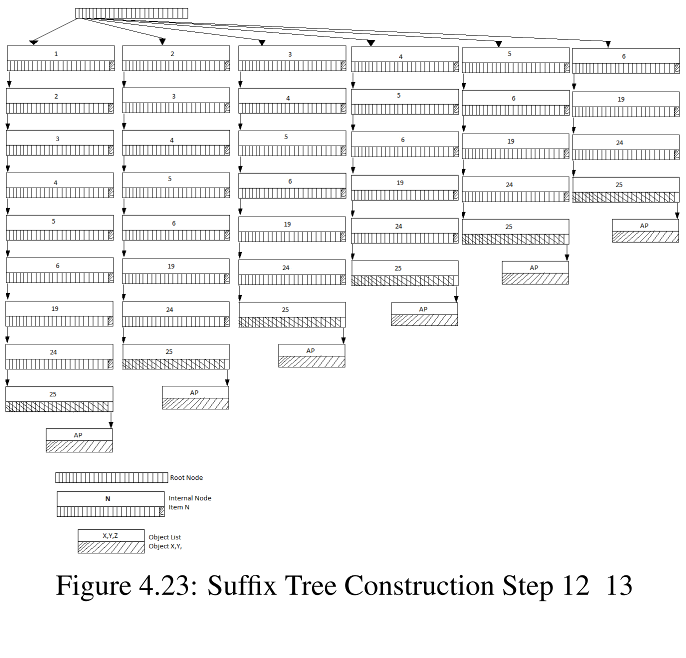


Figure 4.23: Suffix Tree Construction Step 12 13

Step 14:- Read Row 1 of SFD where Item Number:- 1 2 3 4 5 6 19 24 25 and Object:- AP

Step 15:- As next item number is 19, start building next branch from next root node and delete previous item number and build suffix 19 24 25 AP(Figure 4.24)

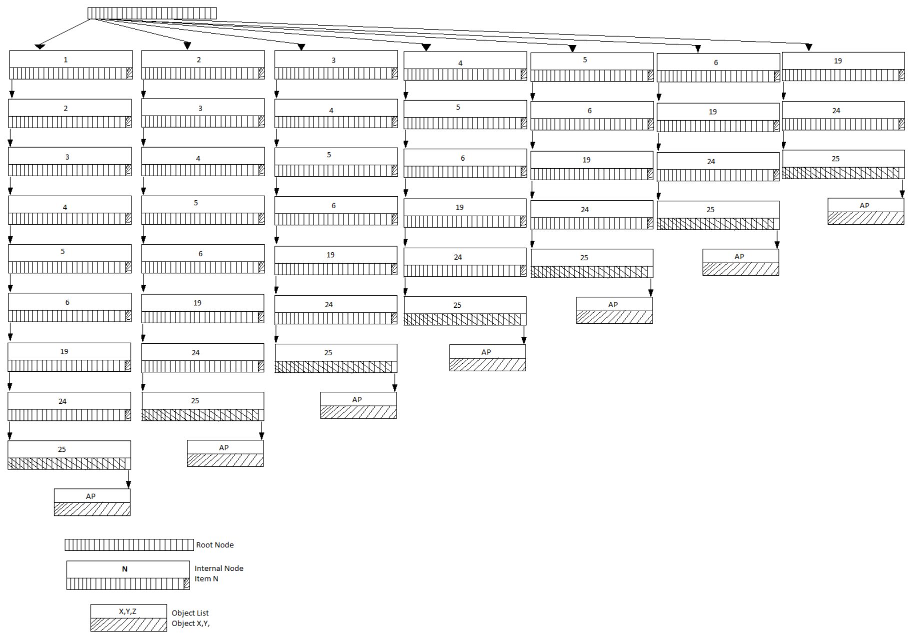


Figure 4.24: Suffix Tree Construction Step 14 15

Step 16:- Read Row 1 of SFD where Item Number:- 1 2 3 4 5 6 19 24 25 and Object:- AP

Step 17:- As next item number is 24, start building next branch from next root node and delete previous item number and build suffix 24 25 AP(Figure 4.25)

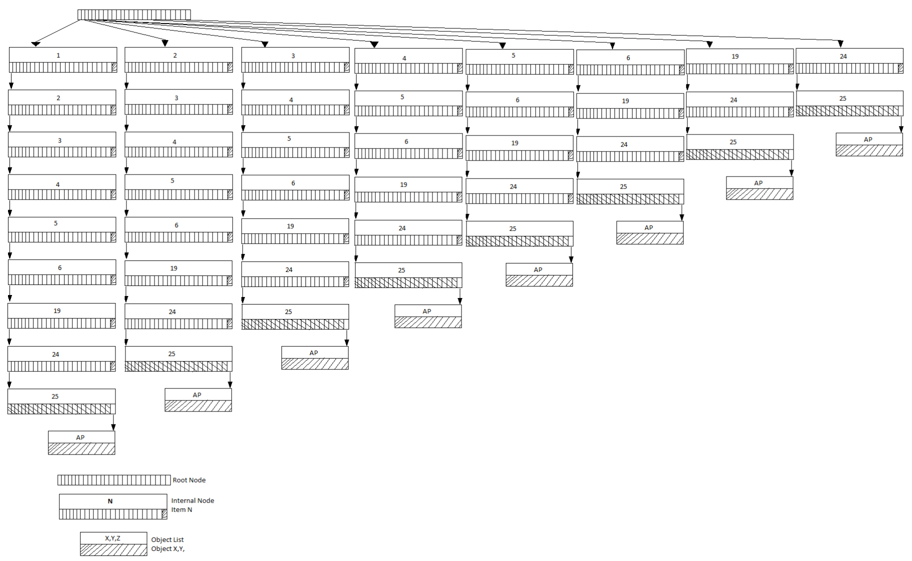


Figure 4.25: Suffix Tree Construction Step 16 17

Step 18:- Read Row 1 of SFD where Item Number:- 1 2 3 4 5 6 19 24 25 and Object:- AP

Step 19:- As next item number is 25, start building next branch from next root node and delete previous item number and build suffix 25 AP(Figure 4.26)

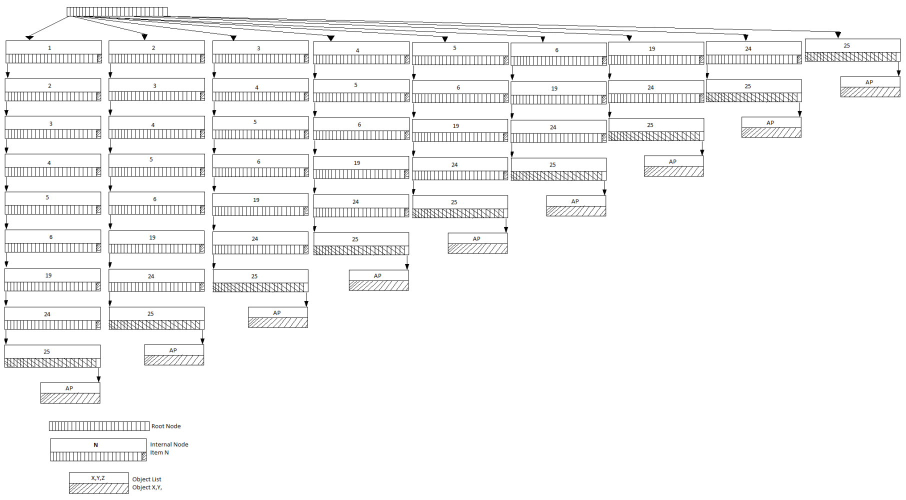


Figure 4.26: Suffix Tree Construction Step 18 19

Step 20:- Read Row 2 of SFD where Item Number:- 1 2 4 7 9 10 15 and Object:- GOA

Step 21:- As first item number is 1, start building next branch from first root node and build suffix 1 2 4 7 9 10 15 GOA (Figure 4.27)

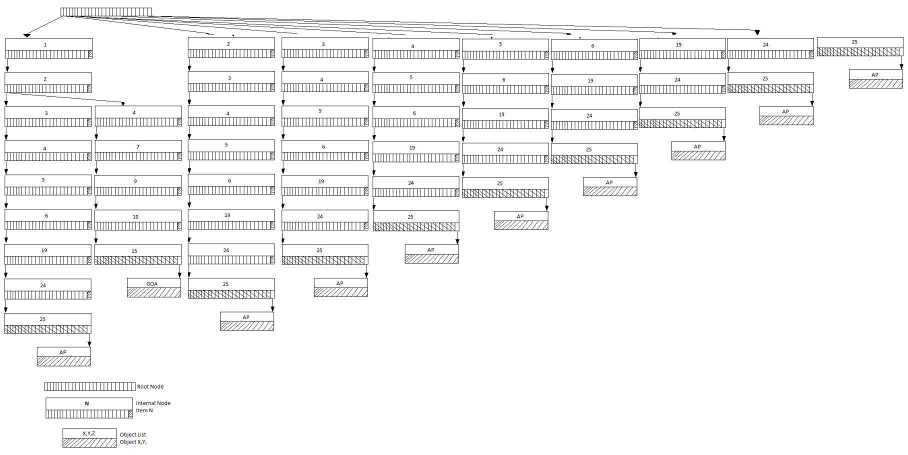


Figure 4.27: Suffix Tree Construction Step 20 21

Step 22:- Read Row 2 of SFD where Item Number:- 1 2 4 7 9 10 15 and Object:- GOA

Step 23:- As next item number is 2, start building next branch from next root node and delete previous item number and build suffix 2 4 7 9 10 15 GOA (Figure 4.28)

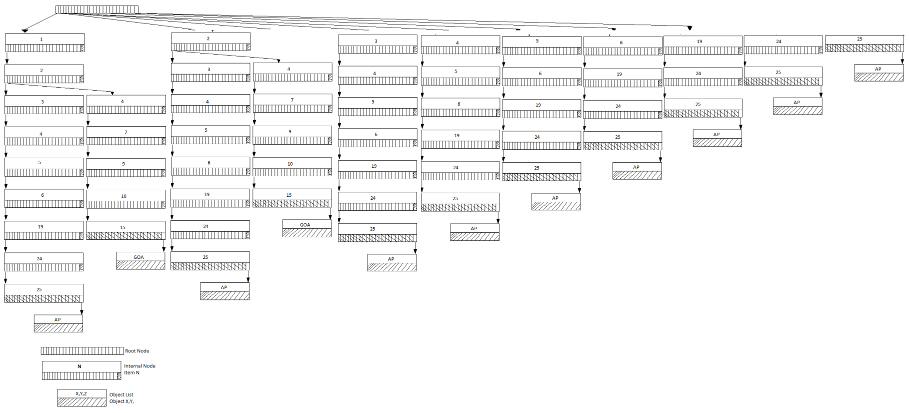


Figure 4.28: Suffix Tree Construction Step 22 23

Step 24:- Read Row 2 of SFD where Item Number:- 1 2 4 7 9 10 15 and Object:- GOA

Step 25:- As next item number is 4, start building next branch from next root node and delete previous item number and build suffix 4 7 9 10 15 GOA(Figure 4.29)

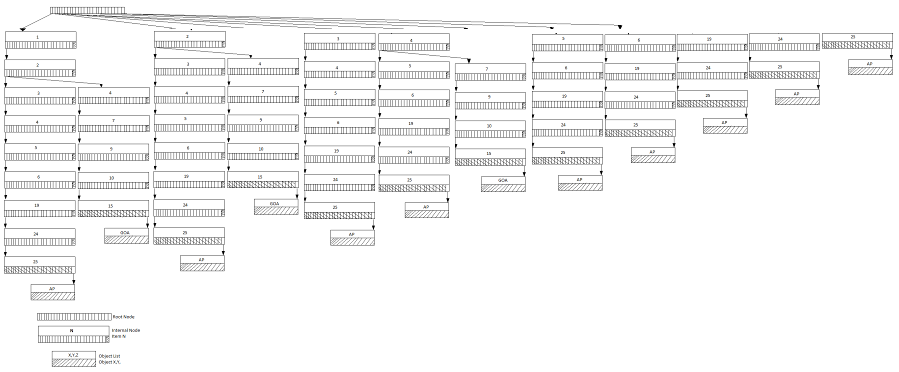


Figure 4.29: Suffix Tree Construction Step 24 25

Step 26:- Read Row 2 of SFD where Item Number:- 1 2 4 7 9 10 15 and Object:- GOA

Step 27:- As next item number is 7, start building next branch from next root node and delete previous item number and build suffix 7 9 10 15 GOA(Figure 4.30)

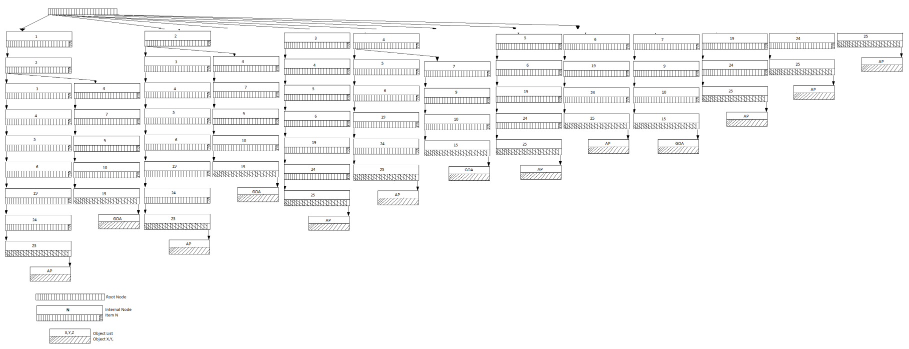


Figure 4.30: Suffix Tree Construction Step 26 27

Step 28:- Read Row 2 of SFD where Item Number:- 1 2 4 7 9 10 15 and Object:- GOA

Step 29:- As next item number is 9, start building next branch from next root node and delete previous item number and build suffix 9 10 15 GOA(Figure 4.31)

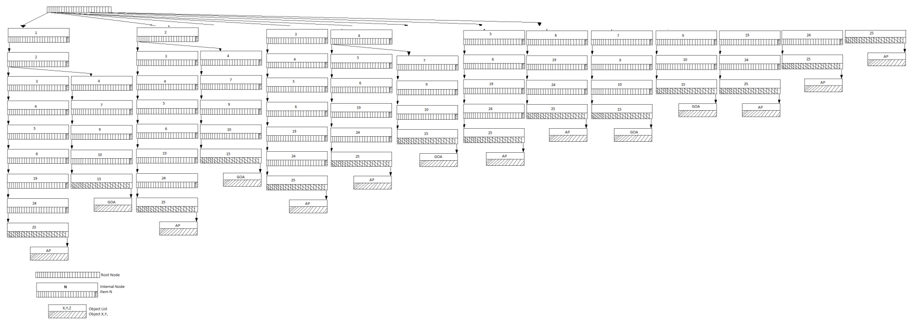


Figure 4.31: Suffix Tree Construction Step 28 29

Step 30:- Read Row 2 of SFD where Item Number:- 1 2 4 7 9 10 15 and Object:- GOA

Step 31:- As next item number is 10, start building next branch from next root node and delete previous item number and build suffix 10 15 GOA(Figure 4.32)

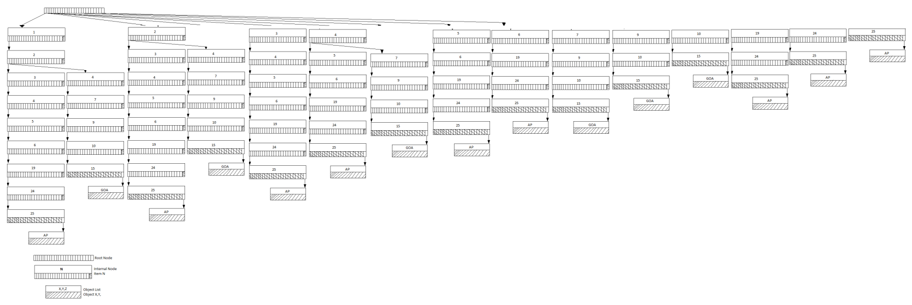


Figure 4.32: Suffix Tree Construction Step 30 31

Step 32:- Read Row 2 of SFD where Item Number:- 1 2 4 7 9 10 15 and Object:- GOA

Step 33:- As next item number is 15, start building next branch from next root node and delete previous item number and build suffix 15 GOA(Figure 4.33)

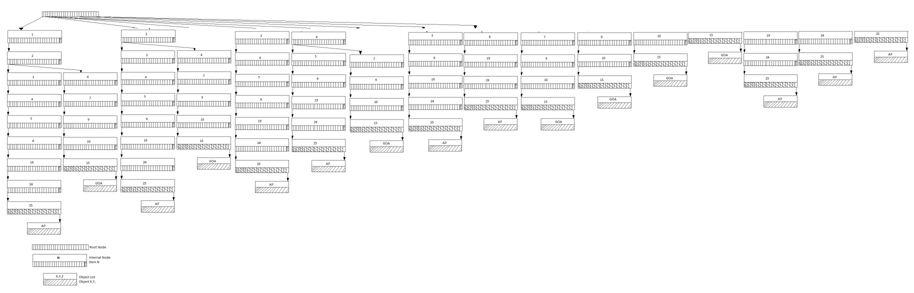


Figure 4.33: Suffix Tree Construction Step 32 33

In this way all the other rows of SFD are inserted into the suffix tree. After inserting all the rows of SFD into suffix tree, if there are any match found the object list create at the end, then that object list will be delete as mentioned in the Algorithm 3 we get (Figure 4.34 and 4.35):-

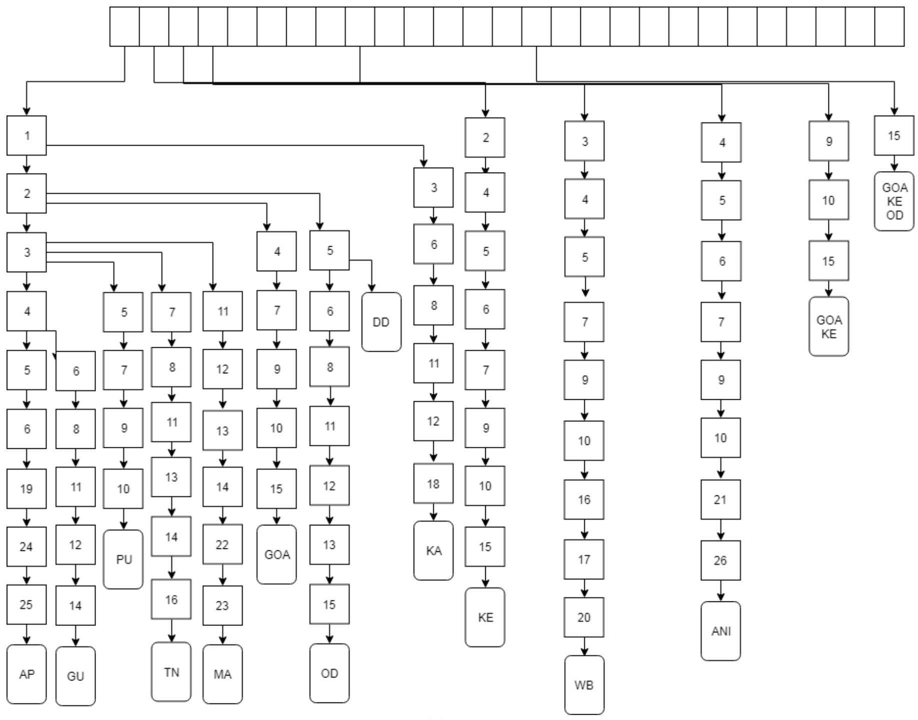


Figure 4.34: 2003's suffix tree before checking current.ObjectList=prefix.ObjectList

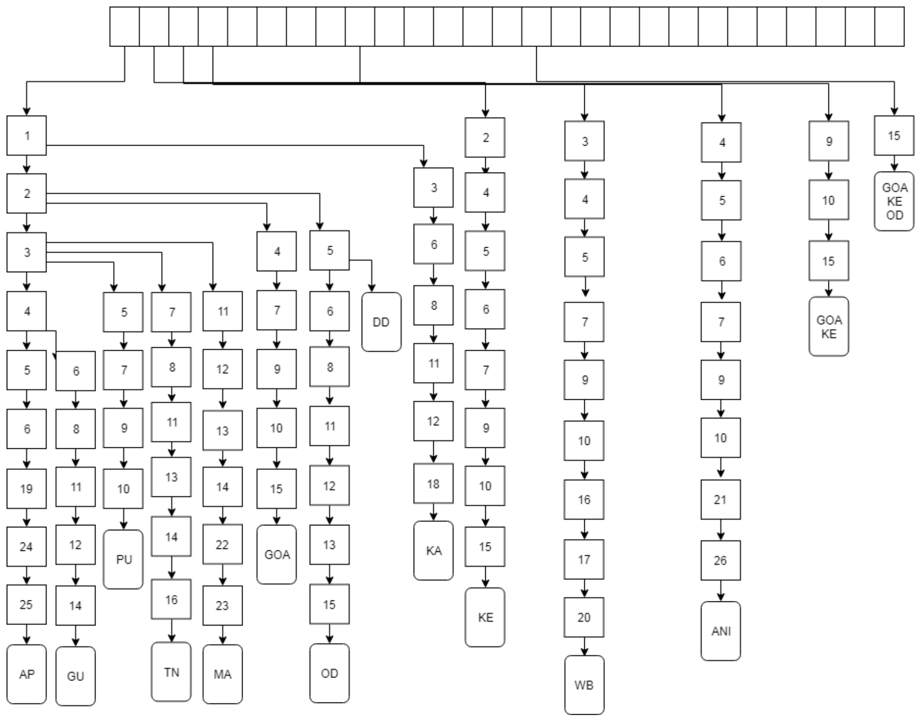


Figure 4.35: 2003's final suffix tree after deleting duplicate object list

All final suffix tree of year 2005, 2009, 2011, 2013, 2015 and 2017 are shown below (Figure 4.36, 4.37, 4.38, 4.39, 4.40 and 4.41):-

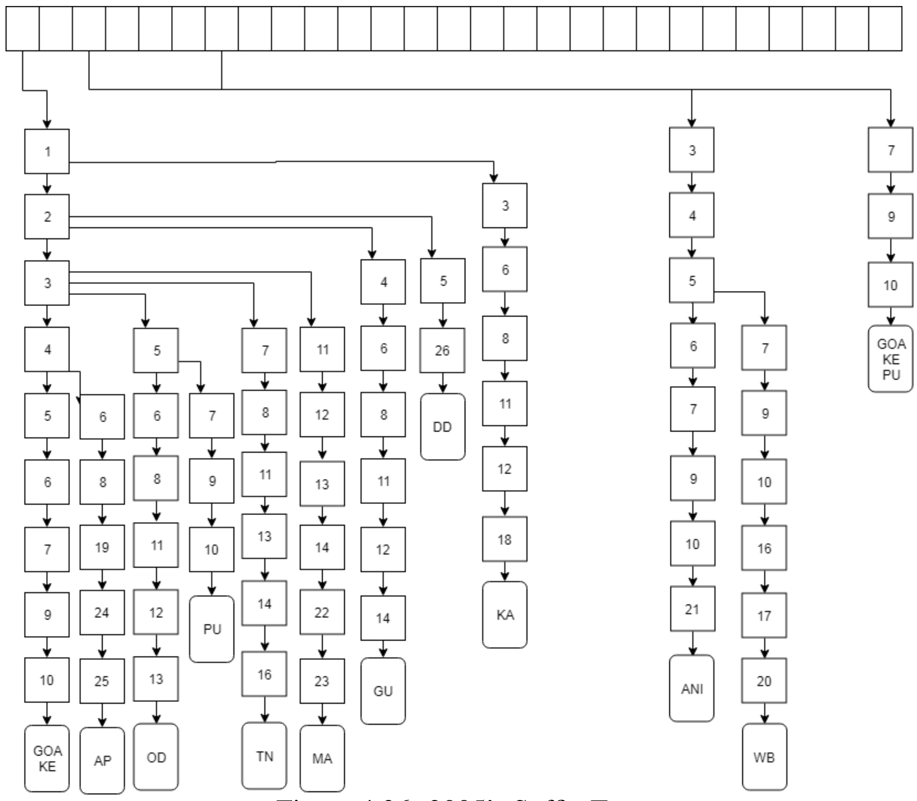


Figure 4.36: 2005’s Suffix Tree

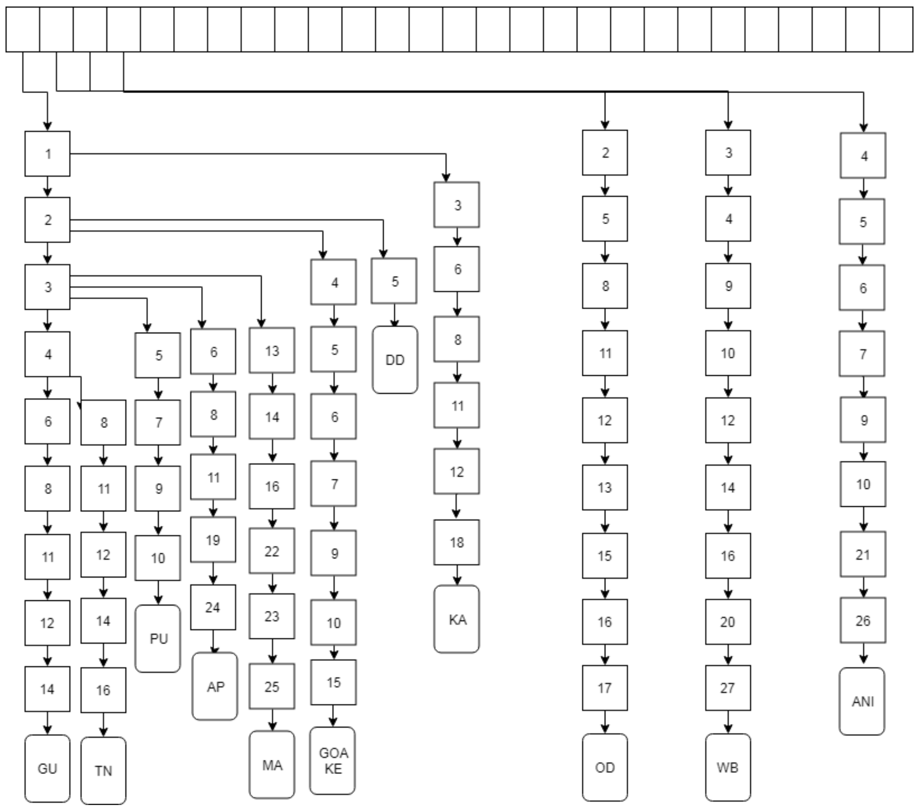


Figure 4.37: 2009’s Suffix Tree

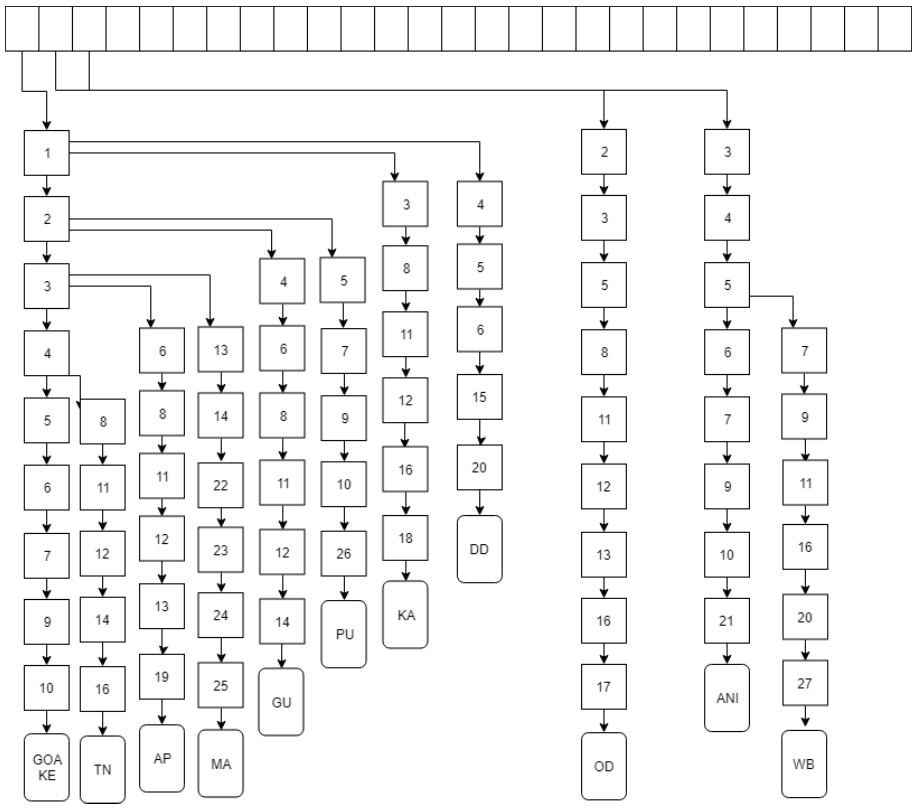


Figure 4.38: 2011's Suffix Tree

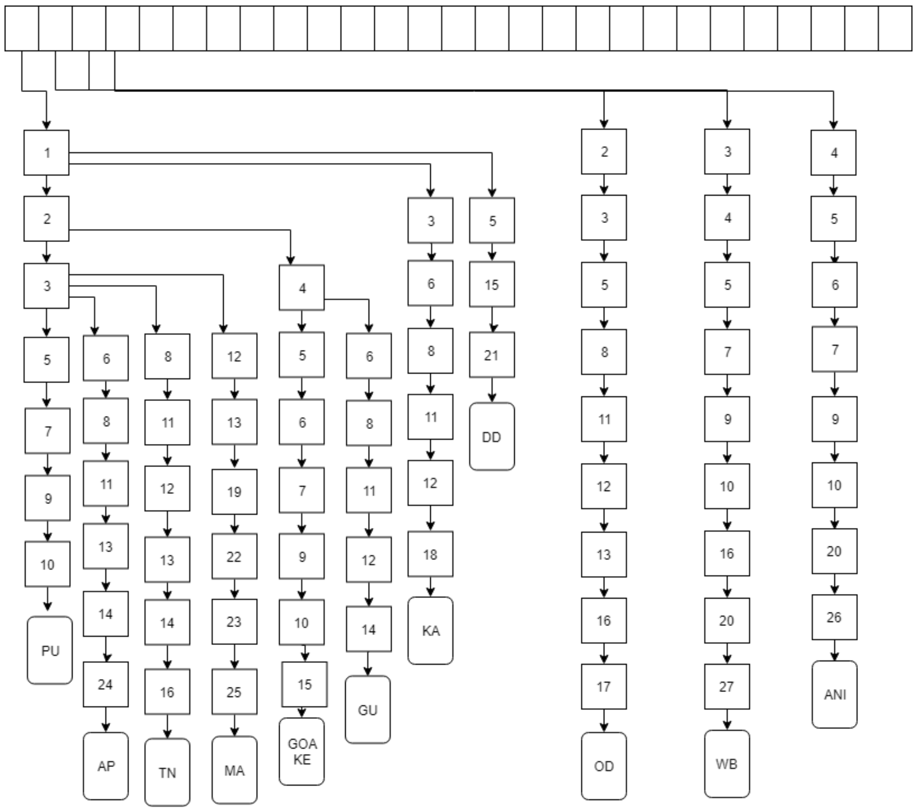


Figure 4.39: 2013's Suffix Tree

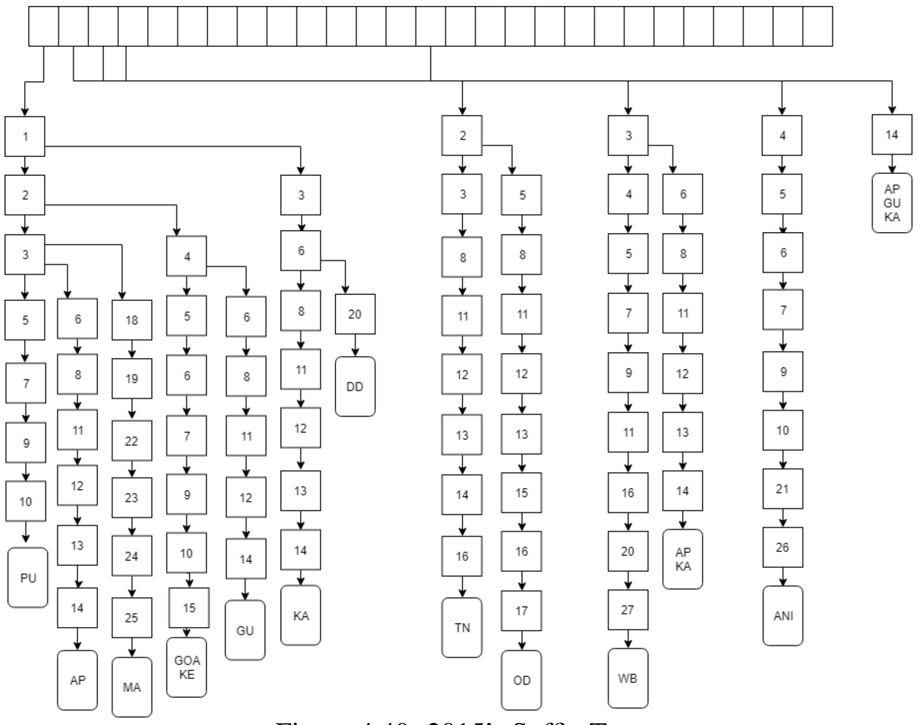


Figure 4.40: 2015's Suffix Tree

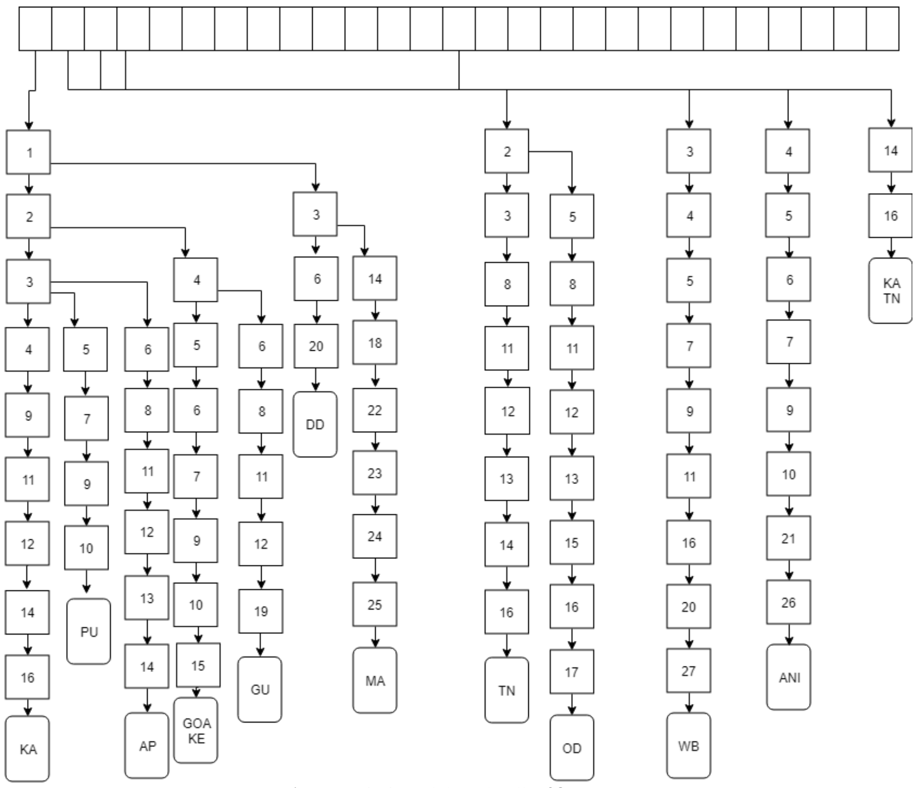


Figure 4.41: 2017's Suffix Tree

4.5 Merging Two Suffix Tree:

In Algorithm 7 merging two suffix tree procedure is described. Here in this example also, the algorithm 7 is followed to merge two suffix trees. But the difficulty is, in this example 7 suffix trees are involved. So, to merge these suffix trees into one suffix tree Huffman Encoding procedure is taken (Figure 4.42).

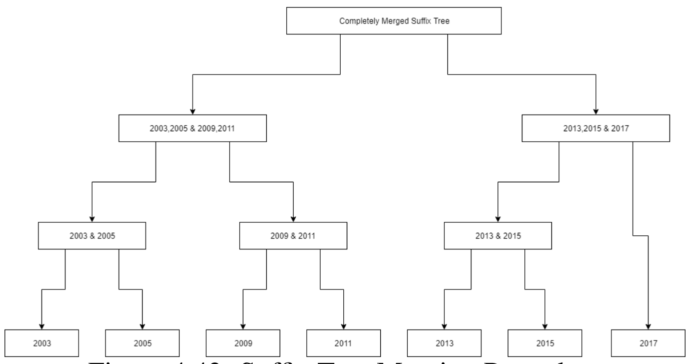


Figure 4.42: Suffix Tree Merging Procedure

In the above mentioned manner the suffix trees are merged using algorithm 7 (Figure 4.43, 4.44, 4.45, 4.46, 4.47 and 4.48)

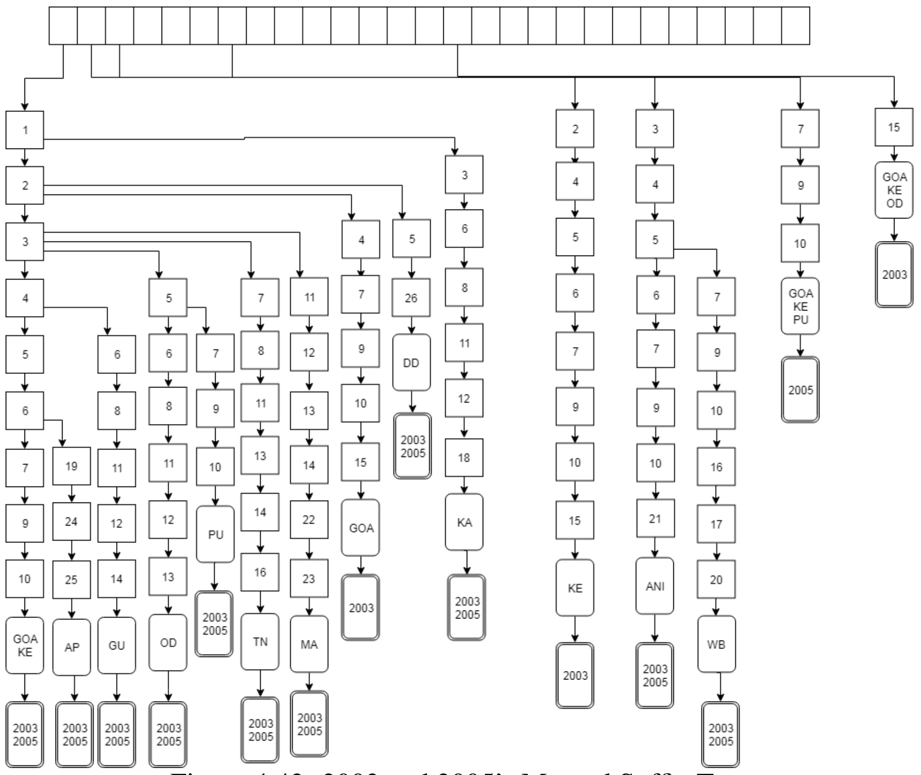


Figure 4.43: 2003 and 2005's Merged Suffix Tree

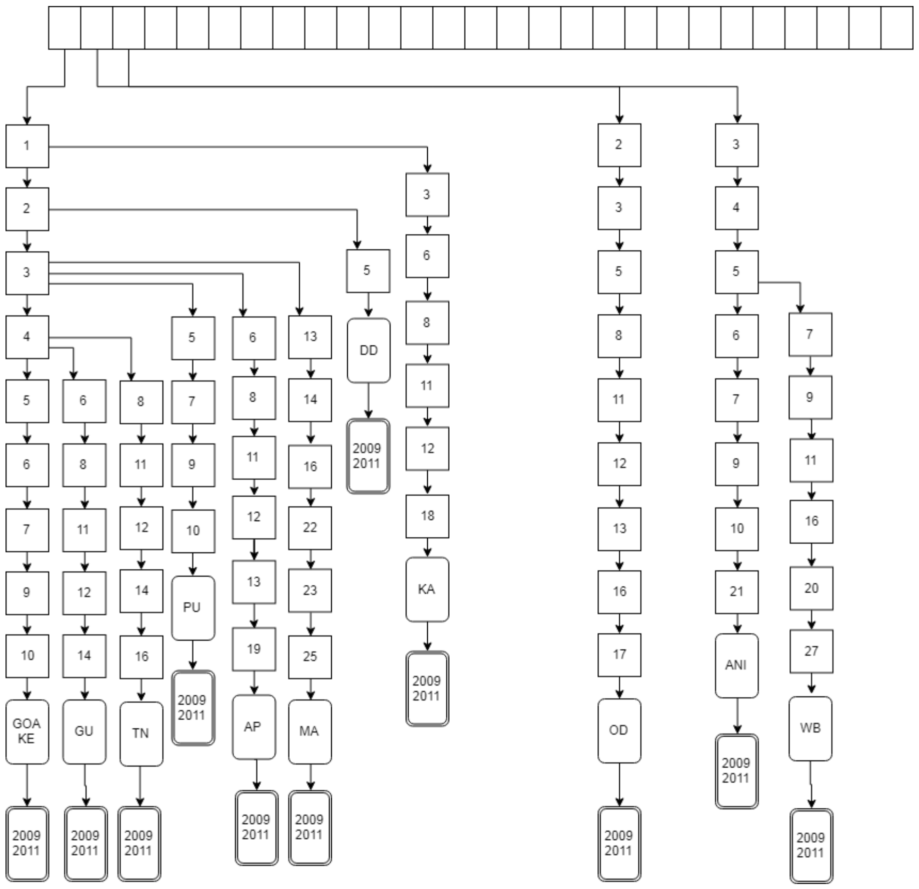


Figure 4.44: 2009 and 2011's Merged Suffix Tree

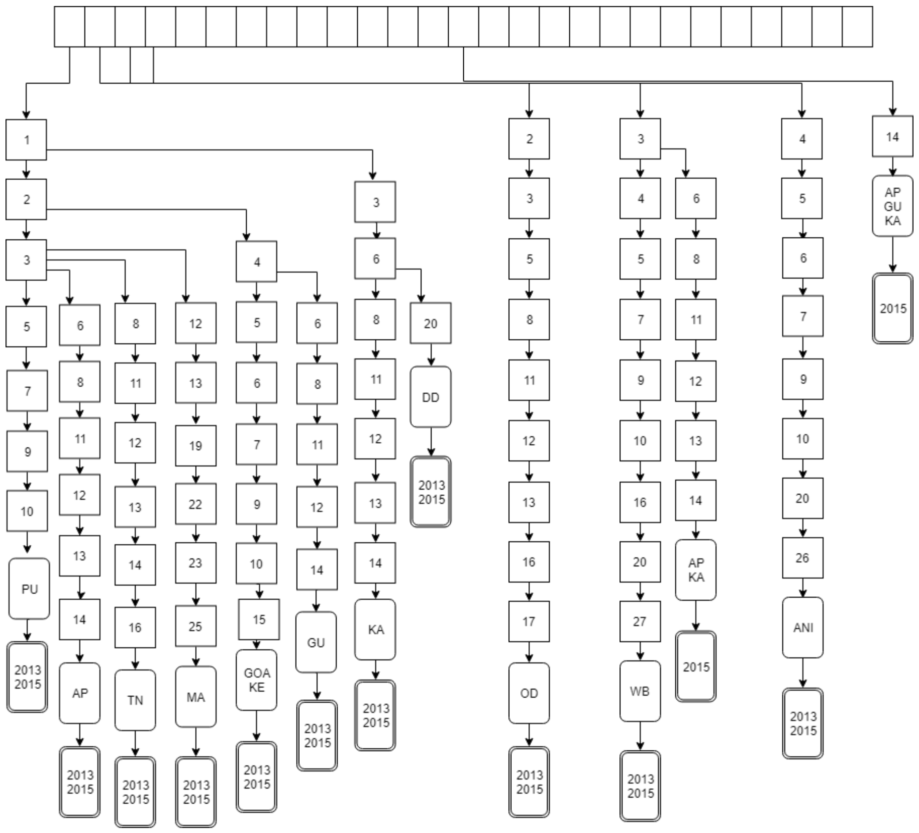


Figure 4.45: 2013 and 2015's Merged Suffix Tree

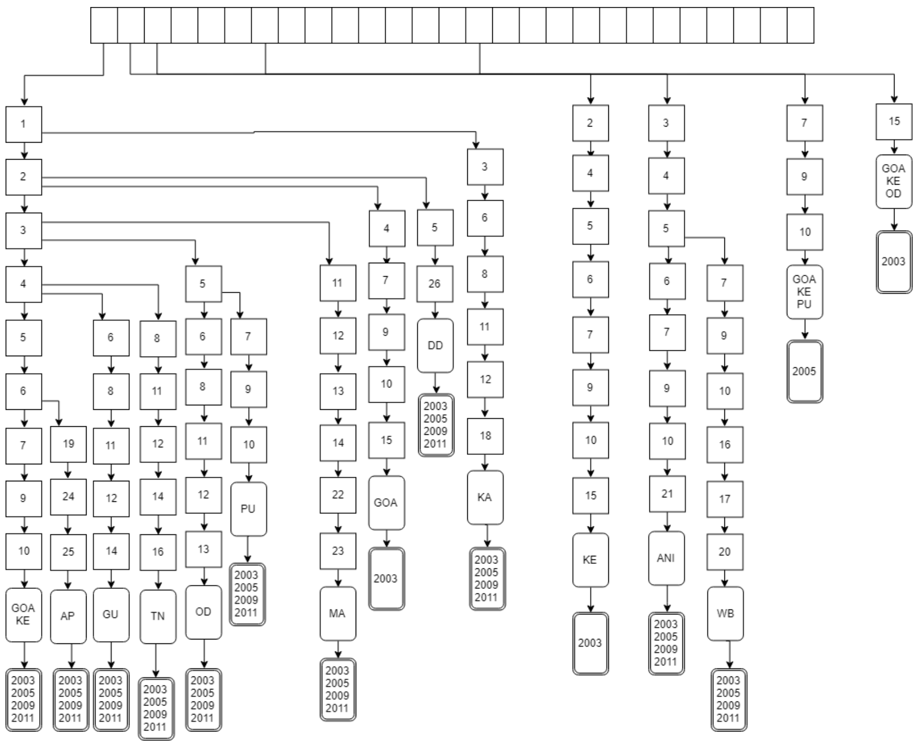


Figure 4.46: 2003,2005 and 2009,2011's Merged Suffix Tree

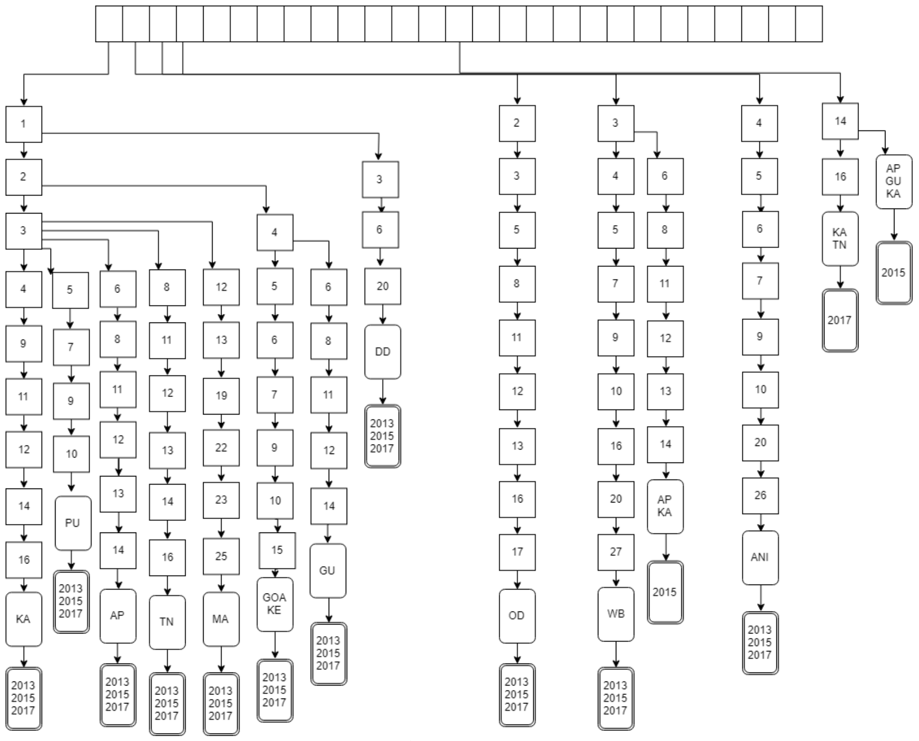


Figure 4.47: 2013,2015 and 2017's Merged Suffix Tree

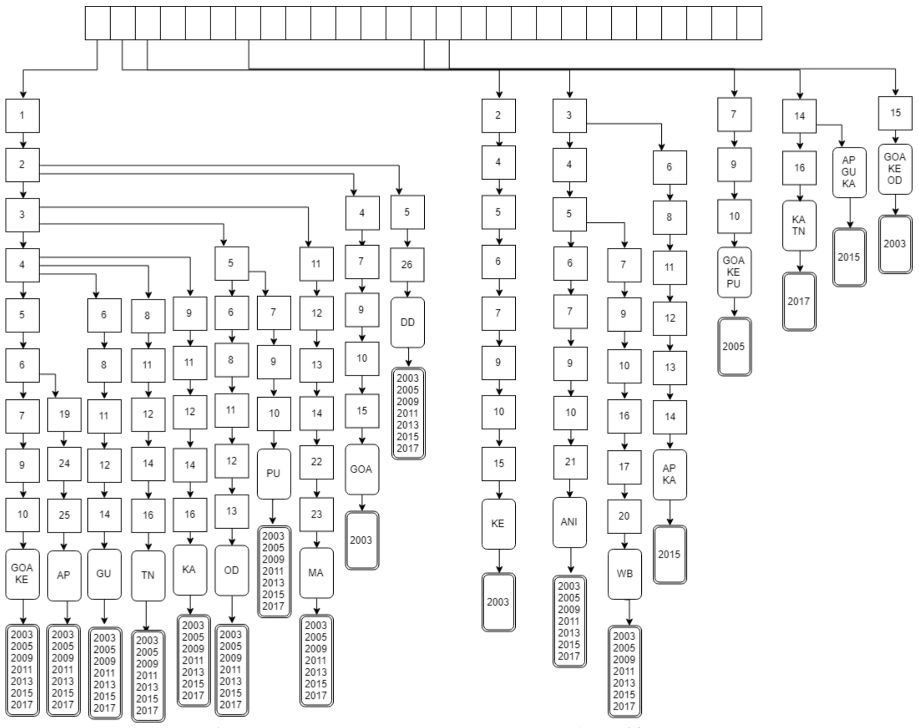


Figure 4.48: Completely Merged Suffix Tree

4.6 Extracting Tri-Clusters from Completely Merged Suffix Tree

This phase will work same as mentioned in Algorithm 8. A table will be created, which has six columns and number of rows same as number of Leaf Node of the Completely merged suffix tree. This phase will take the 'Item Number added with sorted Frequent Item Table with $minSupport$' table also. Traverse from leaf node to root node of Completely merged suffix tree and store them into specific column according to their node type. After that matching of ItemNumber from Tri-cluster table with Item Number from 'Item Number added with sorted Frequent Item Table with $minSupport$' table will fill the Attribute:Value pair of Tri-cluster table. Now concatenate row wise, the content of Attribute:Value, ObjectList and ConditionList column values and store them into Tri-cluster column (Table 4.12). This process is able to give proper tri-clusters, which can be generated from these exampled datasets.

Table 4.12: Tri-Cluster Table

Item	ObjectList	ConditionList	Attribute:Value	Tri-cluster
1 2 3 4 5 6 7 9 10	GOA KE	2003 2005 2009 2011 2013 2015 2017	VDM:1 TM:2 TF:2 MDM:2 OM:2 VDF:2 MDF:2 GA:2 OF:2	VDM:1 TM:2 TF:2 MDM:2 OM:2 VDF:2 MDF:2 GA:2 OF:2, GOA KE, 2003 2005 2009 2011 2013 2015 2017
1 2 3 4 5 6 19 24 25	AP	2003 2005 2009 2011 2013 2015 2017	VDM:1 TM:2 TF:2 MDM:2 OM:2 VDF:2 OM:5 MDF:5 OF:5	VDM:1 TM:2 TF:2 MDM:2 OM:2 VDF:2 OM:5 MDF:5 OF:5, AP, 2003 2005 2009 2011 2013 2015 2017
1 2 3 4 6 8 11 12 14	GU	2003 2005 2009 2011 2013 2015 2017	VDM:1 TM:2 TF:2 MDM:2 VDF:2 GA:4 OF:4 MDF:4 OM:4	VDM:1 TM:2 TF:2 MDM:2 VDF:2 GA:4 OF:4 MDF:4 OM:4, GU, 2003 2005 2009 2011 2013 2015 2017
1 2 3 4 8 11 12 14 16	TN	2003 2005 2009 2011 2013 2015 2017	VDM:1 TM:2 TF:2 MDM:2 GA:4 OF:4 MDF:4 OM:4 VDF:4	VDM:1 TM:2 TF:2 MDM:2 GA:4 OF:4 MDF:4 OM:4 VDF:4, TN, 2003 2005 2009 2011 2013 2015 2017
1 2 3 4 9 11 12 14 16	KA	2003 2005 2009 2011 2013 2015 2017	VDM:1 TM:2 TF:2 MDM:2 GA:2 OF:4 MDF:4 OM:4 VDF:4	VDM:1 TM:2 TF:2 MDM:2 GA:2 OF:4 MDF:4 OM:4 VDF:4, KA, 2003 2005 2009 2011 2013 2015 2017
1 2 3 5 6 8 11 12 13	OD	2003 2005 2009 2011 2013 2015 2017	VDM:1 TM:2 TF:2 OM:2 VDF:2 GA:4 OF:4 MDF:4 MDM:4	VDM:1 TM:2 TF:2 OM:2 VDF:2 GA:4 OF:4 MDF:4 MDM:4, OD, 2003 2005 2009 2011 2013 2015 2017
1 2 3 5 7 9 10	PU	2003 2005 2009 2011 2013 2015 2017	VDM:1 TM:2 TF:2 OM:2 MDF:2 GA:2 OF:2	VDM:1 TM:2 TF:2 OM:2 MDF:2 GA:2 OF:2, PU, 2003 2005 2009 2011 2013 2015 2017
1 2 3 11 12 13 14 22 23	MA	2003 2005 2009 2011 2013 2015 2017	VDM:1 TM:2 TF:2 OF:4 MDF:4 MDM:4 OM:4 GA:5 VDF:5	VDM:1 TM:2 TF:2 OF:4 MDF:4 MDM:4 OM:4 GA:5 VDF:5, MA, 2003 2005 2009 2011 2013 2015 2017
1 2 4 7 9 10 15	GOA	2003	VDM:1 TM:2 MDM:2 MDF:2 GA:2 OF:2 TF:4	VDM:1 TM:2 MDM:2 MDF:2 GA:2 OF:2 TF:4, GOA, 2003
1 2 5 26	DD	2003 2005 2009 2011 2013 2015 2017	VDM:1 TM:2 OM:2 TF:5	VDM:1 TM:2 OM:2 TF:5, DD, 2003 2005 2009 2011 2013 2015 2017
2 4 5 6 7 9 10 15	KE	2003	TM:2 MDM:2 OM:2 VDF:2 MDF:2 GA:2 OF:2 TF:4	TM:2 MDM:2 OM:2 VDF:2 MDF:2 GA:2 OF:2 TF:4, KE, 2003
3 4 5 6 7 9 10 21	ANI	2003 2005 2009 2011 2013 2015 2017	TF:2 MDM:2 OM:2 VDF:2 MDF:2 GA:2 OF:2 TM:5	TF:2 MDM:2 OM:2 VDF:2 MDF:2 GA:2 OF:2 TM:5, ANI, 2003 2005 2009 2011 2013 2015 2017
3 4 5 7 9 10 16 17 20	WB	2003 2005 2009 2011 2013 2015 2017	TF:2 MDM:2 OM:2 MDF:2 GA:2 OF:2 VDF:4 VDM:5 TM:4	TF:2 MDM:2 OM:2 MDF:2 GA:2 OF:2 VDF:4 VDM:5 TM:4, WB, 2003 2005 2009 2011 2013 2015 2017

Continued on next page

Table 4.12 – continued from previous page

Item	ObjectList	ConditionList	Attribute:Value	Tri-cluster
3 6 8 11 12 13 14	AP KA	2015	TF:2 VDF:2 GA:4 OF:4 MDF:4 MDM:4 OM:4	TF:2 VDF:2 GA:4 OF:4 MDF:4 MDM:4 OM:4, AP KA, 2015
7 9 10	GOA KE PU	2005	MDF:2 GA:2 OF:2	MDF:2 GA:2 OF:2, GOA KE PU, 2005
14 16	KA TN	2017	OM:4 VDF:4	OM:4 VDF:4, KA TN, 2017
14	AP GU KA	2015	OM:4	OM:4, AP GU KA, 2015
15	GOA KE OD	2003	TF:4	TF:4, GOA, KE, OD, 2003

Chapter 5

Result Analysis

5.1 Result Analysis from Data Mining Perspective

Cluster analysis is an unsupervised learning mechanism. Unsupervised learning is mainly used when learning data contains only indicative signals without any description attached, it is up to us to find the structure of the data underneath, to discover hidden information, or to determine how to describe the data. This kind of learning data called unlabeled data. Unsupervised learning can be used to detect anomalies, such as fraud or defective equipment, or to group behavior of some similar kind of data. This third property of unsupervised learning is used in case of cluster analysis. In this project, three-dimensional clustering is used to group some forest dataset. Here the above mentioned resulting triclusters (Table 4.12) can give a very useful message about Indian forest distribution.
In that table, one can observe that mangrove forest density is only high in West Bengal, not in other states and the forest density remains almost the same in between the year 2003 and 2017. In a similar way, total forest are in Andhra Pradesh is very high in the year 2003, 2005, 2009, 2011 and in 2013, bt started decreasing from the year 2015. As well as, we can easily say that, the total forest area of Odisha remain high throughout the experimental period.
This kind of useful data retrieval may leave a very effective impact on future data prediction.

5.2 Comparison with Main Dataset

From the main datasets we can clearly observe that Total Mangrove, Total Forest, Medium Dense Mangrove, Open Mangrove, Very Dense Forest, Medium Dense Forest, and Open Forest area are moderately low for Goa and Kerala in the year 2003, 2005, 2009, 2011, 2013, 2015, 2017 and the Geographic Area is also moderately low compared to other states in those years. Here the first row of the tri-cluster table shows the same result.
In the second row of tri-cluster table we can observe that, for the year 2003, 2005, 2009, 2011, 2013, 2015 and 2017 Andhra Pradesh has moderately low total mangrove area, moderately low total forest area, moderately low medium dense mangrove area, moderately low very dense mangrove area, very high medium dense mangrove area, very high open mangrove area and very high open forest area. The same thing we can observe in our main dataset.
Now suppose in fourteenth row of the tri-cluster table, it is stated that Andhra Pradesh and Karnataka have moderately low total forest area, moderately low very dense forest area, moderately high open forest area, moderately high medium dense forest area, moderately high medium dense mangrove area and moderately high open mangrove area in the year 2005 and this state has a moderately high geographic area with respect to other states consider in that year. But in the 2015's main dataset, we can observe that medium dense mangrove and open mangrove area are totally occupied different measurement of area. There Andhra Pradesh occupies moderately high medium dense mangrove area and open mangrove area but Karnataka occupies the very low almost negligible area for medium dense mangrove and open mangrove. This kind of dissimilarities we can observe in some tri-clusters.
But except the fourteenth and seventeenth tri-cluster, all other tri-clusters satisfy the main datasets observation result.

5.3 Statistical Result Analysis

According to Indian govertment forest data[1] we can observe that, in the states Andhra Pradesh, Goa, Gujrat, Karnataka, Maharastha, Kerala, Oddisha, Tamil Nadu, West Bengal, Andaman Nicobbor Iceland, Damn and Diu and Puducherry, the forest cover was

[1]`https://community.data.gov.in/forest-cover-of-india-from-1987-to-2015/`

250776 square kilometer in 2003. In percentage terms, it was 17.96% of total geographic area. The forest cover was 250875 square kilometer in 2005. There was an increase of 99 square kilometer in forest cover as compared to 2003. The percentage of forest cover to the total geographical area was 18.04% in 2005. The total mangrove forest cover in 2003 and in 2005 is 0.32% in both the year, that means there was no change total mangrove forest area.

The forest cover was 257936 square kilometer in 2009. There was an increase of 7061 square kilometer in forest cover as compared to 2005. The percentage of forest cover to the total geographical area was 18.55% in 2009. The forest cover was 265388 square kilometer in 2011. There was an increase of 7452 square kilometer in forest cover as compared to 2009. The percentage of forest cover to the total geographical area was 19.09% in 2011. The total mangrove forest cover in 2009 and in 2011 was 0.33% and 0.32% respectively, that means there was a decrease in total mangrove forest area.

The forest cover was 291943.3 square kilometer in 2013. There was an increase of 26555.3 square kilometer in forest cover as compared to 2011. The percentage of forest cover to the total geographical area was 20.59% in 2013. The forest cover was 247949 square kilometer in 2015. There was a decrease of 43994.3 square kilometer in forest cover as compared to 2013. The percentage of forest cover to the total geographical area was 19.35% in 2015. The total mangrove forest cover in 2013 and in 2015 was 0.33% and 0.36% respectively, that means there was an increase in total mangrove forest area.

The forest cover was 254975.2 square kilometer in 2017. There was an increase of 7026.2 square kilometer in forest cover as compared to 2015. The percentage of forest cover to the total geographical area was 19.85% in 2013. The total mangrove forest cover in 2017 was 0.38%, that means there was an increase in total mangrove forest area.

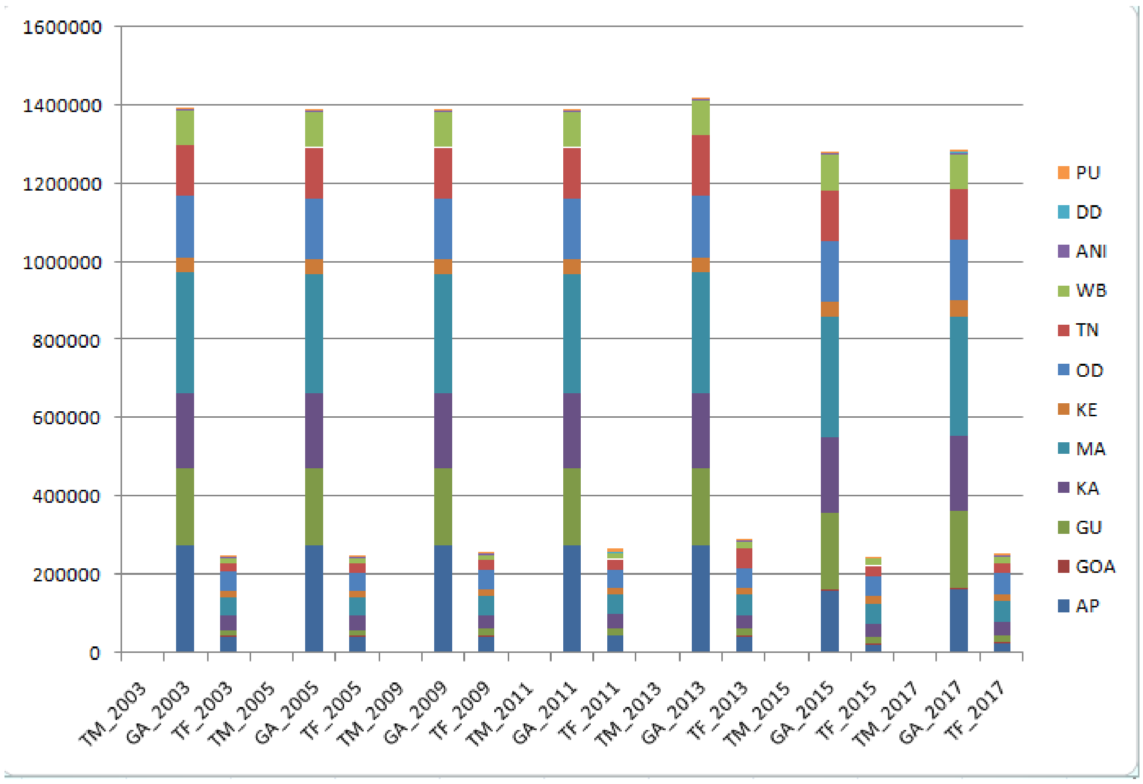


Figure 5.1: Year Wise Result analysis

Chapter 6

Conclusion

In this thesis, a new method of tri-clustering is proposed. Any kind of clustering technique needs some computer memory storage structure. Generalized suffix forest is a totally new kind of data structure proposed in this thesis. Previously mainly hash tables are used as data structure method for storing clusters. Generalized suffix forest utilize less memory space than a hash table.

In Chapter 3, the first proposed algorithm may vary upon datasets. So, after pre-processing of any dataset we can apply algorithm 2 to algorithm 8. Algorithm 2 will take $O(n+m)^p$ time complexity, here n is for sorting algorithm, m is for number of rows in SFD and p is for number of input pre-processed database. Algorithm 3 will take $O(n)^2$ time to create frequent generalised suffix tree. Here first n is used to create nodes in the suffix tree, and second, n is used to store the strings. Algorithm 7 will take $O(n + m)$ time to execute, here n time is required to merge two suffix tree and m is for adding conditions to $NewObjectList$. This 7^{th} algorithm will run $p - 1$ times, where p is the number of suffix tree. The 8^{th} algorithm which is used for extracting the tri-clusters from previously generated fully complete generalized suffix forest will take, $O(mn)$ times, here m is for number of rows created by extracting all the nodes, and n is for number of items per row, so that matching is done from Item Table.

These algorithms are used on Indian forest distribution dataset for checking purpose whether these algorithms give desired output or not.

Chapter 7

Partial Development of Proposed Algorithm

Online Prime OAC Tri-Clustering using Suffix Tree

7.1 Online version of Prime OAC tri-clustering

7.1.1 For first iteration

For the online version of prime OAC-tri-cluster, at first the user has no prior knowledge of the elements and even cardinalities of G,M,B and I. At first step of this below mentioned Algorithm 9 user will receive a tri-context $\kappa = (G, M, B, I)$ (For each combination of elements from each two sets of we compute the result of applying the corresponding prime operator.). After that a user define minimal density threshold (ρ_{min}) is considered as input (to check whether the newly generated tri-cluster is present in the previously generated tri-cluster or not, which are in suffix tree.). Now user can apply the prime operator on each of the combinations which are generated from tri-context κ.(let consider the prime operation is left shift by 5 in this example) (line 2-10). After this step we get a tri-cluster $T = ((m, b)', (g, b)', (g, m)')$ which is a prime OAC tri-cluster and $(g, m, b) \in I$ is called generating triple of tri-cluster T. It is important in this setting to consider every pair of tri-clusters is different if they have different generating triples, even if their extent, intent and modus are equal, because any other triple can change only one of them thus making them different.

Now we will check the newly generated tri-cluster is in suffix tree or not using minimal density threshold and density of the generated tri-cluster. If it is not in the suffix tree then we will insert the reference of that tri-cluster into the suffix tree and the actual tri-cluster will be stored into a dictionary (line 11-19).

Algorithm 9 Prime OAC tri-cluster for generating triple

```
Input: $\kappa = (G, M, B, I)$, $\rho_{min}$
Output: Triplate T
 1: begin:
 2: for all$(g, m) : g \in G \mathbf{and} m \in M$ do
 3:     $PrimeOA[g, m] = (g, m)'$
 4: end for
 5: for all$(m, b) : m \in M \mathbf{and} b \in B$ do
 6:     $PrimeAC[m, b] = (m, b)'$
 7: end for
 8: for all$(b, g) : b \in B \mathbf{and} g \in G$ do
 9:     $PrimeOC[b, g] = (b, g)'$
10: end for
11: $IndexD = 1$
12: for all$(g, m, b) \in I$ do
13:     T= $(PrimeOA[g, m] \parallel PrimeAC[m, b] \parallel PrimeOC[b, m])$
14:     $\mathrm{T}_{key} = hash(\mathrm{T})$
15:     if $\mathrm{T}_{key} \notin T^{\rho(T)}_{IndexD} \geq \rho_{min}$ then
16:         $T[\mathrm{T}_{key}]$ =T
17:     end if
18:     $IndexD++$
19: end for
```

7.1.2 For second iteration and above

In each iteration user receive a set of triple $J(J \subseteq I)$ in Algorithm 10. After that the current set of tri-cluster (T) and the dictionary (T) of prime set is taken as input. Then the algorithm process each triple (g, m, b) of J sequentially and add each of them with dictionary prime sets.

- Adds b to $(g, m)'$
- Adds g to $(b, m)'$
- Adds m to $(g, b)'$ [line 6-8]

Now add the reference of currently generated tri-cluster into the tri-cluster set nothing but add the terminator of suffix tree and update the dictionary with newly generated tri-cluster.[line 9-14]

Algorithm 10 Online version of Prime OAC tri-cluster for incoming triples based on generating triple

```
Input: $J$ {triple coming from source}, T {current set of tri-cluster}, $T$ {current dictionary}
Output: Full dictionary $T$ and final triplet T
1: begin:
2: $IndexT = 1$
3: $IndexD$ {current index value of dictionary}
4: $PGT = IndexD$
5: for all$(g, m, b) \in J$ do
6:     $PrimeOA[g, m] = PrimeOA[g, m] \cup b$
7:     $PrimeAC[m, b] = PrimeAC[m, b] \cup g$
8:     $PrimeOC[b, g] = PrimeOC[b, g] \cup m$
9:     T= $(PrimeOA[g, m] \parallel PrimeAC[m, b] \parallel PrimeOC[b, m])$
10:    T$_{key} = hash$(T)
11:    $IndexT{+}{+}$
12:    $T = T \cup (PrimeOA[g, m], PrimeAC[m, b], PrimeOC[b, m])$
13:    $IndexD{+}{+}$
14: end for
```

Redundancy Checking Phase After this we have to check the newly generated triple is present on the full tri-cluster set or not (Algorithm 11). If it is not there then we have to insert that. Through this algorithm if there is any duplicate triple generate then we can remove that easily.

Algorithm 11 Post processing algorithm

```
Input: T {full set tri-cluster}, $\Gamma$ {newly generated triple}
Output: Non repeated triples in tri-cluster set
1: for all T$\in$ do
2:     calculate hash(T)
3:     if hash(T)$\notin \Gamma$ then
4:         T=T$\cup\Gamma \& T = \cup\Gamma$
5:     end if
6: end for
```

7.1.3 Jenkin's Hash Function

The Jenkin's hash fuction algorithm stated below (Algorithm 12)

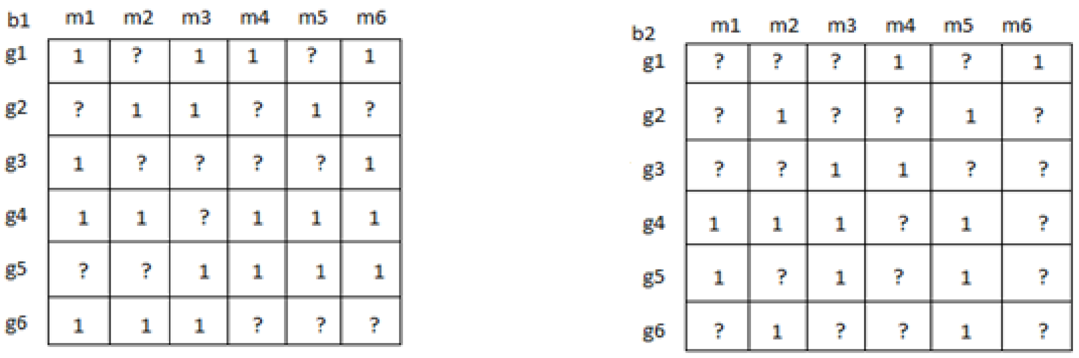
Algorithm 12 Jenkin's Hash Funclion

```
Input: Message, length of the message (say n)
Output: Encrypted message
 1: $i = 0$
 2: $hash = 320's$
 3: while $i \neq n$ do
 4:     $hash = hash + message$
 5:     $hash = hash + (hash << 10)$
 6:     $hash = hash^{(}hash >> 6)$
 7:     $i + +$
 8: end while
 9: $hash = hash + (hash << 3)$
10: $hash = hash^{(}hash >> 11)$
11: $hash = hash + (hash << 15)$
```

7.2 Explain with Example

7.2.1 For first iteration

Step 1:- Let consider a data matrix consist of 6 objects, 6 attributes under 1 condition; and another data matrix consist of 6 objects, 6 attributes under another condition. Therefore G=6, M=6, B=2, I=72. In Figure 7.1 '1' means that cell contain some data and '?'means that cell does not contain any data. This implies that we can make tri-clusters of only those (g, m, b)'s which contain a 1 in their cell.



b1	m1	m2	m3	m4	m5	m6
g1	1	?	1	1	?	1
g2	?	1	1	?	1	?
g3	1	?	?	?	?	1
g4	1	1	?	1	1	1
g5	?	?	1	1	1	1
g6	1	1	1	?	?	?

b2	m1	m2	m3	m4	m5	m6
g1	?	?	?	1	?	1
g2	?	1	?	?	1	?
g3	?	?	1	1	?	?
g4	1	1	1	?	1	?
g5	1	?	1	?	1	?
g6	?	1	?	?	1	?

Figure 7.1: Two sample binary value dataset

Step 2:- Now we have to take a minimal density threshold, let $\rho_{min} = 0.08037$.

Step 3:- Now let's consider two a triples (g_5, m_4, b_1) and (g_6, m_2, b_2)[1]. All combination of these triples are $((g_5, m_4), (m_4, b_1), (b_1, g_5))$ and $((g_6, m_2), (m_2, b_2), (b_2, g_6))$. Now let g_5 = 0000000010000101, m_4 = 0000001000011000 and b_1 = 0000000101010010 &

[1]$(g_5, m_4, b_1) \in I \& (g_6, m_2, b_2) \in I$ is called generating triple of this tri-cluster T

$g_6 = 1001101001011110$, $m_2 = 0110010110100001$ and $b_2 = 1111000010100101$.

After applying the prime operator on this combinations we get T= $((g_5, m_4)', (m_4, b_1)', (b_1, g_5)') =$ $((0001000010100000, 0100001100000000), (0100001100000000, 0010100010000000), (001010001000000$
and $((g_6, m_2)', (m_2, b_2)', (b_2, g_6)') = ((1001101001011110000000, 1100101101000010000 0), (11001011010$
(Here in this example all the values of g_5, m_4 and $b_1 \& g_6, m_2$ and b_2 are represented as 16bit binary number but in reality it represented as 32bit binary number.)

Step 4:- After that a tri-cluster set (T) table is being prepared which have an index value, hash value of tri-cluster set. Here hash values are generated by using Jenkin's hash function. Before applying the Jenkin's hash function we convert these 16-bit binary value to 32-bit binary value(by adding 16 0's at front) (Table 7.1).

Table 7.1: Tri-cluster set table

IndexT	Tri-cluster-sets(T) (hash value)
1	((E0B0FA6C, A4E77B5E), (A4E77B5E, C2761DEB), (C2761DEB, E0B0FA6C))
2	((D436FB87, 407AD7ED),(407AD8ED, 8D34E241), (8D34E241, D436FB87))

Step 5:- Now we have to calculate T_{IndexD} i.e. $1^{\rho(T)} = 1 \geq \rho_{min}$. So we can insert these $IndexT$ into the suffix tree. At first as a root 4294967296 (which is 2^{32}) cells are created. This suffix tree structure is shown below Figure 7.2.

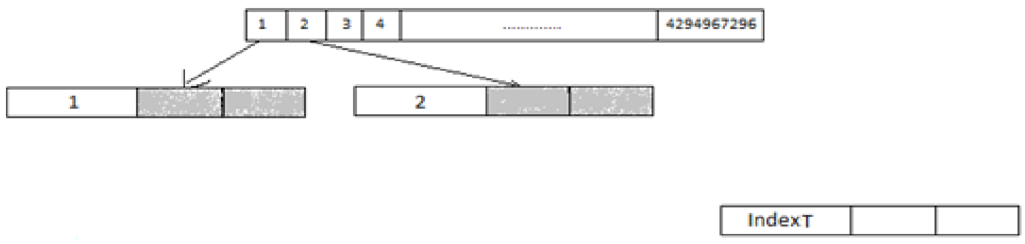


Figure 7.2: Primarily Created Suffix Tree

This process continued for all the entries of the data matrix.

7.2.2 For second iteration Example 1

Step 1:- In this iteration we receive a triple J. Suppose $(g, m, b) \in I$.

Step 2:- Now we have to make all combinations of this triple. That is $((g, m), (m, b), (b, g))$.

Suppose $g = 0000001000001111, m = 0000000111110001 and b = 0000010000000101$

Step 3:- In this step we have to add each item of the coming triple to the tri-cluster sets.

- Add b to $(g_6, m_2)'$

  $b = 0000010000000101$

  $(g_6, m_2)' = (10011010010111000000, 11001011010000100000)$

  After addition we get: $(10011010100111100101, 11001011100000100101)$

- Add g to $(m_2, b_2)'$

  $g = 0000001000001111$

  $(m_2, b_2)' = (11001011010000100000, 11110000101001010000 0)$

  After addition we get: $(11001011011000101111, 11110000101101010 1111)$

- Add m to $(b_2, g_6)'$

  $m = 0000000111110001$

  $(b_2, g_6)' = (11110000101001010000 0, 10011010010111000000)$

  After addition we get: $(11110000101101001 0001, 10011010011111010001)$

Step 4:- Now we have to create a Dictionary table 7.2 where the hash value of newly generated tri-cluster is stored with its index value (IndexD).

Table 7.2: Dictionary table 1

IndexD	Dictionary Set (hash value)
1	((BBC14F38, 407AD7ED), (FDC7E34C, 1EC0FDBD), (44EB2436, 0094A210))

Step 5:- Now we have to update the tri-cluster set table and add a column here which hold the IndexD value which is actually a pointer to dictionary table that the newly generated tri-cluster is generated from that tri-cluster set (Table 7.3).

Table 7.3: Updated hash table 1

IndexT	Tri-cluster-sets(T) (hash value)	Pointers of generating triple (IndexD)
1	((E0B0FA6C, A4E77B5E), (A4E77B5E, C2761DEB), (C2761DEB, E0B0FA6C))	null
2	((D436FB87, 407AD7ED), (407AD8ED, 8D34E241), (8D34E241, D436FB87))	1

Step 6:- Now in suffix tree will look like this where the suffix terminator will hold the pointers of dictionary table (Figure 7.3).

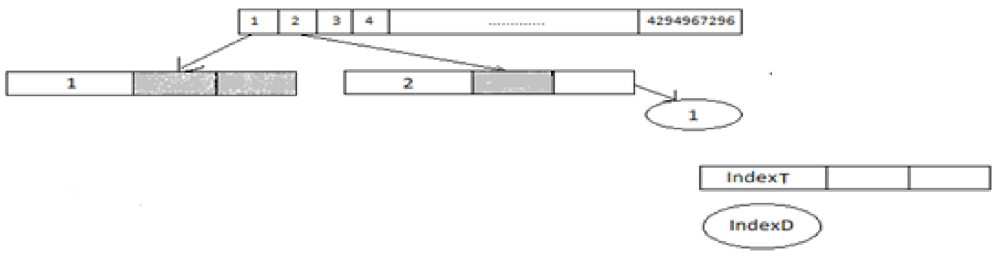


Figure 7.3: Primarily Created Suffix Tree for Iteration 2 (example 1)

7.2.3 For second iteration Example 2

Step 1:- In this iteration we receive a triple J. Suppose $(g, m, b) \in I$.

Step 2:- Now we have to make all combinations of this triple. That is $((g, m), (m, b), (b, g))$.

Suppose $g = 1111001001001111, m = 1111110111110001 and b = 0000010011100101$

Step 3:- In this step we have to add each item of the coming triple to the tri-cluster sets.

- Add b to $(g_5, m_4)'$

  $b = 0000010011100101$

  $(g_5, m_4)' = (0001000010100000, 0100001100000000)$

  After addition we get: $(0001010110000101, 0100011111100101)$

- Add g to $(m_4, b_1)'$

  $g = 1111001001001111$

  $(m_4, b_1)' = (0100001100000000, 0010100010000000)$

  After addition we get: $(10011010101001111, 10001101011001111)$

- Add m to $(b_1, g_5)'$

  $m = 1111110111110001$

  $(b_1, g_5)' = (0010100010000000, 0001000010100000)$

  After addition we get: $(10010011001110001, 1000011101001000)$

Step 4:- Now we have to create a Dictionary table 7.4 where the hash value of newly generated tri-cluster is stored with its index value (IndexD).

Table 7.4: Dictionary table 2

IndexD	Dictionary Set (hash value)
1	((BBC14F38, 407AD7ED), (FDC7E34C, 1EC0FDBD), (44EB2436, 0094A210))
2	((5AF19CAC, C5A723DD), (1802436D, 3B20ABA9), (2A43301C, F14436CE))

Step 5:- Now we have to update the tri-cluster set table and add a column here which hold the IndexD value which is actually a pointer to dictionary table that the newly generated tri-cluster is generated from that tri-cluster set (Table 7.5).

Table 7.5: Updated hash table 2

IndexT	Tri-cluster-sets(T) (hash value)	Pointers of generating triple (IndexD)
1	((E0B0FA6C, A4E77B5E), (A4E77B5E, C2761DEB), (C2761DEB, E0B0FA6C))	2
2	((D436FB87, 407AD7ED), (407AD8ED, 8D34E241), (8D34E241, D436FB87))	1

Step 6:- Now in suffix tree will look like this where the suffix terminator will hold the pointers of dictionary table (Figure 7.4).

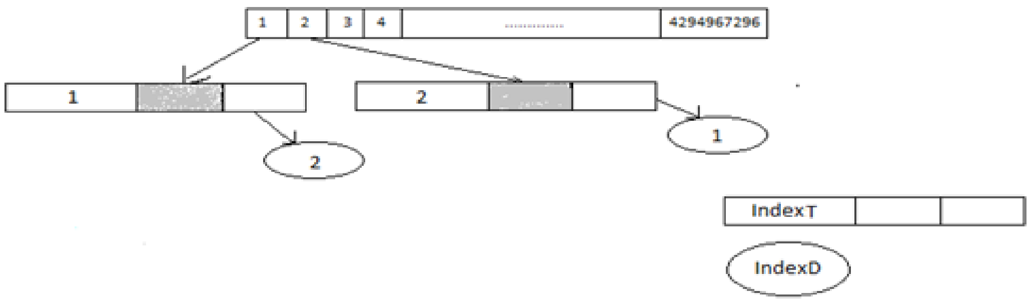


Figure 7.4: Primarily Created Suffix Tree for Iteration 2 (example 2)

This process will be continued for other entries of data matrix and incoming triple. For other entries from data matrix will be generating triple and pointing from root node of the suffix tree. And after that which triple will come from outer source, will generate a tri-cluster using that generating triple.

Bibliography

[1] C Robert et al. Cluster analysis: correlation profile and orthometric (factor) analysis for the isolation of unities in mind and personality. *Edwards Brothers*, 1939.

[2] Vladimir Estivill-Castro. Why so many clustering algorithms: a position paper. *SIGKDD explorations*, 4(1):65–75, 2002.

[3] Kartick Chandra Mondal. *Algorithms for Data Mining and Bio-informatics*. PhD thesis, Université Nice Sophia Antipolis, 2013.

[4] Edward M McCreight. A space-economical suffix tree construction algorithm. *Journal of the ACM (JACM)*, 23(2):262–272, 1976.

[5] Martin Farach. Optimal suffix tree construction with large alphabets. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 137–143. IEEE, 1997.

[6] Marie-France Sagot. Spelling approximate repeated or common motifs using a suffix tree. In *Latin American Symposium on Theoretical Informatics*, pages 374–390. Springer, 1998.

[7] Alberto Apostolico, Costas Iliopoulos, Gad M. Landau, Baruch Schieber, and Uzi Vishkin. Parallel construction of a suffix tree with applications. *Algorithmica*, 3(1-4):347–365, 1988.

[8] Robert Giegerich and Stefan Kurtz. From ukkonen to mccreight and weiner: A unifying view of linear-time suffix tree construction. *Algorithmica*, 19(3):331–353, 1997.

[9] Marcel H Schulz, Sebastian Bauer, and Peter N Robinson. The generalised k-truncated suffix tree for time-and space-efficient searches in multiple dna or pro-

tein sequences. *International journal of bioinformatics research and applications*, 4(1):81–95, 2008.

[10] Tristan Snowsill and Florent Nicart. A hard-disk based suffix tree implementation. Technical report, Technical Report 133088, University of Bristol, Bristol, UK, 2011.

[11] Lifeng Jia, Chunguang Zhou, Zhe Wang, and Xiujuan Xu. Suffixminer: Efficiently mining frequent itemsets in data streams by suffix-forest. In *International Conference on Fuzzy Systems and Knowledge Discovery*, pages 592–595. Springer, 2005.

[12] Sara C Madeira and Arlindo L Oliveira. Biclustering algorithms for biological data analysis: a survey. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 1(1):24–45, 2004.

[13] Evan M Adams. Using migration monitoring data to assess bird population status and behavior in a changing environment. 2014.

[14] Kartick Chandra Mondal, Nicolas Pasquier, Anirban Mukhopadhyay, Ujjwal Maulik, and Sanghamitra Bandhopadyay. A new approach for association rule mining and bi-clustering using formal concept analysis. In *International Workshop on Machine Learning and Data Mining in Pattern Recognition*, pages 86–101. Springer, 2012.

[15] Ning Li, Li Zeng, Qing He, and Zhongzhi Shi. Parallel implementation of apriori algorithm based on mapreduce. In *2012 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, pages 236–241. IEEE, 2012.

[16] Gerd Stumme. Formal concept analysis. In *Handbook on ontologies*, pages 177–199. Springer, 2009.

[17] Ali Jaoua and Samir Elloumi. Galois connection, formal concepts and galois lattice in real relations: application in a real classifier. *Journal of Systems and Software*, 60(2):149–163, 2002.

[18] Wei Yao and Ling-Xia Lu. Fuzzy galois connections on fuzzy posets. *Mathematical Logic Quarterly*, 55(1):105–112, 2009.

[19] Peter D Turney. The identification of context-sensitive features: A formal definition of context for concept learning. *arXiv preprint cs/0212038*, 2002.

[20] Gerd Stumme. Efficient data mining based on formal concept analysis. In *International Conference on Database and Expert Systems Applications*, pages 534–546. Springer, 2002.

[21] Sergei O Kuznetsov. Machine learning and formal concept analysis. In *International Conference on Formal Concept Analysis*, pages 287–312. Springer, 2004.

[22] Isabelle Bloch. Information combination operators for data fusion: a comparative review with classification. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 26(1):52–67, 1996.

[23] Rudolf Wille. Formal concept analysis as mathematical theory of concepts and concept hierarchies. In *Formal concept analysis*, pages 1–33. Springer, 2005.

[24] Yiyu Yao and Yaohua Chen. Rough set approximations in formal concept analysis. In *Transactions on rough sets V*, pages 285–305. Springer, 2006.

[25] Lizhuang Zhao and Mohammed J Zaki. Tricluster: an effective algorithm for mining coherent clusters in 3d microarray data. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 694–705. ACM, 2005.

[26] P Mahanta, HA Ahmed, DK Bhattacharyya, and Jugal K Kalita. Triclustering in gene expression data analysis: a selected survey. In *2011 2nd National Conference on Emerging Trends and Applications in Computer Science*, pages 1–6. IEEE, 2011.

[27] Ao Li and David Tuck. An effective tri-clustering algorithm combining expression data with gene regulation information. *Gene regulation and systems biology*, 3:GRSB–S1150, 2009.

[28] Tian Wang, Hamid Krim, and Yannis Viniotis. A generalized markov graph model: Application to social network analysis. *IEEE Journal of Selected Topics in Signal Processing*, 7(2):318–332, 2013.

[29] Jinhui Tang, Xiangbo Shu, Guo-Jun Qi, Zechao Li, Meng Wang, Shuicheng Yan, and Ramesh Jain. Tri-clustered tensor completion for social-aware image tag refine-

ment. *IEEE transactions on pattern analysis and machine intelligence*, 39(8):1662–1674, 2017.

[30] David Gutiérrez-Avilés, Cristina Rubio-Escudero, Francisco Martínez-Álvarez, and José C Riquelme. Trigen: A genetic algorithm to mine triclusters in temporal gene expression data. *Neurocomputing*, 132:42–53, 2014.

[31] Rui Henriques and Sara C Madeira. Triclustering algorithms for three-dimensional data analysis: A comprehensive survey. *ACM Computing Surveys (CSUR)*, 51(5):95, 2018.

[32] David Gutierrez-Aviles and Cristina Rubio-Escudero. Lsl: A new measure to evaluate triclusters. In *2014 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 30–37. IEEE, 2014.

[33] Seokjong Lee, Wonchae Lee, Yoonseok Choe, Dowon Kim, Gunyeon Na, Jinoh Kim, Moonkyu Kim, Jungchul Kim, and Joonyong Cho. Gene expression profiles in varicose veins using complementary dna microarray. *Dermatologic surgery*, 31(4):391–395, 2005.

[34] David Gutiérrez-Avilés and Cristina Rubio-Escudero. Triq: A comprehensive evaluation measure for triclustering algorithms. In *International Conference on Hybrid Artificial Intelligence Systems*, pages 673–684. Springer, 2016.

[35] David Gutiérrez-Avilés, Raúl Giráldez, Francisco Javier Gil-Cumbreras, and Cristina Rubio-Escudero. Triq: a new method to evaluate triclusters. *BioData mining*, 11(1):15, 2018.

[36] Gerd Stumme. A finite state model for on-line analytical processing in triadic contexts. In *International Conference on Formal Concept Analysis*, pages 315–328. Springer, 2005.

[37] Graham Cormode and Divesh Srivastava. Anonymized data: generation, models, usage. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 1015–1018. ACM, 2009.

[38] Ankush Gupta, Andrea Vedaldi, and Andrew Zisserman. Synthetic data for text localisation in natural images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2315–2324, 2016.

[39] Robert Jaschke, Andreas Hotho, Christoph Schmitz, Bernhard Ganter, and Gerd Stumme. Trias–an algorithm for mining iceberg tri-lattices. In *Sixth International Conference on Data Mining (ICDM'06)*, pages 907–911. IEEE, 2006.

[40] Huaiguo Fu and Engelbert Mephu Nguifo. A parallel algorithm to generate formal concepts for large data. In *International Conference on Formal Concept Analysis*, pages 394–401. Springer, 2004.

[41] Sabine Krolak-Schwerdt, Peter Orlik, and Bernhard Ganter. Tripat: a model for analyzing three-mode binary data. In *Information Systems and Data Analysis*, pages 298–307. Springer, 1994.

[42] Dmitry I Ignatov, Sergei O Kuznetsov, Ruslan A Magizov, and Leonid E Zhukov. From triconcepts to triclusters. In *International Workshop on Rough Sets, Fuzzy Sets, Data Mining, and Granular-Soft Computing*, pages 257–264. Springer, 2011.

[43] Dmitry Gnatyshak, Dmitry I Ignatov, and Sergei O Kuznetsov. From triadic fca to triclustering: Experimental comparison of some triclustering algorithms. In *CLA*, volume 1062, pages 249–260. Citeseer, 2013.

[44] Sergey Zudin, Dmitry V Gnatyshak, and Dmitry I Ignatov. Putting oac-triclustering on mapreduce. In *CLA*, pages 47–58, 2015.

[45] André Crosnier and JR Rossignac. Tribox bounds for three-dimensional objects. *Computers & Graphics*, 23(3):429–437, 1999.

[46] Fumito Yamaguchi and Hiroaki Nishi. Hardware-based hash functions for network applications. In *2013 19th IEEE International Conference on Networks (ICON)*, pages 1–6. IEEE, 2013.

[47] Dmitry Gnatyshak, Dmitry I Ignatov, Sergei O Kuznetsov, and Lhouari Nourine. A one-pass triclustering approach: Is there any room for big data? In *CLA*, pages 231–242, 2014.

[48] Dmitry V Gnatyshak. Greedy modifications of oac-triclustering algorithm. *Procedia Computer Science*, 31:1116–1123, 2014.

[49] DV Gnatyshak. A single-pass triclustering algorithm. *Automatic Documentation and Mathematical Linguistics*, 49(1):27–41, 2015.

[50] Rokia Bendaoud, Amedeo Napoli, and Yannick Toussaint. Formal concept analysis: A unified framework for building and refining ontologies. In *International Conference on Knowledge Engineering and Knowledge Management*, pages 156–171. Springer, 2008.

[51] Dmitry I Ignatov, Dmitry V Gnatyshak, Sergei O Kuznetsov, and Boris G Mirkin. Triadic formal concept analysis and triclustering: searching for optimal patterns. *Machine Learning*, 101(1-3):271–302, 2015.

[52] Petri Kontkanen, Petri Myllymäki, Wray Buntine, Jorma Rissanen, and Henry Tirri. 1 an mdl framework for data clustering. *Minimum*, page 323, 2005.

[53] Romain Guigourès, Marc Boullé, and Fabrice Rossi. A triclustering approach for time evolving graphs. In *2012 IEEE 12th International Conference on Data Mining Workshops*, pages 115–122. IEEE, 2012.

[54] Romain Guigourès, Marc Boullé, and Fabrice Rossi. Discovering patterns in time-varying graphs: a triclustering approach. *Advances in Data Analysis and Classification*, 12(3):509–536, 2018.

[55] Yongli Liu, Tengfei Yang, and Lili Fu. A partitioning based algorithm to fuzzy tricluster. *Mathematical Problems in Engineering*, 2015, 2015.

[56] Dmitry I Ignatov, Sergei O Kuznetsov, Jonas Poelmans, and Leonid E Zhukov. Can triconcepts become triclusters? *International Journal of General Systems*, 42(6):572–593, 2013.

[57] Rudolf Wille. The basic theorem of triadic concept analysis. *Order*, 12(2):149–158, 1995.

[58] Fritz Lehmann and Rudolf Wille. A triadic approach to formal concept analysis. In *International Conference on Conceptual Structures*, pages 32–43. Springer, 1995.

[59] Radim Belohlavek and Petr Osicka. Triadic fuzzy galois connections as ordinary connections. *Fuzzy Sets and Systems*, 249:83–99, 2014.

[60] Hassan Chafi, Arvind K Sujeeth, Kevin J Brown, HyoukJoong Lee, Anand R Atreya, and Kunle Olukotun. A domain-specific approach to heterogeneous parallelism. *ACM SIGPLAN Notices*, 46(8):35–46, 2011.

[61] Sheng-yi Jiang, Xia Li, Qi Zheng, and Lian-xi Wang. Approximate equal frequency discretization method. In *2009 WRI Global Congress on Intelligent Systems*, volume 3, pages 514–518. IEEE, 2009.

[62] José M Jerez, Ignacio Molina, Pedro J García-Laencina, Emilio Alba, Nuria Ribelles, Miguel Martín, and Leonardo Franco. Missing data imputation using statistical and machine learning methods in a real breast cancer problem. *Artificial intelligence in medicine*, 50(2):105–115, 2010.

[63] Forest survey of india, ministry of environment, forest climate change, state of forest report, 2003, 2005, 2009, 2011, 2013, 2015, 2017.

[64] Joseph E Beck and Beverly Park Woolf. High-level student modeling with machine learning. In *International Conference on Intelligent Tutoring Systems*, pages 584–593. Springer, 2000.

Appendix A

Kind of Data to be used

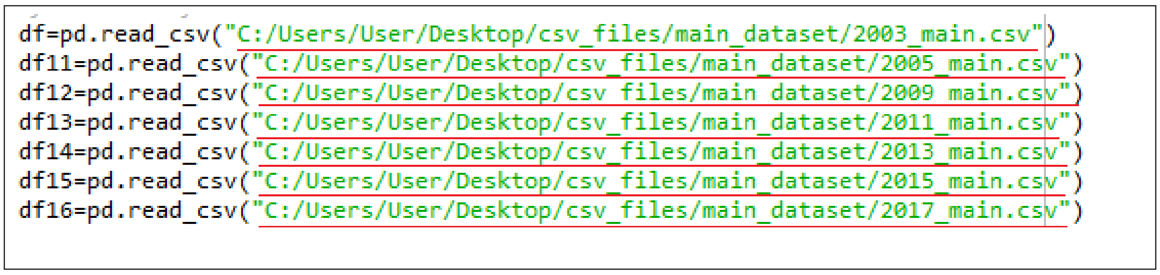
If the user wants to see the default dataset's execution value, then the user has to give the dataset, which contains the same object, attribute, condition valued dataset as like default datasets which are given or which is used in this experiment. Otherwise, if the user wants to see other's dataset's execution value, which has the totally different object, attribute, condition values, then the user may first pre-process the dataset (the desired procedure is given in next section) and then insert it on runtime when the program will ask for data.

Appendix B

Data Pre-processing Techniques

B.1 For default dataset

For the default dataset or same object, attribute, condition valued dataset, user just have to upload the raw datasets in described manner. (Figure B.1)



```
df=pd.read_csv("C:/Users/User/Desktop/csv_files/main_dataset/2003_main.csv")
df11=pd.read_csv("C:/Users/User/Desktop/csv_files/main_dataset/2005_main.csv")
df12=pd.read_csv("C:/Users/User/Desktop/csv_files/main_dataset/2009_main.csv")
df13=pd.read_csv("C:/Users/User/Desktop/csv_files/main_dataset/2011_main.csv")
df14=pd.read_csv("C:/Users/User/Desktop/csv_files/main_dataset/2013_main.csv")
df15=pd.read_csv("C:/Users/User/Desktop/csv_files/main_dataset/2015_main.csv")
df16=pd.read_csv("C:/Users/User/Desktop/csv_files/main_dataset/2017_main.csv")
```

Figure B.1: Default Dataset Upload

The user just have to change red underlined part of the above figure in python code to see the output of the default dataset. The .csv file name must be same for the different data valued dataset, which has same condition, object and attribute name but has different numeric values for these entities.

B.2 For user given dataset

User must aware of his/her dataset. This algorithm will take only labeled data or binary data as its input. To make one dataset labeled or binary user can use the following steps as per requirement until they get their desired output $\rightarrow$

1. **One Hot Encoding:-** This can be done to transform one dataset into binary dataset. But one drawback is number of objects and attributes are increased here so, user

may not get desired output.

2. **Equal Frequency Discretization:-** This can be done when user wants some labeled data. This may not ensure that, user will get 1,2,3,4,5 labeled dataset. But this method discretize a dataset in such manner that user can easily understand their datset behaviour. This kind of discretization method does not depend on actual values, it depend on number of values.

3. **Equal Width Discretization:-** This method is also same like Equal Frequency Discretization, only difference is that, it is concerned with actual values, rather than number of values. All these discretization methods comes under unsupervised learning technique.

4. **Domain Knowledge Application:-** This method is totally intuition based method. As long as user will be in the feature engineering field, he/she will become much expert to apply domain knowledge in his/her dataset. Domain knowledge application is nothing but mathematical rule application upon required features, such that addition, multiplication, division, subtraction, sin, cos, logarithm, square, square root, backward selection (used for binary classifier), majority votes (used for probability classifier) and many others to get desired output.

5. **Mean or Median value replacement:-**

   (a) If user wants binary valued dataset then he/she simply can replace, higher value of mean/median with '1' and lower value of mean/median with '0'.

   (b) If user wants labeled valued dataset, i.e. 1,2,3,4,5 valued dataset then, user can replace the maximum value of a feature with '5', less than maximum but more than mean/median with '4', equals to mean/median with '3', less than mean/median but more than minimum value of a feature with '2' and minimum value of a feature with '1'.

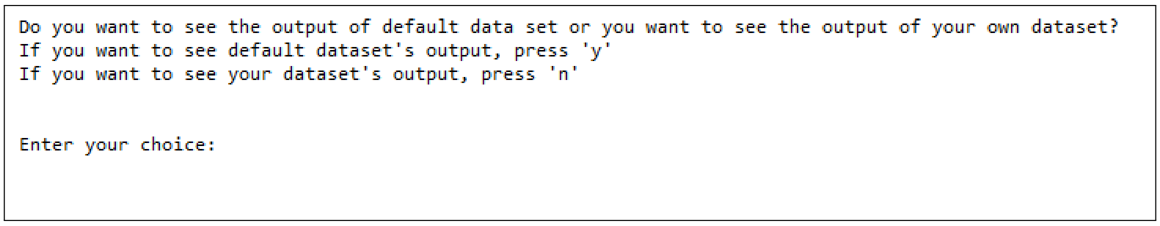
There are lots of other way to pre-process one dataset to make binary or labeled one. But one important caution should be maintained, that the dataset must not loss any important aspects.

Appendix C

User Guide

Before start running the program user should download the datasets and set the path mentioned in section B.1. After that user have to create $tri\_cluster\_output.csv$ and $bi\_clusters\_output.csv$ empty csv files under $.spyder - py3$ folder to store the output of user given dataset.

Now user can start running the program. The program will first ask the user whether the user wants to see the default dataset's output or user wants to give data by his own. (Figure C.1)



```
Do you want to see the output of default data set or you want to see the output of your own dataset?
If you want to see default dataset's output, press 'y'
If you want to see your dataset's output, press 'n'


Enter your choice:
```

Figure C.1: User's Choice

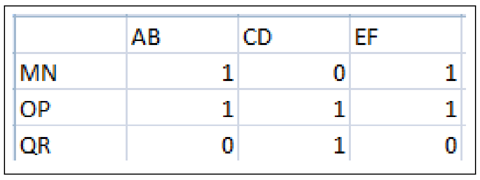
C.1 For default dataset

Here if user press $'y'$ then the program shows the desired output to the user in console, as well as some $.png$ and $.csv$ files will be stored in $.spyder - py3$ folder (those files are $\rightarrow$ 2003_dka.csv, 2005_dka.csv, 2009_dka.csv, 2011_dka.csv, 2013_dka.csv, 2015_dka.csv, 2017_dka.csv(these are the output after applying domain knowledge), 2003_dis.csv, 2005_dis.csv, 2009_dis.csv, 2011_dis.csv, 2013_dis.csv, 2015_dis.csv, 2017_dis.csv(these are the output after discretization), 2003_mr.csv, 2005_mr.csv, 2009_mr.csv, 2011_mr.csv, 2013_mr.csv, 2015_mr.csv, 2017_mr.csv(these are the output after mean value replacement), 2003_Item_Table.csv, 2003_Frequent_Item_Table_with_minsupport.csv, 2003_Item_Number_Added_with_sfi_Table.csv, Item_Number_Added_with_sfi_Table.csv, 2003_sfd.csv, 2005_sfd.csv, 2009_sfd.csv, 2011_sfd.csv, 2013_sfd.csv, 2015_sfd.csv, 2017_sfd.csv(these are the output for sorted frequent database), 2003_lattice.png, 2003_bi-cluster.csv, 2005_lattice.png, 2005_bi-cluster.csv, 2009_lattice.png, 2009_bi-cluster.csv, 2011_lattice.png, 2011_bi-cluster.csv, 2013_lattice.png, 2013_bi-cluster.csv, 2015_lattice.png, 2015_bi-cluster.csv, 2017_lattice.png, 2017_bi-cluster.csv(these are the output for individual bi-clusters), 200305_lattice.png, 200305.csv, 200911_lattice.png, 200911.csv, 201315_lattice.png, 201315.csv, 2003200520092011_lattice.png, 2003200520092011.csv, 201320152017_lattice.png, 201320152017.csv(these are for merging suffix trees), complete_lattice.png, complete.csv(these are for completely merge suffix forest), tricluster.csv(this is the final tri-cluster table output))

C.2 For user given dataset

Now if user press $'n'$ then the program will first ask for conditions, then object, then attribute name(if pre-processed data is binary then user can only give attribute name, and if pre-processed data is labeled then user have to give attribute value pair).

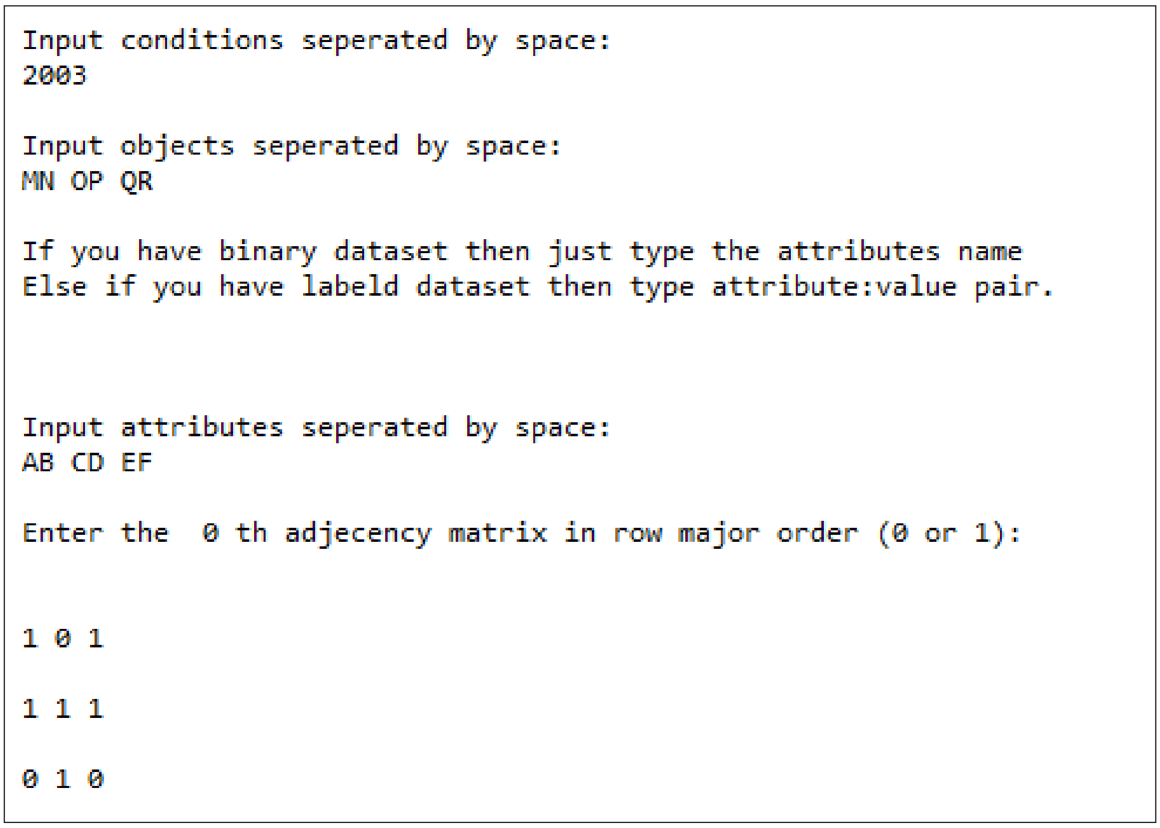
- If user have binary dataset, for condition, suppose 2003 (Figure C.2).



	AB	CD	EF
MN	1	0	1
OP	1	1	1
QR	0	1	0

Figure C.2: Pre-processed Binary Dataset

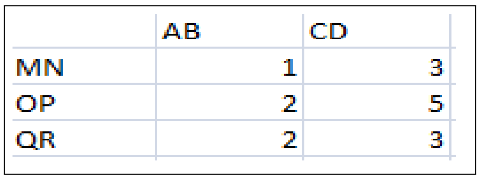
- Then user can give the input in below mentioned manner. (Figure C.3)



```
Input conditions seperated by space:
2003

Input objects seperated by space:
MN OP QR

If you have binary dataset then just type the attributes name
Else if you have labeld dataset then type attribute:value pair.



Input attributes seperated by space:
AB CD EF

Enter the  0 th adjecency matrix in row major order (0 or 1):


1 0 1

1 1 1

0 1 0
```

Figure C.3: Pre-processed Binary Dataset Input

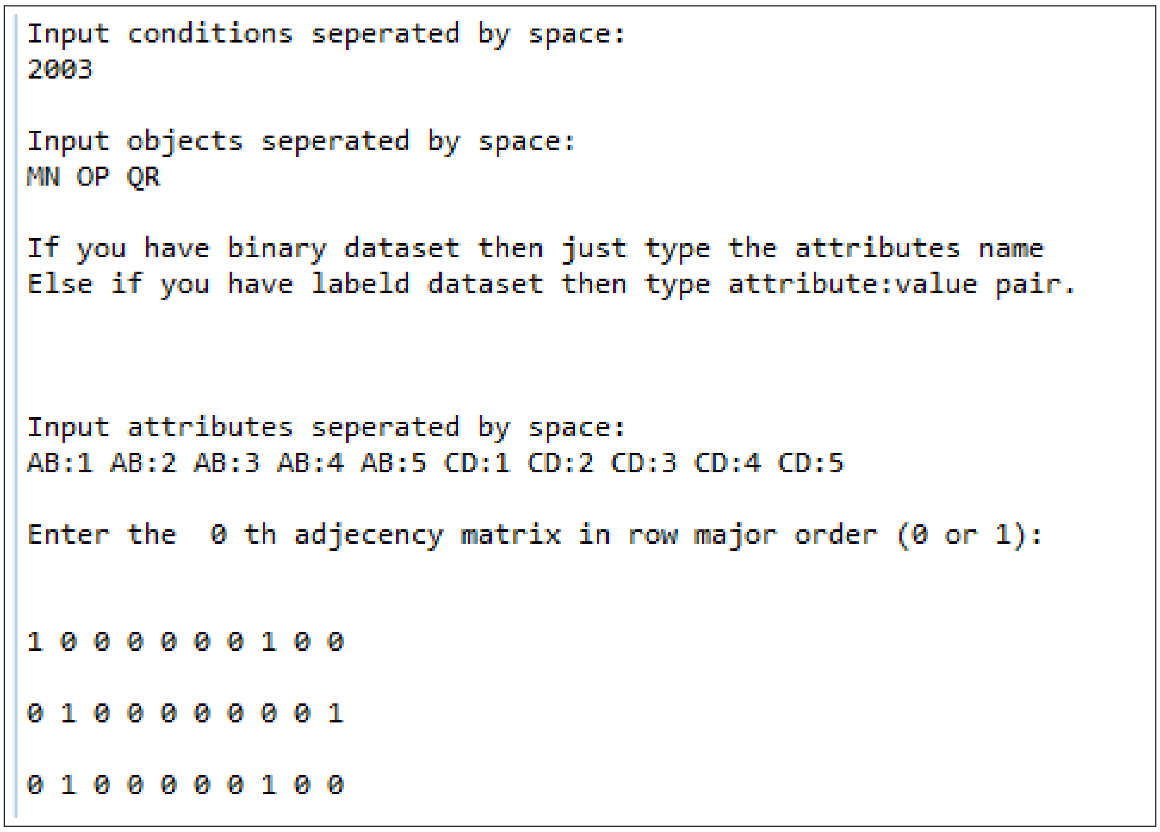
- If user have labeled dataset, for condition, suppose 2003 (Figure C.4).



	AB	CD
MN	1	3
OP	2	5
QR	2	3

Figure C.4: Pre-processed Labeled Dataset

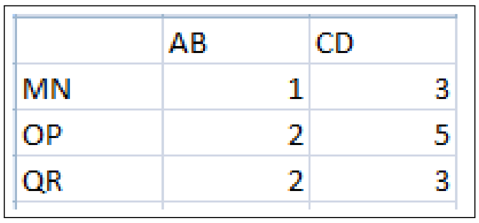
- Then user can give the input in below mentioned manner. (Figure C.5)



```
Input conditions seperated by space:
2003

Input objects seperated by space:
MN OP QR

If you have binary dataset then just type the attributes name
Else if you have labeld dataset then type attribute:value pair.



Input attributes seperated by space:
AB:1 AB:2 AB:3 AB:4 AB:5 CD:1 CD:2 CD:3 CD:4 CD:5

Enter the  0 th adjecency matrix in row major order (0 or 1):


1 0 0 0 0 0 0 1 0 0

0 1 0 0 0 0 0 0 0 1

0 1 0 0 0 0 0 1 0 0
```

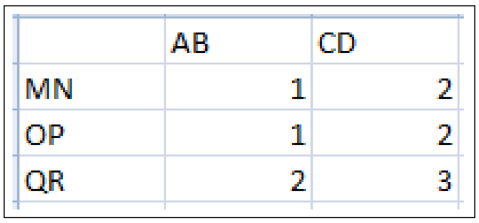
Figure C.5: Pre-processed Labeled Dataset Input

Now lets suppose user have two labeled dataset (Figure C.6 and Figure C.7) for condition 2003 and 2005.



	AB	CD
MN	1	3
OP	2	5
QR	2	3

Figure C.6: Labeled dataset for condition 2003

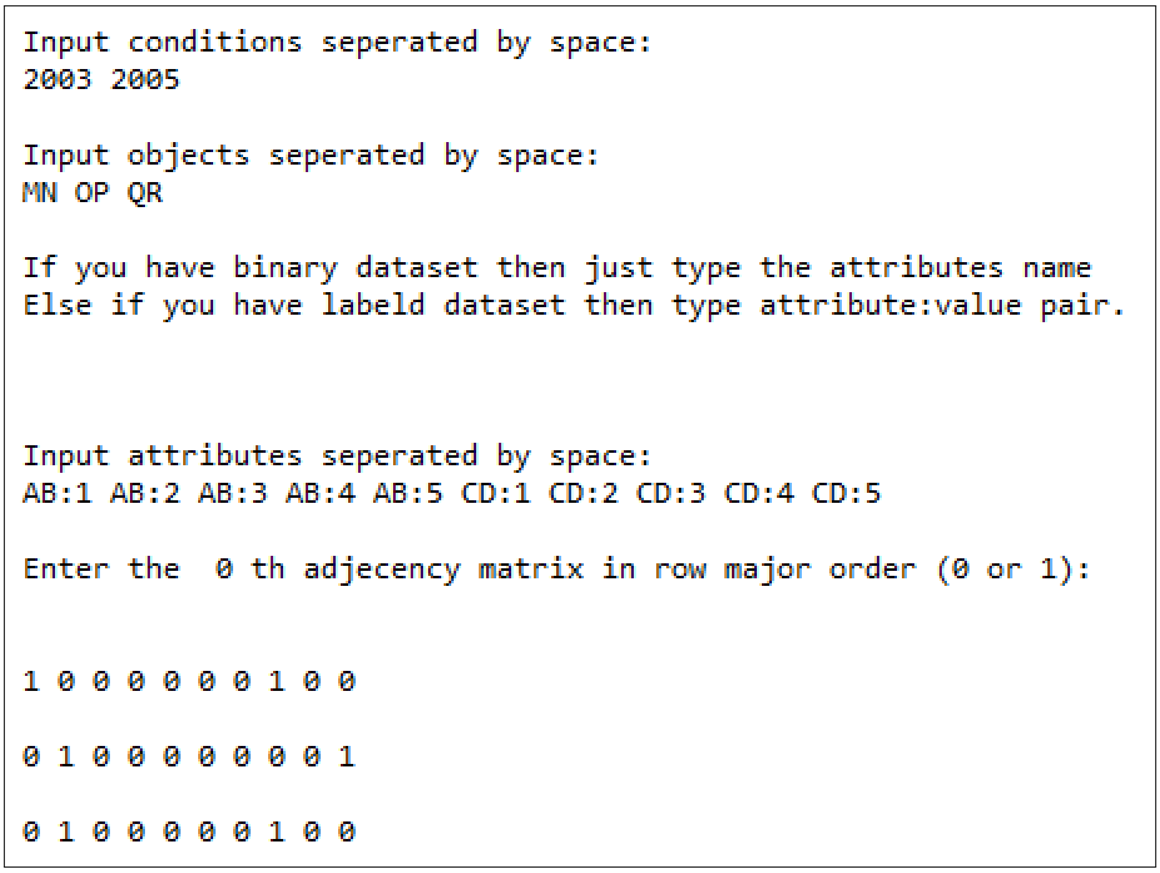


	AB	CD
MN	1	2
OP	1	2
QR	2	3

Figure C.7: Labeled dataset for condition 2005

Then program will run in below mentioned manner

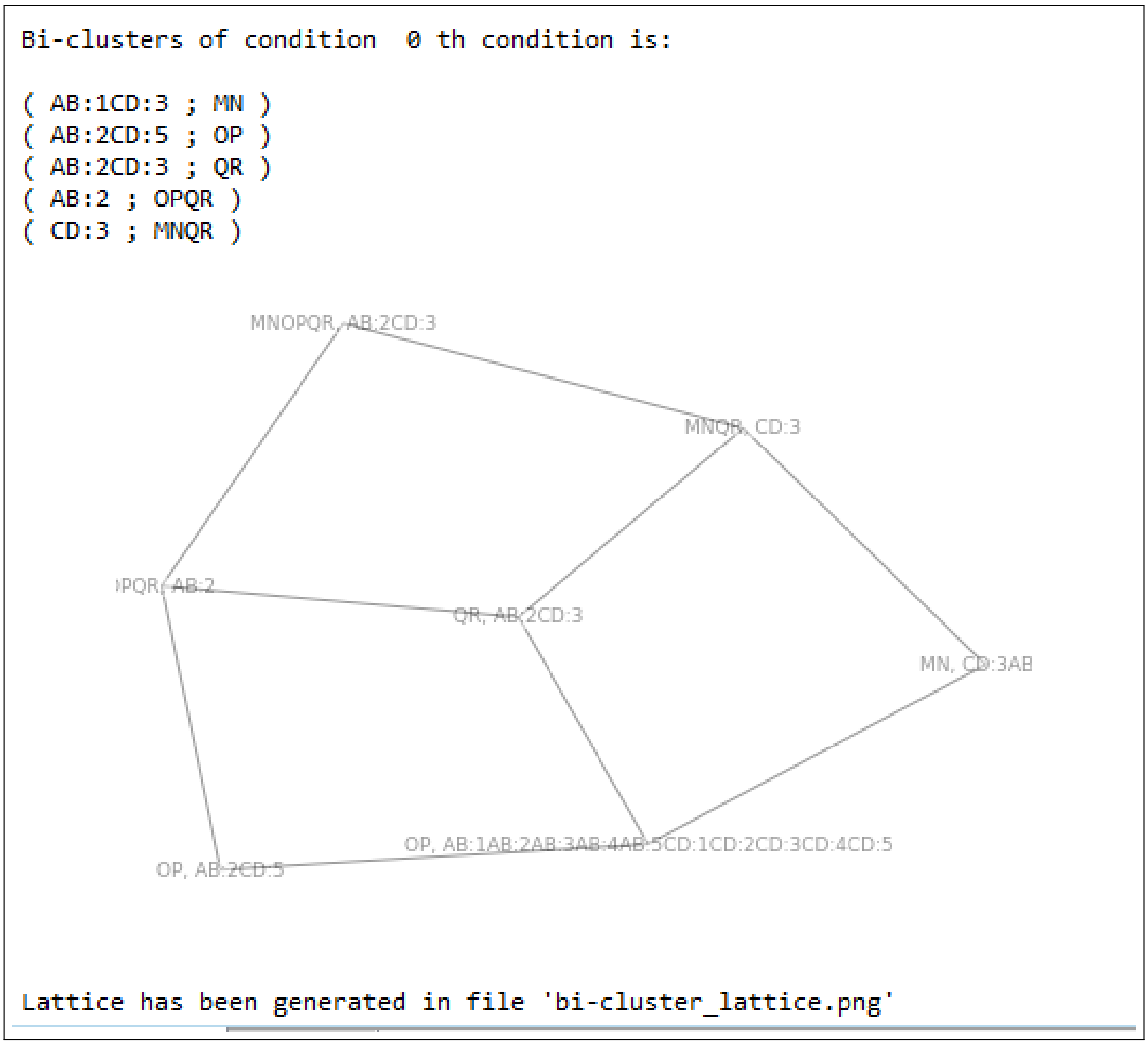
1. First it will take conditions, object and attributes and then first matrix from user (Figure C.8).



```
Input conditions seperated by space:
2003 2005

Input objects seperated by space:
MN OP QR

If you have binary dataset then just type the attributes name
Else if you have labeld dataset then type attribute:value pair.



Input attributes seperated by space:
AB:1 AB:2 AB:3 AB:4 AB:5 CD:1 CD:2 CD:3 CD:4 CD:5

Enter the  0 th adjecency matrix in row major order (0 or 1):


1 0 0 0 0 0 0 1 0 0

0 1 0 0 0 0 0 0 0 1

0 1 0 0 0 0 0 1 0 0
```

Figure C.8: For user given dataset, compilation step 1

2. Then it will show the bi-clusters and lattice for the first matrix and save the lattice in a $.png$ file under $.spyder-py3$ folder (Figure C.9).

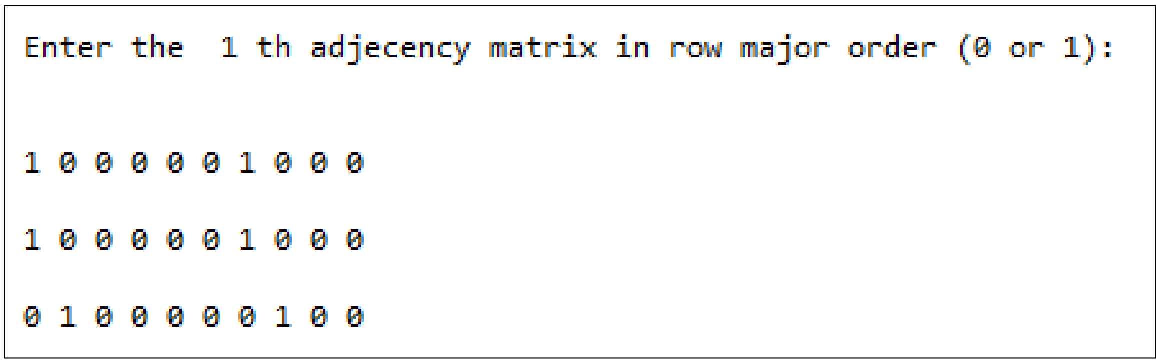


```
Bi-clusters of condition  0 th condition is:

( AB:1CD:3 ; MN )
( AB:2CD:5 ; OP )
( AB:2CD:3 ; QR )
( AB:2 ; OPQR )
( CD:3 ; MNQR )
```

```
Lattice has been generated in file 'bi-cluster_lattice.png'
```

Figure C.9: For user given dataset, compilation step 2

3. After that it will take the second matrix from the user(as here only two condition mentioned, so the program will take only two matrix. If 7 conditions are given then it will take 7 matrices from the user) (Figure C.10).

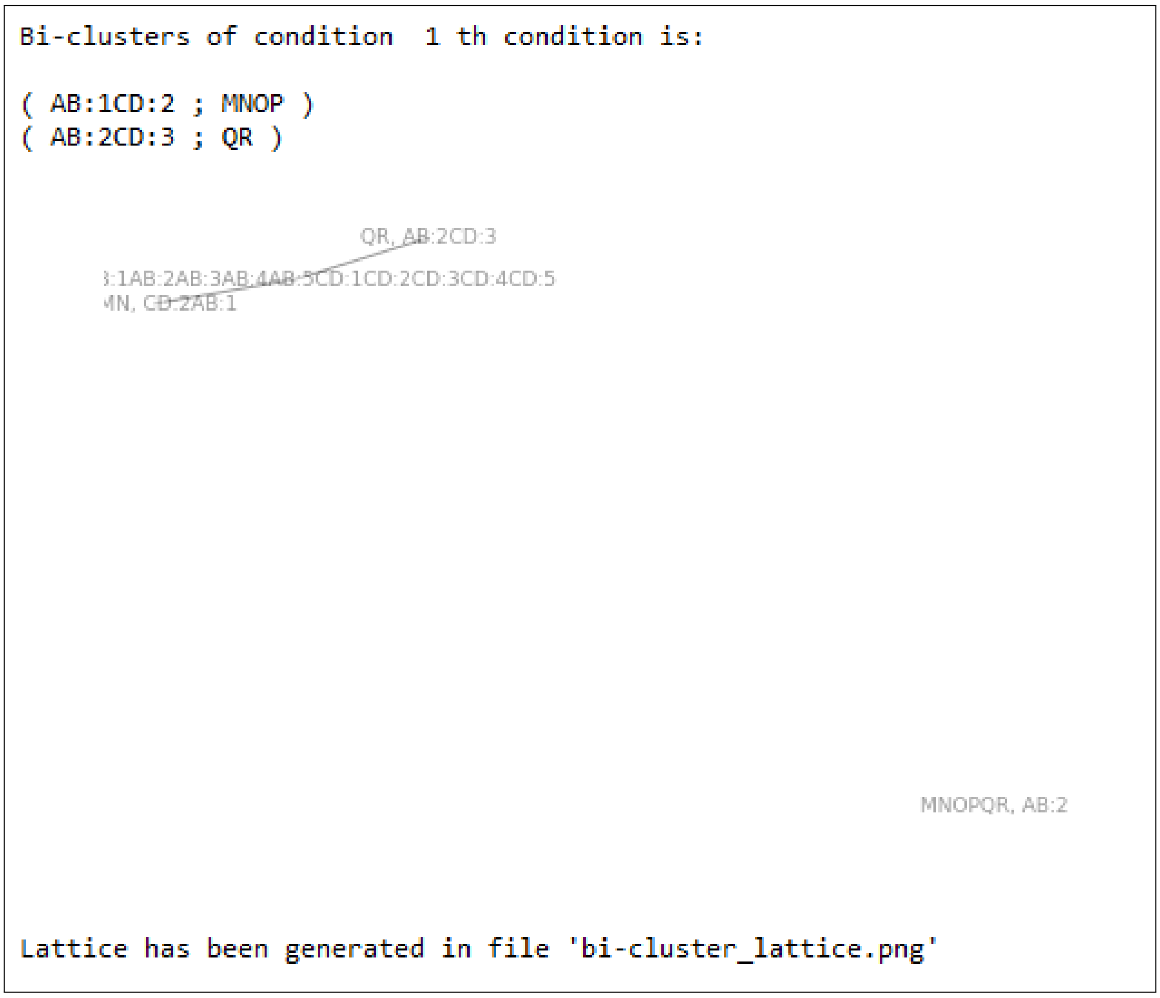


```
Enter the  1 th adjecency matrix in row major order (0 or 1):


1 0 0 0 0 0 1 0 0 0

1 0 0 0 0 0 1 0 0 0

0 1 0 0 0 0 0 1 0 0
```

Figure C.10: For user given dataset, compilation step 3

4. Now it will show the bi-clusters and lattice for the second matrix and save the lattice

in a *.png* file under *.spyder* − *py*3 folder (Figure C.11).

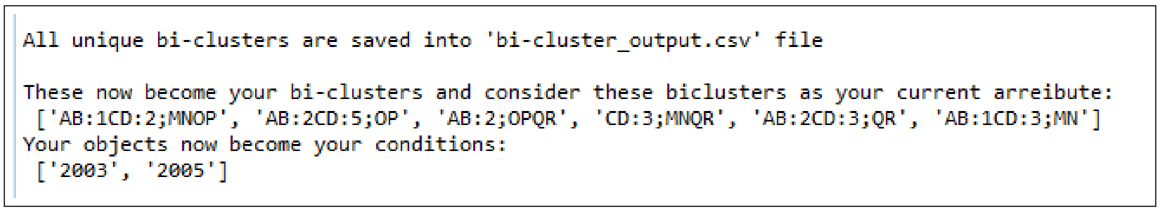


```
Bi-clusters of condition  1 th condition is:

( AB:1CD:2 ; MNOP )
( AB:2CD:3 ; QR )
```

```
Lattice has been generated in file 'bi-cluster_lattice.png'
```

Figure C.11: For user given dataset, compilation step 4

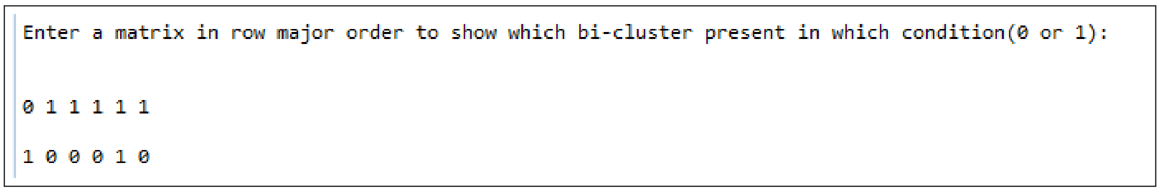
5. Now this program will save all the uniquely created bi-clusters into *bi_cluster_output.csv* file under *.spyder* − *py*3 folder, and take all the unique bi-clusters as attribute and conditions as object (Figure C.12).



```
All unique bi-clusters are saved into 'bi-cluster_output.csv' file

These now become your bi-clusters and consider these biclusters as your current arreibute:
 ['AB:1CD:2;MNOP', 'AB:2CD:5;OP', 'AB:2;OPQR', 'CD:3;MNQR', 'AB:2CD:3;QR', 'AB:1CD:3;MN']
Your objects now become your conditions:
 ['2003', '2005']
```

Figure C.12: For user given dataset, compilation step 5

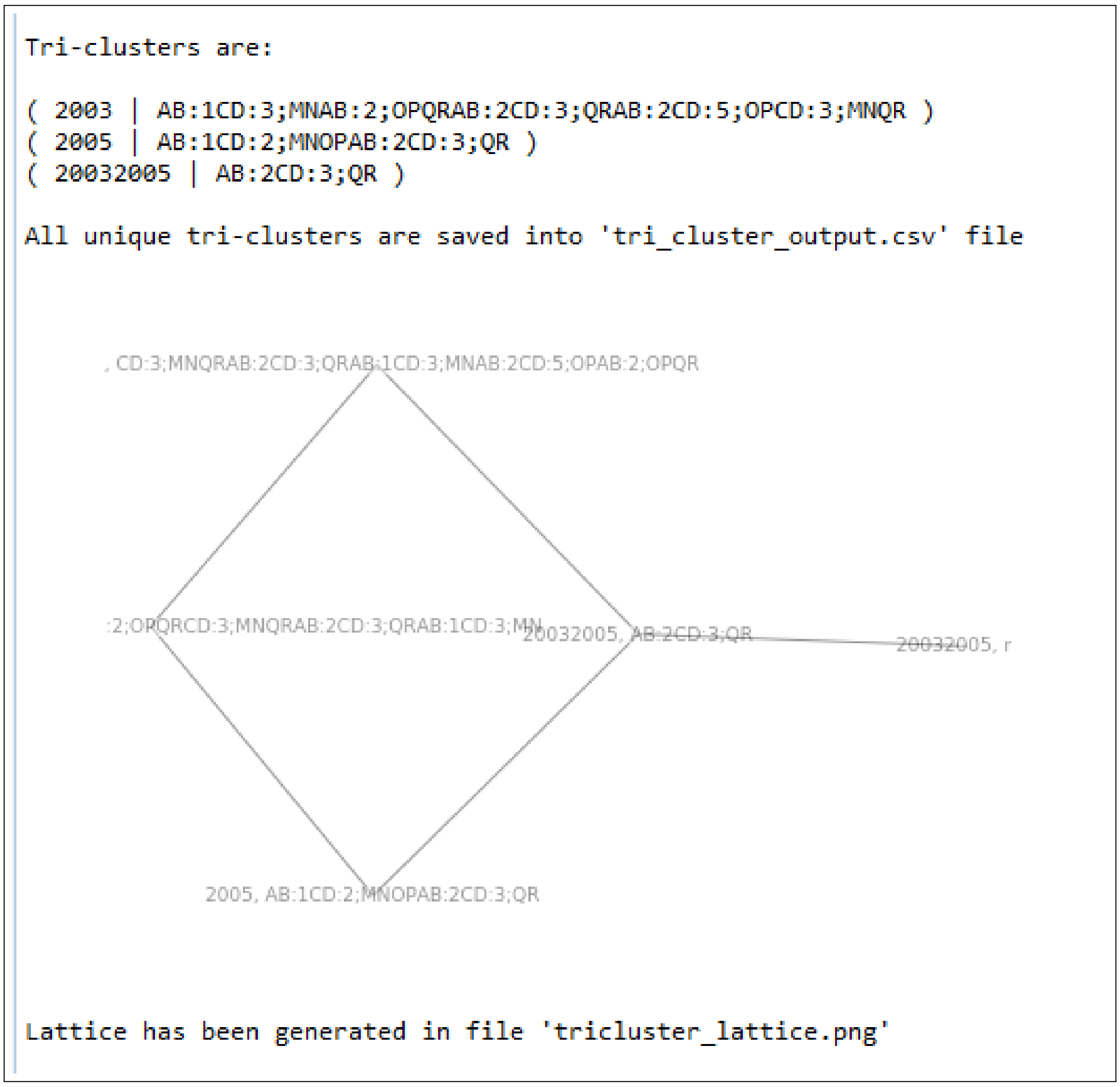
6. Next the program will ask the user if a bi-cluster is present in for a condition or not. User have have to answer this in 0,1 adjacency matrix format. User can now can think all bi-clusters (just shown before) as attributes and all conditions (just shown before) as conditions and fill the matrix (Figure C.13).



```
Enter a matrix in row major order to show which bi-cluster present in which condition(0 or 1):

0 1 1 1 1 1

1 0 0 0 1 0
```

Figure C.13: For user given dataset, compilation step 6

7. Now the program will show all the tri-clusters, final lattice and save them into *tri_cluster_output.csv* and *tricluster_lattice.png* file under *.spyder − py − 3* folder (Figure C.14).



```
Tri-clusters are:

( 2003 | AB:1CD:3;MNAB:2;OPQRAB:2CD:3;QRAB:2CD:5;OPCD:3;MNQR )
( 2005 | AB:1CD:2;MNOPAB:2CD:3;QR )
( 20032005 | AB:2CD:3;QR )

All unique tri-clusters are saved into 'tri_cluster_output.csv' file
```

, CD:3;MNQRAB:2CD:3;QRAB:1CD:3;MNAB:2CD:5;OPAB:2;OPQR
:2;OPQRCD:3;MNQRAB:2CD:3;QRAB:1CD:3;MN
20032005, AB:2CD:3;QR
20032005, r
2005, AB:1CD:2;MNOPAB:2CD:3;QR

```
Lattice has been generated in file 'tricluster_lattice.png'
```

Figure C.14: For user given dataset, compilation step 7

Appendix D

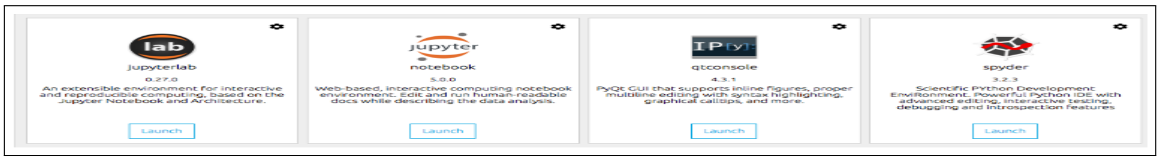
Experimental Setup

At first Python 3.7[1] should be downloaded as per operating system. Python is the programming language which will be installed on the machine and on top of that different IDEs and packages can be installed. So, Anaconda Distribution[2] needs to be installed as per OS type.

Now some packages needs to be installed to run this project and to install those packages, the below mentioned lines should be performed in command prompt $\rightarrow$

- pip3 install pandas
- pip3 install matplotlib
- pip3 install nltk
- pip3 install numpy
- pip3 install plotly
- pip3 install networkx
- pip3 install python-csv

Now this project code is ready to run in any of this below mentioned IDE (Figure D.1).



An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

Figure D.1: IDE

[1]`https://www.python.org/downloads/`
[2]`https://www.anaconda.com/distribution/`