PROCEEDINGS

International IEEE Symposium on INTELLIGENCE in NEURAL and BIOLOGICAL SYSTEMS

May 29-31, 1995

Herndon, VA

SPONSORED BY IEEE COMPUTER SOCIETY TECHNICAL COMMITTEE ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE

Establishing a Framework for Comparative Analysis of Genome Sequences[1]

Arvind K. Bansal
Department of Mathematics and Computer Science
Kent State University
Kent, OH 44242 - 0001, USA
E-mail: arvind@mcs.kent.edu

Abstract

This paper describes a framework and a high level language tool for comparative analysis of genome sequence alignment. The framework integrates the information derived from multiple sequence alignment and phylogeny to derive new properties about homologous sequences. Multiple sequence alignments are treated as an abstract data type. Abstract operations have been described to manipulate a multiple sequence alignment, and to derive mutation related information from a phylogenetic tree by superimposing parsimony. The framework has been applied to derive constrained columns which exhibit evolutionary pressure to preserve a common property in a column despite mutation. A Prolog tool based on the framework has been implemented and demonstrated on alignments containing 3000 sequences and 3904 columns.

Keywords: Artificial Intelligence, Genome Sequence, Prolog, Sequence Alignment, Parsimony, Phylogeny

1. Introduction

The interpretation of genetic sequence data represents one of the most interesting puzzles facing scientists today. This will almost certainly be the year in which we gain access to complete microbial genomes, and it is likely that hundreds of complete genomes will be completed during the coming decade. Each such genome represents the complete blueprint of a living organism; that is, it holds the information characterizing a life form. Deepening our understanding of such genomes amounts to advancing our grasp of life itself. Such understanding will almost certainly drive many applications in areas ranging from medicine and agriculture to bio-remediation of the environment.

One fundamental approach towards understanding genetic sequence data is based on comparative analysis which uses phylogeny - hypothesizing an evolution tree having common ancestral sequences. For example, the corresponding gene from a mouse and from a human look quite similar because they may have evolved from the same common ancestral genome. An essential step in performing comparative analysis is to form an alignment of similar sequences which are believed to be homologous - having a common ancestry. The alignment amounts to an assertion of correspondence between the characters in the sequences. For example, consider the following fragment of similar protein sequences from five distinct organisms:

```
>Tuna
VQKCAQCHTVENGGKHKVGPNLWGLFGRKTGQAEGYSYTD
>Hippopotamus
VQKCAQCHTVEKGGKHKTGPNLHGLFGRKT GQS PGFSYTD
>Horse
VQKCAQCHTVEKGGKHKTGPNLHGLFGRKT GQAPGFTYTD
>Human
I MKCSQCHTVEKGGKHKTGPNLHGLFGRKT GQAPGY SYTA
>Yeast
KTRCAQCHT IE AGGPHKVGPNLHG IF SRHS GQAEG - SYTD
```

What is being shown here is the relationship between similar pieces of genetic material from five distinct organisms. The sequences have been "aligned"; that is, they have been positioned so that the characters in each column are believed to be derived from a single character in some common ancestral sequence. In the case of the last sequence, this required introducing a '-' as a spacer to maintain the hypothesized correspondence.

The construction, maintenance, and interpretation of such alignments will play a central role in the interpretation of genomes. This paper describes a framework and the corresponding toolkit of Prolog routines that allows users to gain efficient access to such alignments in a high-level language, and to hypothesize the ancestral characters and mutation events using a parsimonious analysis of phylogeny. This paper also describes an application of the framework to derive

[1] This research was supported in part by DOE through Argonne National Laboratory's summer research faculty program

constrained columns which exhibit evolutionary pressure to preserve common properties despite mutations.

The major contributions of this paper are

1. the development of a generic high level language library for complex analysis of multiple sequence alignment and phylogenetic tree;
2. the derivation of group of amino acids in homologous proteins which share some important common properties despite their differences; and
3. the identification of constrained columns (in an alignment of homologous sequences) which conserve some common properties despite mutations resulting into different types of amino-acids in the column.

Further application of the framework, when combined with heuristic biochemical knowledge [2], will facilitate understanding of structure-function relationship of genome sequences.

The rest of the paper is as follows. Section 2 discusses some background necessary for computer scientists to understand standard biological terminology, and to introduce definitions used to explain our concept. Section 3 discusses the multiple sequence alignment as an abstract data type. Section 4 describes parsimonious labeling of a phylogenetic tree. Section 5 describes a heuristics to derive constrained columns which preserve a common property despite independent mutations. Section 6 briefly describes other applications of the tool. The last section concludes the paper.

2. Background and Definitions

In this section, some basic terminology are defined for genome analysis [1, 7] and Prolog [8]- the target language for the development of analysis tools. Familiarity with terminology related to amino-acids and their classification is assumed [6, 9].

For representation convenience, sequences are represented within square brackets [...], a generic value is represented within angular brackets < ... >, elements of a set are represented within curly brackets {...}, union of sets is denoted by $\cup$, intersection is denoted by $\cap$, difference of two sets is denoted by -, membership is denoted by $\in$, forall is denoted by $\forall$, and abbreviated characters for various molecules are represented within parenthesis.

2.1 Terminology for Genome Analysis

The genome of an organism is encoded within molecules of DNA. A molecule of DNA is a sequence of four nucleotides adenine ('A'), thymine ('T'), guanine ('G'), and cytosine ('C'). For our purposes, a DNA molecule is represented by a sequence of characters from the alphabet '{A', 'C', 'G', 'T'}.

RNA molecules are very similar to DNA molecules. The major difference relevant to this discussion is that uracil occurs rather than thymine. Thus, may be represented by a sequence of characters from the alphabet {'A', 'C', 'G', 'U'}. The generated tools process both DNA and RNA sequence data. Hence, it is convenient to speak of nucleotides as characters from the alphabet {'A', 'T', 'G', 'U', 'C'} denoted by $\mathcal{N}$.

A protein is a sequence of different types of molecules collectively known as amino-acids. More commonly, Alanine (A), arginine (R), asparagine (N), aspartic acid (D), cysteine (C), glutamic acid (E), glutamine (Q), glycine (G), histidine (H), isoleucine (I), leucine (L), lysine (K), methionine (M), phenylalanine (F), proline (P), serine (S), threonine (T), tryptophan (W), tyrosine (Y), and valine (V) constitute the sequence of protein. The alphabet {'A', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'K', 'L', 'M', 'N', 'P', 'Q', 'R', 'S', 'T', 'V', 'W', 'Y'} is denoted by $\mathcal{A}$.

A genome sequence, denoted by $\mathcal{S}$, is of the form [s_1, ... s_N] ($1 \le I \le N$) where $s_I \in \mathcal{N}$ for DNA or for RNA and $s_I \in \mathcal{A}$ for a protein. Two genetic sequences are similar under some scoring scheme, if there is a significant match between them after limited shifting of characters. Sequence alignment is the process of aligning similar sequences together in a way that asserts a correspondence between characters that are thought to derive from a common ancestral sequence. Aligning a set of sequences will require the introduction of spacing characters which are referred to as *indels*. If the aligned sequences did indeed derive from a common ancestral sequence, then indels represent evolutionary events in which characters were either inserted or deleted.

A valid nucleotide alignment character is an element of $\mathcal{N} \cup$ {'-'}. A valid protein alignment character is an element of $\mathcal{A} \cup$ {'-'}. A multiple sequence alignment, denoted by $\mathcal{M}$ is of the form [<Label$_1$, $\mathcal{S}^A_1$>, ..., <Label$_M$, $\mathcal{S}^A_M$>]. Each $\mathcal{S}^A_I$ ($1 \le I \le M$) is a transformed version of $\mathcal{S}_I$ such that length($\mathcal{S}^A_I$) = length($\mathcal{S}^A_J$) and every element of $\mathcal{S}^A_I$ is a valid nucleotide alignment character; or every element is a valid protein alignment character. Each sequence in the alignment must have an associated unique identifier. A multiple sequence alignment is represented as two dimensional matrix such that each row represents $\mathcal{S}^A_I$. Each column of $\mathcal{S}^A_I$ is denoted by C_J where $1 \le J \le$ length($\mathcal{S}^A_I$). Alternately, a multiple sequence alignment is represented as an association of columns, denoted by $C_1 \oplus ... \oplus C_N$ (where N = length($\mathcal{S}^A_I$)) such that picking character at index I from every column gives I_{th} sequence $\mathcal{S}_I$. For convenience, multiple sequence alignments will be referred to as alignments. A sub-alignment is a portion of alignment which is derived by selecting a subset of the rows and a subset of the columns.

In a conserved column the cumulative occurrence of other characters is insignificant compared to the occurrences of the most occurring character. A set of columns $\{C_M, ..., C_N, ..., C_P\}$ are strongly related if the cumulative occurrence of other sub-sequences of characters in the set are insignificant compared to few most occurring sub-sequences.

A control-sequence is used to mark columns in an alignment. Markers are assigned to the subset of columns corresponding to non-indels in the control-sequence. Thus, one might speak of "E.coli position 413" meaning "the column in the alignment in which the 413_{th} nonindel character in the sequence with identifier E.coli occurs".

Alignments are stored in files using FASTA format. In this format, the file contains a sequence of alignment entries. Each alignment entry begins with a line containing '>' immediately followed by the identifier of the sequence. The remainder of the alignment entry is composed of one or more lines containing the characters of the sequence itself.

```
1          11          21   26   31          41          51      57    61       69
> P25112
MDTSRVQPIK LARVTKVLGR TGSQGQCTQV RVEFMDDTSR SIIRN VKGPV REGDVLTLLE SEREARRLR
>P33285
--MDTKTPVT LAKVI KVLGR TGSRGGVTQV RVEF LEDTTR TIVRNVKGPV REGD ILVLME SEREARRLR
>P33286
--MDS KTPVT LAKV I KVLGR TGSRGGVTQV RVEF LEDTTR TIVRNVKGPV REGD ILVLME SEREARRLR
>P02380
--MDNKTPVT LAKV I KVLGR TGSRGGVTQV RVEF LEDTSR TIVRNVKGPV REND ILVLME SEREARRLR
>P34789
----MDSQIK HAVVVKVMGR TGSRGQVTQV RVKF TDSD- R FIMRNVKGPV REGDVLTLLE SEREARRLR
```

Figure 1: A multiple sequence alignment in FASTA format

A phylogenetic tree depicts a hypothesized evolutionary history of the objects at the leaves of the tree. If the leaves are organisms, then the tree shows a set of ancestral organisms, along with the points at which distinct lineage's diverged. If the leaves are homologous sequences, then the nodes of the tree correspond to ancestral sequences. This paper is concerned with phylogenetic trees in which the leaves are sequences from an alignment. A phylogenetic tree asserts a hypothesized evolutionary history for the sequences of an alignment that is the leaves of the tree are identifiers from an alignment. Now consider the characters which occur in a specific column C_I of the alignment, and suppose that the characters of C_I are added to the appropriate leaves of the tree. What can be inferred about the ancestral values corresponding to the column? There are different approaches to inferring ancestral values. One approach is to label the internal nodes - assign values to the ancestral sequences- in a way that minimizes the number of evolutionary events required to explain the values of the leaves; that is, one chooses a parsimonious labeling. Often, there are several equivalent parsimonious labeling.

Two genetic sequences which have been derived by evolution from a common ancestral sequence are called homologous. Detection of homology is normally achieved by first recognizing similarity using some algorithmic process and then inferring homology by weighing different forms of evidence.

To make this all a bit more concrete, let us look at some examples.

Example 1

A protein alignment is shown in Figure 1. The sequence identifiers are P25112, P33285, P33286, P02380, P34789. Each block of protein contains ten characters. The alignment contains 5 rows and 69 columns. The columns C_{26}, C_{57} and c_{69} are equal to $[Q, G, G, G, Q]$, $[R, R, R, R, R]$, and $[V, I, I, I, V]$ respectively. The column C_{69} is a conserved column, and the columns C_{26} and C_{57} are strongly related.

The corresponding phylogenetic tree is given in Figure 2. The leaf nodes are marked with sequence identifiers P25112, P34789, P33285, P33286, P02380; internal nodes are labeled as N0, N1, and N2, and the arcs are marked with the numeric values corresponding to approximate evolutionary distances such as .11540 (between P25112 and N1) and .16202 (between N1 and N0).

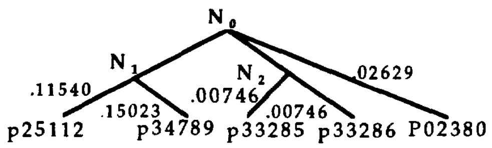

Figure 2: A phylogenetic tree for the alignment

2.2 Prolog

Prolog is a popular high level declarative AI language [8] based on the logic programming paradigm. Prolog facilitates programming due to the natural manipulation of high level data structures such as lists, sets, and trees. Prolog supports rapid prototyping, heuristic reasoning, and an elegant programming style. However to reduce any performance penalty associated with use of high level language, a small kernel of functions (necessary for overall performance) have been implemented in C. The result is a high level language library which facilitates , rapid code development, and reasonable performance.

3. Alignment as Abstract Data Type

In this section alignment has been described as an abstract data types, and a kernel set of abstract commands have been described. These abstract commands (or functions) have been used to read large alignments (often alignments do now contain millions of characters), verify alignments for the presence of invalid characters, perform statistical computation on alignments, and integrate analysis of the alignment with processing of an associated phylogenetic tree. Sequences from different genome banks may have non-standard characters: characters representing ambiguity between amino-acids at a specific position, characters representing unidentifiable amino-acid at a specific position, and erroneous characters introduced during information processing. This necessitates the need for validating the characters in a sequence.

During alignment analysis there is a need for conversion from DNA sequence to RNA sequence and vica versa, and conversion of ASCII representation to internal representation for fast counting and statistical computation.

Cumulative count of different valid characters in a column are needed for statistical computation, to identify a set of conserved columns, and to identify strongly related columns.

During sequence alignment, the actual characters in a sequence are shifted due to the presence of indels. A control sequence is used to refer to positions in an alignment: a position in the alignment is referred as a pair of the form (*position of the last valid character of control sequence, number of indels following the character*).:

3.1 Abstract Operations on Alignment

read_alignment(in: *File*; out: *Alignment*)
reads an alignment from File (in FASTA format), constructs an internal representation, and binds the alignment to a "handle" for easy access.

display_alignment (in: *Alignment*)
displays an alignment.

size_of_alignment(in: *File*; in-out: *Rows, Columns*)
reads a multiple sequence alignment from a file (in FASTA Format), and returns the number of sequences and number of columns in the alignment.

alignment_columns(in: *Alignment*, in-out: *Column*)
returns (or verifies) the number of columns in an alignment

alignment_rows(in: *Alignment*; in-out: *Rows*)
returns (or verifies) the number of rows in an alignment

alignment_type(in: *Alignment*, in-out: *Type*)
returns (or verifies) the type of an alignment.

save_alignment(in: *Alignment*, out: *File*)
saves an alignment in *File* using FASTA format.

free_alignment (out: *Alignment*)
removes an alignment from the environment.

copy_alignment(in: *Alignment*, out: *Alignment_copy*)
creates a new copy of an alignment, and is useful when the original version has to be preserved.

normalize_nucleotide_alignment(in-out: *Alignment*)
converts ASCII version of valid nucleotide characters of a nucleotide alignment to internal representation for efficient statistical computation.

normalize_protein_alignment(in-out: *Alignment*)
. converts ASCII version of valid protein characters of a protein alignment to internal representation for efficient statistical computation

transform_to_ascii(in: *Internal_chars*, out: *Ascii_chars*)
transforms the internal representation of characters to ASCII representation for user interface.

dna_to_rna(in-out: *Alignment*)
converts all occurrences of 'T' by 'U'.

rna_to_dna(in-out: *Alignment*)
converts all occurrences of 'U' by 'T'.

validate_nucleotide_alignment(<u>in:</u>*Alignment*, <u>out:</u> *Chars*)
returns the set of invalid nucleotide characters in an alignment.

validate_protein_alignment(<u>in:</u> *Alignment*, <u>out:</u> *Chars*)
returns the set of invalid protein characters in an alignment.

column_charcaters(<u>in:</u> *Alignment*, *Column*; <u>in-out:</u> *Sequence*)
returns (or verifies) the sequence of characters for the specified column of an alignment.

row_charcaters(<u>in:</u> *Alignment*, *Row*; <u>in-out:</u> *Sequence*)
returns (or verifies) the sequence of characters for the specified row of an alignment.

sub_alignment(<u>in:</u>*Alignment*,*Rows*,*Columns*;<u>out:</u>*Sub_align*)
returns a sub-alignment by selecting the set of sequences in an alignment, and then selecting the characters from the given columns. A sub-alignment is also treated like an alignment, and all abstract operations are applicable on sub-alignments.

identifier_to_row(<u>in:</u> *Alignment*; <u>in-out:</u> *Identifier*, *Row*)
returns (verifies) the row number for an identifier in an alignment;

row_to_identifier(<u>in:</u> *Alignment*; <u>in-out:</u> *Identifier*, *Row*)
returns (verifies) the identifier associated with a row in an alignment

row_identifiers(<u>in:</u> *Alignment*; <u>out:</u> *Identifier_list*)
returns the list of identifiers for all the sequences in an alignment. The function is used to test whether the given sequence is a member of an alignment.

char(<u>in:</u> *Alignment*, *Row*, *Column*; <u>in-out:</u> *Character*)
returns (or verifies) a character present in a given row and column of an alignment.

set_char(<u>in:</u> *Row*, *Column*, *Character*; <u>in-out:</u> *Alignment*)
writes a character in the position identified by row and column.

Count_sequence_chars(<u>in:</u> *Alignment*, *Rows*; <u>out:</u> *Counts*)
returns a sequence of pairs of the form [$< s_1, c_1 >$, ..., $< s_N, c_N >$] for the given set of sequences where s_I is a valid nucleotide (or protein) character for nucleotide (or protein) sequence, and c_I is the count.

count_column_chars(<u>in:</u> *Alignment*, *Columns*; <u>out:</u> *Counts*)
returns a sequence of pairs of the form [$< s_1, c_1 >$, ..., $< s_N, c_N >$] for the given set of columns where s_I is a valid nucleotide (or protein) character for nucleotide (or protein) sequence, and c_I is the count. This function is used to identify conserved columns and strongly related columns in an alignment.

map_control_seq(<u>in:</u> *Control_sequence*; <u>in-out:</u> *Alignment*)
builds up the markers for each column position, and stores the markers in a vector of triples of the form < Column position, markers>.

column_reference(<u>in:</u> *Align*, *Column*; <u>in-out:</u> *Index* + *Offset*)
returns the marker for the a column in the form (index of last valid character of control sequence + Offset) where offset is the number of indels after the last valid character.

4. Labeling of a Phylogenetic Tree

Parsimony labeling has been superimposed on phylogenetic tree to derive the possible set of characters at the ancestral nodes, and to derive mutation events during evolution. Parsimony has been imposed using following two principles:

1. Majority rule is used to identify the value of the parent for the given values of children,
2. Probability of mutation is equal for every child.

These two principles ensures minimum cumulative mutation count, and are available in literature [3, 4].This paper describes a two pass algorithm for labeling: the first phase generates set of all possible labels in a bottom up manner using majority rule at every internal node; the second pass uses top down approach to restrict to one value using a random selection among equi-probable characters, and identifies the mutation arcs by comparing dissimilar values at parent node and children nodes.

Value restriction during second pass is summarized in Table 1. For convenience, restricted values of the parent nodes are denoted as V^P, the value of the child node derived during the first pass is denoted as V^C_1 and the restricted value of child node during second phase is denoted as V^C_2. Note that V^C_1 could either be a single value or a set of equi-probable values.

	V^C_1 Type	Relation	V^C_2	Mutated
1	single	$V^P = V^C_1$	V^P	no
2	single	$V^P \neq V^C_1$	V^C_1	yes
3	set	$V^P \in V^C_1$	V^P	no
4	set	$V^P \notin V^C_1$	$\in V^C_1$	yes

Table 1: Rules for parsimonious labeling

Example 2:
Figure 3 demonstrates the parsimonious analysis of the

phylogenetic tree given in Figure 2 for the column 57. The result after Pass I is given in Figure 3a, and both the possibilities of parsimonious labeling after pass 2 is given in Figures 3b and 3c. The arcs with circle mark the mutation events.

4.1 Abstract Commands for Labeling

parsimonious_labels(in: *Tree, Alignment, Column*; out: *Labeled_tree*)

builds a labeled phylogenetic tree for a column of an alignment and the corresponding phylogenetic tree. Every internal node of the labeled tree carries the set of equi-probable values derived after pass I, randomly selected value derived during pass II, and total mutation count (based on the selected value) for the subtree rooted at the node.

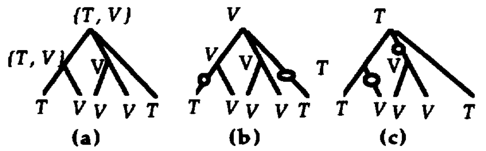

Figure 3: Parsimonius labeling

min_mutations(in: *Labeled_tree*, out: *Mutations*)

returns the total count of mutations for the labeled phylogenetic tree.

pars_values(in: *Labaled_tree*, out: *Node_labels*)

returns the set of equi-probable values (derived during Pass I) for an internal node of the labeled phylogenetic tree.

5. Identifying Constrained Columns

In this section, a heuristics to identify a significant portion of the sequence using a notion of *mutationally independent* nodes has been described. Two nodes of a labeled phylogenetic tree are mutationally independent if

1. the nodes have the same character which is different from the background character - character occurring at the root node of the labeled phylogenetic tree, and
2. the root node of the minimal subtree covering the nodes has background character, and
3. there are only two or three types of characters in a column.

Since there are multiple possible parsimonious labeling for a pair of phylogenetic tree and multiple sequence alignment, choice of independent nodes depends upon the choice of the labeled tree. However, the presence of independent nodes is not affected by the choice of the labeled trees.

Example 3:

Consider labeled phylogenetic trees in Figure 2b and 2c for the phylogenetic tree in Figure 2 and multiple sequence alignment in Figure 1. In Figure 2b, the background character is V; the nodes P25112 and nodes P02380 are independent nodes since the node N_0 has amino-acid character V, and both P25112 and P02380 have characters T. In Figure 2b, the background character is T; and the nodes P34789 and N_2 are independent. Note that the number of mutational events between the independent nodes is two.

The constrained columns possibly play a major role in marking the portions necessary for structure or function of a genome (or protein) sequence since the presence of the same character despite independent mutations exhibits evolutionary pressure to preserve a common property. necessary for stabilization of the molecule. Moreover, the difference in the properties of two amino-acids (or nucleotides for genes) present at mutationally independent nodes may be responsible for function preserving variations in physical and chemical properties between homologous proteins.

The algorithm for identifying constrained columns is given in Figure 4. In the algorithm, function *parsimonius_labels* generates a labeled phylogenetic tree, the function *string* concatenates all the label in a set, the function *character* returns the parsimonious value at the node.

The algorithm was coded in Prolog and executed on various homologous protein sequence alignments available in protein data banks available at Argonne National Laboratory. The corresponding unrooted phylogenetic tree for every protein alignment was created using a typical software available at Argonne National Laboratory.

5.1. Results and Discussions

Table 2 exhibits the pairs of amino acids and their relative occurrence. The strong substitutability between two amino acids is related to specific common properties being preserved. Simple rules of equivalence, and transitivity do not work in this domain since two amino acids may share more than one common property. For example, D is substituted by E and by N quite regularly. However, E is rarely substituted by N.

```
Algorithm constrained_columns
Input:   1. multiple sequence alignment $\mathcal{M}$;
         2. a phylogenetic tree $\mathcal{T}$
Output:  1. $S^C$ - a set of constrained columns;
         2. $S^2$ - a set of group of substitutable characters
            in all the constrained columns.
begin
  1. $S^C = \Phi$; $S^2 = \Phi$; /* initialize */;
  2. Let the set of columns with two or three types of
     characters be $S^C$.
  3. $\forall C_I \in S^C$
  do /* generate labeled tree */
    4. $\mathcal{T}^M$- = parsimonius_labels($\mathcal{T}$,$\mathcal{M}$);
    5. $\forall$ internal node $n \in \mathcal{T}^M$
    do
      6. Let the set of charcaters in left subtree be $S^L$;
      7. Let the set of characters in right subtree be $S^R$;
      /* get the common set $S^M$ of characters after
         mutation from left and right subtrees */
      8. $S^M = S^L \cap S^R$ -{character(n)};
      9. if $S^M \neq \Phi$ then /* non-empty set */
        10. $S^C = S^C \cup \{C_I\}$; /* mark the columns */
        11. $S^2 = S^2 \cup$ {string({character(n)} $\cup S^M$ )};
            goto 2;
        endif
    od
  od;
end.
```

Figure 4: identifying constrained columns

The analysis shows that major portions of constrained columns are covered by three amino acids: isoleucine, leucine, and valine. which form the non-polar hydrophobic region in proteins. Some frequently occurring pairs are 'AG', 'AP', 'AS', 'AT', 'AV', 'DE', 'DN', 'EQ', 'FL', 'GS', 'GN', 'IL', 'IV', 'KQ', 'KR', 'LM', 'LV', 'NS', , 'QR', 'ST'. Many of these pairs belong to similar subclass of amino acids. For example, the pairs 'AG'[2], 'IL', 'IV', 'LV', and 'AV' are aliphatic amino acids, and exhibit hydrophobicity; the amino-acids 'DE' are acidic and have a fully ionized second carboxyl group; the amino-acids 'KR' are positively charged basic amino acids; and the amino-acids 'GS', 'GN', and 'ST' are uncharged polar hydroxyamino acids. Similarly, the amino-acid 'N' is an amide of 'D', and the amino-acid 'Q' is an amide of 'E'. Although, some of the pairs have amino-acids from different sub classes such as 'AS', 'AT', 'LM', 'FL', and VT. However, these pairs share a common property. For example, 'S' is the hydroxyl version of the 'A'; 'T' is a hydroxyl version of 'V'; 'and both 'LM' and 'FL' have non-polar hydrophobic side-chain. Some of the less frequent pairs such as 'KQ' have common potentiality of hydrogen acceptance [9]. A further study of relaxing the constraints to allow three different types of charcaters in a column exhibited that there are constrained columns with triples such as 'ILV' which is to be expected since all three are highly hydrophobic aliphatic acids.

[2] G is a border line case between non-polar aliphatic amino-acids and uncharged polar amino-acids.

6. Other Applications

The tool has also been used to derive conserved columns and strongly related group of columns.. For example, conserved columns are derived using the function *count_columns*, and identifying the character with maximum count with a percentage value above a threshold value. Strongly related columns are identified by the iterative derivation of the counts of character-groups for a set of columns using the function *count_columns* such that the set of character-groups are restricted to very few possibilities from Cartesian product $\mathcal{N}$ X ...X $\mathcal{N}$ for nucleotides and $\mathcal{A}$ X ... X $\mathcal{A}$.

7. Conclusion

In this paper, a high level language framework for comparative analysis of homologous sequence alignment and the corresponding toolkit have been. described. The tool-set is suitable for complex analysis of large size multiple sequence alignment and large size phylogenetic tree, and the integration of information derived from multiple sequence alignment and phylogeny. The framework has been applied to identify a new set of constrained columns which exhibit some evolutionary pressure to preserve a common property despite mutations resulting into different amino acids The set of these constrained columns are significant since their common properties characterize the portion of homologous protein sequences.

Currently, the properties of different amino-acid groups (based on their chemical structure) are being studied in detail, and strongly related constrained columns are being studied in order to integrate this information with secondary and tertiary structures predicted using other techniques.

	A	C	D	E	F	G	H	I	K	L	M	N	P	Q	R	S	T	V	W	Total	Probability
A		11	5	11	2	102		2	14	8	2		37	1	4	176	51	51		477	0.088
C	11				2	1		1		2		1			2	14		2		36	0.006
D	5			165		12	6				2	72	1	2	2	11	3	1		282	0.052
E	11		165			6	1		15	1	1	3	5	43	1	2	1	3		258	0.047
F	2	2					3	8		118	3		1			3		5	7	152	0.028
G	102	1	12	6					6			11	4	2	5	31	5	3	1	189	0.035
H			6	1	3				3	1	1	9		17	11	3	1			56	0.010
I	2	1			8				1	213	25	2			1	1	16	582	1	853	0.158
K	14			15		6	3	1		4	2	14	2	23	198	6	6	3		297	0.055
L	8	2		1	118		1	213	4		81		13	4	6	8	4	87		550	0.102
M	2		2	1	3		1	25	2	81				2		1		10		130	0.024
N		1	72	3		11	9	2	14					5	5	28	9			159	0.029
P	37		1	5	1	4			2	13				5	3	15	9	2		97	0.018
Q	1		2	43		2	17		23	4	2	5	5		20		2	1		127	0.023
R	4	2	2	1		5	11	1	198	6		5	3	20		2	2		4	266	0.049
S	176	14	11	2	3	31	3	1	6	8	1	28	15		2		118			419	0.077
T	51		3	1		5	1	16	6	4		9	9	2	2	118		20	1	248	0.046
V	51	2	1	3	5	3		582	3	87	10		2	1			20			770	0.143
W					7	1		1							4		1			14	0.002
	477	36	282	258	152	189	56	853	297	550	130	159	97	127	266	419	248	770	14	5380	1

Table 2: Occurrences of Amino-acid pairs at mutationally independent nodes.

Acknowledgments

A major part of this project was supported DOE through Argonne National Laboratory's summer research faculty program.. I thank Ross Overbeek for many useful discussions related to this project, and for providing me with necessary data. I also thank Terry Gaasterland, Natalie Maltsev, and Raja Rajshekar for useful discussion during the development of this tool.

Bibliography

[1] P. Berg and M. Singer, "Dealing with Genes - "The Language of Heredity", Publisher: University Science Books, 1992.

[2] J. U. Bowie, R. L. Luthy, and D. Eisenberg, "A Method to Identify Protein Sequences That Fold into a Known Three-Dimensional Structure, Science, Volume 253, July 1991, pp. 164- 170

[3] J. Felenstein, "Evolutionary Trees from DNA Sequences - A Maximum likelihood approach", The Journal of Molecular Evolution, Publisher: Springer verlag, (1981) 17:368-376

[4] J. Felenstein, "Phylogenies from Molecular Sequences: Inference and Reliability", Annu. Rev. Genet. (1988) 22:521-65.

[5] W. M. Fitch, "Numerical Methods for Inferring Evolutionary Trees", The Quarterly Review of Biology, Volume 57, No. 4, pp. 379-404.

[6] A. L. Lehninger, "Biochemistry", Third Printing, Publisher: Worth Publisher Inc., 1977, pp. 55-155.

[7] W. H. Li and D. Graur, "Fundamentals of Molecular Evolution", Publisher: Sinauer Associates Inc., Massachussets, 1991

[8] L. Sterling, "The Art of Prolog", Second edition, Publisher: MIT Press, 1994

[9] L. Stryer, "Biochemistry", Publisher: W. H. Freeman and Company, Third edition, 1988, pp. 15-41.