BIPARTITION VISUALIZATION

USING SELF ORGANIZING MAPS

BY

NEHA NAHAR

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENTS OF

MASTER OF SCIENCE

IN

COMPUTER SCIENCE

UNIVERSITY OF RHODE ISLAND

2007

MASTER OF SCIENCE THESIS

OF

NEHA NAHAR

APPROVED:

Thesis Committee:

Major Professor _____

_____

_____

_____

Dean of the Graduate School

UNIVERSITY OF RHODE ISLAND

2007

ABSTRACT

The *Tree of Life* has provided a framework to study the evolution of organisms. Phylogenetic trees are used to depict the evolution of organisms or of molecules. Comparative genome analyses have shown that genomes are mosaics where different parts have different histories. This discovery calls into question the notion of a unified evolutionary tree for an organism and gives rise to the notion of an evolutionary consensus tree based on the evolutionary patterns of the majority of genes in a genome. The tree topologies that are in conflict with the majority consensus are strong indicators of horizontal gene transfer events. Horizontal gene transfer is the process in which genes from a species are transferred across species border or to different species that are not an offspring. Due to horizontal gene transfer the *Tree of Life* concept is transforming to a *Web of Life* where different parts of the genomes can be traced separately from the accepted history of the species. Clustering gene families based on the phylogenetic information they retained, allows extracting a majority consensus for the genomes' evolutionary history, and determination of genes that have a conflicting phylogeny.

The purpose of this thesis is to design and implement a framework for an interactive web-based tool that facilitates comparative genome analysis of different species. This tool performs the analysis on a bipartition matrix and generates results as an interactive self-organizing map that allows users to interpret the data and highlight consensus relationships among the organisms. Another important advantage of the tool is an interactive and visual identification of horizontally transferred genes. Additionally, it

allows researchers to submit their bipartition data online and provides them with the resources and the computation power to perform analyses.

ACKNOWLEDGEMENTS

I would like to express deepest gratitude towards my advisor, Dr. Lutz Hamel for his unflinching guidance and support. He challenged me and encouraged me to set higher benchmarks and boosted my confidence to solve all the problems. His expertise and experience improved my research skills and prepared me for future work. I would like to thank him for giving me a unique opportunity to work on this interesting and intriguing project.

I am indebted to Dr. Maria Popstova and Dr. Peter Gogarten from University of Connecticut, Storrs. They provided me with all the biological knowledge required to complete this dissertation. I am grateful to Maria for always being there to answer all my questions.

I am especially thankful to my committee members Dr. Joan Peckham and Dr. Yana Reshetnyak for reading and commenting on my dissertation. I had immense pleasure working with them. I would also like to thank Kevin Bryan for his suggestions and all the technical and administrative help he provided.

This dissertation work was financially supported through the NASA Applied Information System Research Program (NNG04GP90G).

I would like to thank my friends and family for being there for me. "*High achievement always takes place in a framework of high expectations".* I am deeply indebted to my beloved husband Dhiraj, my parents in-law Mr. Rushabh and Mrs. Asha Nahar, my parents Dr. Pradeep and Mrs. Aruna Bafana and my brothers Pankaj and Gaurav for their

high expectations and confidence in me. Their immeasurable encouragement helped me achieve my goal. Without their support and confidence, I would not have completed this dissertation and my higher education.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

This thesis involves the design and implementation of a web-based tool that performs bipartition visualization using Self-Organizing Maps (SOM) [1]. Bipartition is the division of phylogenetic tree into two parts connected by a single branch. Given the bipartition [2] matrix, the tool is useful for the comparative genomic analysis of any living organisms, especially prokaryotic (bacteria and archaea). The tool generates a consensus tree for given organisms and also reports the strongly supported and conflicting bipartitions.

The *Tree Of Life* has provided a framework to study the evolution of organisms [3]. Phylogenetic trees are used to depict the evolution of organisms or of molecules. However, comparative genome analyses have shown that genomes are mosaics where different parts have different histories [4]. These findings questioned the validity of the tree concept, especially for prokaryotic species [2, 5]. While individual genes might be exchanged between species, in many instances a core of genes in a set of genomes might represent a tree-like organismal history; however, in other instances, genomes that had independent evolutionary histories are fused to form a new line of descent. The *Tree of Life* concept needs to be amended by fusing lines of descent and by connecting threads, representing gene transfer events, that embed the organismal lines of descent into a *Web of Life* [6]. Thus the task of comparative genomics is to identify genes that share common history, genes whose evolution is different from the majority consensus, and to identify groups of genes that might have been transferred together [7]. The last are of special interest because they point towards crucial events in evolutionary history.

The developed tool facilitates web-based comparative genome analysis by allowing visual and interactive inspection of either individual gene histories or groups of closely related gene families identified as those through clustering based on a self-organizing map [1] approach. The tool also allows locating gene families whose evolutionary histories are in significant disagreement with a majority consensus. This type of phylogenetic conflict is in most cases attributed to horizontal gene transfer.

Number of genomes	Number of trees	Number of bipartitions
4	3	3
6	105	25
8	10,395	119
10	2,075,025	501
13	1.37E + 10	4,082
20	2.22E + 20	5.24E + 05
50	2.84E + 74	5.63E + 14

Table 1: Number of trees and bipartitions required to represent phylogenetic data

Evolutionary relationships between organisms are usually represented as a phylogenetic tree. For *n* different genomes the number of possible trees grows very fast. With *n* genomes there are *(2n-5)! /[2(n-3)(n-3)!]* different unrooted tree topologies (as seen in Table 1), and it is an impossible computational task to iterate through all possible trees *(1.37E+10)* even for only 13 genomes. Alternatively, a phylogenetic tree can be divided into quanta of phylogenetic information such as a bipartition.

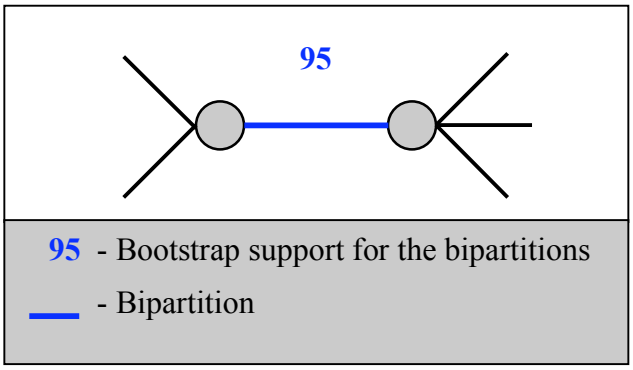


Figure 1: Bipartition of a phylogenetic tree

A bipartition as shown in Figure 1, is the division of a tree into two parts that are connected by a single branch. The number of possible bipartitions for *n* genomes is given by the formula: $2^{(n-1)}-n-1$, and it grows much slower with an increasing number of species than the number of different trees (as seen in Table 1). Another advantage of bipartition analysis is that different bipartitions can either be compatible or conflicting [8, 9]. Compatible or non-conflicting bipartitions are the ones that can co-exist in one tree and conflicting or non-compatible bipartitions are those that cannot co-exist in one tree. Section 2.1.2 describes compatible and non-compatible bipartitions in detail with an example. By identifying compatible bipartitions that are supported by the majority of gene families one can find a majority consensus phylogenetic signal. Bipartitions that are in significant conflict with the majority consensus bipartitions are most likely related to the horizontal gene transfer or to systematic artifacts of phylogenetic reconstruction [10].

The web-based tool is implemented using CGI/Perl environment. The user can upload data file (bipartition matrix file; as described in section 2.1) and the cutoff for bootstrap value via convenient web interfaces. The cutoff value specified by the user ranges from

0 to 100. Only gene families with bootstrap support values higher than the cutoff value are considered during the analysis. Once the server performs the analyses using the analysis scripts and generates the required results, a hyperlink to the results is displayed to the user. The results are accessible to the user for a given period of time before they expire.

Broadly the work of this thesis concentrates on the design and implementation of this interactive web-based tool for biologists to perform analysis and knowledge discovery on the genomic data.

The main work of this thesis is as outlined:

- Design of the framework for the interactive web-based tool. The tool architecture was expected to have a directory structure to handle multiple uploads of user data files and analysis results storage for each user.

- Implementation of the framework for the web-based tool. The developed tool handles multiple uploads of user data files, user management and results generation and storage for users.

- Additionally the tool facilitates security features to protect the web server against malicious attacks over the Internet.

The remainder of this thesis is composed of three chapters.

- First, we discuss some background information regarding bipartition matrices, self-organizing maps, consensus trees and related work. This material will give the context for understanding the implementation and results.

- Next, the thesis will discuss the key parts of the implementation. This will include details about the tool characteristics, framework, technologies used and consensus tree generation.

- Finally, we discuss the results generated by the tool. This will include gene maps, consensus tree and strongly supported and conflicting bipartitions for 14 archaeal taxa. The chapter ends with concluding remarks and thoughts for future research.

2 BACKGROUND

This chapter provides basic background material in order to give the reader some context for understanding the tool purpose, architecture, usage and the results discussed later. By the end of this section, the reader should have a broad understanding of bipartitions, self-organizing maps and consensus trees.

2.1 Assembling Gene Families and Generating Bipartition Matrix

A gene family is a collection of genes from different genomes that are related to each other and share a common ancestor. Orthologs are genes that arise due to speciation events and paralogs are genes that arise due to duplication. In general, a gene family may include both orthologs and paralogs [11]. Only the sets of putatively orthologous genes where each species contributes only one gene into a family are considered in this study [7]. The evolutionary history of an individual gene family is a phylogenetic tree that can be represented as a set of bipartitions. The developed tool uses a bipartition matrix as input to represent the phylogenetic information contained in gene families.

2.1.1 Representation of a Bipartition

A bipartition divides the tree into two groups, but it does not consider the relationship within each of the two groups. Bipartitions can be trivial or non-trivial. Trivial bipartitions are those that have only one species on one side and the remaining on the other. Non-trivial bipartitions are the ones that have more than one species on both sides. There are *(n-3)* non-trivial bipartitions for any given tree with *n* species. Figure 2 shows the way a bipartition is represented using special characters "." and "*". Here a

bipartition is represented as a bit pattern, with one bit position for each species. The bit is set to ”*” if the corresponding species belongs to one side of the bipartition and to ”.” if it belongs to the other side. Hence, all the species on one side of the bipartition are represented using “*” and all the species that are on the other side are represented using the ”.” character. The black arrow shows the bipartition in both cases A and B. As seen in Figure 2, case A, species 1 and 2 are on one side of the bipartition and species 3, 4, and 5 are on the other side. The representation of this bipartition shows that species 1 and 2 are represented as “*” and remaining species that lie on the other side of the bipartition are represented by “.” character.

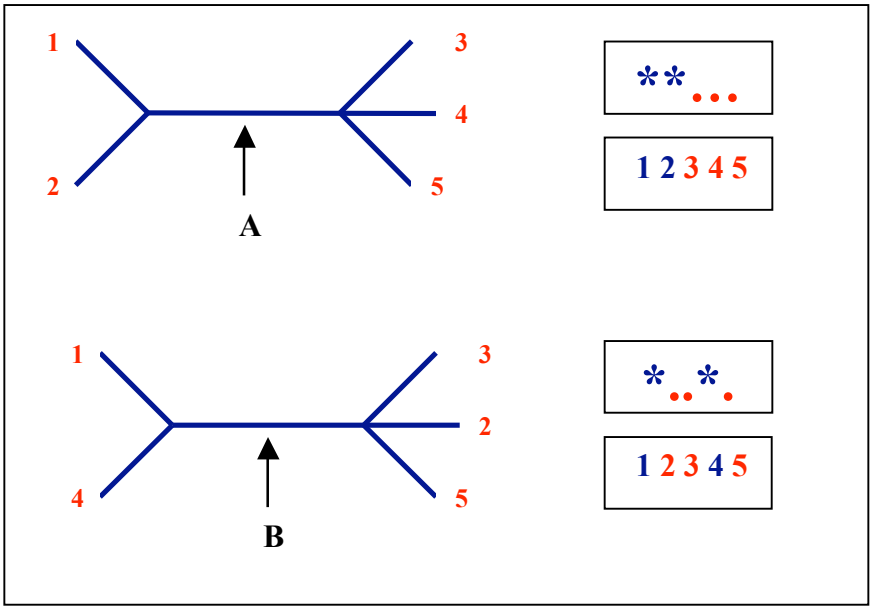


Figure 2: Representation of a bipartition using “*. “ notation

Figure 2, case B shows the arrangement of the species similar to case A, but with species 2 and 4 swapped. Hence the representation of this bipartition shows “*” at positions 1 and 4 and “.”character otherwise.

2.1.2 Compatible and Conflicting Bipartitions

As mentioned in the introduction section, compatible bipartitions are the ones that can co-exist in one tree and conflicting or non-compatible bipartitions are those that cannot co-exist in one tree. Consider an arrangement of 5 species labeled 1,2,3,4, and 5 as shown in Figure 3. The non-trivial bipartitions (as shown by blue arrows in Figure 3) in this case are represented as "****…**" and "*****..**".

To formally interpret the compatibility and non-compatibility of bipartitions, consider a bipartition "**…" or (A,B) such that A = {1,2} and B = {3,4,5}. It divides the tree with species 1 and 2 on one side of the bipartition and species 3,4, and 5 on the other side. Formally two bipartitions (A,B) and (C,D) are compatible if and only if at least one of the intersections (A ∩ C), (A ∩ D), (B ∩ C) or (B ∩ D) is empty. If not, then the two bipartitions are conflicting or non-compatible with each other.

For example consider bipartition (A,B) represented as "**…"and (C,D) represented as "***..". In this case A = {1,2} , B = {3,4,5}, C = {1,2,3} and D = {4,5}. These two bipartitions are compatible as (A ∩ D) is empty. Compare bipartition (A,B) represented as "**…" with bipartition (E,F) represented as "*…*". Here, A = {1,2} , B = {3,4,5}, E = {1,5} and F = {2,3,4}. It can be seen that bipartition (A,B) is non-compatible with (E,F) as (A ∩ E) = {1}, (A ∩ F) = {2}, (B ∩ E) = {5} and (B ∩ F) = {3,4}. As none of the intersections are empty, it can be said that these two bipartitions are non-compatible or conflicting bipartitions.

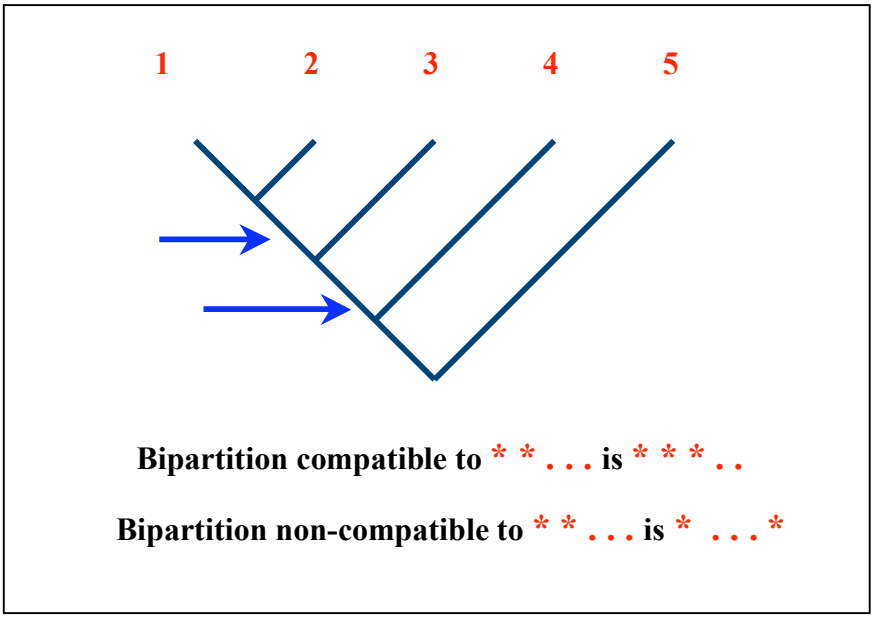


Figure 3: Tree with 5 species and compatible and non-compatible bipartitions

2.1.3 Assembling Gene Families

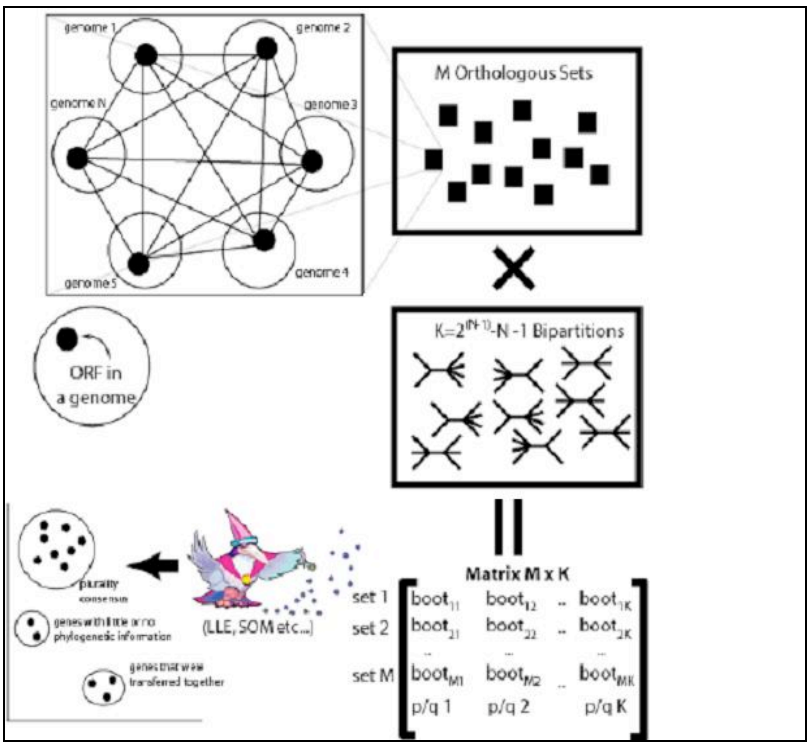


Figure 4: Generation of bipartition matrix and gene family clusters

Assembling gene families is the first step in the generation of the bipartition matrix. As

shown in Figure 4, complete genomes of N species are downloaded. Reciprocal best BLAST [12] hit criteria [13, 14] with relaxation is used to detect set of a M orthologous genes. BLAST finds regions of local similarity between sequences, calculates statistics of the alignments and provides measures of significance for the locally aligned regions. The reciprocal best BLAST hit method requires strong conservative relationships among the orthologs so that if a gene from species 1 selects a gene from species 2 as a best hit when performing a BLAST search with genome 1 against genome 2, then the gene 2 must in turn select gene 1 as the best hit when genome 2 is searched against genome 1. The requirement of reciprocity is very strict and often fails in the presence of paralogs. To select more orthologous sets the criteria of strict reciprocity was relaxed by allowing broken connections. Every genome is BLAST against every other genome. The top hit of every BLAST search is selected. Gene families are selected such that genes select each other as their top-scoring hit.

2.1.4 Generating Bipartition Matrix

As described in the methodology and result sections in [8], the next step in generating the bipartition matrix is to align sequences for each gene family and reconstruct a Maximum Likelihood tree for each gene family.

Then, 100 bootstrapped replicates are generated for each gene family tree, and are evaluated with the Phyml [15] program. All 100 generated trees are split into a set of bipartitions and corresponding bootstrap support values are assigned to each bipartition by calculating how many times each bipartition is present in a family. The bipartition matrix is composed from bootstrap values for k bipartitions for each gene families with

rows corresponding to M gene families and columns to all possible (k) bipartitions for a given set of taxa. The format of a bipartition matrix is shown in Table 2. The format of a bipartition matrix is shown in Table 2. It is a matrix where rows represent gene families and columns give the bootstrap support values for the particular bipartitions calculated for each gene family.

	Bipartition #1		**Bipartition #k**
Support value vector for gene family #1	BP_{11}	…	BP_{1k}
Support value vector for gene family #2	BP_{21}	…	BP_{2k}
…			
Support value vector for gene family #M	BP_{m1}	…	BP_{mk}

Table 2: Bipartition Matrix

2.2 Self-Organizing Maps

This section provides an insight into the SOM algorithm along with some conceptual details regarding SOM.

2.2.1 Clustering and Data Visualization

In bipartition analysis, each gene family can be treated as a point in a high dimensional space of all possible bipartitions. The coordinates of such a point are the bootstrap support values of individual bipartitions. The number of possible bipartitions for *n* taxa is given by the formula: $2^{(n-1)}-n-1$. It is difficult for humans to visualize and understand high dimensional data. Data visualization solves this problem by reducing the number of dimensions of the data.

Clustering is the classification of objects into different groups, or, partitioning of a data set into clusters, so that the data in each subset share some common characteristics. Visual clustering using self-organizing maps is able to project the high dimensional bipartition space of gene family vectors onto a two-dimensional plane in a topology-preserving manner such that gene families with similar phylogenetic signals are grouped together. Conflicting and non-conflicting families are discovered through a visual and interactive investigation of the map.

2.2.2 SOM Algorithm

SOM is one of the data visualization techniques that reduces the dimensionality of the data by producing a map (usually 2 dimensional) that illustrates similarity by clustering similar data points together. It is a neural network based algorithm that detects the structure of the input data based on the similarity between points in high dimensional space. It is based on unsupervised learning where no human intervention is required during training.

In our case a SOM consists of a rectangular grid of neurons. The input or training data is given by an n x k data matrix where n is the number of rows representing the training instances and k is the number of attributes (here bipartitions) describing each training instance. Two parts define each neuron; one is its data vector with same size as the input vector and the second is its position on the grid. During training each row in the input is presented to the map (as shown in Figure 5) repeatedly as can be seen by the following algorithm.

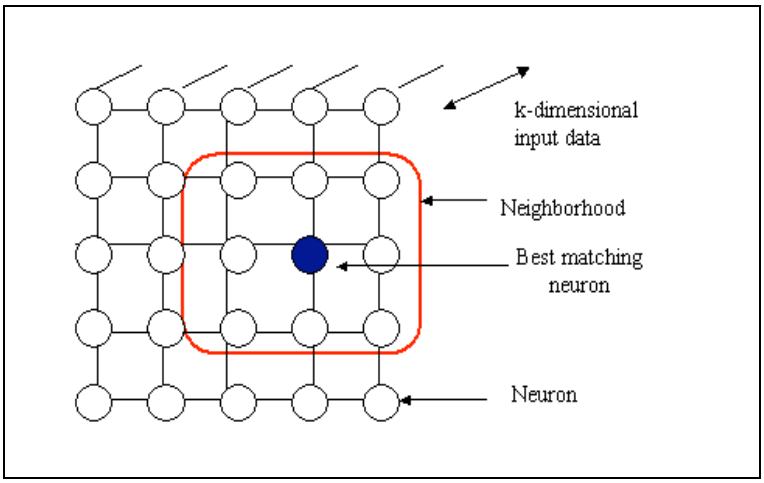


Figure 5: SOM two-dimensional grid

A high level sketch of the SOM algorithm:

```
Initialize the neurons.
Repeat
      For each record r in the input dataset
            Find a neuron on the map that is similar to record r.
            Make the neuron look more like record r.
            Determine the neighboring neurons and make them look
            more like record r.
      End For.
Until Converged.
```

In the first step, for input row r the best matching index c is found using the following: c = arg min $\| r - m_i \|$ for every i, where m_i is the data of the i^{th} neuron and the $\| \cdot \|$ operator computes the Euclidean distance. Let m_c be the best matching neuron. In the next step, the neuron m_c is updated to look more like record r. Finally, the neighborhood of neuron m_c is determined and the neighboring neurons are made to look more like record r. At time step t the neighboring neurons are updated as $m_i^{(t+1)} = m_i^{(t)} + h_{ci} [r - m_i^{(t)}]$ for all i where h_{ci} is the neighborhood function such that,

$h_{ci} = 0$ if $|c - i| > \beta$

$= \alpha$ if $|c - i| <= \beta$

$m_i^{(t)}$ is the sample data vector of the i^{th} neuron at time t. α is the learning rate and β is the neighborhood distance. To make the map converge faster, the learning rate and the neighborhood distance are often also decreasing functions of time. This process is carried out for large number of iterations (usually more than 10000). After the learning process finishes, each input record is assigned to its closest neuron. A 2-dimensional map is obtained showing the similarity between the data.

2.2.3 U-matrix

The map generated by the self-organizing map algorithm can be interpreted via the canonical U-matrix [16, 17] (unified distance matrix) representation. This representation visualizes the distance between the neurons. The distance between the adjacent neurons is computed and presented with different shades of coloration between the adjacent nodes of the 2-dimensional grid. Dark coloration between the neurons corresponds to a large distance and a light coloration between the neurons signifies that the neurons are close to each other in the input space. Light areas can be thought as clusters and dark areas as cluster separators

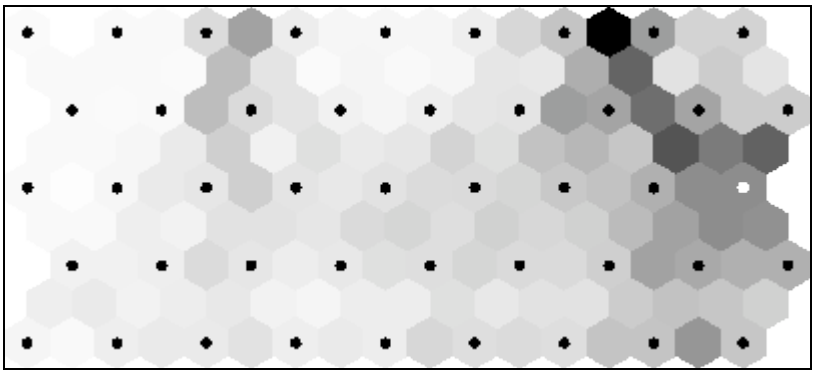


Figure 6: U-matrix

Figure 6 shows a U-matrix representation of the self-organizing map. Neurons are marked as black dots. This representation shows that neurons in the upper right corner are separate clusters as there is dark coloration gap between them and other neurons. U-matrix representation helps find clusters in the input data without having any prior information regarding the clusters.

2.2.4 Emergent Self-Organizing Map (ESOM)

Small SOMs with the number of neurons equal to the number of expected clusters behave very similar to the k-means algorithm. The goal of k-means algorithm is to cluster data points based on attributes into k partitions. The number of expected clusters has to be specified at the start. In this approach the number of expected clusters ranges from 2-3 into the tens. Here we use Emergent SOMs (ESOM) [16, 18], which use a much larger number of neurons than the expected number of clusters, usually at least an order of magnitude more than expected clusters. The larger the map more interesting and revealing the emergent structure. ESOMs not only visualize inter-cluster relationships displayed but also intra-cluster relationships are visualized. Another advantage of ESOM over other clustering algorithms such as k-means is that no a priori knowledge of how

many clusters to expect is required. Finally, with the ESOM, cluster membership of a gene family is not restricted to one particular cluster.

2.3 Consensus Tree

In context of phylogenetics, a tree [19] is a mathematical structure that is used to model the actual evolutionary history of a group of sequences or organisms. The tree consists of nodes connected by branches. A rooted tree as shown in Figure7A has a node identified as root. All other nodes descend from this node called root. A rooted tree has direction that corresponds to the evolutionary time. The closer the node to the root of tree, the older it is in time.

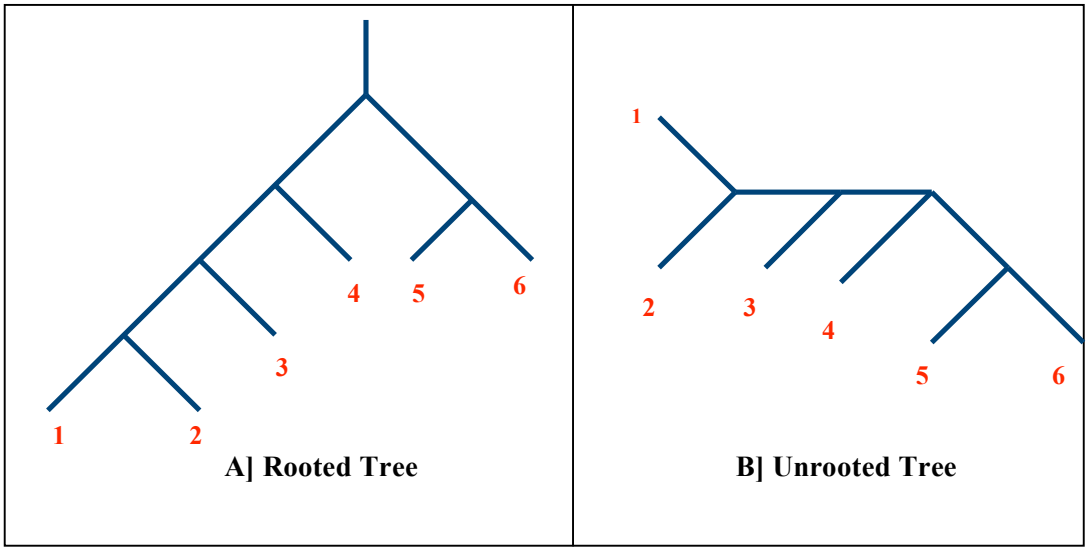


Figure 7: Rooted and unrooted trees with 6 species

In contrast, unrooted trees have no root and hence they do not specify evolutionary relationships and time of existence of the organisms. Figure 7B shows the unrooted version of the same tree.

Consensus tree is a tree that shows features common in a given set of trees. In this case

we search for an evolutionary tree that best explains the given set of trees for given species. There are three different consensus methods; strict consensus, majority-rule consensus and Adams consensus. The strict consensus tree contains exactly those groups or splits that occur in all the given set of input trees. The strict consensus tree is generally a rooted tree. The majority-rule consensus tree [20] contains exactly those groups that occur in more than half the given set of trees. It typically has more resolution than the strict consensus tree. Adams consensus tree can be viewed as a set of leaf subset nestings. A group nests within a larger group if the most recent common ancestor of the smaller group is a descendant of the most recent common ancestor of the larger group. We use the majority-consensus method to reconstruct the consensus tree [20] from the set of bipartitions. This consensus tree presents the features shared by all or most of the trees. Here we describe the algorithm designed to compute a consensus tree from given bipartitions.

Initially, a mask is assigned to each bipartition. This mask is a string of 0s and 1s and has length equal to the number of species used in the analysis. It depends on the positions of the character "*" in the bipartition representation. Each character in the bipartition representation is scanned. The mask is computed by shifting bits of 1 left, by *i* bits and then using logical OR operation with 0. Here *i* represents each position of character "*" in the bipartition. Each bipartition in the output file is scanned along with the support value. Its mask is computed. Then, we check if the bipartition is compatible with the existing set or is non-compatible. The masks of the bipartitions are used to decide the compatibility between them. Two masks are compatible if there is inclusive

disjunction between them. An inclusive disjunction is true if either, or both, of its components are true. If the masks of the bipartition are not compatible, they are listed as conflicting bipartitions. The existence of compatibility in the masks of the bipartitions indicates that they agree with each other. The scanned bipartition is then added to a list of compatible bipartitions.

Each compatible bipartition is then processed to compute the consensus tree. A tree node is initialized and a child node is created for each species as a leaf node. Active leafs are computed by using the mask of the leaf and performing logical AND operation with 1. A common parent node for these active leaves is found by identifying a parent node whose mask subsumes the current mask of the leaves. Marked active leaves indicate that they were processed as a part of a previous active leaf in the form of a subtree. They are already added in the consensus tree. If the active leaf is unmarked then they are unlinked from the common parent and inserted in the consensus tree as a child node of the new parent. This process is performed for each compatible bipartition and the consensus tree is generated in New Hampshire (NH) format. In NH tree format, a pair of parentheses indicates that the species are grouped together. A colon and bootstrap support value follows each species name as seen in the example below:

Tree in NH format:

```
((((((aeropyrum:100,sulfolobus:100):52,pyrobaculum:100):48,nanoarcheum:100):31
,thermoplasma:100):14,(pyrococcus:100,thermococcus:100):95):7,(archaeoglobus:1
00,((haloarcula:100,halobacterium:100):96,methanosarcina:100):46):31,((methano
coccus:100,methanothermobacter:100):24,methanopyrus:100):7);
```

2.4 Related Work

There are many approaches to detect horizontally transferred genes such as AU test

(approximately unbiased test), Lento plots and bipartition spectra. In this section we describe each of these approaches and the advantages of the tool over them.

2.4.1 Approximately Unbiased (AU) Test

The AU test approach is based on the confidence of phylogenetic tree selection. For each tree that describes a dataset, the AU test estimates the probability that the tree might be the true tree describing the history of the dataset under consideration. As have been tested in [8] bipartition analysis outperforms AU test [21] for detection of conflicting signals in phylogenetic data.

2.4.2 Lento Plots

Lento plot [9] summarizes the phylogenetic information present in a given set of gene families. Each column in the Lento plot depicts the number of gene families that support a particular bipartition. Bipartitions are represented on the X-axis of the plots.

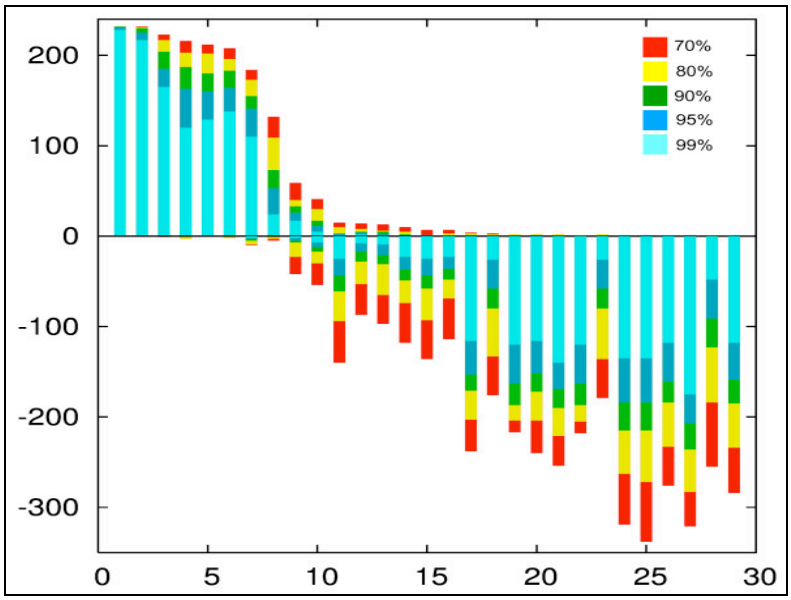


Figure 8: Lento plot showing phylogenetic information retained in 13 gamma proteobacterial genomes

Figure 8 shows the Lento plot that depicts the phylogenetic information retained in 13 gamma proteobacterial genomes. As shown in Figure 8, for each bipartition the bars in the positive direction give the number of gene families that support the bipartition with the indicated (color coded) support value, the bars in the negative direction give the number of supported conflicting bipartitions found in all of the gene families. This number can be greater than the number of gene families, because a single gene family can support several conflicting bipartitions.

As found in [8], first eight bipartitions are supported by the majority of gene families, and only three datasets conflict with these plurality bipartitions at the 99% bootstrap support level.

The developed, visually oriented tool has an advantage over Lento plot [9, 10] analysis by allowing interactive and visual inspection of the areas on the map that comprise families with close phylogenetic signals and those that generate the conflicts. The consensus phylogeny for these families can be investigated by just one click. The group of conflicting families can also be visually divided into clusters according to self-organizing maps. This type of analysis is not provided by Lento plot based approaches.

3 IMPLEMENTATION

This chapter describes the developed tool. We begin with an introduction to the tool characteristics. Next, we present the high-level system architecture followed by the directory structure of the server and the framework. Continuing, we will give more detail about each of the component in the system from an implementation perspective. At the end we describe the process performed to generate results.

3.1 Tool Characteristics

The developed tool is a web-based application that performs bipartition analysis using self-organizing maps. The input to the tool is a bipartition matrix composed from common gene families of a given set of species. Using the given input, the tool will generate clusters of gene families with similar phylogenetic signals. A phylogenetic signal is generally used to denote whether the related organisms resemble each other in terms of their genetic material. The developed tool allows an interactive reconstruction of phylogenetic trees for any combination of the resulting clusters of gene families. Finally, it also reports strongly supported and conflicting bipartitions.

The developed tool is an online service that allows users to upload their bipartition matrices and perform the genomic analysis interactively. Users can submit the bipartition matrix along with the desired cut-off value. Once the server performs the analyses and generates the required results, a hyperlink to the results is displayed to the user. The results shall be accessible to the user for a given period of time before they expire. Usage of the tool shall allow the user to use the resources and the computational

power of the server and perform their analyses.

3.1.1 Gene Map

The tool facilitates the user to view the SOM generated map. This map as shown in Figure 9, displays the gene family clusters.

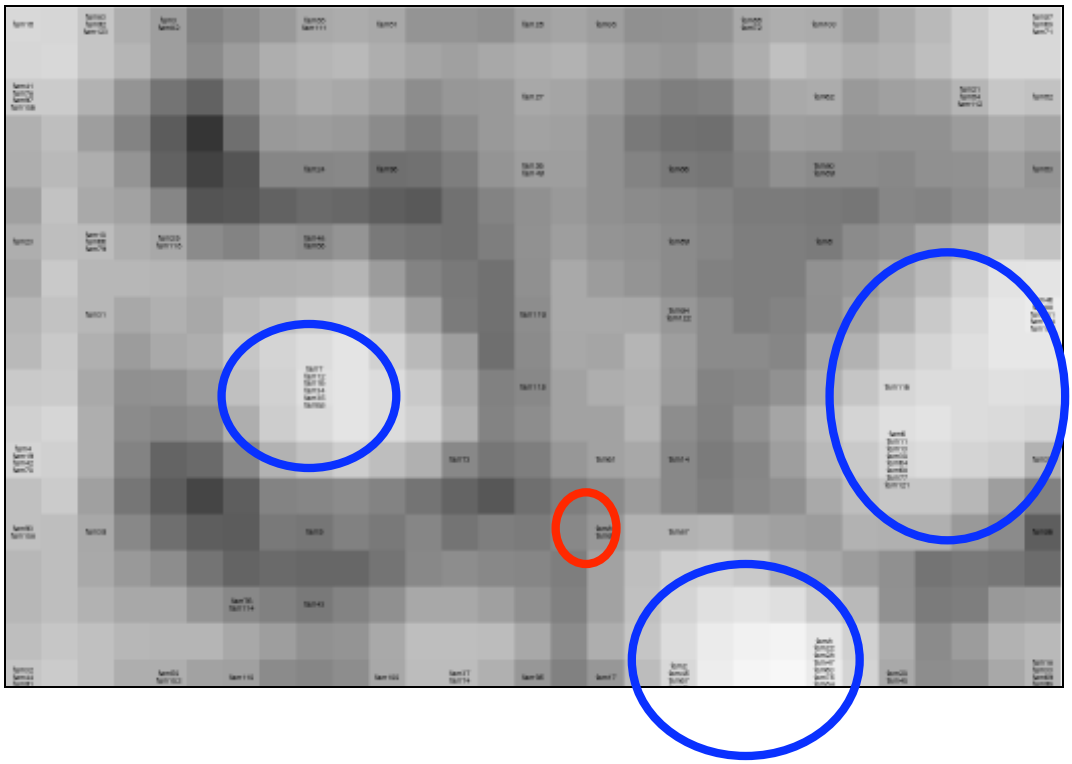


Figure 9: SOM generated map from bipartition matrix

The map displays light colored areas, grey areas, and dark areas. Light colored areas (shown with blue circles) show clusters that are dense and are different from the surrounding clusters. Grey areas display clusters that are sparse, and dark areas represent areas with very little similarity information. In the above map white areas indicate groups of gene families whose phylogenetic signal is distinctly different from the surrounding genes. Genes that are in conflict (shown in red) with the set of compatible

bipartitions are grouped separately from the clusters containing genes conforming to the consensus phylogeny. A clearer picture of this emerges when one studies the support of individual bipartitions.

3.1.2 Consensus Tree

The tool provides an interactive way for users to select specific clusters on the map (as shown in Figure 10) and generate majority consensus tree using those clusters. As shown in Figure 10 (blue arrow), the user also can view the cluster number, map co-ordinates of the cluster and the families in that cluster by moving the mouse on the neuron belonging to the cluster.

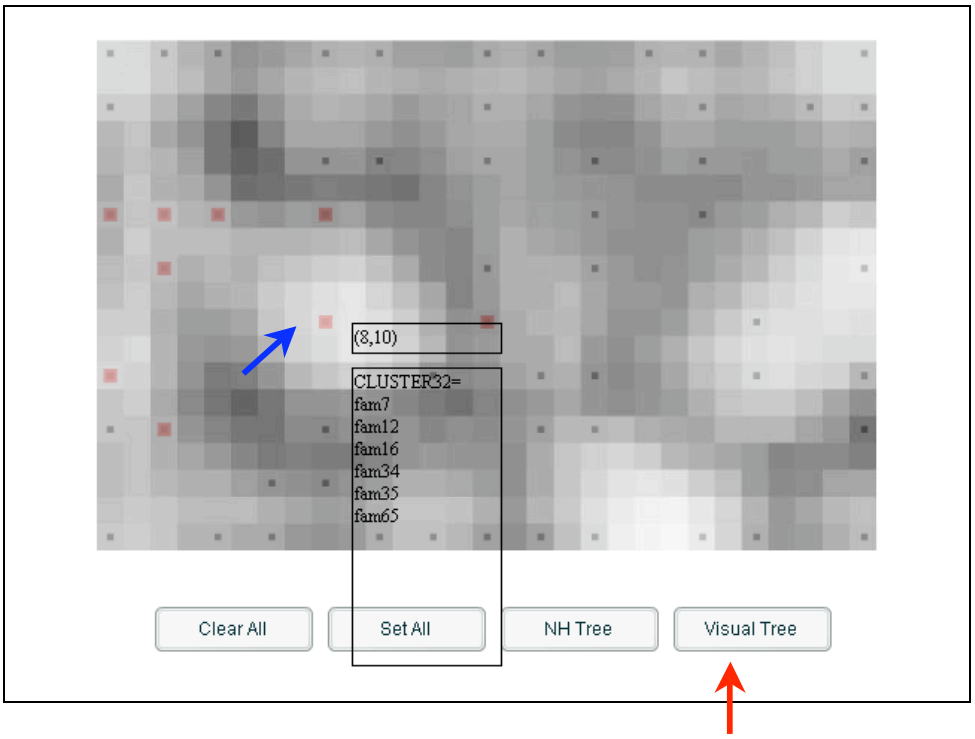


Figure 10: Cluster map

The consensus tree shown in Figure 11 is generated on the fly using the "Visual Tree" button (red arrow), from the combined set of supported bipartitions of the families in the

selected clusters. It is convenient for the biologists to generate a consensus tree with all clusters and a consensus tree with few selected clusters of interest and then compare them. For ease of use, the cluster numbers used to generate the consensus tree are displayed in the window caption. If all clusters have been selected then the text "All clusters" is displayed as the window caption. In this case clusters 23-25, 28, 32-34 and 42 on the map are selected as shown by red squares in Figure 10. As few clusters are selected, the ATV tree viewer window displays these numbers in the window caption. The consensus tree is calculated according to the majority consensus rule described in section 2.3 and as implemented in CONSENSE [22].

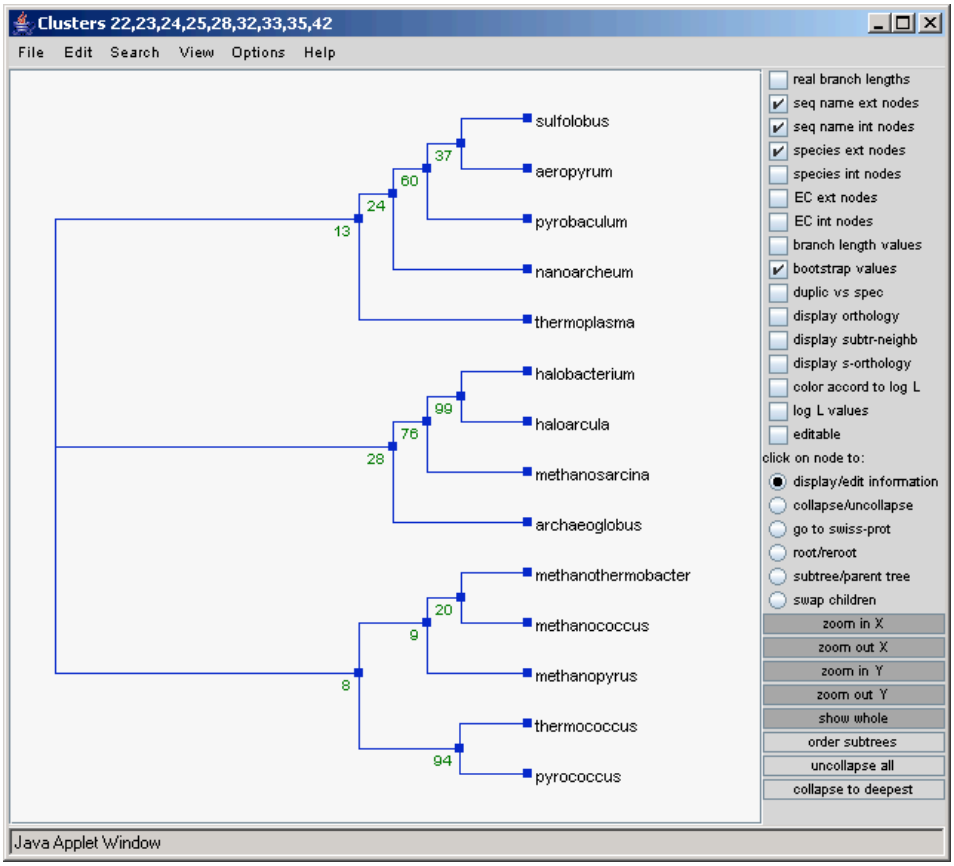


Figure 11: Consensus tree (ATV tree viewer)

The consensus tree as shown in Figure 11 can be viewed and manipulated using the ATV tree viewer applet [23]. ATV has many options that allow the user to modify the view of the tree. Users can re-root a tree with particular outgroup. User has the option to view the bootstrap values at each tree node. Collapsing the tree at any node, zoom in and zoom out in both X and Y directions and displaying the branch length values are additional options of interest provided by ATV tree viewer.

3.1.3 Strongly Supported and Conflicting Bipartitions

The tool facilitates the user to view a list of bipartitions that are supported by the majority of the families followed by a list of conflicting bipartitions. As shown in Figure 12, one way to view these plots is as slices from the SOM generated map where each slice corresponds to a particular bipartition. Each such slice shows exactly where on the map that particular bipartition is or is not supported and by which gene families. These representations can be further investigated to identify horizontal gene transfer events. For easier comprehensibility these slices are also represented as 3D plots.

The 3D plot for a particular bipartition can be interpreted as a contour map with three dimensions, x-dimension of the SOM map (15 in Figure 12), y-dimension of the SOM map (10 in figure 12) and height as the bootstrap support ranging from 0 -100. In the 3D plot in Figure 12, red color indicates the highest bootstrap support value for that bipartition. All the lines shown in different colors can be interpreted as contour lines showing elevation or height in terms of bootstrap support. The mountains shown in red indicate the gene families that highly support the shown bipartition (156 in this case). The valleys in red indicate the gene families that strongly disagree with the bipartition

(156 in this case). Other colored lines shown in green, blue, purple and cyano blue color indicate the elevation for different support values such as 80, 60 and so on.

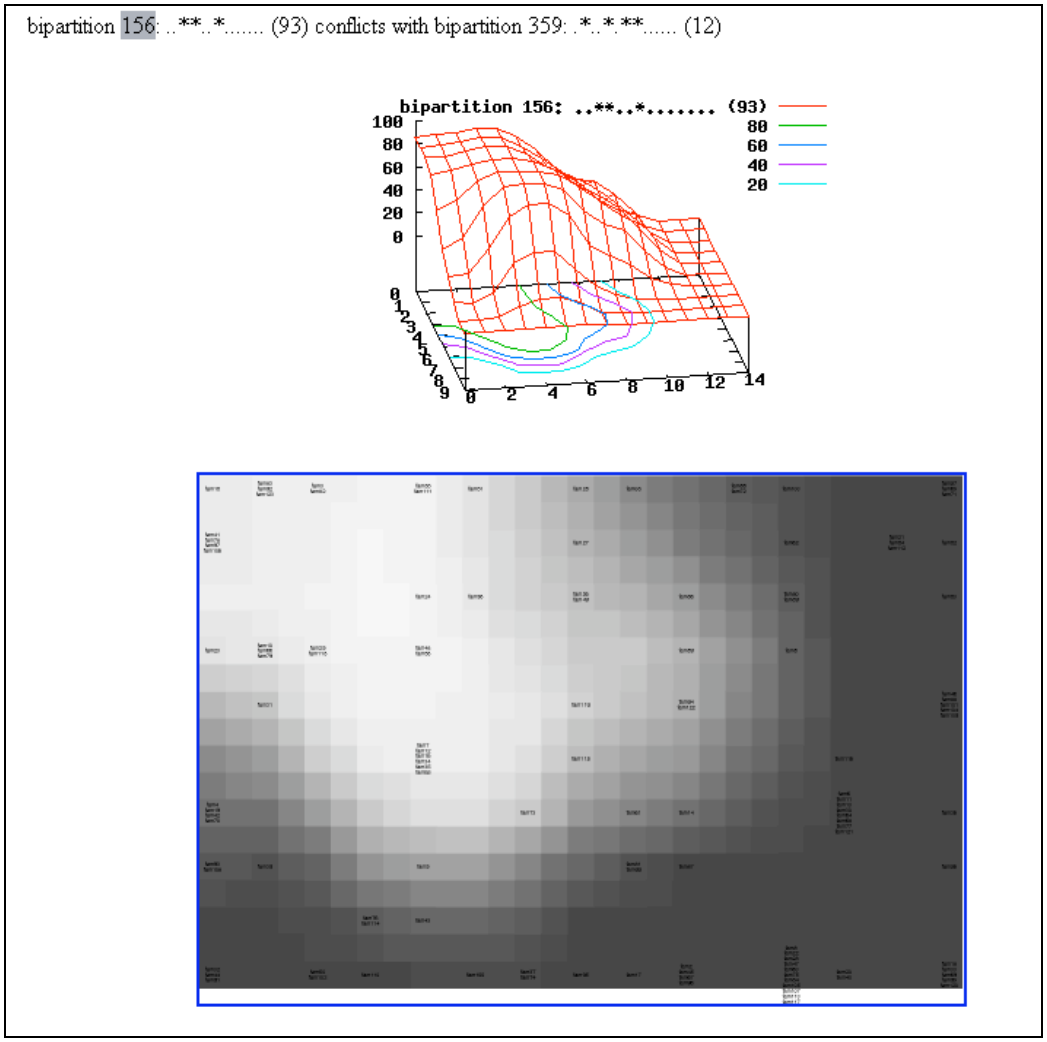


Figure 12: Conflicting bipartition

3.2 Tool Architecture

In this section we describe the architecture of the tool server. We begin with the high level architecture of the system. Then, we describe the directory structure for the server. Finally, we discuss the issues related to web security and the daemons implemented on

the server.

3.2.1 High Level Tool Architecture

The web-based tool is composed of different components (as shown in Figure 13) such as web server, interface, analysis programs, and user data storage space.

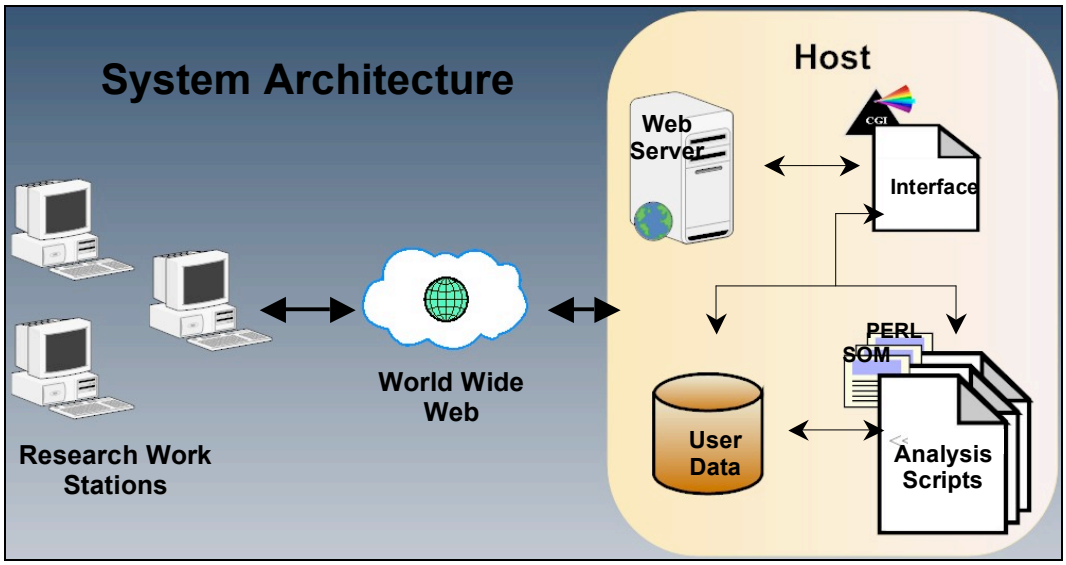


Figure 13: High-level system architecture

The web server component provides the web interface. The interface component provides the user management capabilities. Analysis programs are scripts that perform analyses and generate the visual output. These results are stored in the data storage for the user to access for a given period of time. The user can upload data file (bipartition matrix file) and the cutoff for bootstrap support value via convenient web interfaces. Once the server performs the analyses and generates the required results, a hyperlink to the results is displayed to the user. The results are accessible to the user for a given period of time before they expire.

3.2.2 Directory Structure

In this section we describe the directory structure of the server. The tool server is hosted on a Linux machine. It is implemented using the CGI-Perl [24] environment. The directories on the server are organized as shown in Figure 14. The "*template*" directory consists of all the analysis scripts and the "*users*" directory. It is called template as it serves as a model directory for each user specific directory structure in "*users*".

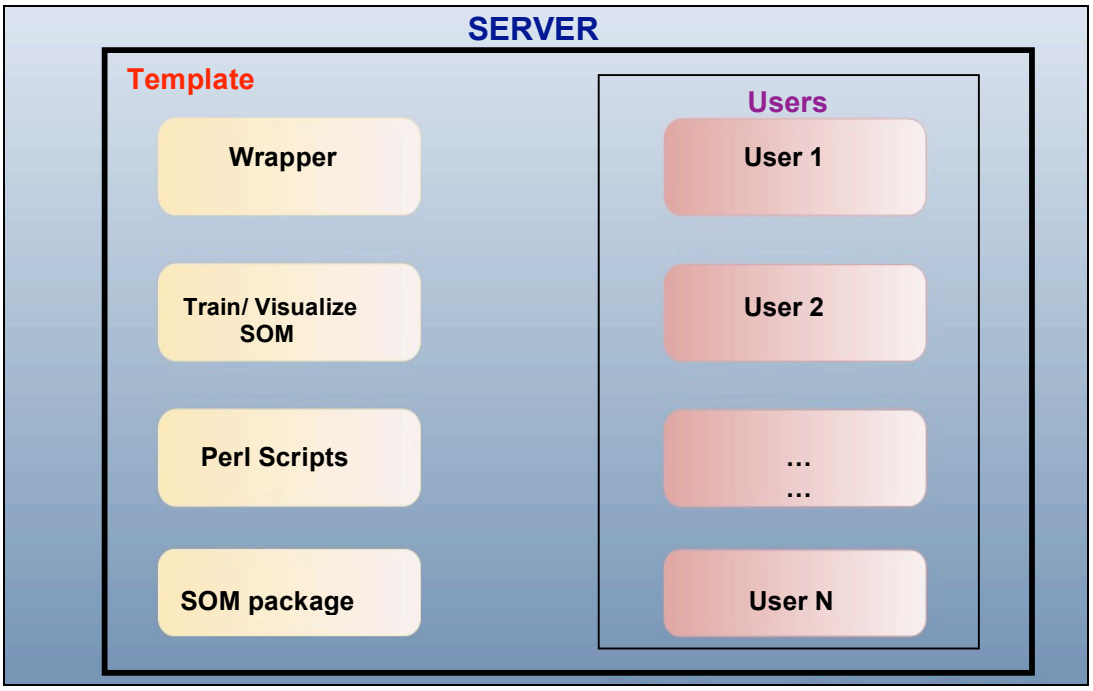


Figure 14: Directory structure of the server

The analysis scripts include the wrapper, SOM programs and programs that train the SOM and visualize the results generated by SOM. The wrapper invokes all these external programs to perform the analysis. The SOM scripts are in a directory called "*SOM*" in the template directory. These scripts execute the SOM algorithm and generate the map. The train script is invoked by the wrapper script to train the SOM algorithm and the visualize script is invoked to generate and visualize the SOM map.

The "*users*" directory acts as a parent directory for all the user specific directories. Whenever a request for analysis is made to the server, a new directory associated with that request and user is created in this directory. The bipartition matrix file and the list of species file are uploaded in the user specific directory. A link to the results for this analysis is displayed to the user. The next section describes the structure of the "*users*" directory and each user specific directory is described in section 3.3, along with the user management tasks performed on the server.

3.2.3 Web Security

The web server component of the tool is the Apache web server. As we know, the Internet is vulnerable to attacks on the web servers. It should be protected against various threats and attacks such as integrity threats, confidentiality threats, denial of service attacks and so on. Integrity threats comprise of modification of the user data on the server or modification of server memory. Confidentiality threats include eavesdropping on the Internet, gaining private information from the server and gaining network server configuration information. One of the forms of denial of service attacks is disrupting the entire network by disabling the network.

In our case, when the bipartition matrix file is uploaded, it is copied in the user specific directory. The web server component (Apache) writes this user file to the directory. If the Apache web server is given "write" permission for all other directories such as "*template*" directory on the server, then any user can intrude the server and modify, add, or delete data from the server. The server becomes vulnerable to integrity, confidentiality and denial of service attacks. The intruder can modify the existing scripts

on the server, or can disrupt the entire network by disabling the network, or run malicious programs in the background to access all the confidential information on the server.

To protect the server against all the malicious attacks described above, it is very important to set the appropriate directory and file permissions for different processes in the system. In our implementation, Apache web server can write only to the some selected subdirectories (described in section 3.3) in the "*users*" directory and not to the "*template*" directory. Hence an intruder will not be able to modify any scripts on the server or upload malicious programs on the server that hack the information residing on the server.

3.2.4 Daemon process

In Linux, a daemon is a computer program that runs in the background, rather than under the direct control of a user. It is usually initiated as a process and launched at the boot time of a system. Two daemons have been implemented on the tool server. One daemon handles the creation of user specific directories and execution of analysis scripts. The other daemon performs directory cleanup in the "*users*" directory.

The implemented tool enables the users to upload their bipartition matrix and provides the capability to perform analyses. A daemon is implemented in Python, on the tool server to handle each request made by the users to perform analysis. This daemon runs in the background and polls for new requests. As soon as a new request is detected it creates a new directory in the "*users*" directory and starts the analysis process for the

user. The details regarding the actions performed by this daemon are described in section 3.3.2.

The implemented tool also facilitates cleanup of the user specific directories after they expire, that is after a given period of time. This function is also implemented using a daemon in Python that runs in the background, looks for user directory creation date and deletes the directory as soon as it expires.

3.3 User Management

In this section we discuss the structure of the user specific directories in the "*users*" directory. Then we discuss the upload functionality in details. Finally the directory cleanup functionality is described.

3.3.1 "Users" Directory Structure

The "*users*" directory is in the "*template*" directory and Apache has the permissions to write to this directory and a few subdirectories. Every time a user uploads the bipartition matrix on the server, the daemon creates a new directory in the "*users*" directory. To ensure that this user specific directory is unique, it's name is composed of a large random number. As shown in Figure 15, two new directories are created in this user specific directory *User1* (called User1 for simplicity) called "*data*" and "*result*". In Figure 13, these two directories as shown in different colors (purple and green). Purple indicates that Apache server has permissions to write to that directory (*data*). Green indicates that Apache does not have permissions to write to this directory and only admin process can write to this ("results") directory.

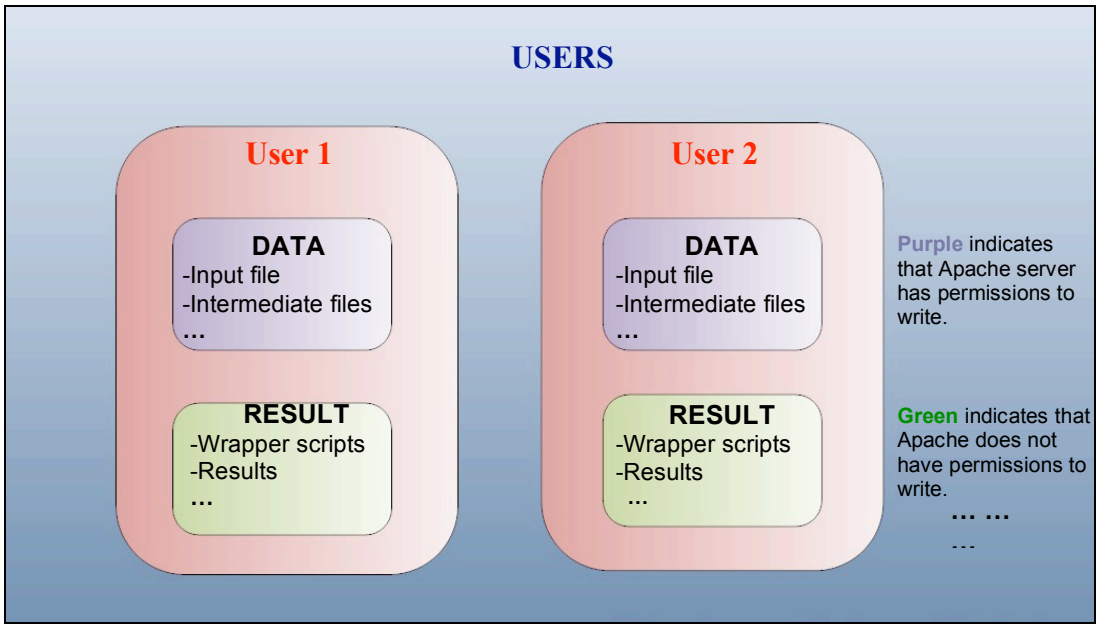


Figure 15: Structure of a user specific directory.

The reason why these two directories are created in the "*users*" directory is described next. When the user specific directory is created, the analysis scripts are invoked via a wrapper script. Two options were considered to run this wrapper script in the background. First, the wrapper script runs as an Apache process (process owned by Apache). Second, the wrapper runs as an admin (specific user with administrative access control rights) process. Generally bipartition analysis requires at least a few hours to complete as complex computations are involved along with huge amount of data.

The first approach of running the wrapper script as an Apache process has two main disadvantages. First, as the analysis runs for a long time, Apache web server is overloaded with requests, which degrades its performance. Other processes using Apache services may get affected due to this overload. Second disadvantage is the case when Apache server is restarted. The analysis scripts running as Apache processes will

be terminated. The user will not be able to view the final results. Also the computational power used and the memory used to store the intermediate results will be redundant and wasted as the analysis aborts abruptly.

The second approach of running the wrapper script as an admin process was found to be more effective. The analysis can run for the required time without any interruptions. Additionally only the admin user can restart the wrapper script. No other user has an authority to do so. Hence two subdirectories "*data*" and "*result*" are created in the "*users*" directory. Apache server has permissions to write only to the "*data*" directory and not to "*result*" directory. Apache writes the bipartition matrix file (input file) uploaded by the user to this "*data*" directory. After the bipartition matrix file is uploaded to the "*data*" directory, the daemon running as an admin process copies the wrapper script in the "*result*" directory and starts the analysis. Using this approach, the server is protected against the malicious attacks over the Internet, the analysis is performed without any interruptions and the results generated are stored in the "result" directory for the user to access.

3.3.2 Upload Functionality

The implemented tool facilitates users to upload their bipartition matrices and perform analyses. Users can browse the file system on their machine and select an input file (bipartition matrix file) to be uploaded. Along with the bipartition matrix file, the user also has to upload a file containing the names of the species one on separate line. The user can specify a cut-off for the bootstrap support values to be used in the analysis.

As mentioned in section 3.2.4, the daemon runs in the background as an admin process. This daemon is implemented in Python. It walks through the directory structure of the "*users*" directory. It checks for creation of new user specific directories that do not have the "*result*" directory. The presence of "*result*" directory indicates that the analysis is either complete or in process. But if the "result" directory is not present then it creates the "*result*" directory and starts the analysis.

Once the "*result*" directory is created, all input files are copied to the "*data*" directory. As the daemon creates the "*result*" directory with admin user as the owner, only admin processes can write to this directory. After the "*result*" directory is created, the parameters required for the analysis scripts are written in a file in the "*data*" directory. The parameters include the bipartition matrix file name, the species list file name and the cutoff for bootstrap support value uploaded by the user. The wrapper script is then copied to "*result*" directory and is executed from there as an admin process. All the generated results such as gene map, cluster map, strongly supported and conflicting bipartitions, bipartition images with their 3 D view are stored in the result directory. When the analysis is complete the users can visit the link provided and view the results.

3.4 Technologies

In this section we describe the technologies used to develop and implement the tool. We describe in details, the SOM_PAK package and daemon script on the server.

3.4.1 CGI-Perl Web Environment

CGI stands for "Common Gateway Interface". It is a standard for interfacing external

programs with the web server. CGI scripts are executed in real time and generate dynamic information. In our case, CGI acts as an interface between the web server and the analysis scripts. CGI scripts can be written in many different programming languages such as C++, Perl, TCL and BASIC. In our case the CGI scripts are written in Perl. Perl is very powerful in terms of text manipulation and also has many web support modules available.

CGI programs reside in the "*cgi-bin*" directory on the tool server. The "*ScriptAlias*" directive in the "*httpd.conf*" configuration file tells Apache that the specified directory is set aside for CGI programs. Apache assumes that every file in this directory is a CGI program and attempts to execute it when a request is made. The programs that generate the consensus tree on the fly are CGI scripts written in Perl. These scripts are often restricted to the directories specified in "*ScriptAlias*" so that the administrator can tightly control who is allowed to execute these scripts and protect the web server against malicious attacks.

3.4.2 SOM_PAK package

The SOM algorithm is implemented using the SOM_PAK package [25]. It is a public-domain software package. This package provides C programs that implement the SOM algorithm. The details about the programs used, the parameters and the output files generated are presented in this section. We have added substantial customization to this package for the visualization (3D map of bipartition support and specialized postscript header).

When the user uploads the bipartition matrix, input data file for SOM is created.

The first line in this file is the dimensionality of the input vectors. Subsequent lines consist of n floating-point numbers followed by an optional class label (the name of the gene family in our case). The "*train*" script is a shell script that acts as a wrapper to call the SOM programs. The SOM parameters used in the analysis are x-dimension, y-dimension, topology, and the neighborhood function. The x-dimension and y-dimension are set on the fly depending on the number of columns in the input bipartition matrix. These are set according to the following rule:

```
Number of columns~=2*x-dimension*y-dimension iff x*y <= 1,500
```

Rectangular topology along with bubble neighborhood function and 100,000 training iterations are used.

The first step in training a SOM is to initialize the map using the *randinit* program in the SOM package. The lattice type selected is rectangular (rect) and the neighborhood function type is step function (bubble). The algorithm then trains the map using the program *vsom*. The training is done in two phases. The first is the ordering phase during which the reference vectors of the map units are ordered. During the second phase the values of the reference vectors are fine-tuned. After the map is fine-tuned, the quantization error is evaluated using the *qerror* program. The map units are then calibrated using known input data samples.

The map trained according to the above steps is then used for visualization. The visualization programs in the SOM package makes an image of the map (one selected component plane of it). Then they plot the trajectory of the best-matching units vs. time on it. The program *visual* generates a list of coordinates corresponding to all the best-

matching units in the map for each data sample in the data file. The list of these coordinates can then be processed for various graphical outputs. We use the Linux command line utility called *a2ping.pl* to convert the map into a PNG image.

3.4.3 Consensus Tree Generation

A FLASH application is implemented to allow the users to select specific gene family clusters and generate the consensus tree from those gene families on the fly. As seen in section 3.1.3, this application provides an interactive way for users to select specific clusters on the map and generate majority consensus trees using those clusters. It computes the co-ordinates of the selected clusters on the map and sends the co-ordinates and the cluster numbers to a CGI script. This CGI script computes the consensus tree in NH format as described in section 2.3.

An ATV tree viewer applet [23] reads this consensus tree in NH format and displays the tree in graphical format as show in Figure 16. This tree viewer applet is implemented in JAVA and uses the FORESTER package to visualize annotated phylogenetic trees.

The CGI script executes a Perl script called "*cluster-bipart.pl*" on the fly when the user selects specific clusters and presses the "NH-tree" or "Visualize Tree" button. The co-ordinates of the selected clusters are read from the FLASH application along with the cluster numbers. The dimensions of the map are extracted from the file generated by SOM. A label list containing the selected clusters and gene families is then constructed. Support values are computed for each bipartition for each selected gene family. A log file is written that contains a list of bipartitions, bootstrap support values along with the genes that support each bipartition. An output file is also generated with the

number of species, names of all the species and list of bipartitions and bootstrap support values that are supported by the selected gene families.

A ‘C’ program called “*add-bipart*” then reads this output file. This program generates the consensus tree in NH format and also computes the conflicting bipartitions as described in section 2.3. This consensus tree is read by a tree viewer applet called ATV and displayed in graphical form.

The original source code for ATV tree viewer applet found at [23] was modified as per the requirements of our tool. As stated earlier in section 3.1.2, it is convenient for the biologists to generate a consensus tree with all clusters and a consensus tree with few selected clusters of interest and then compare them. For ease of use, we updated the ATV applet source to display the cluster numbers used to generate the consensus tree as the window caption. As shown in Figure 11, if few clusters are selected, then the selected cluster numbers are displayed in the caption window and if all clusters have been selected then the text “All clusters” is displayed as the window caption as shown in Figure 16.

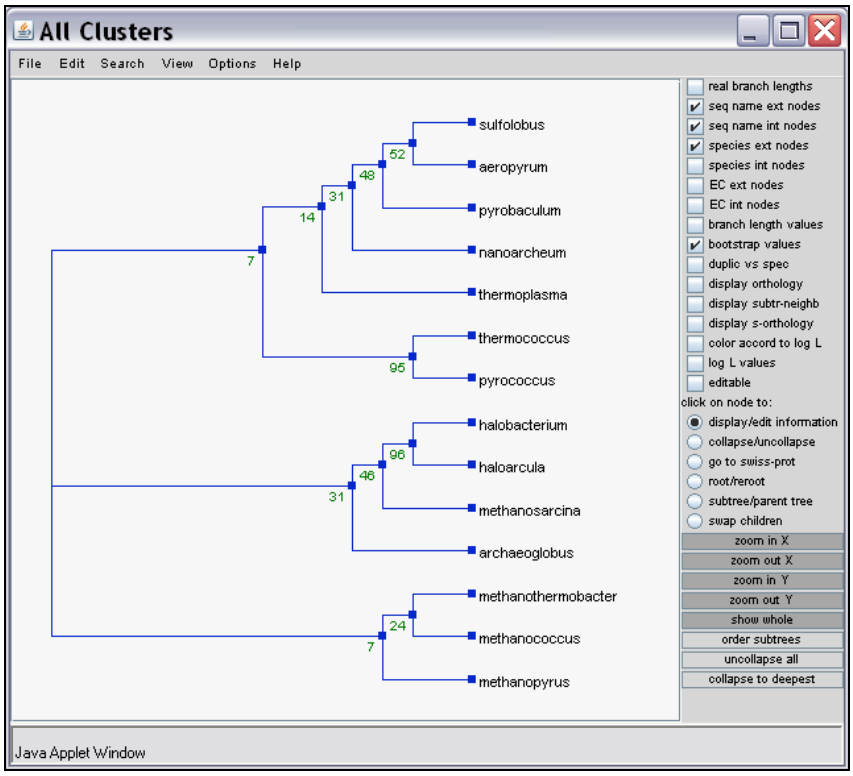


Figure 16: Consensus tree generated with all clusters.

3.4.4 Python Daemon

The daemon that handles the creation of the user specific directories and invokes the analysis process is written in Python. Two options were considered for the implementation of this daemon.

The initial approach was to write the daemon using the Linux::Inotify module. Inotify is a file system event-monitoring mechanism included in the Linux kernel. It is used to monitor files and directories on the file system. It is useful to watch certain files and directories on the system in order to identify suspicious behavior. Inotify2 is a Perl module for Linux operating system that implements an interface to Linux::Inotify file/directory change notification system. Linux::Inotify2 is portable and well documented. Hence Linux::Inotify2 was used initially. This module created a notify

object that acted as a container to store watches on file system names and was responsible for handling event data.

But during extensive testing it was observed that when the server CPU is overloaded, sometimes the kernel did not generate the required notification, even when the directory was created. As a result, the user specific directory is created with no results. The disk space is wasted, as the created directory will no longer be used. So to avoid this problem and achieve reliability, the daemon was implemented using Python.

Python is a very powerful language. It has clear syntax and high-level dynamic data types. The function *walk ()* walks through a particular directory and returns filenames in the directory tree. For each directory it returns a 3 tuple: *(dirpath,dirnames,filenames)*. "dirpath" is path to the directory. "dirnames" is the list of all subdirectories in "dirpath" and filenames are all files in "dirpath".

The implemented daemon checks for all directories in the "*users*" directory. It checks for a file called *done* indicating that the upload of the input files is complete in the "*data"* directory. Then if the "result" directory is present in the user specific directory, it checks if the status of the analysis is "complete" or is "in process". In these cases the daemon performs no actions and walk through the next directory. If no "result" directory is found in the user specific directory, then the daemon creates one and copies the wrapper script to it. The status of the analysis becomes "in process" and the wrapper script is executed in the background. This script invokes all the required analysis scripts and generates the results for the user in an orderly manner.

This approach was found to be more reliable than the initial one. Multiple users can upload their files. A unique link is displayed to the user once the analysis is started. After completion, results can be viewed by visiting the given link.

4 RESULTS

In this chapter we present a case study of 14 complete genomes of archaea containing all genes as amino acid sequences. We will begin by how we assemble the gene families and generate the bipartition matrix. Next, we walk through the screenshots of upload functionality provided by the tool. Then we describe SOM map obtained for different cutoff values for bootstrap support. Next, we work through a simple example that demonstrates the capabilities of the implemented tool, the web interface and the generated results. We end with some concluding remarks and thoughts about future research.

4.1 Generating Bipartition Matrix

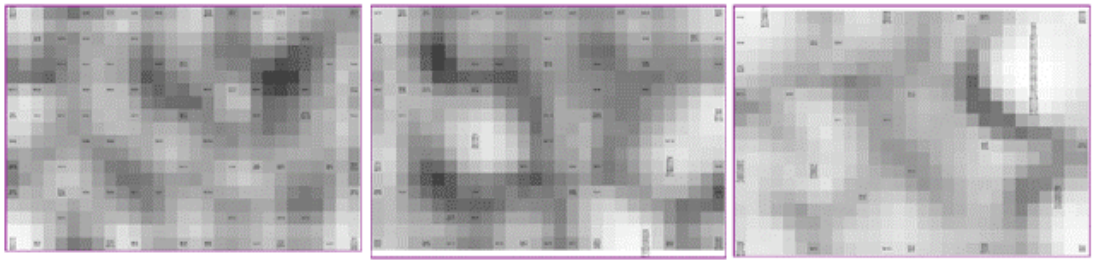
According to the process described in the methods and results section in [8], 14 complete genomes of archaea containing all genes as amino acid sequences were downloaded from NCBI ftp-site [26]. Common gene families were selected based on reciprocal best BLAST [12] hit criteria [13, 14] with relaxation (as described in section 2.1.3). To select more orthologous sets the criteria of strict reciprocity was relaxed by allowing broken connections. Applying this approach to 14 archaea, 123 gene families were selected with up to 3 broken connections (109 families were selected by strict RBH method and 14 families with up to 3 broken connections). Gene families were aligned with Clustalw [27] version 1.83 using default parameters. For each gene family tree, 100 bootstrapped replicates were generated and evaluated with the Phyml program. All 100 generated trees were split into a set of bipartitions and corresponding bootstrap support values were

assigned to each bipartition by calculating how many times each bipartition is present in a family. The bipartition matrix was composed from bootstrap values for bipartitions for each gene family with rows corresponding to gene families and columns to all possible bipartitions for a given set of taxa. There are total 8177 different bipartitions for 14 species. In theory, the bipartition matrix should contain 123 x 8191 elements. In practice, the bipartitions that are not supported by a single family were not included, so columns with all zeros were removed. As a result a matrix with 123 x 1646 elements was generated with bootstrap support values for remaining bipartitions for each family.

4.2 Cut-off Value for Bootstrap Support

The cutoff value specified by the user ranges from 0 to 100. Only bootstrap support values higher than the cutoff value are considered during the analysis. The SOM algorithm [16] was applied to bipartition matrix using the following SOM parameters: x-dimension=15, y-dimension=10,rectanglular topology, with radius equal to the x-dimension and 100,000 training iterations [23].

Three different cutoffs for bootstrap support values were tested: 0%, 70% and 90% (i.e support values below the cut-off were treated as zero). Examples of three SOM maps that resulted from three different cutoff values are given in Figure 17.



A **B** **C**

Figure 17: SOM map generated for 3 different cutoff values: 0%, 70%and 90%

As seen from the Figure 17, clusters become more pronounced with the increase of the cutoff value as smaller numbers of bipartitions are left in the matrix for the analysis, and families are grouped together only according to highly supported splits. For 90% cutoff (only bootstrap support values higher than 90% are considered) there exists one big cluster (upper right corner in figure C) that includes majority of the families with a similar phylogenetic signal. For a tested case of 14 archaea these splits are *Haloarcula-Halobacterium* and *Thermococcus-Pyrococcus*. Families that fall into one big cluster in the top right corner are the families that share high support for these two bipartitions.

A bootstrap support of 70% is considered to be still reliable in analyzing phylogenies and it allows more bipartitions to be included in the analysis. Next we demonstrate how the web-tool works on the example of 70% cutoff value.

4.3 Tool Interface: Uploading Bipartition Matrix

This section describes the dynamic analysis for 14 archaea with 70% cutoff value. Screenshots of the web interface of the tool are shown. Figure 18 shows the homepage of the tool. The “Perform Bipartition Analysis” link shown by red arrow directs the user

to the upload file page.

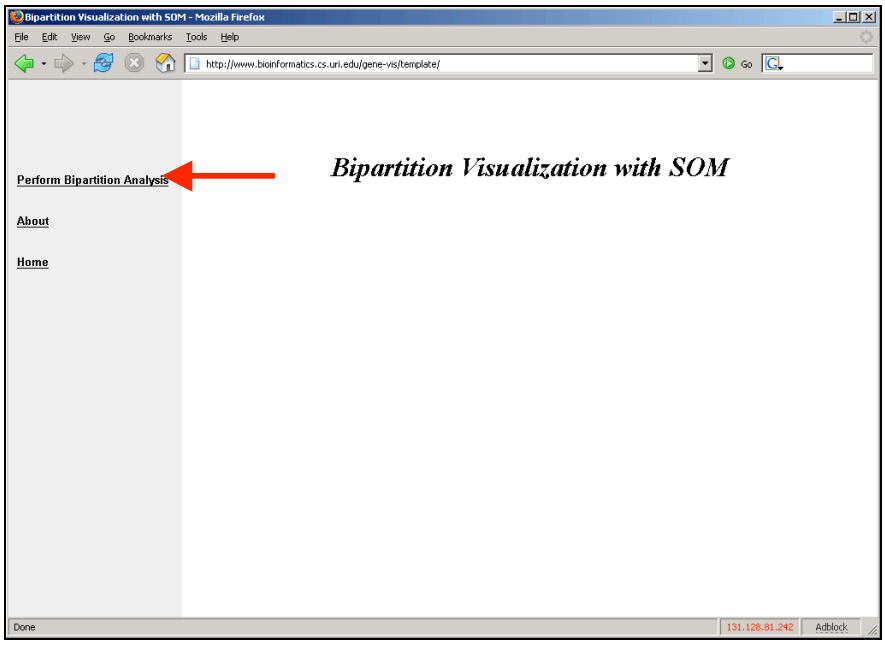


Figure 18: Homepage of the tool

The links on the left hand side frame directs the user to static results of analyses performed on different species with different cutoff values and different training iterations.

The upload web page is as shown in Figure 19. The users can browse the file system on their machine and select the bipartition matrix file. Additionally the user also selects the genomes list file, where each species is listed on a separate line. Lastly the user enters the cutoff for bootstrap support values.

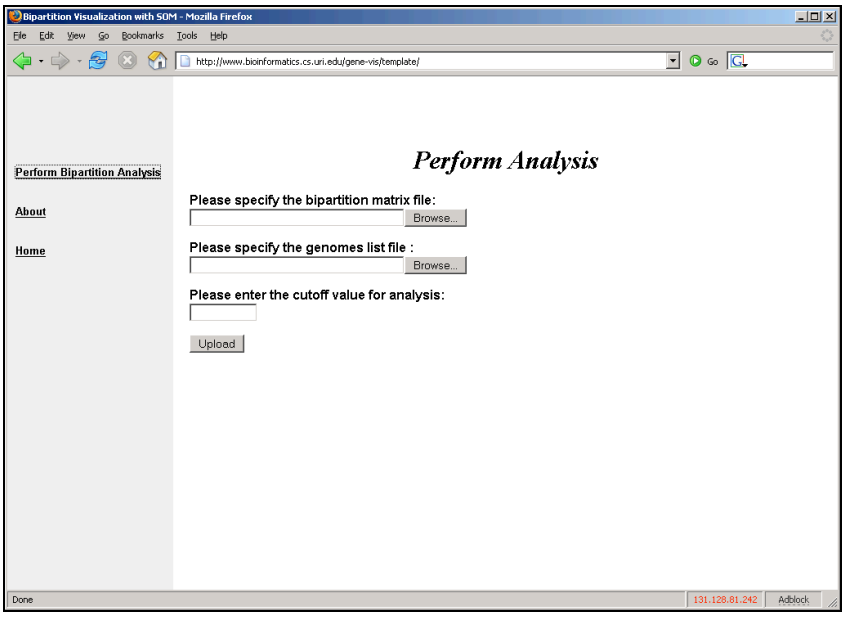


Figure 19: Web page to upload bipartition matrix and cutoff

Once the user clicks the upload button, as described in section, a new directory is created in the "*users*" directory and the analysis is started. As shown in Figure 20 (red arrow), a link to the results is provided to the user. The user can visit the link to view the results and perform further analysis. The results generated are discussed in the following sections.

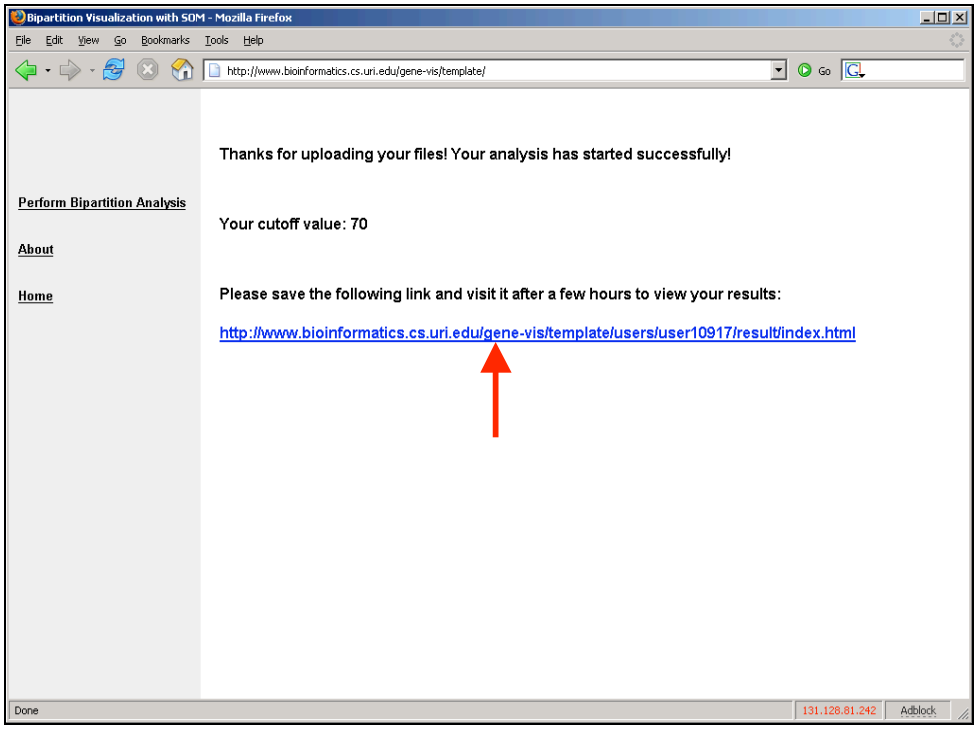


Figure 20: Link to results displayed to the user

4.4 Tool Interface: Map

After the bipartition matrix file is uploaded, analysis is performed using the train and visualize SOM analysis script. The generated results can be viewed through the tool web interface. The "Map" link directs the user to the SOM generated map. The map displays light colored areas (shown with blue circles), grey areas, and dark areas. Light colored areas show clusters that are dense and are separated from the surrounding clusters represented in grey and dark areas. Grey areas display clusters that are sparse, and dark areas represent areas with very little similarity information. In the map shown in Figure 21, white areas indicate groups of gene families whose phylogenetic signal is distinctly different from the surrounding genes. Genes that are in conflict (shown in red) with plurality bipartitions are grouped separately from the clusters containing genes

conforming to the plurality phylogeny. A clearer picture of this emerges when one studies the support of individual bipartitions.

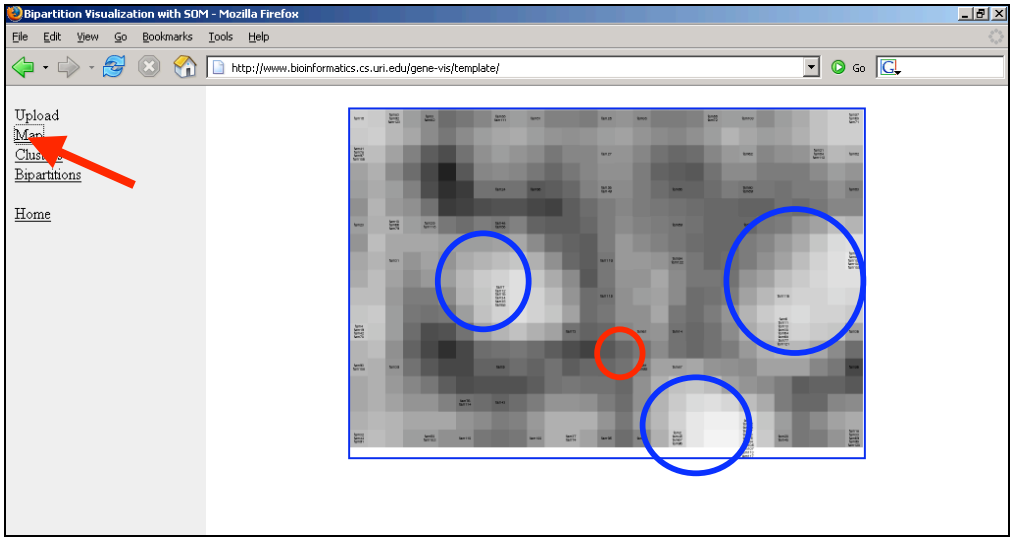


Figure 21: SOM generated map for bipartition matrix of 123 families with a cutoff of 70%.

4.5 Tool Interface: Clusters

The Clusters link directs the user to the map with clusters of gene families (as shown in Figure 22). When the user moves the mouse over a neuron that contains gene families, a pop-up window displays the cluster number, coordinates of the neuron on the SOM map and the gene families it contains. By clicking on the map, user can select any set of clusters of interest and then visualize a consensus phylogenetic tree. Selected neurons are highlighted as red squares on the map.

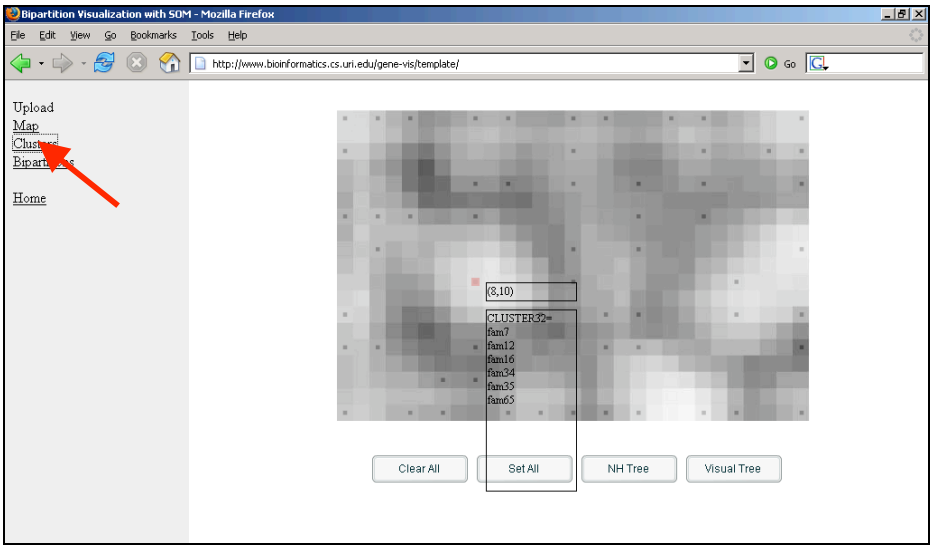


Figure 22:Clusters of gene families created by SOM.

The "Set All" button selects all the neurons on the cluster map as shown in Figure 22. After selecting all clusters, the consensus tree can be visualized using "Visual Tree" button shown in red arrow in Figure 23.

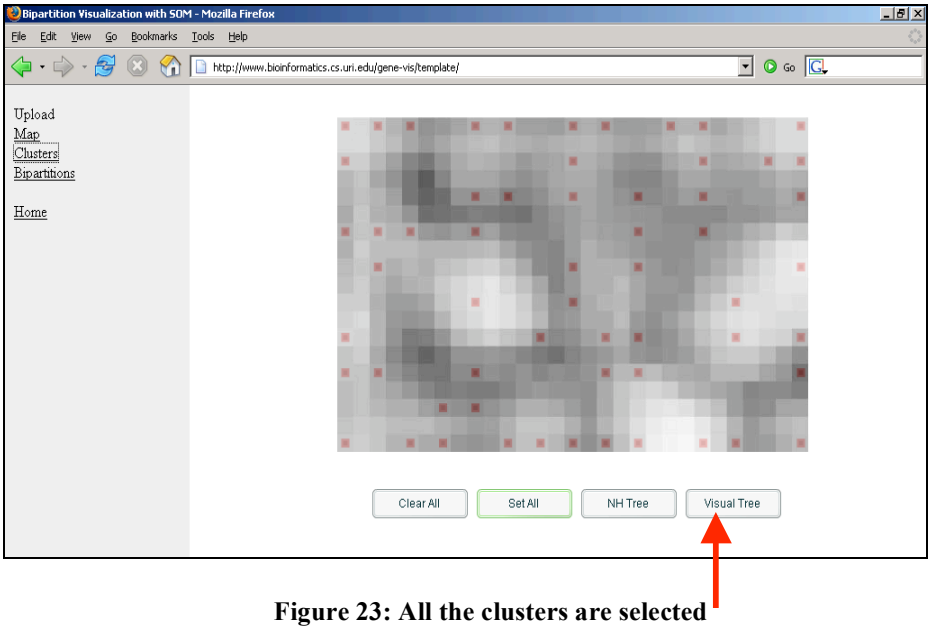


Figure 23: All the clusters are selected

As shown in Figure 24, list of the selected clusters and the gene families present in those

clusters are displayed in the browser and the consensus tree as shown in Figure 25 is displayed in a separate window using the ATV tree viewer applet.

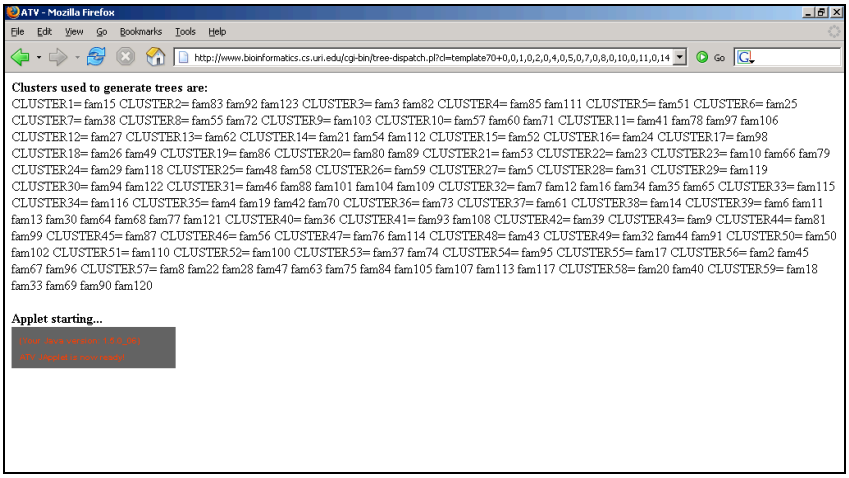


Figure 24: List of selected clusters and gene families

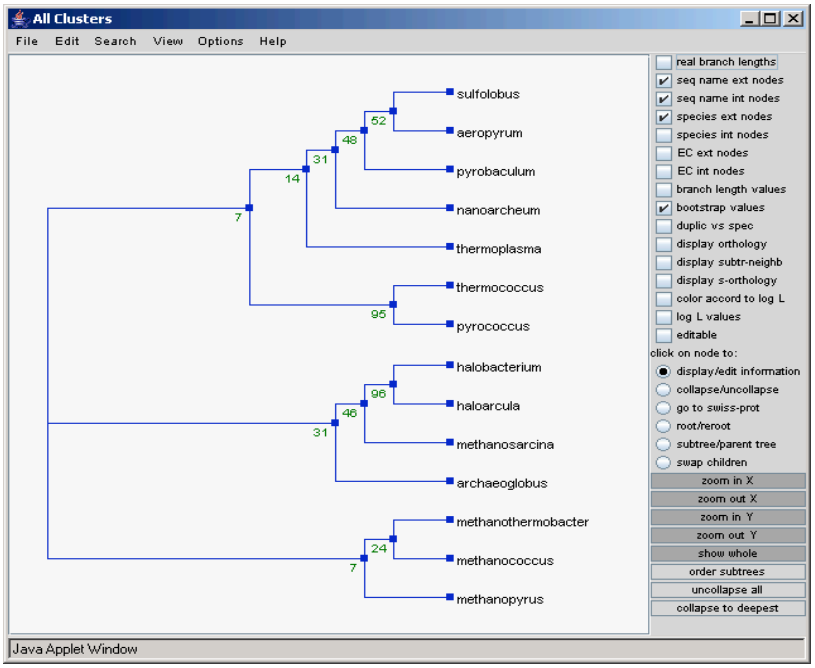


Figure 25: Consensus tree for all clusters

The ATV tree viewer applet used to visualize the tree. ATV has many options that allow the user to modify the view of the tree. User can re-root a tree with particular outgroup.

User has the option to view the bootstrap values at each tree node. Collapsing the tree at any node, zoom in and zoom out in both X and Y directions and displaying the branch length values are additional options of interest provided by ATV tree viewer.

4.6 Tool Interface: Bipartitions

The Bipartitions link directs the user to the list of bipartitions with their graphical representations. This web page begins with a list of bipartitions that are supported by the majority of the families (as seen in Figure 26). Support values for a given bipartition are given in brackets (red arrow in Figure 26).

The 3D function in Figure 26 shows bipartition support over map. It shows the slice of the SOM map that corresponds to the bipartition where white areas signify high and black areas – low support. A three-dimensional bipartition support representation is used to depict areas on the map that highly support a given bipartition. The same information is given by a 2-D representation with the gene family cluster information added in where white areas correspond to the regions that highly support this bipartition while black represent regions of conflicts.

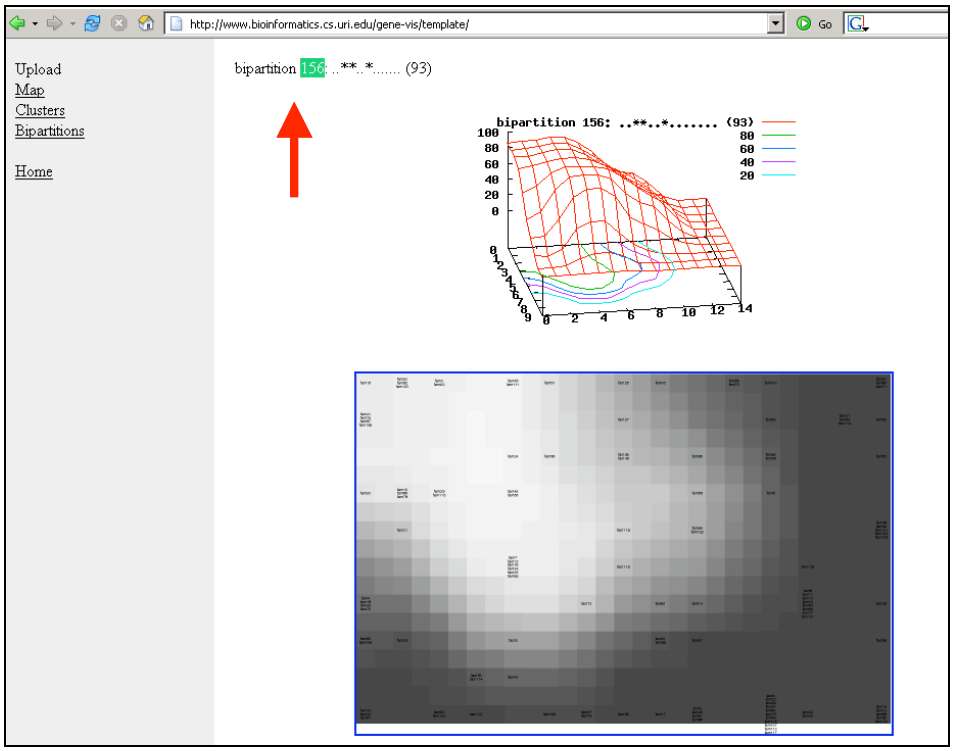


Figure 26: Bipartition 156 that groups Haloarcula, Halobacterium and Methanosarcina together.

As an example we explore in details the bipartition 156, which groups *Halobacterium, Haloarcula* and *Methanosarcina* together. This bipartition is in agreement with the small subunit ribosomal RNA phylogeny [28] and the consensus calculated from the transcription and translation machinery [29]. By selecting all clusters from the white area one can reconstruct a phylogenetic tree that would represent a history of families that are in agreement with the selected bipartition (as seen in Figure 27).

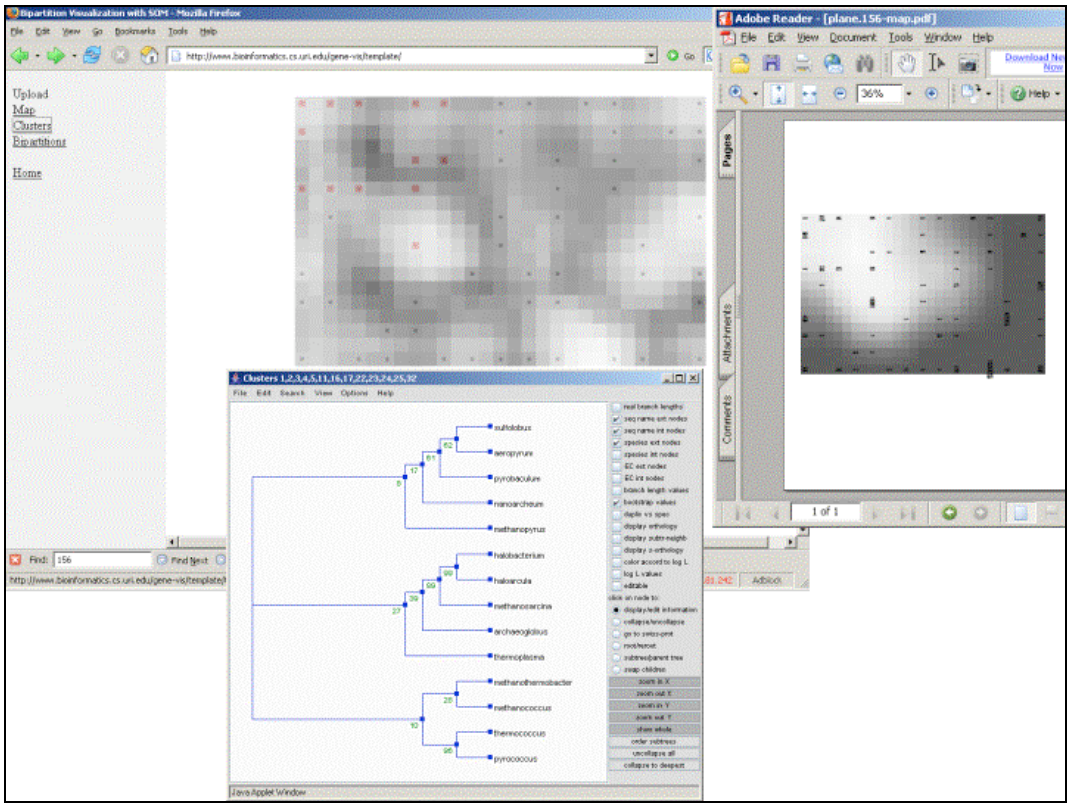


Figure 27: Tree reconstructed from the selected clusters (red dots on the left map) that fell into white areas on bipartition superposition map (shown on right)

As shown in Figure 28, bipartition 15 conflicts with bipartition 156 (left). Selected clusters from the white area that support conflicting bipartition are selected as shown in right window. Reconstructed phylogenetic tree that shows bootstrap support for conflicting bipartitions is shown in central window.

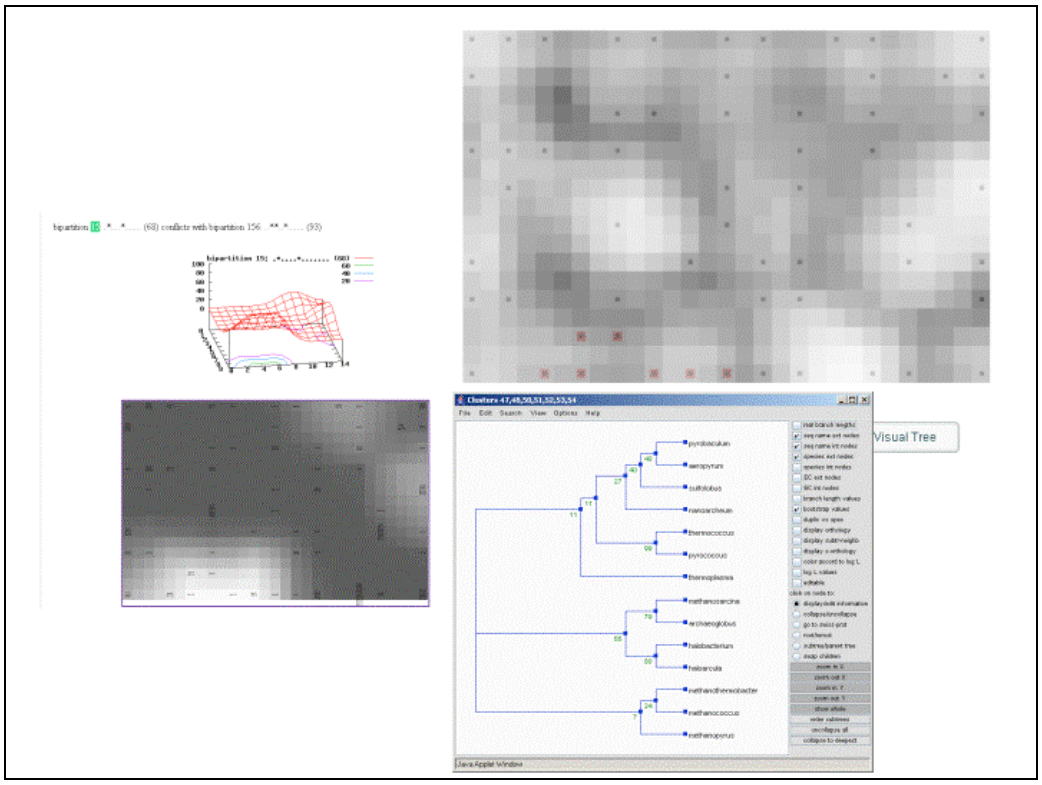


Figure 28: Analysis of a conflicting bipartition.

To find families whose phylogenetic histories are in conflict with bipartition 156, one can scroll down in the list of bipartitions and find bipartitions that show conflicts with bipartition 156. For example, bipartition 15 in Figure 28 is the one of them. Bipartition 15 corresponds to a split where *Archaeoglobus* groups together with Methonosarcina a bipartition that is in conflict with the consensus phylogeny of conserved genes [29]. White areas on a bipartition support map shows clusters that support this conflicting bipartition. The resulting phylogenetic tree confirms that *Archaeoglobus* - *Methanosarcina* branch has a high bootstrap support value in the gene families of these clusters. This finding suggests a highway of gene sharing [30] between the *Methanosarcina* and *Archaeoglobus* lineages.

4.7 Conclusions and Future Work

The work of this thesis shows that the developed web-based tool provides an interface for biologists to perform analysis and knowledge discovery on genomic data. The user can interactively conduct a phylogenetic analysis of any gene family with the ease of a single click. The web-service allows the researcher to view the genomic information contained in the bipartition matrix from a number of different perspectives:

- Gene family clusters based on similarity of bipartition support.
- Gene family consensus tree.
- Bipartition support together with conflicting bipartitions.

An important advantage of the tool is an interactive and visual identification of horizontally transferred genes. This function is implemented to detect conflicting signals in a bipartition matrix. This kind of functionality is not available in other techniques or tools.

There are some areas in which this study can be improved:

- Locally linear embedding (LLE) algorithm [31] can be used instead of the SOM algorithm. LLE is an unsupervised learning algorithm that computes low-dimensional, neighborhood-preserving embeddings of high-dimensional inputs. Unlike SOM, LLE maps its inputs into a single global coordinate system of lower dimensionality, and its optimizations do not involve local minima. By exploiting the local symmetries of linear reconstructions, LLE learns the global structure of nonlinear manifolds.

- A quartet-based approach can be explored instead of bipartitions. Bipartition analysis requires gene families containing representatives from all species of interest, thus only a relatively small number of families can be included in an analysis. A quartet-based approach will allow inclusion of incomplete gene sets where only four or more species are present [32, 33]. The idea of a quartet analysis is essentially the same as that of bipartition with [9] the one difference that only four species are considered at a time. Each quartet can have three possible topologies. The number of possible quartets is given by the formula: n!/(n-4)!4!.

- In SOM, at the end of training of the algorithm, all neurons equally represent all data vectors the neurons in the neighborhood tend to model the similar regions of the data space. The neurons in the border of the SOM map have fewer neighboring neurons than the ones inside the map. Hence they have fewer chances to be updated. The neurons at the border are therefore more similar to the ones located inside rather than the data they represent. This phenomenon is known as border effect. Border effect makes SOM harder to converge and reduces the accuracy of the map. To avoid these border effects in the current map generated by SOM, boundless maps (such as toroid maps) [17] can be used to improve the quality of the generated SOM map.

REFERENCES

[1] T. Kohonen, *Self-Organizing Maps.* Springer, 2001,

[2] J. P. Gogarten, W. F. Doolittle and J. G. Lawrence, "Prokaryotic Evolution in Light of Gene Transfer," *Mol. Biol. Evol.,* vol. 19, pp. 2226-2238, 2002.

[3] C. R. Darwin, *The Origin of Species by Means of Natural Selection. Or the Preservation of Favoured Races in the Struggle for Life.* Adamant Media Corporation,

[4] E. Hilario and J. P. Gogarten, "Horizontal transfer of ATPase genes--the tree of life becomes a net of life," *BioSystems,* vol. 31, pp. 111-119, 1993.

[5] W. F. Doolittle, "Phylogenetic classification and the universal tree," *Science,* vol. 284, pp. 2124-2129, Jun 25. 1999.

[6] W. Martin, "Mosaic bacterial chromosomes: a challenge en route to a tree of genomes," *Bioessays,* vol. 21, pp. 99-104, Feb. 1999.

[7] N. Nahar, M. S. Poptsova, J. P. Gogarten and L. Hamel, "Visualization of genome evolution using self organizing maps,"

[8] M. S. Poptsova and J. P. Gogarten, "The power of phylogenetic approaches to detect horizontally transferred genes," *BMC Evol. Biol.,* vol. 7, pp. 45, Mar 21. 2007.

[9] G. M. Lento, R. E. Hickson, G. K. Chambers and D. Penny, "Use of spectral analysis to test hypotheses on the origin of pinnipeds," *Mol. Biol. Evol.,* vol. 12, pp. 28-52, Jan. 1995.

[10] O. Zhaxybayeva, P. Lapierre and J. P. Gogarten, "Genome mosaicism and organismal lineages," *Trends Genet.,* vol. 20, pp. 254-260, May. 2004.

[11] W. M. Fitch, "Homology," *TRENDS IN GENETICS,* vol. 16, pp. 227-231, 2000.

[12] S. F. Altschul, W. Gish, W. Miller, E. W. Myers and D. J. Lipman, "Basic local alignment search tool," *J. Mol. Biol.,* vol. 215, pp. 403-410, 1990.

[13] O. Zhaxybayeva and J. P. Gogarten, "Bootstrap, Bayesian probability and maximum likelihood mapping: exploring new tools for comparative genome analyses," *Feedback,* 2004.

[14] M. G. Montague and C. A. Hutchison 3rd, "Gene content phylogeny of herpesviruses," *Proc. Natl. Acad. Sci. U. S. A.,* vol. 97, pp. 5334-5339, May 9. 2000.

[15] S. Guindon and O. Gascuel, "A Simple, Fast, and Accurate Algorithm to Estimate Large Phylogenies by Maximum Likelihood," *Syst. Biol.,* vol. 52, pp. 696-704, 2003.

[16] A. Ultsch and F. Morchen, "ESOM-Maps: tools for clustering, visualization, and classification with Emergent SOM," *Data Bionics Research Group, University of Marburg,* vol. 17, 2005.

[17] A. Ultsch, "Maps for the Visualization of high-dimensional Data Spaces," *Proc.Workshop on Self Organizing Maps,* pp. 225-230, 2003.

[18] M. Nocker, F. Morchen and A. Ultsch, "An algorithm for fast and reliable ESOM learning,"

[19] R. D. M. Page, E. Holmes and D. E. C. Holmes, *Molecular Evolution: A Phylogenetic Approach.* Blackwell Publishing, 1998,

[20] T. Margush and F. R. McMorris, "Consensusn-trees," *Bull. Math. Biol.,* vol. 43, pp. 239-244, 1981.

[21] H. Shimodaira, "An Approximately Unbiased Test of Phylogenetic Tree Selection," *Syst. Biol.,* vol. 51, pp. 492-508, 2002.

[22] J. Felsenstein, "PHYLIP (phylogeny inference package) version 3.65 Seattle: Department of Genome Sciences," *University of Washington, Distributed by the Author,* 2005.

[23] C. M. Zmasek and S. R. Eddy, "ATV: display and manipulation of annotated phylogenetic trees," *Bioinformatics,* vol. 17, pp. 383-384, 2001.

[24] J. Dwight and M. Erwin, *Using CGI.* Que, 1996,

[25] T. Kohonen, J. Hynninen, J. Kangas and J. Laaksonen, "SOM_pak: the selforganizing map program package, release 3.1," *Laboratory of Computer and Information Science, Helsinki University of Technology, Finland,* 1998.

[26] National Center for Biotechnology Information (NCBI). Ftp-site.

ftp://ftp.ncbi.nlm.nih.gov/genomes/Bacteria. Access date: July 2005

[27] J. D. Thompson, D. G. Higgins and T. J. Gibson, "CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice," *Nucleic Acids Res.,* vol. 22, pp. 4673-4680, 1994.

[28] C. Woese, "Bacterial evolution." *Microbiology and Molecular Biology Reviews,* vol. 51, pp. 221-271, 1987.

[29] C. Brochier, P. Forterre, S. Gribaldo, Y. Zivanovic and F. Confalonieri, "An emerging phylogenetic core of Archaea: phylogenies of transcription and translation machineries converge following addition of new genome sequences," *Feedback,* 2006.

[30] R. G. Beiko, T. J. Harlow and M. A. Ragan, "Highways of gene sharing in prokaryotes," *Proceedings of the National Academy of Sciences,* vol. 102, pp. 14332-14337, 2005.

[31] S. T. Roweis and L. K. Saul, "Nonlinear Dimensionality Reduction by Locally Linear Embedding," 2000.

[32] O. Zhaxybayeva and J. P. Gogarten, "An improved probability mapping approach to assess genome mosaicism," *Feedback,* 2004.

[33] O. Zhaxybayeva, J. P. Gogarten, R. L. Charlebois, W. F. Doolittle and R. T. Papke, "Phylogenetic analyses of cyanobacterial genomes: Quantification of horizontal gene transfer events," *Genome Res.,* vol. 16, pp. 1099, 2006.

BIBLIOGRAPHY

S. F. Altschul, W. Gish, W. Miller, E. W. Myers and D. J. Lipman, "Basic local alignment search tool," *J. Mol. Biol.,* vol. 215, pp. 403-410, 1990.

R. G. Beiko, T. J. Harlow and M. A. Ragan, "Highways of gene sharing in prokaryotes," *Proceedings of the National Academy of Sciences,* vol. 102, pp. 14332-14337, 2005.

C. Brochier, P. Forterre, S. Gribaldo, Y. Zivanovic and F. Confalonieri, "An emerging phylogenetic core of Archaea: phylogenies of transcription and translation machineries converge following addition of new genome sequences," *Feedback,* 2006.

C. R. Darwin, *The Origin of Species by Means of Natural Selection. Or the Preservation of Favoured Races in the Struggle for Life.* Adamant Media Corporation,

W. F. Doolittle, "Phylogenetic classification and the universal tree," *Science,* vol. 284, pp. 2124-2129, Jun 25. 1999.

J. Dwight and M. Erwin, *Using CGI.* Que, 1996,

J. Felsenstein, "PHYLIP (phylogeny inference package) version 3.65 Seattle: Department of Genome Sciences," *University of Washington, Distributed by the Author,* 2005.

W. M. Fitch, "Homology," *TRENDS IN GENETICS,* vol. 16, pp. 227-231, 2000.

J. P. Gogarten, W. F. Doolittle and J. G. Lawrence, "Prokaryotic Evolution in Light of Gene Transfer," *Mol. Biol. Evol.,* vol. 19, pp. 2226-2238, 2002.

S. Guindon and O. Gascuel, "A Simple, Fast, and Accurate Algorithm to Estimate Large Phylogenies by Maximum Likelihood," *Syst. Biol.,* vol. 52, pp. 696-704, 2003.

E. Hilario and J. P. Gogarten, "Horizontal transfer of ATPase genes--the tree of life becomes a net of life," *BioSystems,* vol. 31, pp. 111-119, 1993.

T. Kohonen, *Self-Organizing Maps.* Springer, 2001,

T. Kohonen, J. Hynninen, J. Kangas and J. Laaksonen, "SOM_pak: the selforganizing map program package, release 3.1," *Laboratory of Computer and Information Science, Helsinki University of Technology, Finland,* 1998.

G. M. Lento, R. E. Hickson, G. K. Chambers and D. Penny, "Use of spectral analysis to test hypotheses on the origin of pinnipeds," *Mol. Biol. Evol.,* vol. 12, pp. 28-52, Jan. 1995.

T. Margush and F. R. McMorris, "Consensusn-trees," *Bull. Math. Biol.,* vol. 43, pp. 239-244, 1981.

W. Martin, "Mosaic bacterial chromosomes: a challenge en route to a tree of genomes," *Bioessays,* vol. 21, pp. 99-104, Feb. 1999.

M. G. Montague and C. A. Hutchison 3rd, "Gene content phylogeny of herpesviruses," *Proc. Natl. Acad. Sci. U. S. A.,* vol. 97, pp. 5334-5339, May 9. 2000.

N. Nahar, M. S. Poptsova, J. P. Gogarten and L. Hamel, "Visualization of genome evolution using self organizing maps,"

National Center for Biotechnology Information (NCBI) ftp-site. ftp://ftp.ncbi.nlm.nih.gov/genomes/Bacteria July 2005

M. Nocker, F. Morchen and A. Ultsch, "An algorithm for fast and reliable ESOM learning,"

R. D. M. Page, E. Holmes and D. E. C. Holmes, *Molecular Evolution: A Phylogenetic Approach.* Blackwell Publishing, 1998,

M. S. Poptsova and J. P. Gogarten, "The power of phylogenetic approaches to detect horizontally transferred genes," *BMC Evol. Biol.,* vol. 7, pp. 45, Mar 21. 2007.

S. T. Roweis and L. K. Saul, "Nonlinear Dimensionality Reduction by Locally Linear Embedding," 2000.

H. Shimodaira, "An Approximately Unbiased Test of Phylogenetic Tree Selection," *Syst. Biol.,* vol. 51, pp. 492-508, 2002.

J. D. Thompson, D. G. Higgins and T. J. Gibson, "CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice," *Nucleic Acids Res.,* vol. 22, pp. 4673-4680, 1994.

A. Ultsch, "Maps for the Visualization of high-dimensional Data Spaces," *Proc.Workshop on Self Organizing Maps,* pp. 225-230, 2003.

A. Ultsch and F. Morchen, "ESOM-Maps: tools for clustering, visualization, and classification with Emergent SOM," *Data Bionics Research Group, University of Marburg,* vol. 17, 2005.

C. Woese, "Bacterial evolution." *Microbiology and Molecular Biology Reviews,* vol. 51, pp. 221-271, 1987.

O. Zhaxybayeva and J. P. Gogarten, "Bootstrap, Bayesian probability and maximum likelihood mapping: exploring new tools for comparative genome analyses," *Feedback,* 2004.

O. Zhaxybayeva and J. P. Gogarten, "An improved probability mapping approach to assess genome mosaicism," *Feedback,* 2004.

O. Zhaxybayeva, J. P. Gogarten, R. L. Charlebois, W. F. Doolittle and R. T. Papke, "Phylogenetic analyses of cyanobacterial genomes: Quantification of horizontal gene transfer events," *Genome Res.,* vol. 16, pp. 1099, 2006.

O. Zhaxybayeva, P. Lapierre and J. P. Gogarten, "Genome mosaicism and organismal lineages," *Trends Genet.,* vol. 20, pp. 254-260, May. 2004.

C. M. Zmasek and S. R. Eddy, "ATV: display and manipulation of annotated phylogenetic trees," *Bioinformatics,* vol. 17, pp. 383-384, 2001.