An Annotated *k*-deep Prefix Tree for (1-*k*)-mer Based Sequence Comparisons

Adrienne Breland[1], Karen Schlauch[2], Monica Nicolescu[1], Frederick C. Harris Jr.[1]

Dept. of Comp. Sci. & Engr.[1]
001-775-784-6974
{breland,monica,fredh}@cse.unr.edu

University of Nevada, Reno
Reno, NV 89557

Department of Biochemistry[2]
001-775-784-8974
schlauch@unr.edu

ABSTRACT

In this report, we describe an algorithm for a *k*-deep annotated prefix tree. The algorithm provides an alignment-free method for comparing nucleotide sequences in a computationally efficient manner. Differences in genomic sequences are measured by recording and comparing counts of words of length *k* or less in each sequence using the algorithm. Tree nodes are annotated with lists to store the number of times each word occurs in each of a group of sequences. Count differences among multiple sequences may be computed in a single tree traversal. Such a tree is built in linear time and spatially bounded by tree depth rather than sequence length(s). We then compare sequence groups of both *E. coli* and *Influenza A virus* H1N1 to demonstrate the power of a *k*-deep prefix tree when used as sequence comparison tool.

Categories and Subject Descriptors

E.1 [**Data Structures**]: Trees

General Terms

Algorithms, Measurement, Experimentation, Languages.

Keywords

K-mer, prefix-tree, d square distance.

1. INTRODUCTION

A prefix-tree is a string compression algorithm which reduces prefix redundancy in a given set of strings. Prefix-trees are tree graphs similar to tries and dictionaries in natural language processing and data compression. A prefix-tree is built on a set of strings $X \in \sum^*$ where $\sum$ is a finite alphabet. Each path from the root to any existing node is labeled by the spelling of a non-empty prefix of any strings in X. In natural language processing, tries enable linear time string matching [9] while dictionaries are used in the GNU zip (gzip) compression algorithm [18]. Tries built on the English alphabet, $\sum = \{a,...,z\}$ grow quickly with a branching factor of 26, and minimization algorithms are common [1].When comparing nucleotide sequences, prefix-trees may be built on a smaller alphabet, $\sum = \{A, C, G, T\}$ and can provide fast indexing of nucleotide words of multiple lengths. A comparable structure was used in [3] to compare the 12-mer "languages" of human chromosomes 21 and 22. Prefix-trees are also used in the assembly program SSAKE [18] to locate overlapping 25-mers between short nucleotide fragments.

The *k*-deep prefix tree we present in this report is limited in depth to include only *k* levels. It is constructed from the *k* first characters of all non-empty prefixes of a single or set of genomic sequences. The number of nodes required is exponentially proportional to tree height (*k*) rather than the total length of sequences, as is the case with suffix trees [14]. This provides a compressed and partially or fully dynamically allocated index into all substrings up to a given length (*k*) found in a single or groups of sequences. While index based hash tables are generally used for this same purpose [2,4,11,12,15], prefix-trees can be more comprehensive because they may include information regarding nucleotide words of multiple lengths. Hash tables generally represent nucleotide word of a single length, and require large, contiguous blocks of memory for fast look up times. A prefix-tree may be implemented with dynamic memory and is equivalent to multiple hash tables for each word length (1,..,*k*), with direct links between each word and its prefix and suffix(es).

Our algorithm also includes node annotation. Tree nodes are annotated with substring occurrence counts, which record the number of times that a substring terminating at each node occurs in each of a set of sequences. Lists at each node allow the storage of information pertaining to multiple sequences in a single tree. This facilitates all-against-all sub word count differencing among a set of sequences in a single tree traversal. The use of tree node annotation is also seen in Generalized Suffix Trees to enables suffix comparisons among multiple sequences [6], and is discussed in detail in [5].

A *k*-mer denotes a substring (word) of a genomic nucleotide sequence of length *k*; a (1-*k*)-mer is a word of length *k* or less. Nucleotide word counts can form the basis of alignment-free sequence comparisons. Sequence comparisons derived from *k*-mer compositions have been used to construct phylogenies which encompass the tree of life [13], as well as enabled sub-species clustering of viral isolates [7]. The word length *k* best suited in *k*-mer based comparative measures is often arbitrarily chosen, and must be addressed as a research question in its own right [16]. A fully annotated *k*-deep prefix tree allows data exploration and the inclusion of multiple word lengths in a single analysis.

In the following, we describe annotated *k*-deep tree construction and its algorithmic complexity. We then describe how (1-*k*)-mer composition comparisons among multiple genomic sequences may be conducted in a single tree traversal. Finally, we illustrate these methods on two sample data sets.

2. K-DEEP PREFIX TREE

2.1 Tree Construction

A single sequence *k*-deep prefix tree is built from all substrings of length k or less in a genomic sequence $S_j = \left[s_{j_1} \dots s_{j_{l_j}}\right]$, where $s_j \in \{A, C, G, T\}$, and l_j denotes the length of S_j. Let w be a substring of S_j, and let $|w|$ denote its length. In the resulting tree, each path from the root to any internal or leaf node at level ℓ, 1≤ ℓ ≤ k, spells a substring w of S_j such that $[w_0 \dots w_{\ell-1}]$ = $\left[s_{j_m} \dots s_{j_{m+\ell}}\right]$, $0 < m \leq |S_j| - \ell - 1$. s_{j_m} denotes the character at position in S_j where substring w begins. The tree is built in linear time proportional to $|S_j|$ by parsing S_j once with a sliding window.

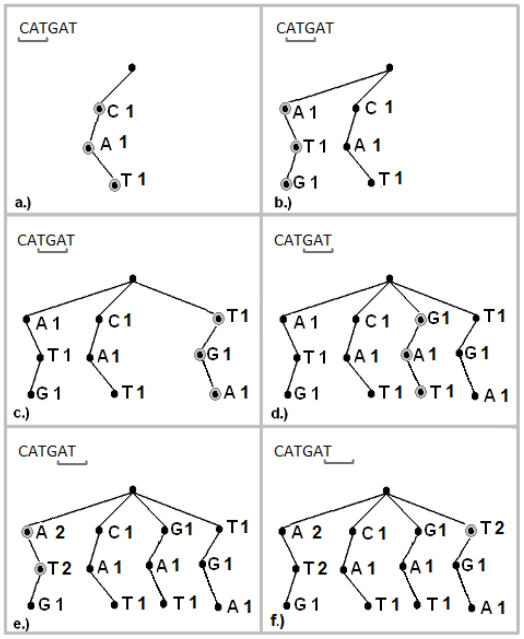

Figure 1(a-e). Building a 3-deep annotated prefix-tree from the sequence "CATGAT".

In Figures 1(a-f), building a 3-deep prefix-tree from the sequence "CATGAT" is illustrated. Because each node is regarded as a prefix string termination, sentinel nodes marking end of strings are not required. A sequence is parsed by a single spaced sliding window of length k, where in the given example k = 3. Each overlapping nucleotide string determined by the window is inserted into the tree. Each tree node may point to up to four children {'a'_child, 'c'_child, 'g'_child, 't'_child}. The default value for all of a node's children is set to NULL. If a word path in the tree does not yet exist, it is built upon insertion. This removes the potential for wasting memory on nodes which represent non-existent words in the set of sequences being examined.

As a word is inserted into the tree, existing nodes may be traversed, or new nodes may be created. At each node involved, either in a traversal or creation, a count at that node is incremented. This process of editing each node that is passed through yields the inclusion of information regarding all substrings of length 1, ... ,k. Nodes at any level of the tree, ℓ , $1 \leq \ell \leq k$ contain counts of all unique strings of length ℓ found in the sequence used to build the tree. The length of a unique substring terminated at a node is implicitly represented by the nodes level in the tree.

Count lists at each node allow the inclusion of multiple sequences in a single tree. If N sequences are examined, each node contains an integer count list of length N. When parsing sequences S_j, j ∈ N, passing through a node causes an increment only at position j in the count list located at that node. Figure 2 illustrates a 3-deep tree built from the sequences "CATGA" and "ATCAT".

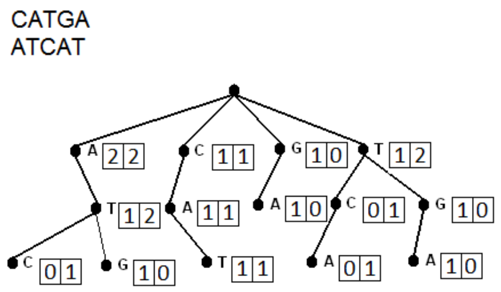

Figure 2. Node annotation for 2 sequences .

In Figure 2, (1-*k*)-mer counts for "CATGA" are stored in the first index of the count list at each node. Counts for "ATCAT" are stored in the second index. By maintaining node based count lists, a single tree can contain the complete (1-*k*)-mer composition of multiple sequences. This can then lead to all-against-all *k*-mer based sequence comparisons in a single tree traversal.

2.2 Algorithmic Complexity

Assuming that N sequences are used to build a depth limited prefix tree, the asymptotic time required to build the tree is linear, $\ominus\left(\sum_{j=1}^{j=N} |S_j|\right)$ as each sequence S_j need be parsed only once. If k is the maximum word length of interest, the maximum number of nodes in the tree is given by K, where $K = \sum_{x=0}^{x=k} 4^x$. Each non-root node contains a list of counts containing the number of times that the prefix terminated at that node occurs in each sequence. Thus, the space required is asymptotically bounded by $O(1 + (K-1) * N) \approx O(K * N)$.

This can allow a space reduction of very long sequences. For example, a 10-deep tree constructed from 5 human sequences of chromosome 11 (approximate length $\approx 1.3\,x\,10^8$ base pairs) could maximally contain approximately $1.4\,x\,10^6$ nodes with 5 integer counts, four pointer addresses (parent, 'a'_child, 'c'_child, 'g'_child, 't'_child) and one character stored per node. The sequences alone would contain a sum of approximately $1.3\,x\,10^9$ characters. A fully expanded 10-deep tree would reduce the total sequence space by at least 2 orders of magnitude.

Space requirements may further be reduced by omitting word counts of nodes in upper levels of the tree. These nodes represent shorter *k*-mers and may sometimes be nonspecific enough for analysis. Assuming k-mers where $k \leq u$ are deemed non interesting, the space required would be bounded by $O(c + K * N), c = \sum_{x=0}^{x=u} 4^x, K = \sum_{x=u+1}^{x=k} 4^x$. While this does not offer drastic improvement, it can prove useful when space becomes a limiting factor for large values of k.

3. NODE BASED DISTANCE SCORING

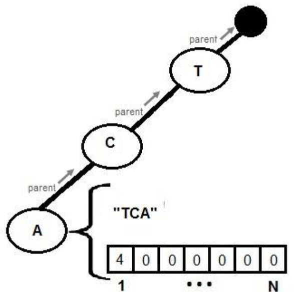

Figure 3. The specific word terminated and its occurrence counts may be determined at each node.

The k-deep prefix-tree is a succinct method that enables sequence exploration and comparison. Each non-root tree node includes the representation of a sequence substring, and the number of times the substring appears in a set of sequences (Figure 3). For example, at the node indicated in Figure 3, we may determine that the substring "TCA" appears only four times in the first of a set of sequences. Note that if "TCA" did not occur in any sequence, the node in Figure 3 would not have been created. Determining the nucleotide word terminated by any node requires a trace back through parent pointers from that node to the root. This is accomplished in time linear to the tree level of the node, which is the length of the word.

From a k-deep prefix-tree built upon N sequences, an NxN distance matrix M tabulating pair-wise distances may be completed by visiting each tree node only once while incrementally modifying matrix positions. Each distance matrix entry $M_{ij}, i \in N, j \in N$ will be computed as the sum of the distance for each word w between sequences S_i and S_j, for all words w in the set W of all (1-k)-mers found in the sequence set. Distance may be defined by the researcher as any count based measure. Distance matrices in this form provide a simple platform for sequence clustering and phylogenetic tree construction.

Figure 4 provides the pseudocode for a recursive all against all comparison of non-root nodes. In this pseudocode, a pre-order tree traversal is conducted. Pair-wise distance scores among all sequence pairs (i,j) are computed at each node. Resulting values are added to each corresponding matrix position. Thus after a tree traversal is complete, at each resulting matrix position:

$$M_{ij} = M_{ji} = \sum_{w}^{W} Tree\_distance\ (w)$$

```
Tree_distance (node * curnode){
        $for\ i = 1{:}N-1$
            $for\ j = i+1{:}N$
                $M_{ji} = M_{ij} += distance(i,j)$

    if(curnode →a_child != NULL)
        Tree_distance (curnode →a_child)
    if(curnode →c_child != NULL)
        Tree_distance (curnode →c_child)
    if(curnode →g_child != NULL)
        Tree_distance (curnode →g_child)
    if(curnode →t_child != NULL)
        Tree_distance (curnode →t_child)
}end Tree_distance
```

Figure 4. Pseudocode for distance scoring pre-order tree traversal.

3.1 Example Comparisons

As an example, we build k-deep prefix trees and compute distance matrices for two genomic data sets. The data sets are composed of microbial subspecies strains and contain sequences which are all similar in length. These data sets include; 1.) 12 HA segments of human H1N1 *Influenza* A *virus* and 2.) five complete *Escherichia coli* chromosomal sequences. Pair- wise sequence distances within data sets are computed using weighted d^2 distance [17] calculations, a count based differencing measure. In the following section, the d^2 distance, as presented by [17] is described.

Tree depths are dependent on the data set. We follow the approximation in [16] suggesting that setting $k \approx log_4(|S_n|)$, where $|S_n|$ is sequence length allows for a sufficiently descriptive k value of prokaryotic sequences. As our data sets contain multiple sequences or varying length, we use average sequence length per data set so that $k \approx log_4(|\overline{S_n}|)$. The best integer fit to our data yield k=5 for the H1N1 sequences and k=13 for the *E.coli* sequences.

3.2 d^2 Distance

The d^2 distance is a method for scoring k-mer count differences between genomic sequences. The d^2 comparison is an example of a comparison metric which can be implemented on a node by node basis. Comparing k-mer counts between genomic sequences is referred to as the d_k^2 distance. This measure has been used to cluster expressed sequence tabs (EST's) [8]. The d_k^2 distance between two sequences S_i and S_j is described in [17] as:

$$d_k^2(i,j) = \sum_{x=1}^{4^k} p_x\ (c_x(i) - c_x(j)\ )^2 \qquad (1)$$

where k is a fixed integer word length, $c_x(i)$ and $c_x(j)$ indicate counts of word w_x in sequences S_i and S_j respectively.

The information required for the calculation of d_k^2 distances between sequence pairs is contained within the described *k*-deep nucleotide tree. Confining k to a single value equates to examining nodes at a single tree depth. Current usage of this measure is often limited to nucleotide words of length six, i.e. $k = 6$ [10]. However, in [17] it is suggested that it may be advantageous to allow a range of values for *k*, with:

$$d^2(i,j) = \sum_{k=m}^{u} d_k^2(i,j) \qquad (2)$$

where m is the user defined minimum word length and u is the user defined maximum. If $m = 1$ and $u = k$, computing this distance score is achieved by visiting each node in a *k*-deep prefix tree in a single traversal. It is this measure that we use to compute distance matrices for our example datasets. The asymptotic time required for a node based d^2 traversal is:

$$O(K\,x\left(N + \frac{Nx(N-1)}{2}\right) = O(KN^2) \qquad (3)$$

where $K = \sum_{x=1}^{x=k} 4^x$ and *N* is the number of sequences examined.

The *k*-deep tree allows the repeated iterative trial of any number of comparison subsets and or weights for computing pair-wise d^2 distances among multiple sequences. All *k*-mer counts are maintained and can be returned to as a starting point. The weight associated with a unique (1-*k*)-mer may be derived from descriptive measures regarding it in the data set. Using this approach, weights can be derived independently on a node by node basis requiring no additional tree traversals. In this report, two different node based weighting schemes are applied to d^2 distances computed for each data set. This allows that the effect of changing weighting schemes can be examined in resulting Neighbor Joining (NJ) cladograms.

As a reference to other weighting schemes, we also compute unweighted d^2 distances within data sets. Thus $p_x = 1$ (eq.1) at all nodes to compute unweighted distance matrices.

3.3 GC-content Weighting

Because each non root node in a *k*-deep prefix tree represents a unique (1-*k*)-mer, it is possible to weight difference values computed at that node based on the *k*-mer sequence it represents. For example, repetitive sequences may be ignored or *k*-mers containing specific sub words may be selected. In this example, we perform weighting based on GC content. GC content is the percentage of G or C nucleotides in a word divided by the total number of nucleotides it contains. This yields a value between zero and one. We combine both GC content weighting and cutoff values in this first weighting scheme. The weight for any (1-*k*)-mer with at least 80% GC content is equivalent to its GC content. Any (1-*k*)-mer with less than 80% is given a zero weight. Thus p_x in equation 1 is defined as:

$$p_x = \begin{cases} \dfrac{n_x(G,C)}{n_x(A,C,T,G)} & if\ \dfrac{n_x(G,C)}{n_x(A,C,T,G)} \geq 0.8 \\ 0 & else \end{cases}$$

where $n_x(G,C)$ denotes the number of occurrences nucleotide bases G and C in subword W_x. This weighting scheme requires a trace back from each node to the tree root to compute GC content of the (1-*k*)-mer indexed by that node. The time required for this is linearly proportional to node level, or (1-*k*)-mer length. An all-against-all comparison of count list values may be conducted based on the determined GC score and resulting weight. All-against-all comparisons may be omitted by determining a zero weight. While the asymptotic computation time does not change, in our analysis this weighting scheme greatly reduced the total number of nodes examined in an all-against-all fashion in both datasets.

3.4 Presence vs. Absence Weighting

We define a weighting scheme based on word presence and absence across sequence sets. We define nucleotide words which are present in at least one sequence and absent in at least one other sequence to exhibit presence/absence variation. Weights are given a value of zero or one based on this observation. Thus if p_x represents the weight for W_x and $C_x(j)$ the count of W_x in sequence S_j, then:

$$\boldsymbol{p_x} = \begin{cases} \mathbf{1} & \boldsymbol{if} \sum_{j}^{N} \boldsymbol{C_x(j)} > 0 \text{ and } \prod_{j}^{N} \boldsymbol{C_x(j)} = \mathbf{0} \\ \mathbf{0} & \boldsymbol{else} \end{cases}$$

For example, assume a given word only occurs in three out of five sequences. This word would be given a weight equal to one and would contribute to final pair-wise distance scores. If however, it was found at least once in all five sequences, it would be assigned a zero weight and yield no contribution. This weighting scheme also allows a potential reduction in the number of nodes which must be examined fully. At each node, the entire count list of N sequences must be examined to determine if at least one count is zero and at least one is greater than zero. As in the GC content weighting scheme, all-against-all comparisons may be avoided if a zero weight is determined.

4. RESULTS

4.1 Neighbor Joining Cladograms

Figures 5 and 6(a-c) display Neighbor Joining (NJ) [16] cladograms computed with no weight, GC-content weighting, and presence/absence weighting. The H1N1 dataset contains three sequences representing each of four locations; Alaska, Texas, Mexico, and California. Differences exist among all three cladograms with regards to branched cluster formations among geographic groups. For example, GC-content weights resulted in MX_2 and MX_3 being closest neighbors, while presence/absence weighting resulted in MX_1 and MX_3 being closest neighbors. The unweighted d^2 distance resulted in Texas_05 being included in a branched cluster with samples originating from Mexico.

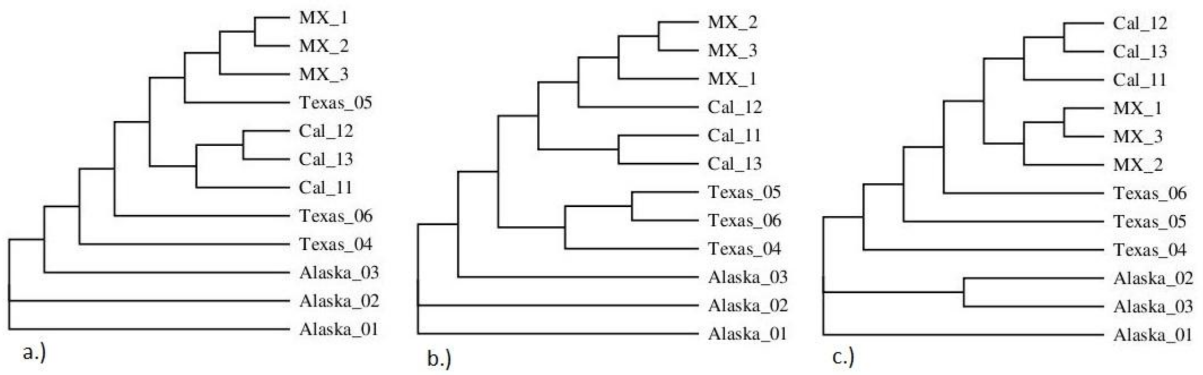

Figure 5. Cladogram of H1N1 sequences using unweighted(a), GC-content weighting(b) and presence/absence weighting(c) schemes.

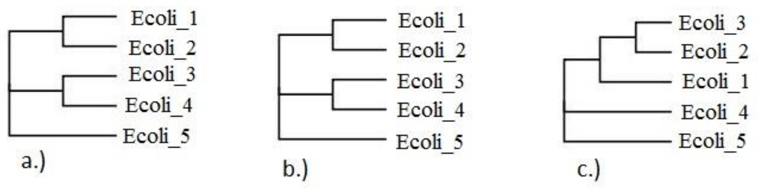

Figure 6. Cladogram of *E. coli* sequences using unweighted(a), GC-content weighting(b) and presence/absence weighting(c) schemes.

Table 1. Actual vs. total possible number of nodes.

$Dataset$	k	$possible\ nodes\ (K)\ \sum_{i=0}^{k} 4^i$	$actual\ nodes(K^1)$	$\%\left(\frac{K^1}{K}\right)$
H1N1	5	1,364	1,181	86.6%
E.coli	13	89,478,484	22,311,042	2.9%

Table 2. Number of nodes weights (p) found to be greater than zero using GC-content weighting scheme. $K^1 - 1$ is the total number of non-root tree nodes.

$Dataset$	p	$\%\left(\frac{p}{K^1 - 1}\right)$
H1N1	135	11.4%
E.coli	462,681	2.1%

Table 3. Number of nodes weights (p) found to be greater than zero using presence/absence weighting scheme. $K^1 - 1$ is the total number of non-root tree nodes.

$Dataset$	p	$\%\left(\frac{p}{K^1 - 1}\right)$
H1N1	296	25.1%
E.coli	18,654,648	83.6%

Cladograms of the five *E. coli* samples were identical between the unweighted and GC-content weighted d^2 distance calculations. Presence/absence weighting primarily resulted in a difference in the placing of the Ecoli_3 sample.

Determining the accuracy of these cladograms is not in the scope of this research. Instead, our goal is to present an algorithm for experimentation with various weighting schemes and more generally, *k*-mer based differencing measures.

4.2 Space Compression

For each data set, the number of nodes created in *k*-deep prefix trees were a fraction of the total number of possible nodes. That is, the actual number of unique (1-*k*)-mers encountered in each data set verses the total number of possible nucleotide *k*-mers of lengths (1,..,*k*).

Table 1 shows this ratio for each data set. In each case, using a prefix-tree allowed a reduction in the total required memory space which would be necessary if using uncompressed hash tables. The ratio of actual vs. total number of possible nodes was 86.6% and 2.9% for the H1N1 and *E .coli* datasets respectively. The degree of compression enabled by the *k*-deep prefix tree is particularly reflected in the *E. coli* data set.

4.3 Weight Based Computation Reduction

Each weighting scheme described includes zero weights. This dictates that certain nodes do not contribute to total distance scores and all-against–all comparisons were not required at those nodes. Tables 2 and 3 report the actual number of nodes with greater than zero weights found in GC-content and presence/absence weighting. Tables 2 and 3 also list the percentage of tree nodes at which full computations were required.

While both weighting schemes enabled a reduction in number of nodes fully examined, the GC-content weight scheme provided the greatest reduction. Only 11.4% and 2.1 % of all tree nodes contributed to distance matrices in the H1N1 and *E.coli* datasets using GC-content weights. The percentage of nodes examined fully using the presence/absence weighting scheme were 25.1% and 83.6%.

These results suggest that definitive word based features may be extracted through experimentations such as these. The *E .coli* cladograms based on the GC-content and the unweighted d^2 distance were identical. However GC-content distances were based on only approximately 2% of all nodes used to compute the unweighted scores.

5. CONCLUSIONS and FUTURE WORK

We described an algorithm for the construction of an annotated *k*-deep prefix tree. In a *k*-deep prefix tree, each non-root tree node represents a unique (1-*k*)-mer which exists in the genomic sequence(s) from which the tree was constructed. In this report, we annotated each node with occurrence counts, denoting the number of times that the (1-*k*)-mer terminated at that node occurs in each sequence.

We then provided an example usage of the tree by computing weighted and unweighted d^2 distances matrices for sample data sets. The d^2 distance is a *k*-mer based distance measure, and the information required to compute d^2 distances is contained within the described *k*-deep tree. We presented two weighting schemes: 1.) the GC nucleotide content of each word terminated at non-root nodes and 2.) presence/absence occurrence variation derived from count lists at each non-root node. Resulting Neighbor Joining cladograms were then compared.

In the *E. coli* dataset, two cladograms based on unweighted and GC-content weighted distance matrices were identical. This was surprising given that the GC-weighted distances were derived from only approximately 2% of the nodes used to compute unweighted distances. This illustrates the potential for extraction of important *k*-mer based features through experimentation with annotated *k*-deep prefix trees.

Future work will include research into the potential for reliance on more static rather than dynamic memory allocation. This would allow faster tree traversal by ensuring that traversing links between nodes resulted in fewer page faults. Future work will also use our *k*-deep tree algorithm to examine differences between sets of sequence based on several weighting schemes and *k*-values.

6. REFERENCES

[1] Aho A.V. and Corasick, A. J. 1975. Efficient string matching: An Aid to Bibliographic Search. Communications of the ACM 18,333-341.

[2] Altschul S.F., Gish, W., Miller, W., Myers, E.W., and Lipman, D.J. 1997. Basic Local Alignment Search Tool. Mol. Bio. 215, 403-410.

[3] Arnau V. and Marin, M. 2003. A fast Algorithm for the Exhaustive Analysis of 12-Nucleoitde-Long DNA Sequences. Applications to Human Genomic. Proceedings of the IEEE International Parallel Distributed Processing Symposium(IPDPS'03).

[4] Arnau V., Gallach M., and Marin I. 2008. Fast comparison of DNA sequences by oligonucleotide profiling. Proceedings of the IEEE International Parallel and Distributed Processing Symposium.

[5] Apostolico A, Bock, M.E., Leonardi, S., and Xu, X. 2000. Efficient Detection of unusual words. J. Comput. Biol.

[6] Bieganski, P., Reidl, J., Cartis, J., V., Retzel, E. F. 1994. Generalized suffix trees for biological sequence data: applications and implementation. in System SciencesV: Biotechnology Computing, Proceedings of the Twenty-Seventh Hawaii International Conference on 5, 35-44.

[7] Breland, A., Nasser, S., Schlauch, K., Nicolescu, M., Harris, F.C. Jr. 2008. Efficient Influenza A Virus Origin Detection. Journal of Electronics and Computer Science 10, 1-8.

[8] Burke, J., Davison, D., and Hide, W. 1999. d2_cluster: a validated method for clustering EST and full-length cDNAsequences. Gen. Res. 9, 1135-1142.

[9] Daciuk, J., Mihov, S., Watson, B.W., and Watson, R.E. 2000. Incremental Construction of Minimal Acyclic Finite_State Automata. Association for Computational Linguistics 28, 207-216.

[10] Hazelhurst, S.2004. An efficient implementation of the d2 distance function for EST clustering:preliminary investigations. Proceedings of the 2004 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries 7, 229-233.

[11] Iseli C., Ambrosini, G., Bucher, P., and Jongeneel, C. V. 2007. Indexing Strategies for Rapid Searches of Short Words in Genome Sequences. PLoS One 2, e579.

[12] Kent, W.J. 2002. BLAT-The BLAST-Like Alignment Tool. Gen. Res. 12, 656-664.

[13] Kirzhner, V., Bolshoy, A., Volkovich, Z., Korol, A., and Nevo, E. 2005. Large-scale genome clustering across life based on a linguistic approach. Biosystems 81, 208-222.

[14] McCreight, E.M.1976. A space-economical suffix tree construction algorithm. JACM 23, 262-272.

[15] Ning, Z., Cox, A., Mullikin, J. 2001. SSHAHA:A fast search method for large DNA databases. Gen. Res. 11, 1725-1729.

[16] Saitou, N and Nei, M. 1987. The neighbor-joining method: a new method for reconstructing phylogenetic trees. Mol. Bio. Evol. 4, 406-425.

[17] Torney, D.C., Burke, C., Davidson, D., Sirkin, K.M. 1990. Computation of d2:A measure of sequence dissimilarity, computers and DNA, SFI Studies in the sciences of complexity. Bell, G., Marr, T., editors, VII. New York, NY: Addison-Wesley.

[18] Wayner P. 2000. Compression algorithms for real programmers. Morgan Kaufman Publishers Inc., San Franciso, CA.