VERBUMCULUS User's Guide

Alberto Apostolico Stefano Lonardi
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907-1398

Version 1.12 - January 2001

Verbumculus is a collection of software tools for the efficient and fast detection and visualization of words that occur unusually often or seldom as substrings in biomolecular sequences. The inner core of VERBUMCULUS rests on subtly interwoven properties of statistics, pattern matching and combinatorics on words. These properties enable one to limit drastically and *a priori* the set of over- or under-represented candidate words of all lengths in a given sequence, thereby rendering it more feasible both to detect and visualize such words in a fast and practically useful way.

The VERBUMCULUS software is accessible at

`http://www.cs.purdue.edu/homes/stelo/Verbumculus/`

The executable code for Solaris (Sparc or Intel architectures) and Linux (Intel architectures) can be downloaded from the site and run locally on the user's machine. Alternatively, the web server is available at the same address.

1 Method

The central data structure of our software is the *suffix tree*. The suffix tree is a digital search tree that collects compactly all suffixes of a given text (see, e.g., [1, 9]) which allows many efficient, and often surprisingly simple and elegant solutions to problems on strings. While the leaves of the tree correspond one-to-one with the suffixes of the text, the internal nodes correspond one-to-one to all distinct words occurring in the text.

There are several algorithms for building the suffix tree in linear time with respect to the size of the input (see, e.g., [14, 19]), and using an appropriate encoding of the words on the edges of the tree, the space required for the suffix tree is also linear. The advantage of the tree over standard tables of words is twofold: (1) the suffix tree is time- and space-efficient resulting in fast analysis and low usage of memory, (2) the tree can be visualized compactly and therefore conveys effectively the information to the user.

For example, once the tree is built, one can easily compute the number of observed occurrences. With a simple traversal we collect the number of leaves in the subtrees rooted at each node of the tree. The number of leaves corresponds exactly to the number of observed occurrences of the word spelled out on a path from the root to that node.

It turns out that all the other parameters of interest can be computed efficiently on the suffix tree. The expected value and variance of *all* substrings in a given sequence can be computed and stored using the (optimal) quadratic-time given in [2] exploiting the combinatorial structure of the periods of the strings.

In [2] we also show that under several scores the candidates over- or under-represented oligonucleotides are restricted to a linear number, as opposed to the quadratic number of possible substrings. Based on this surprising fact, we have designed VERBUMCULUS to detect favored and unfavored word in our probabilistic framework in overall linear time and space.

The suffix tree for a single string can be easily extended to represent the suffixes of a set of strings. These suffixes are collected in a tree called *generalized* suffix tree [9]. Here we are interested in the number of "colors", that is the count of distinct sequences that contain a particular word. The computation of the colors uses the linear time algorithm by Hui [10].

2 Software Structure

VERBUMCULUS is composed by three modules: the tree builder VERBUM, the graph drawing program DOT, and the graphic interface TREEVIZ. The entire package consists of more than ten thousand lines of code.

VERB is written in C++ using the Standard Template Library (STL) [15] which should allow us easy portability under different platforms. The development of the software has been facilitated by the use of debuggers and a version control system (CVS). We have compiled the code, without any change, under Solaris and Linux.

VERB reads the input sequence(s) and the various parameters supplied by the user, and creates a (possibly pruned) suffix tree annotated with the score selected at the beginning by the user (see next Section for the list of available scores). The output is a text file representing the tree in the `dot` format (see below). VERB is particularly fast: although the time taken for the analysis depends on the score and the other parameters, it is usually in the order of a few seconds for the most common choices.

DOT is a graph drawing program developed by AT&T Labs as part of the GRAPHVIZ package [7, 8]. It reads graphs in the `dot` representation and outputs drawings in a dozen of formats, among which Postscript and GIF. The source code and binary executables for common platforms are freely available from the GRAPHVIZ site.

Finally, TREEVIZ is the graphical user interface that runs on the client size, and more specifically on the browser of the user. It is entirely written in Java, and uses the GRAPPA libraries by AT&T Labs [8].

A couple of thousands lines of PERL code glue everything together. PERL scripts generate the HTML for the input forms and control the execution of the various stages, handling exceptions and errors.

3 Command Line Usage

Users can download the binary executables VERB and DOT for Solaris (Intel and Sparc architecture) from VERBUMCULUS' site. The most common usage involves running VERB

```
:::::::::::: General
A          : read the files in FASTA format
N          : data is protein
I          : consider both strands (DNA input)
M <order>  : set the Markov order (default 0)
B <label>  : add a label to the .dot graph
S <file>   : use the statistics of <file> instead of [file]
z <type>   : z-score (1) counting occurrences: obs - exp
                     (2) counting occurrences: obs / exp
                     (3) counting occurrences: (obs - exp) / approx_var
                     (4) counting occurrences: (obs - exp) / var_complete
                     (5) counting occurrences: Trifonov
                     (6) counting occurrences: Tree2Tree (need ext model)
                     (7) counting sequences: Obs - Exp
                     (8) counting sequences: Obs / Exp
                     (9) counting sequences: (Obs - Exp)^2 / Exp
                    (10) counting sequences: Tree2Tree (need ext model)
:::::::::::: Filters
l <length> : filter out words shorter than <length> (default 2)
L <length> : filter out words longer than <length> (default 10)
x (value)  : filter out words having exp. # of occurrences < <value>
X (value)  : filter out words having score < <value> (default 0)
D "word"   : filter out words having "word" as substring
f          : do not exclude strings with symbols not in {a,t,c,g,u,A,G,T,C,U}
:::::::::::: Sliding window
w <window> : examines the specified window (1,2,...)
W <size>   : sets the window size
:::::::::::: Misc
I          : consider both strands
H <size>   : randomly shuffle the sequence (keeping constant the count
             of substrings of size up to <size>)
G <size>   : generate a random file of a given size reading
             the statistics from [file]
F <order>  : generate a Fibonacci strings of <order>-th order
```

Figure 1: Command-line options of VERB

on the file containing the sequences to produce the `dot` file, and then using DOT to create a graphical representation (PostScript, GIF, FrameMaker, etc.) of the tree.

Figure 1 shows the command line options of VERB. Options `-l` and `-L` define respectively the minimum and the maximum length of the motifs (default is 2 and 6 bps respectively). Flag `-X` sets the threshold on the score: words which score is lower than the threshold (in absolute value) are filtered out. Flag `-x` is used to mask words that have expectation lower than a given value. Flag `-D` tells VERB to reject words that contain a specified pattern. `-w` and `-W` are used to analyze a specific window, and they control the position and the size of the window respectively. `-B` allows the user to add a text label to the tree.

Flag `-z` controls the type of score we want to use to annotate the tree. We discuss first those based on a single sequence, where VERBUMCULUS supports the following six different

scores. For each oligonucleotide w we define[1]

- $z_1(w) = Obs(w) - Exp(w)$
- $z_2(w) = \frac{Obs(w)}{Exp(w)}$
- $z_3(w) = \frac{Obs(w)-Exp(w)}{\sqrt{\hat{Var}(w)}}$
- $z_4(w) = \frac{Obs(w)-Exp(w)}{\sqrt{Var(w)}}$
- $z_5(w) = \frac{Obs(w)-\mathcal{E}(w)}{\max\{\sqrt{\mathcal{E}(w)},1\}}$ as defined by Brendel *et al.*, where $\mathcal{E}(w)$ is the expected frequency of w based a Markov model of order $m-2$ (see [6] for details)
- $z_6(w)$ is computed by comparing the number of occurrences of the words of two suffix trees (requires the submission of an external model with the parameter `-S`)

where we denote by $Obs(w)$ the number of observed occurrences of w in the input sequence, by $Exp(w)$ the number of expected occurrences of w under a Bernoulli model, by $Var(w)$ the variance on the number of occurrences of w under the same model, and by $\hat{Var}(w)$ a first-order approximation of the true variance that is faster to compute. More specifically, if the size of the sequence is n, and w has length $m \leq (n+1)/2$ then

$$Exp(w) = (n - m + 1)p(w)$$

and

$$\begin{aligned} Var(w) = {} & Exp(w)(1 - p(w)) - p(w)^2(2n - 3m + 2)(m - 1) \\ & + 2p(w)\sum_{l=1}^{s}(n - m + 1 - d_l)\prod_{j=m-d_l+1}^{m} p(w_{[j]}) \end{aligned}$$

where $p(w)$ is the probability of occurrence of w and $\{d_1, d_2, \ldots, d_s\}$ are the *periods* of w (see [3, 2] for a detailed explanation).

Likewise, the analysis of the target sequence may proceed considering the sequence as a whole as well as by performing computations independently within a number of consecutive segments in a suitable covering of the input, and analyzing one such "window" at a time.

We now turn to the scores associated with frequencies defined on a set of sequences. In this class, the following four additional scores are supported.

- $z_7(w) = SObs(w) - SExp(w)$
- $z_8(w) = \frac{SObs(w)}{SExp(w)}$
- $z_9(w) = \frac{(SObs(w)-SExp(w))^2}{SExp(w)}$

[1] we denote with w a generic oligonucleotide that occurs in the genetic sequence we are going to analyze, and $w_{[i,j]}$ a substring of w from position i to position j

- $z_{10}(w)$ is computed by comparing the number of colors of the words of two suffix trees (requires the submission of an external model with the parameter `-S`)

where $SExp(w)$ is the expected number of sequences that contain at least one occurrence of w and $SObs(w)$ is the observed number of sequences that contain at least one occurrence of w. Given k sequences of size n_i for $i \in [1, k]$, Pesole *et al.* [16] define $SExp(w)$ as follows

$$SExp(w) = \sum_{i=1}^{k} \left(1 - e^{-\hat{q}_i(w)(n_i - m + 1)}\right)$$

where $\hat{q}_i(w)$ is the probability of occurrence of w in the i-th sequence.

The parameters of the model, i.e., the probability of the symbols, are estimated from the sequence itself. Flag `-S` allows the user to specify a different file from which VERBUMCULUS will compute parameters of the model.

The flag `-M` sets the order M of the Markov chain. The default is 0, which corresponds to the Bernoulli model. We remark that words smaller than $M + 2$ are not displayed at all, because the model would "predict" their statistics exactly and therefore they will never result surprising. Choosing higher Markov orders does not necessarly mean that you will get better results. As the size of the model grows its ability of prediction grows as well. Therefore there are less and less surprising words.

When the model is very precise on the reproduction of a particular set of observations but cannot capture the general characteristics of the source, we have a situation called *overfitting*. A related problem is that the bigger is M, the larger is the number of parameters to be estimated from the data, and the longer has to be the sequence in order to get statistically meaningful estimates.

Option `-A` is used to tell VERB that the input is in FASTA format. If `-A` is not specified, VERB will assume that the file contains only the sequences with no annotations. We strongly recommend, however, to use the FASTA format. An example of sequence in FASTA format is shown in Figure 2. Any line that begins with `>` or `;` is considered to be an annotation and disregarded. VERB internally converts the sequences to upper case, and resolves the IUPAC-IUB symbols by generating random substitutions (see Table 1). For example, if VERB encounters the character `R` it randomly substitutes the symbol with `A` or `G` with equal probability. Any symbol in the sequence that does not belong to the alphabet of Table 1 is discarded.

Finally, option `-N` tells VERB that the input is a set of proteins, instead of DNA. The alphabet follows the standard naming convention for the amino acids (see Table 2). In the current version, VERB internally converts the sequences to upper case, and reduce the alphabet to three symbols $\mathcal{H}$, $\mathcal{P}$, $\mathcal{C}$ which represent respectively hydrophobic, polar and charged amino acids or to four symbols $\mathcal{H}$, 0, +, − which denote respectively hydrophobic, not charged, positevely charged and negatively charged amino acids. The mappings between the twenty natural amino acids and the reduced alphabet of the electric charge is shown in the last column of Table 2. The rationale of reducing the alphabet size is twofold. One one hand, finding exactly conserved patterns of interesting size over an alphabet of 20 symbols proved unlikely. On the other, amino acids within certain groups share chemical and structural properties in such a way that they can actually swap without changing the function of the

```
>RTS2   RTS2 upstream sequence, from -200 to -1
TCTGTTATAGTACATATTATAGTACACCAATGTAAATCTGGTCCGGGTTACACAACACTT
TGTCCTGTACTTTGAAAACTGGAAAAACTCCGCTAGTTGAAATTAATATCAAATGGAAAA
GTCAGTATCATCATTCTTTTCTTGACAAGTCCTAAAAAGAGCGAAAACACAGGGTTGTTT
GATTGTAGAAAATCACAGCG
>MEK1   MEK1 upstream sequence, from -200 to -1
TTCCAATCATAAAGCATACCGTGGTAATTTAGCCGGGGAAAAGAAGAATGATGGCGGCTA
AATTTCGGCGGCTATTTCATTCATTCAAGTATAAAAGGGAGAGGTTTGACTAATTTTTTA
CTTGAGCTCCTTCTGGAGTGCTCTTGTACGTTTCAAATTTTATTAAGGACCAAATATACA
ACAGAAAGAAGAAGAGCGGA
>NDJ1   NDJ1 upstream sequence, from -200 to -1
ATAAAATCACTAAGACTAGCAACCACGTTTTGTTTTGTAGTTGAGAGTAATAGTTACAAA
TGGAAGATATATATCCGTTTCGTACTCAGTGACGTACCGGGCGTAGAAGTTGGGCGGCTA
TTTTGACAGATATATCAAAAATATTGTCATGAACTATACCATATACAACTTAGGATAAAA
ATACAGGTAGAAAAACTATA
```

Figure 2: The initial portion of a sample set of sequences in FASTA format

Symbol	*Meaning*	*Nucleic Acid*
A	A	Adenine
C	C	Cytosine
G	G	Guanine
T	T	Thymine
U	U	Uracil
M	A or C	
R	A or G	
W	A or T	
S	C or G	
Y	C or T	
K	G or T	
V	A or C or G	
H	A or C or T	
D	A or G or T	
B	C or G or T	
X or N	G or A or T or C	

Table 1: IUPAC-IUB symbols for nucleotide nomenclature

protein. Indeed, it is biologically more meaningful to compare amino acids based on their degrees of similarity rather than in terms of strict equality of residues.

The classification we choose, based on the electric charge of amino acids, is taken from [21, 18]. This is just one of the possible choices since some amino acids belong to more than one class (for example `H`) and some to none (for example `P`). Also, several classifications have appeared in the scientific literature, and they not always agree.

The execution of VERB on the file containing the sequence(s) under study creates a file with the suffix `dot` that holds the representation of the tree. For example, a run of VERB on the file of Figure 2 is shown in Figure 3.

The information printed by VERB on the standard output reflects the choices of the user and summarizes the statistics of the tree. In our example, we submitted 108 sequences for a total of 5407 base pairs. The complete tree of all the words up to size five is composed by 2935 nodes. The annotation took 0.03 seconds on our 300Mhz Intel/Solaris machine. The shape of the distribution of the scores is approximately Gaussian.

We will not describe here all the features of DOT. A user guide can be found at `http://www.research.att.com/sw/tools/graphviz/`. The standard usage of DOT to create a PostScript file is "`dot -Tps <filename>.dot -o <filename>.ps`". The picture of the tree for our running example is shown in Figure 4.

4 Web Server Usage

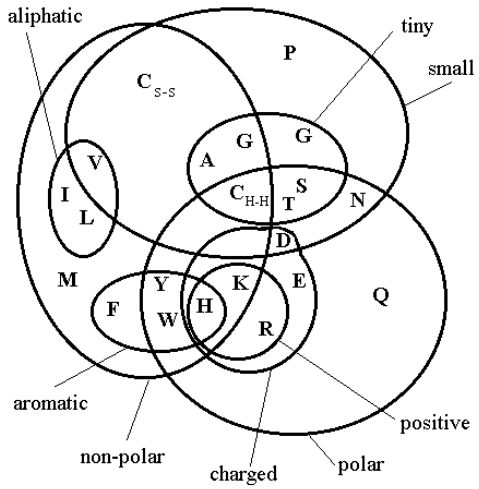
A portion of the main interface of the web server is shown in Figure 5. The user has the option to submit the input as a raw sequence of letters or in FASTA format. The input can be "pasted" into the window or uploaded to the server. In the case of analyzing long sequences, we advise the user to download the executables VERBUM and DOT and work locally, to avoid the overhead of network communication and the relative inefficiencies of Perl scripts and Java.

Various *filters* can be used to reduce the size of the output and mask the irrelevant information. The current filters offered by VERBUMCULUS support the selection of words in which:

- length is within a specified interval;
- z-score (in absolute value) is higher than a fixed threshold;
- expectation is higher that a given threshold (this is to filter out rare words);
- occurrence of a particular substring is forbidden.

These filters can be combined in any way to meet the user's needs.

Another important choice is the type of score used to annotate the tree. Our experience suggests that words which are highly significant in terms of scores based on counting occurrences are usually highly significant in terms of colors, and vice versa. As a result, it does not really matter which score one chooses – the significant words will be discovered. Unfortunately, signals that are more subtle could be missed using the simpler scores. Some preliminary comparative analyses on simulated sequences are reported in [4].



Name	*Symbol*	*hydrophobic*	*positive*	*negative*	*polar*	*charged*	*small*	*tiny*	*aromatic*	*aliphatic*	occurrences	VERB 3-class	VERB 4-class
Alanine	A, ALA	•					•	•			7.49%	$\mathcal{H}$	$\mathcal{H}$
Arginine	R, ARG		•		•	•					5.22%	$\mathcal{C}$	+
Asparagine	N, ASN				•		•				4.53%	$\mathcal{P}$	0
Aspartic acid	D, ASP			•	•	•	•				5.22%	$\mathcal{C}$	−
Cysteine	C, CYS	•					•				1.82%	$\mathcal{P}$	0
Glutamic acid	E, GLU			•	•	•					6.26%	$\mathcal{C}$	−
Glutamine	Q, GLN				•						4.11%	$\mathcal{P}$	0
Glycine	G, GLY	•					•	•			7.10%	$\mathcal{H}$	0
Histidine	H, HIS	•	•		•	•			•		2.23%	$\mathcal{P}$	+
Isoleucine	I, ILE	•								•	5.45%	$\mathcal{H}$	$\mathcal{H}$
Leucine	L, LEU	•								•	9.06%	$\mathcal{H}$	$\mathcal{H}$
Lysine	K, LYS	•	•		•	•					5.82%	$\mathcal{C}$	+
Methionine	M, MET	•									2.27%	$\mathcal{H}$	$\mathcal{H}$
Phenylalanine	F, PHE	•							•		3.91%	$\mathcal{H}$	$\mathcal{H}$
Proline	P, PRO						•				5.12%	$\mathcal{H}$	$\mathcal{H}$
Serine	S, SER				•		•	•			7.34%	$\mathcal{P}$	0
Threonine	T, THR	•			•		•				5.96%	$\mathcal{P}$	0
Tryptophan	W, TRP	•			•				•		1.32%	$\mathcal{P}$	$\mathcal{H}$
Tyrosine	Y, TYR	•			•				•		3.25%	$\mathcal{P}$	0
Valine	V, VAL	•					•			•	6.48%	$\mathcal{H}$	$\mathcal{H}$

Table 2: Amino acids naming and classification (table based on [21], picture based on [18]). $\mathcal{H}$, $\mathcal{P}$, $\mathcal{C}$, 0, +, and − denote respectively hydrophobic, polar, charged, not charged, positevely charged and negatively charged amino acids. Occurence statistics were compiled using the NCBI database

```
~/data> verb -X 9 -L 5 -z 3 EarlyI.100.fasta
Reading data file EarlyI.100.fasta
Total size data set 5407
Number of sequences data set 108
Min_length 2
Max_length 5
Zoom 10
Score 3
Threshold 9
Building Suffix Tree
Annotating the tree
Size of the dataset tree 2935 nodes
Seconds 0.03
Writing EarlyI.100.fasta.dot
--------------------------------
        max: 16.4924
        min: -0.840909
        mean: 3.45274
        std dev: 2.6454
------- HISTOGRAM [-9, 9] ------
        <-9             0
        [-9,-5]         0
        [-5,-1]         0
        [-1,0]          14
        [0,1]           136
        [1,2]           248
        [2,3]           208
        [3,4]           192
        [4,5]           122
        [5,6]           88
        [6,7]           64
        [7,8]           37
        [8,9]           31
        >9              35
--------------------------------
* Printed 35 nodes in the .dot file
```

Figure 3: A run of VERB on a sample FASTA sequence

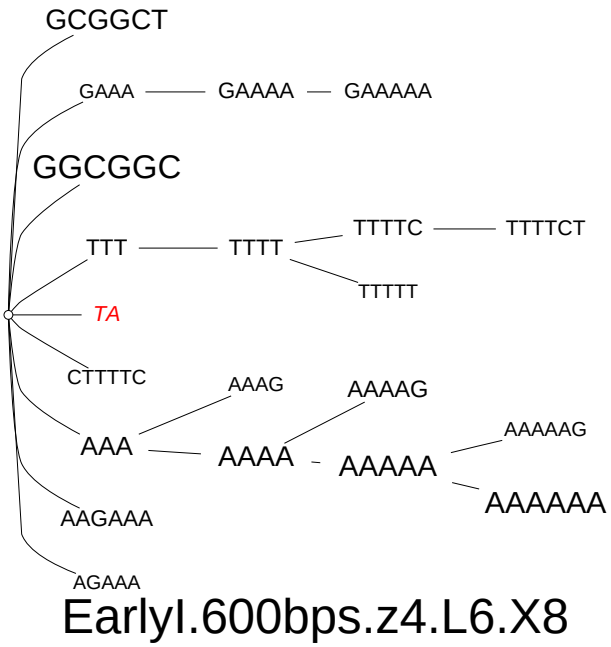


Figure 4: DOT output on a sample set of sequences

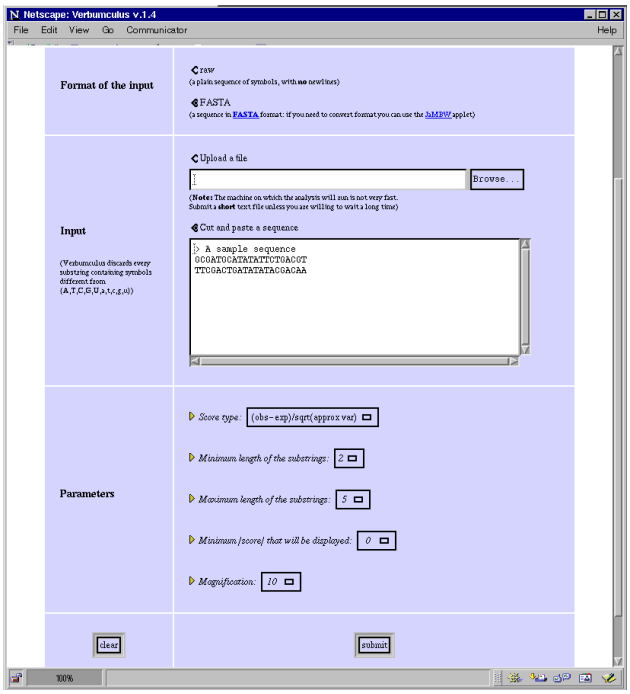


Figure 5: Web interface of VERBUMCULUS

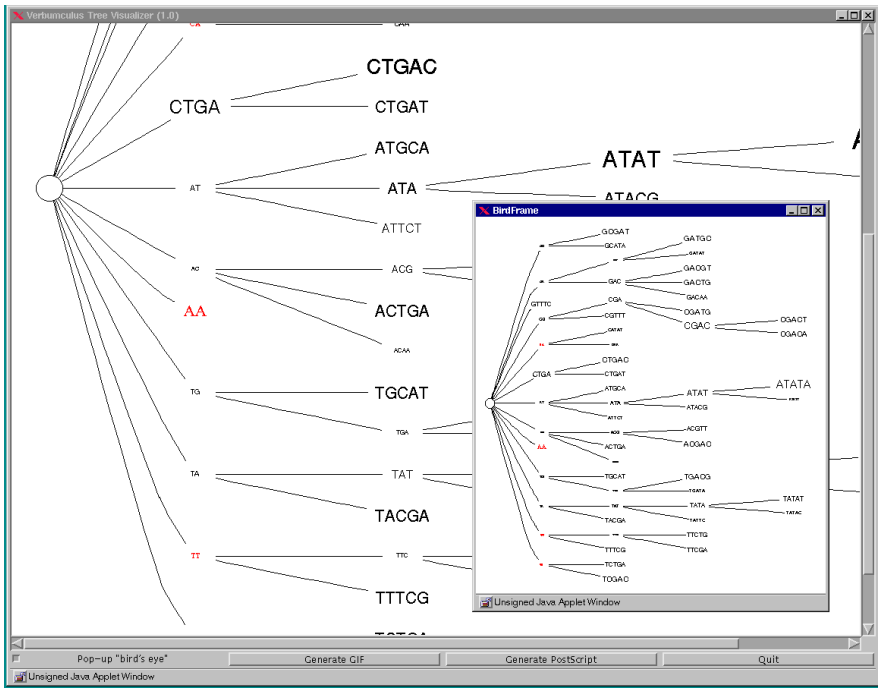


Figure 6: TreeViz output on a sample sequence

For performance reasons, we have limited the visualization of TreeViz to 100 nodes: when the tree becomes bigger, Verbumculus generates a Postscript file with the drawing of the tree. If the user wants to take advantage of the interactive facilities of TreeViz he will have to increase the effectiveness of the filters in order to produce a smaller tree.

Once TreeViz has drawn a tree, the user can wander about it. The magnitude of a score value is transduced through font size, in the sense that for every word w, the higher the absolute value of the score of w, the bigger the font used to represent w. Words with a negative score are, in addition, printed in red italics (see Figure 6). At any time the user can click on a word and get information about the number of occurrences, the expected number of occurrences, and the value of the score. Along with these, a representation of the occurrences in the original sequence is produced (see Figure 8).

Figure 8 refers to the output of the analysis of a set of sequences. The two panels show the usual information about two words picked from the tree, along with their positions inside the submitted sequences. The original set of sequences is displayed in the window where the occurrences of the selected words are highlighed in red. The first row shows the position of each base relative to the beginning the sequence.

Since the tree can be fairly big, TreeViz offers the option to get an overall picture of the tree by clicking on the "bird's eye" button (the small window in Figure 6). Also, TreeViz can generate drawings in Postscript or GIF that can be saved on the user's machine for further scrutiny.

An alternate viewer that uses hyperbolic geometry has been added in the latest version of TreeViz. Figure 9 shows a view of a pruned suffix tree projected on a sphere. The software is an adapted version of the visualization applets by A. Robinson, based on the work by

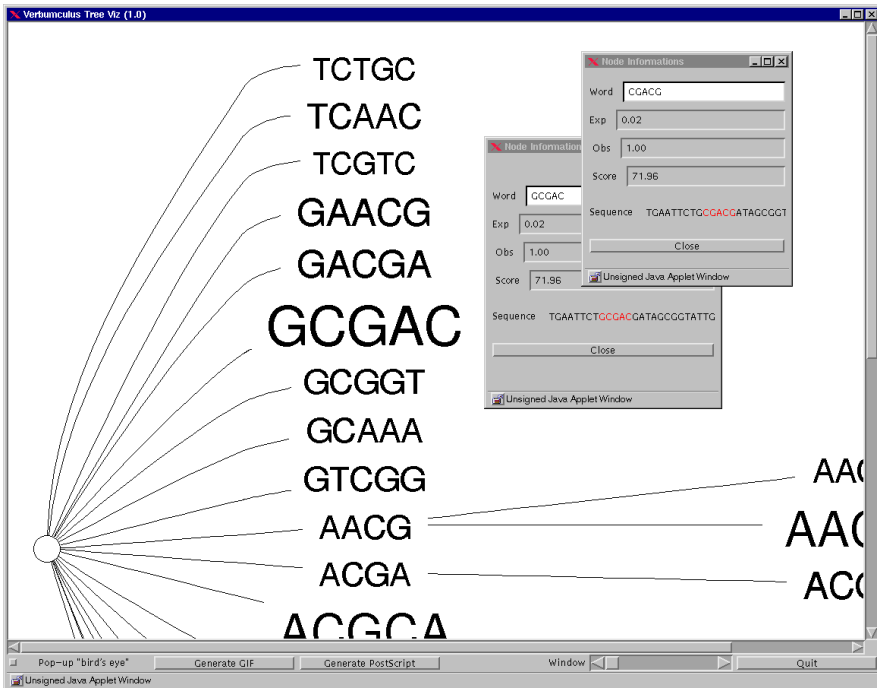


Figure 7: TreeViz output in the case of sliding window

Figure 8: TreeViz output in the case of a set of sequences: note the panels showing the positions of the selected word in the submitted sequences

Lamping *et al.* [11, 12]. The idea is to lay out the tree uniformly on the hyperbolic plane and map the plane onto a circular display region. The projection onto the disk provides a natural mechanism for assigning more space to a portion of the hierarchy while still embedding it in a much larger context. Change of focus is accomplished by translating the structure on the hyperbolic plane, which allows a smooth transition without compromising the presentation of the context.

The beauty of such visualization technique is that it allows the viewer to keep a global perspective of the data (i.e., the context) while examining selected regions in detail (i.e., to focus in). In the hyperbolic projection, as one moves away from the origin, the distance increases, but not in a linear fashion. Thus, the perimeter of the projection actually corresponds to being at infinity and therefore all space may be shown in the projection.

The visualization of the suffix tree decorated with z-scores in the hyperbolic space poses some additional problems. The font size of each word conveys the score, and therefore must be maintained. However, when the words approach the boundary of the hyperbolic plane they could become too small to be seen. In this case, they are drawn with a fixed size font in grey (instead of black, if over-represented) or orange (instead of red, if under-represented). The overall effect is to "dim" words close to the boundary to avoid the cluttering, but at the same time to keep track of them.

The hyperbolic viewer support several interactive facilities:

- dragging the pointer on the display will cause the scene to be translated;
- holding down "shift" while dragging on the display will cause the scene to be rotated about the root node;
- if the "`+`" and "`-`" keys are pressed during the display of the tree, the number of levels of the tree shown will be increased and decreased respectively;
- if the "`=`" is pressed during the display of the tree, the number of levels of the tree shown will reset to the default.

The web server of VERBUMCULUS is available at the address `http://www.cs.purdue.edu/homes/stelo/Verbumculus/`

References

[1] APOSTOLICO, A. The myriad virtues of suffix trees. In *Combinatorial Algorithms on Words*, A. Apostolico and Z. Galil, Eds., vol. 12 of *NATO Advanced Science Institutes, Series F*. Springer-Verlag, Berlin, 1985, pp. 85–96.

[2] APOSTOLICO, A., BOCK, M. E., LONARDI, S., AND XU, X. Efficient detection of unusual words. *J. Comput. Bio. 7*, 1/2 (Jan. 2000), 71–94. Also TR-97-050, Dept. of Computer Sciences, Purdue University.

[3] APOSTOLICO, A., BOCK, M. E., AND XU, X. Annotated statistical indices for sequence analysis (invited paper). In *Compression and Complexity of Sequences* (Positano, Italy, 1998), B. Carpentieri, A. De Santis, U. Vaccaro, and J. Storer, Eds., IEEE Computer Society Press, pp. 215–229.

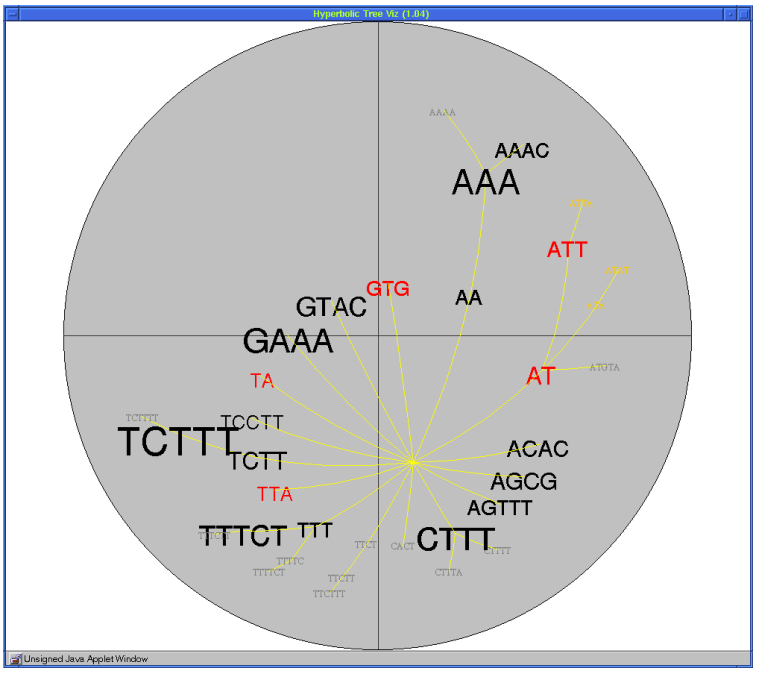


Figure 9: Hyperbolic Tree Viewer

[4] Apostolico, A., Gong, F., and Lonardi, S. Verbumculus and the discovery of unusual oligonucleotides. Tech. Rep. TR-00-022, Dept. of Computer Sciences, Purdue University, 2000.

[5] Bailey, T. L., and Elkan, C. Unsupervised learning of multiple motifs in biopolymers using expectation maximization. *Machine Learning 21*, 1/2 (1995), 51–80.

[6] Brendel, V., Beckmann, J. S., and Trifonov, E. N. Linguistics of nucleotide sequences: Morphology and comparison of vocabularies. *J. Biomol. Struct. Dynamics 4*, 1 (1986), 11–21.

[7] Gansner, E. R., Koutsofios, E., North, S., and Vo, K.-P. A technique for drawing directed graphs. *IEEE Trans. Software Eng. 19*, 3 (1993), 214–230.

[8] Gansner, E. R., and North, S. An open graph visualization system and its applications to software engineering. *Softw. Pract. Exp. 30* (2000), 1203–1233.

[9] Gusfield, D. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge University Press, 1997.

[10] Hui, L. C. K. Color set size problem with applications to string matching. In *Proceedings of the 3rd Annual Symposium on Combinatorial Pattern Matching* (Tucson, AZ, 1992), A. Apostolico, M. Crochemore, Z. Galil, and U. Manber, Eds., no. 644 in Lecture Notes in Computer Science, Springer-Verlag, Berlin, pp. 230–243.

[11] Lamping, J., and Rao, R. Laying out and visualizing large trees using a hyperbolic space. In *Proceedings of the ACM Symposium on User Interface Software and Technology* (1994), Visualization I, pp. 13–14. TechNote.

[12] Lamping, J., and Rao, R. The hyperbolic browser: A focus+context technique for visualizing large hierarchies. *J. Visual Languages and Computing 7*, 1 (Mar. 1996), 33–55.

[13] Leung, M. Y., Marsh, G. M., and Speed, T. P. Over and underrepresentation of short DNA words in herpesvirus genomes. *J. Comput. Bio. 3* (1996), 345–360.

[14] McCreight, E. M. A space-economical suffix tree construction algorithm. *J. Assoc. Comput. Mach. 23*, 2 (Apr. 1976), 262–272.

[15] Musser, D. R., and Stepanov, A. A. Algorithm-oriented generic libraries. *Softw. Pract. Exp. 24*, 7 (July 1984), 623–642.

[16] Pesole, G., Prunella, N., Liuni, S., Attimonelli, M., and Saccone, C. Wordup: An efficient algorithm for discovering statistically significant patterns in DNA sequences. *Nucleic Acids Res. 20*, 11 (1992), 2871–2875.

[17] Schbath, S. An efficient statistic to detect over- and under-represented words in DNA sequences. *J. Comput. Bio. 4* (1997), 189–192.

[18] Taylor, W. R. The classification of amino acid conservation. *J. Theor. Biol. 119* (1986), 205–218.

[19] Ukkonen, E. On-line construction of suffix trees. *Algorithmica 14*, 3 (1995), 249–260.

[20] van Helden, J., André, B., and Collado-Vides, J. Extracting regulatory sites from the upstream region of the yeast genes by computational analysis of oligonucleotides. *J. Mol. Biol. 281* (1998), 827–842.

[21] Zvelebil, M. J., Barton, G. J., Taylor, W. R., and Sternberg, M. J. E. Prediction of protein secondary structure and active sites using the alignment of homologous sequences. *J. Mol. Biol. 195* (1987), 957–961.