Vector Graphics
to improve BLAST graphic representations

by

Rafael Jimenez

Computer Science
Masters Dissertation

School of Computer Science
University of KwaZulu-Natal

December, 2007

Abstract

BLAST reports can be complicated. Viewing them graphically helps to understand them better, especially when the reports are long. At present ”Web BLAST” and the stand-alone ”wwwBLAST” versions, distributed by the NCBI, include graphical viewers for BLAST results. An alternative approach is ”BLAST Graphic Viewer” developed by GMOD as part of the BioPerl library. It provides a more aesthetically pleasing and informative graphical visualization to represent BLAST results. All the strategies mentioned above are based on the use of bitmap graphics and dependent on JavaScript code embedded in HTML. We present Vector Graphic BLAST (VEGRA) a Python object orientated library based on BioPython to yield graphical visualization of results from BLAST utilizing vector graphics. Graphics produced by VEGRA are better than bitmaps for illustration, more flexible because they can be resized and stretched, require less memory, and their interactivity is more effective as it is independent of tertiary technologies due to its integration into the graphic. In addition, the library facilitates a definition of any layout for the different components of the graphic, as well as adjustment of size and colour properties. This dissertation studies previous alternatives and improves them by making use of vector graphics and thus allowing more effective presentation of results. VEGRA is not just an improvement for BLAST visualization but a model that illustrates how other visualization tools could make use of vector graphics. VEGRA currently works with BLAST, nevertheless the library has been written to be extended to other visualization problems.

Acknowledgements

Firstly, I would like to express sincere appreciation to both my advisors, Hugh and Dan for their guidance, patience and encouragement throughout my graduate education. They provided me with valuable help and useful discussion.

I want to thank the School of Computer Science at the University of KwaZulu-Natal, the Central Node of the National Bioinformatics Network and their staff for their constant support without which this dissertation could have never been completed.

Finally many thanks to Alex Garcia (CIAT), Kieran O'Neill (NBN), and Alberto Labarga (EBI) for serving on my advisory committee and providing helpful guidance and critical reviews.

Preface

Submitted in fulfilment of the academic requirements for the degree of Master of Science (M.Sc.) in the School of Computer Science, University of KwaZulu-Natal, Pietermaritzburg campus.
This dissertation represents original work by the author and has not been submitted in part or whole to this or any other university.

Most of this work was carried out at the National Bioinformatics Network in Cape Town under the supervision of Professors Jacobson and Murrell.

Some of this work has been presented at Bioinformatics conferences and appears in conference proceedings [20].

Rafael Jimenez (Candidate)

Hugh Murrell (UKZN supervisor)
School of Computer Science, University of KwaZulu-Natal

Dan Jacobson (NBN co-supervisor)
Central Node, National Bioinformatics Network

Alex Garcia (pro-bono co-supervisor)
Visitor, Central Node, National Bioinformatics Network

Glossary of acronyms

API: Application Programming Interface

BGV: BLAST Graphic Viewer

BLAST: Basic Local Alignment Search Tool

CIAT: International Center for Tropical Agriculture

CSS: Cascading Style Sheets

CVS: Concurrent Versions System

DHTML: Dynamic Hypertext Markup Language

DNA: Deoxyribonucleic acid

DOM: Document Object Model

EBI: European Bioinformatics Institute

EPS: File format is used to transfer PostScript language artwork between applications and is supported by most illustration and page-layout programs

FASTA: File format used as input for DNA and Protein sequences

FLA: A Macromedia Flash source document

FTP: File Transfer Protocol

GI: Gene Info identifiers

GMOD: Generic Model Organism Database

GNU: Recursive acronym for "GNU's Not Unix"

HSP: High-scoring segment pair

HTML: Hypertext Markup Language

Glossary of acronyms

LGPL: The GNU Lesser General Public License

LXR: Linux Cross-Reference

NCBI: National Centre for Biotechnology Information

OBF: Open Bioinformatics Foundation

PDF: Portable Document Format

RNA: ribonucleic acid

SVG: Scalable Vector Graphics

SWF: Graphic file format (a version of the Macromedia Flash Player vector-based graphics format)

UML: Unified Modelling Language

URL: Uniform Resource Locators

W3C: World Wide Web Consortium

WU-BLAST (WuBLAST): Washington University Basic Local Alignment Search Tool

WWW: world wide web

Contents

Chapter 1

Problem statement

The BLAST package tool does not always provide a graphical representation for results. When one is available its visual design and usability is measurably poor. Graphic configuration or customization of the representation is almost non-existent. Documentation about the structure of the code is missing, and even in the case where documentation is present, attempts to improve the tool by re-coding would be a step in the wrong direction in the development of BLAST graphics since it would not use the best file format for this type of representation.

Inspired by the idea of optimizing BLAST graphics, this dissertation reports on the development of a graphic library that enables representation and visualization of Blast results by means of vector graphics. The problem has been addressed in three specific and well-scoped tasks:

1. To improve the graphic design of BLAST graphics using usability and graphic design concepts, taking into consideration previous graphic representation models for BLAST.

2. To develop a library based on object-oriented programming syntax and methodology to create vector graphics to represent BLAST results.

3. To make the graphic customizable by allowing the user to define the properties of the graphic via a simple configuration file.

Chapter 2

Introduction

Design is a set of fields for problem-solving that employs user-centric approaches to understand user needs to create successful solutions that solve real problems [38]. Within the context of computer graphics a graphical user interface (GUI) is a particular type of user interface provided to the user in order to enable interaction with the system. GUIs might be graphical, text-based, or a combination of both [27]. To some extent users gain access to the functionalities provided by the software via a GUI. As graphical user interfaces are communication systems, visual design should play an important role when solving problems related to and with the use of GUIs. Both, aesthetics and functionality should be considered when designing GUIs [38].

Bioinformatics is the application of computer science and technology to the management of biological information [39]. Quite often bioinformatics software is text-based, where letters and numbers are displayed in order to represent molecular sequences [37]. Scientific visualization and computer graphics methods can assist with the visual display of this information [37]. Although information visualization techniques provide possible solutions to the problem of managing and analysing an overwhelming amount of biological information, the visualization of biological information still presents many challenges [37] [39].

This dissertation examines visual representations for BLAST, a bioinformatics tool widely used to search protein and nucleotide databases for sequences similar to a search sequence [26]. It presents a study of the fundamental techniques of visual design, demonstrates their application when designing a graphical interface and proposes alternatives to enhance its visual quality and usability. It also provides a review of suitable graphic technologies that could be used instead to efficiently create this kind of representation. Finally, based on these analyses, this dissertation describes a graphical library that allows the generation of a new visual representation; thus enhancing the visual quality of the graphical user interface for BLAST outputs.

As this dissertation brings together concepts and paradigms from computer science, graphical design, and biology applied to a specific Bioinformatics application it might be considered to be highly interdisciplinary. This raises the challenge of how to present this work in a way in which the reader will not get lost. This has been addressed by providing the necessary biological and computer science background in the initial chapters. The importance of BLAST and the context in which it is being used are both explained in chapter three. Basic biological concepts such as DNA, the genetic code and molecular evolution, as well as computer graphics concepts such as bitmaps and vector graphics are also included in chapter three. In chapter four we briefly review the state of the art in the visualization of BLAST outputs focusing our study on graphical representations. In chapter five we continue with a critical review, analyzing specific problems in the

GUI design of the existing approach to visualizing BLAST outputs. In chapter six we define the methodology used in this dissertation to create a graphic library for BLAST graphic representations. In chapter seven we describe the structure, content and functionality of the graphic library we propose as a solution. Finally the dissertation concludes with chapter eight which discusses the achievements and contributions of this work.

Chapter 3

Background

3.1 A glimpse of the genetic code

An organism's genome carries genetic information that can contain tens of thousands of genes. Most genes serve as a recipe for building a protein molecule. Proteins perform important tasks within cells or serve as building blocks for intra-cellular and extra-cellular components. The flow of information from the genes determines the protein composition and thereby the functions of the cell.

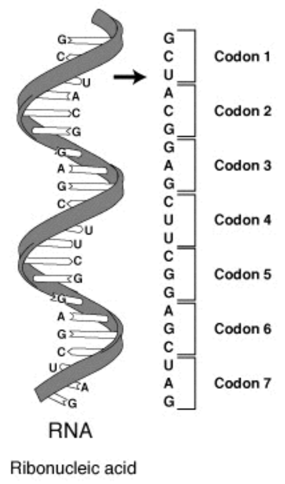


Figure 3.1: RNA codons [18]

Translation is a step along the way to construct proteins from DNA. It is the synthesis of proteins directed by an intermediary template to the DNA called mRNA.

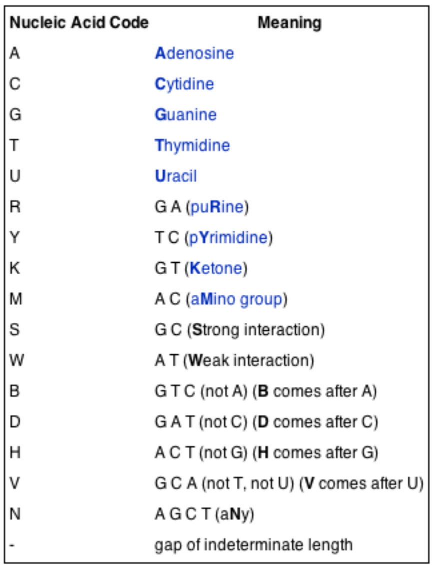
The sequence of mRNA and DNA is composed of four different nucleotides whereas a protein is built up of 20 amino acids. To allow the four nucleotides to specify 20 different amino acids, the nucleotide sequence is interpreted in groups of three nucleotides known as codons [see figure: 3.1]. Thus, each codon specifies one amino acid. This is referred to as the genetic code [see figure: 3.4].

Because codons are three nucleotides long, DNA can be translated in three different reading frames. Since DNA is double stranded, there are really six reading frames for every piece of DNA. Thus every reading frame has the potential to produce a radically different amino acid sequence.

3.1.1 Representation of the genetic code on a computer

As shown in the genetic code table [see figure: 3.4], nucleotides and amino-acids are represented by a specific letter in the alphabet. Sequences of genes and proteins are represented by these alphabetic characters and computers can be used to read, compare and analyse them.

This abstraction and the rules applied to each of these symbols provides a relatively simple computational model which can be used



Nucleic Acid Code	Meaning
A	**A**denosine
C	**C**ytidine
G	**G**uanine
T	**T**hymidine
U	**U**racil
R	G A (pu**R**ine)
Y	T C (p**Y**rimidine)
K	G T (**K**etone)
M	A C (a**M**ino group)
S	G C (**S**trong interaction)
W	A T (**W**eak interaction)
B	G T C (not A) (**B** comes after A)
D	G A T (not C) (**D** comes after C)
H	A C T (not G) (**H** comes after G)
V	G C A (not T, not U) (**V** comes after U)
N	A G C T (a**N**y)
-	gap of indeterminate length

Figure 3.2: Nucleic Acid Code

to read and interpret DNA and protein sequences.

The nucleotide alphabet is quite simple, consisting of just four nucleotides: adenosine, cytidine, guanosine and thymidine (uridine in case of RNA). For simplicity sake, they are usually abbreviated as A, C, G, and T (U). Amino acids from the building blocks for proteins and the 20 amino acids are represented by the symbols, A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, and Y [14]. In addition, with the aim of describing ambiguities in the nucleotide and protein alphabets, the code can be broadened by the addition of more characters. The overall content of this code is defined by the standard IUB/IUPAC nucleic acid codes [see figure: 3.2] and amino acid code [see figure: 3.3].

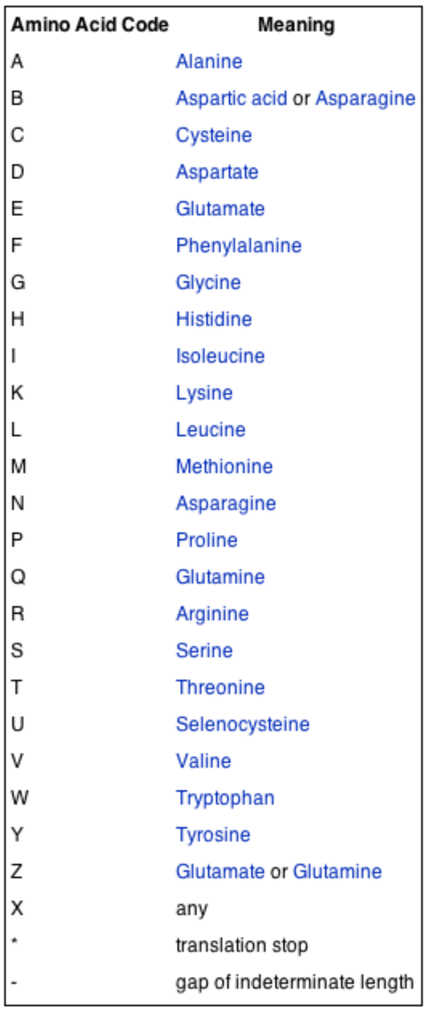
3.2 Molecular evolution

Evolution refers to changes in the gene pool resulting in the progressive adaptation of populations to their environment [15]. Evolution occurs because genetic variation exists in populations and because there is a natural selection favoring organisms that are best adapted to their environment [15].

Genetic sequences basically change over time due to three forces: mutation, natural selection, and genetic drift.

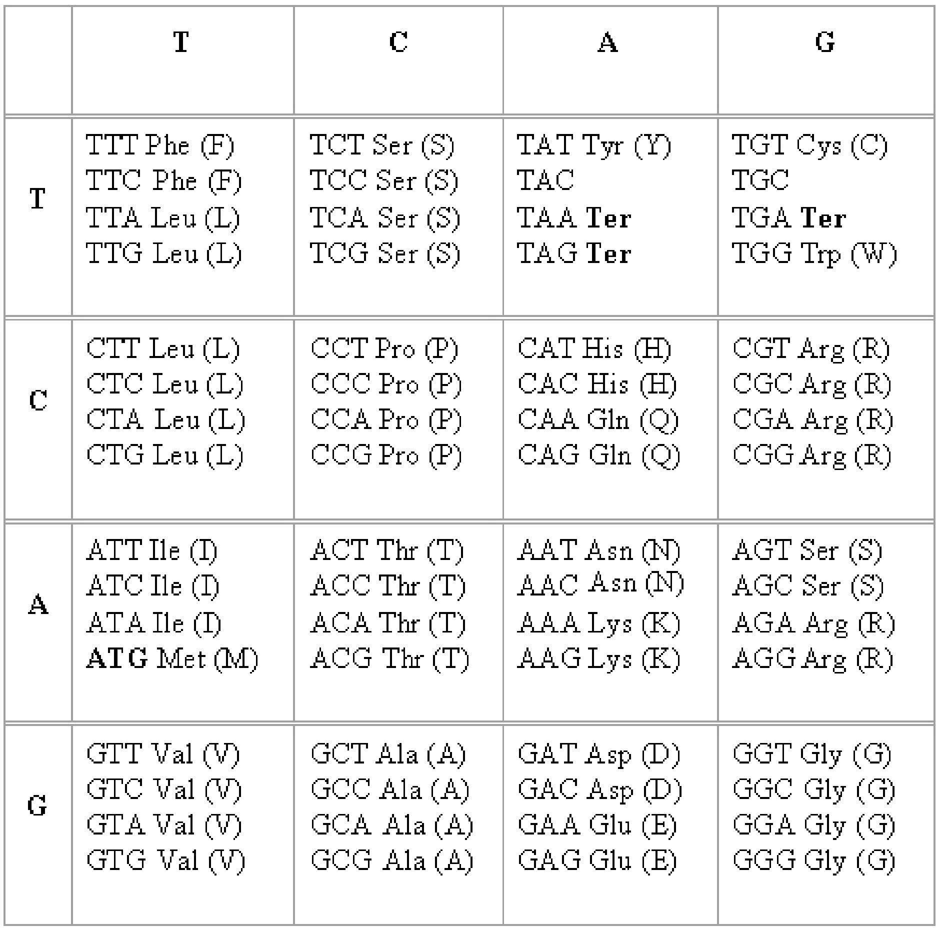
3.2.1 Mutation

A mutation is simply a change in a DNA sequence. It can be due to errors in the replication of the DNA or damage caused by mutagenic agents [30].



Amino Acid Code	Meaning
A	Alanine
B	Aspartic acid or Asparagine
C	Cysteine
D	Aspartate
E	Glutamate
F	Phenylalanine
G	Glycine
H	Histidine
I	Isoleucine
K	Lysine
L	Leucine
M	Methionine
N	Asparagine
P	Proline
Q	Glutamine
R	Arginine
S	Serine
T	Threonine
U	Selenocysteine
V	Valine
W	Tryptophan
Y	Tyrosine
Z	Glutamate or Glutamine
X	any
*	translation stop
-	gap of indeterminate length

Figure 3.3: Amino Acid Code



	T	**C**	**A**	**G**
T	TTT Phe (F) TTC Phe (F) TTA Leu (L) TTG Leu (L)	TCT Ser (S) TCC Ser (S) TCA Ser (S) TCG Ser (S)	TAT Tyr (Y) TAC TAA **Ter** TAG **Ter**	TGT Cys (C) TGC TGA **Ter** TGG Trp (W)
C	CTT Leu (L) CTC Leu (L) CTA Leu (L) CTG Leu (L)	CCT Pro (P) CCC Pro (P) CCA Pro (P) CCG Pro (P)	CAT His (H) CAC His (H) CAA Gln (Q) CAG Gln (Q)	CGT Arg (R) CGC Arg (R) CGA Arg (R) CGG Arg (R)
A	ATT Ile (I) ATC Ile (I) ATA Ile (I) **ATG** Met (M)	ACT Thr (T) ACC Thr (T) ACA Thr (T) ACG Thr (T)	AAT Asn (N) AAC Asn (N) AAA Lys (K) AAG Lys (K)	AGT Ser (S) AGC Ser (S) AGA Arg (R) AGG Arg (R)
G	GTT Val (V) GTC Val (V) GTA Val (V) GTG Val (V)	GCT Ala (A) GCC Ala (A) GCA Ala (A) GCG Ala (A)	GAT Asp (D) GAC Asp (D) GAA Glu (E) GAG Glu (E)	GGT Gly (G) GGC Gly (G) GGA Gly (G) GGG Gly (G)

Figure 3.4: Genetic code table

3.2.2 Natural selection

The basic concept of natural selection is that "nature" (the physical and biological environment) "selects" for variations (as caused by mutations) in characteristics or traits which improve individual survival and reproduction (adaptive traits) and selects against unfavourable traits which burden individuals (maladaptive traits) [12].

3.2.3 Genetic drift

Genetic drift is the term used in population genetics to refer to the statistical drift over time of allele [1] frequencies in a population due to random effects in the formation of successive generations [30]. It means that a neutral allele may be expected to increase or decrease in any given generation with equal probability.

The DNA of a specific species contain signs of its evolutionary history. Two species that share a similar DNA sequence (and thus also the specific protein coded) probably have a common ancestor. Usually the DNA sequence will be slightly different between the two, as each species will have accumulated different mutations once they separate from one another. The number of mutations can be used to indicate how closely the species are related. It can also be used as an indication of how long ago they became separate species. The history of the DNA of each species manifests itself in its genetic code.

[1]In genetics, an allele is any one of a number of viable DNA codings occupying a given locus (position) on a chromosome.

3.3 The importance of similarity searches

Recent advances in molecular biology, particularly high throughput sequencing techniques, have provided researchers with fast, cheap, and simple sequencing techniques so that large genomes can now be determined in relatively short periods of time. In fact, to date, many bacterial genomes, as well as those of some simple and complex eukaryotes have been fully sequenced.

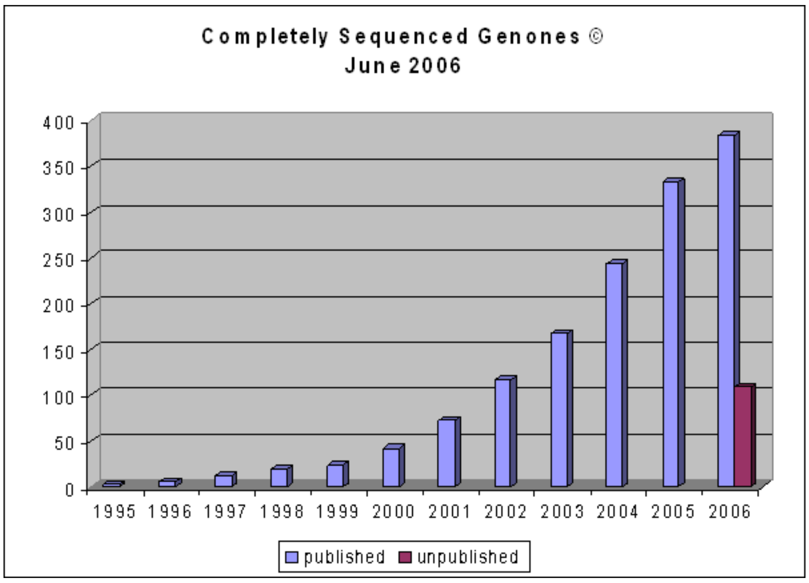


Figure 3.5: Sequenced genomes in the last twelve years [4]

In total 422 genome projects have been completed with all the results being publicly available. Figure 3.5 [25] illustrates the number of completed genomes sequenced for the last ten years. Although the technology we have at hand facilitates data production, it also makes it increasingly hard to organize, manage and analyze all the data being produced. Ultimately we would like to extract knowledge from this data.

One of the most powerful methods for inferring the biological function of a gene (or the protein that it encodes) is sequence similarity searching on protein and DNA sequence databases [21]. Similarity searches provide the most informative and reliable method for inferring the biological function of an anonymous gene [22].

Moreover, prediction of sequence homology by identifying statistically significant sequence similarity can be used to infer common ancestors, assign functions in metabolic pathways, estimate evolutionary distance in phylogeny reconstruction, determine the active sites of enzymes, construct novel mutations and characterize alleles of genetic diseases, to name just a few uses. The degree to which change in sequences follow functional or random paths depends on natural selection and neutral evolution [21].

Several computer programs have been developed to check and analyse similarities among sequences and genomes and the most widely used in sequence comparison is called BLAST [22]

3.4 What is BLAST?

BLAST (Basic Local Alignment Search Tool) is a set of similarity search programs designed to explore all of the available sequence databases regardless of whether the query is a protein (amino acid) or DNA (nucleotide) sequence. The BLAST programs have been designed for speed, with a minimal sacrifice of sensitivity to distant sequence relationships. The scores assigned in a BLAST search have a well-defined statistical interpretation, making biologically relevant matches easier to distinguish from random background hits. BLAST uses a heuristic algorithm that seeks local alignments and is therefore able to detect relationships among sequences that share only isolated regions of similarity [2].

BLAST has become one of the most important pieces of software in the field of computational biology. There are several reasons for this. First, sequence similarity is a powerful tool for inferring function and relationships to novel molecular sequences. Second, BLAST is fast. The number of sequences available is large (over 100 Gigabases) and growing rapidly [4], so speed is important. Third, BLAST is reliable, from both a rigorous statistical standpoint and a software development point of view. Fourth, BLAST is flexible and can be adapted to many sequence analysis scenarios. Finally, BLAST is entrenched in the bioinformatics culture to the extent that the word "BLAST" is often used as a verb. There are other BLAST-like algorithms with some useful features, but the historical momentum of BLAST maintains its popularity above all others [22].

3.4.1 An overview of the BLAST programs

There are many different types of BLAST programs available from the main BLAST package. Choosing the right one depends on the type of query (long, short; nucleotide or protein), and the searched database. A BLAST search has four components: query, database, program, and search purpose/goal.

Databases

To discuss effective BLAST program selection, we first need to know what databases are available and what sequences these databases contain. According to their content, they are grouped into nucleotide and protein databases. These databases and their detailed compositions can be found in the annual paper review called:

The Molecular Biology Database Collection [11].

BLAST Programs

The appropriate selection of a BLAST program for a given search is influenced by the following three factors:

1. the nature of the query
2. the purpose of the search
3. the database intended as the target of the search and its availability

List of the main BLAST programs:

Nucleotide BLAST. Nucleotide BLAST searches take nucleotide sequences as input and compare these against other nucleotide sequences [26].

> **Standard nucleotide-nucleotide BLAST.** This program takes nucleotide sequences in FASTA format [2], GenBank Accession numbers or GI numbers and compares them against nucleotide databases.

Protein BLAST. Protein BLAST takes protein (amino acid) sequences and compares these against other protein sequences [26].

> **Standard protein-protein BLAST.** This program takes protein (amino acid sequences) sequences in FASTA format, GenBank Accession numbers or GI numbers and compares them against protein databases [22].

[2]FASTA format is a text-based format for representing either nucleic acid sequences or peptide sequences, in which base pairs or amino acids are represented using single-letter codes. The format also allows for sequence names and comments to precede the sequences.

Translating BLAST. This BLAST translates either query sequences or databases from nucleotides to proteins (amino acids) so that protein to nucleotide searches can be performed [26].

Translated query - Protein db BLASTx. Converts a nucleotide query sequence into protein sequences in all six reading frames. The translated protein products are then compared against the chosen protein database [22].

Protein query - Translated db tBLASTn. Takes a protein query sequence and compares it against a nucleotide database which has been translated in all six reading frames [22].

Translated query - Translated db tBLASTx. Converts a nucleotide query sequence into protein sequences in all six reading frames and then compares this to a nucleotide database which has been translated in all six reading frames [22].

Other BLAST programs: These include several BLAST derivatives and BLAST wrappers (scripts that run BLAST in a specialized way) with names such as PSI-BLAST, PHI-BLAST, MegaBLAST, BLASTZ, XBLAST, MPIBLAST, HT-BLAST, GENE-BLAST and WuBLAST [22].

3.4.2 How BLAST works

The BLAST algorithm uses heuristics to increase its speed. BLAST performs "local" alignments [2]. Most proteins are modular in nature, with functional domains often being repeated within the same protein as well as across different proteins from different species. The BLAST algorithm is tuned to find these domains or shorter stretches of sequence similarity. The local alignment approach also means that a mRNA can be aligned with a piece of genomic DNA, as is frequently required in genome assembly and analysis. If instead BLAST started out by attempting to align two sequences over their entire lengths (known as a global alignment), fewer similarities would be detected, especially with respect to domains and motifs.

When a query is submitted, the sequence, plus any other input information such as the database to be searched, word size, expect value, and so on, are fed to the algorithm on the BLAST server. BLAST works by first making a look-up table of all the "words" (short subsequences, by default three letters for proteins and eleven letters for nucleotides) and "neighbouring words", i.e., similar words in the query sequence. The sequence database is then scanned for these "hot spots". When a match is identified, it is used to initiate gap-free and gapped extensions of the "word". After the algorithm has looked up all possible "words" from the query sequence and extended them maximally, it assembles the best alignment for each query-sequence pair and writes this information to a BLAST result [22].

3.4.3 BLAST Scores and Statistics

Once BLAST has found a similar sequence to the query in the database, it is helpful to have some idea of whether the alignment is "good" and whether it portrays a possible biological relationship, or whether the similarity observed is attributable to chance alone. BLAST uses statistical theory to produce a bit score and expect value (E-value) for each alignment pair (query to hit) [33].

The bit score gives an indication of how good the alignment is; the higher the score, the better the alignment. In general terms, this score is calculated from a formula that takes into account the alignment of similar or identical residues, as well as any gaps introduced to align the sequences [33]. A key element in this calculation is the “substitution matrix”, which assigns a score for aligning any possible pair of residues. The BLOSUM62 matrix is the default for most BLAST programs, the exception being BLASTn (the program that performs nucleotide - nucleotide comparisons and hence does not use protein-specific matrices). Bit scores are normalized, which means that the bit scores from different alignments can be compared, even if different scoring matrices have been used [26].

The E-value gives an indication of the statistical significance of a given pairwise alignment and reflects the size of the database and the scoring system used [33]. The lower the E-value, the more significant the hit. A sequence alignment that has an E-value of 0.05 means that this similarity has a 5 in 100 (1 in 20) chance of occurring by chance alone. Although a statistician might consider this to be significant, it still may not represent a biologically meaningful result, and analysis of the alignments is required to determine biological significance.

3.4.4 Basic BLAST report structure

A standard BLAST report is composed of four sections:

Header

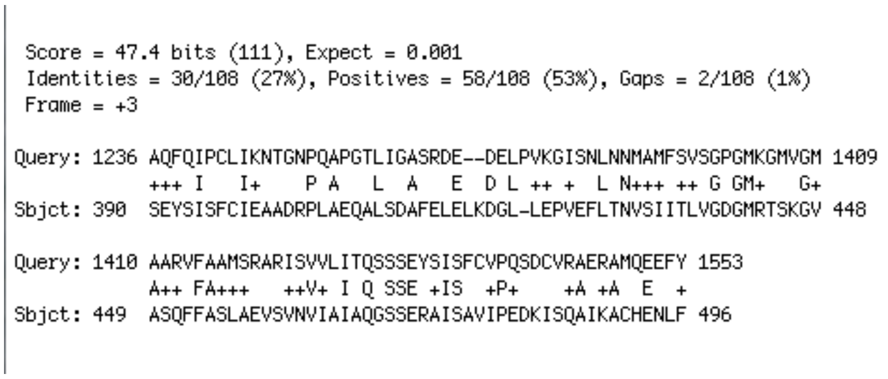
The first line contains the name of the program, its version, and its build date. This is for inclusion in error reports if BLAST crashes or exhibits some kind of unexpected behaviour. The next piece of information is a reference to the scientific literature which should be cited in published research that employs BLAST. The most important information in the header, the names of the query sequence and the database, appear next. The last line is a progress meter that is updated during the search.

One-line summaries

Each line indicates the name of the sequence, the highest scoring alignment found and the lowest E-value for any High-scoring segment pair (HSP) or group of HSPs. The one-line summaries are often hyperlinked to the alignments further below when the output comes from a web page. These summaries are useful for quickly finding out information such as the names of the top matches.

Alignments

The alignments usually make up the bulk of the report, which shows only one alignment [see figure 3.6].



```
 Score = 47.4 bits (111), Expect = 0.001
 Identities = 30/108 (27%), Positives = 58/108 (53%), Gaps = 2/108 (1%)
 Frame = +3

Query: 1236 AQFQIPCLIKNTGNPQAPGTLIGASRDE--DELPVKGISNLNNMAMFSVSGPGMKGMVGM 1409
            +++ I   I+    P A   L  A   E  D L ++ +  L N+++ ++ G GM+   G+
Sbjct: 390  SEYSISFCIEAADRPLAEQALSDAFELELKDGL-LEPVEFLTNVSIITLVGDGMRTSKGV 448

Query: 1410 AARVFAAMSRARISVVLITQSSSEYSISFCVPQSDCVRAERAMQEEFY 1553
            A++ FA+++   ++V+ I Q SSE +IS  +P+    +A +A  E  +
Sbjct: 449  ASQFFASLAEVSVNVIAIAQGSSERAISAVIPEDKISQAIKACHENLF 496
```

Figure 3.6: Alignment

Footer

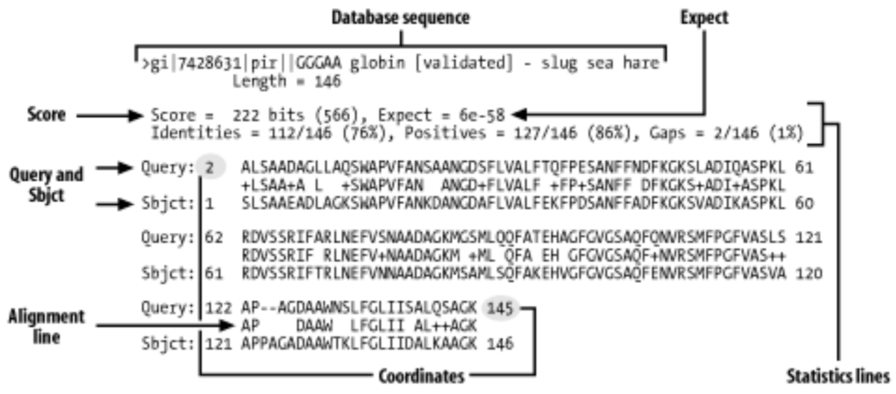
The footer reports search parameters and various other statistics. The most important features are the word size (W), neighbourhood word threshold score (T), Expect (E), and the scoring scheme (scoring matrix or match/mismatch values and the gap costs) because these factors control the sensitivity and specificity of a search. The footer labels these values clearly.

3.4.5 Alignment structure

The alignments and alignment statistics reported by BLAST differ slightly from program to program (BLASTP, BLASTN, BLASTX, TBLASTN, and TBLASTX).

BLASTP alignment

BLASTP alignments are the simplest to understand. Figure 3.7 shows the anatomy of a typical BLASTP alignment.



```
>gi|7428631|pir||GGGAA globin [validated] - slug sea hare
          Length = 146

 Score =  222 bits (566), Expect = 6e-58
 Identities = 112/146 (76%), Positives = 127/146 (86%), Gaps = 2/146 (1%)

Query: 2    ALSAADAGLLAQSWAPVFANSAANGDSFLVALFTQFPESANFFNDFKGKSLADIQASPKL 61
             +LSAA+A L  +SWAPVFAN  ANGD+FLVALF +FP+SANFF DFKGKS+ADI+ASPKL
Sbjct: 1    SLSAAEADLAGKSWAPVFANKDANGDAFLVALFEKFPDSANFFADFKGKSVADIKASPKL 60

Query: 62   RDVSSRIFARLNEFVSNAADAGKMGSMLQQFATEHAGFGVGSAQFQNVRSMFPGFVASLS 121
             RDVSSRIF RLNEFV+NAADAGKM +ML QFA EH GFGVGSAQF+NVRSMFPGFVAS++
Sbjct: 61   RDVSSRIFTRLNEFVNNAADAGKMSAMLSQFAKEHVGFGVGSAQFENVRSMFPGFVASVA 120

Query: 122  AP--AGDAAWNSLFGLIISALQSAGK 145
             AP    DAAW  LFGLII AL++AGK
Sbjct: 121  APPAGADAAWTKLFGLIIDALKAAGK 146
```

Figure 3.7: BLAST anatomy

Score

This value is computed from the scoring matrix and gap penalties. A higher score indicates greater similarity. The raw score is shown without units, and the normalized score is followed by "bits." [33].

Database sequence

The complete FASTA definition line is reported here along with the length of the sequence. All the alignments between the query and a specific database sequence are collectively called a hit. The database in figure 3.7 has one alignment.

Expect

The number of alignments expected at random given the size of the search space, the scoring matrix, and the gap penalties [33]. The lower the E-value, the less likely this is a random similarity.

Statistics lines

The Score, E-value, and percent identity always appear here. Depending on the program, percent positive scoring, P-value, group, gaps, strand, and reading frame may also be reported here.

Query and Sbjct

The query sequence is always listed first. The database sequence is abbreviated as Sbjct (*subject).*

Coordinates

The coordinates of each sequence are indicated at the beginning and ending *of each line. The single alignment in figure 3.7 is long enough to be reported on three separate lines.*

Alignment line

Letters that are identical between two sequences are reported here. Those that have positive scores in the scoring matrix are displayed with a plus sign. Gaps and non-positive scores are blank.

BLASTN alignment

DNA is a double-stranded molecule, and genes may occur on either strand. This fact makes BLASTN alignments a little more difficult to interpret than BLASTP alignments. When a query sequence is searched against a database, both strands of the query are examined. The plus strand is the sequence in the FASTA file. The minus strand is the reverse complement of this sequence. If the similarity between the query and subject sequences is on the same strand, both sequences are labelled as being on the plus strand and the coordinates increase from left to right [see figure 3.8] . Since BLAST just aligns letters and has no model of genes or other features, it is impossible to determine on which strand a gene lies from a BLASTN alignment. Even if an alignment is labelled as "Plus/Plus," the encoded gene may be on the minus strand.

BLASTX alignment

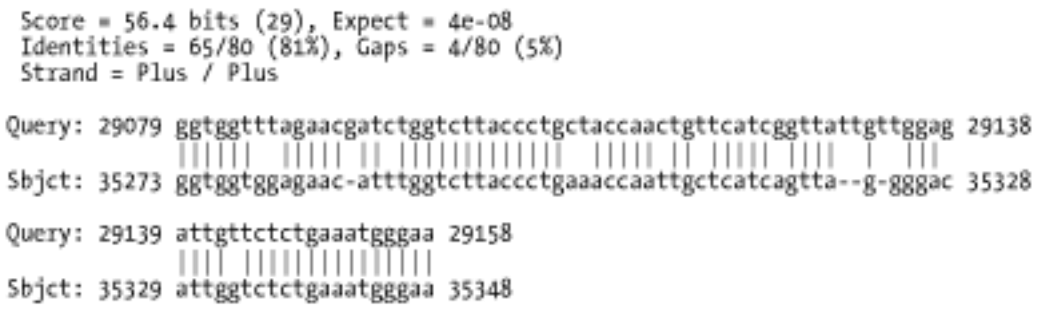
Alignments from BLASTX are complicated by both strand and reading frame. The query sequence is translated in three frames on both the plus and minus strands. With three nucleotides per codon, the coordinates of the query sequence increase by threes [see figure 3.6]. On the plus strand, the reading frame is computed relative to the start of the plus strand; reading frame 1 starts at position 1 and reading frame 2 starts at position 2. On the minus strand, the reading frame is calculated relative to the reverse complement of the plus strand; the last letter of the FASTA file starts frame -1 and the second-to-last letter starts frame -2. Minus strand matches invert the query coordinates.

TBLASTN alignment

TBLASTN alignments are very similar to BLASTX alignments, except that the database and query are exchanged. Therefore, the database sequence increases in threes, and the database sequence has flipped coordinates on the minus strand.

TBLASTX alignment

TBLASTX has more complicated alignments because both the query and the database have strand and frame. One of the most confusing aspects of TBLASTX alignments is that a number of different frames may represent the same region from both the query and subject. A TBLASTX alignment between two genomic



```
 Score = 56.4 bits (29), Expect = 4e-08
 Identities = 65/80 (81%), Gaps = 4/80 (5%)
 Strand = Plus / Plus

Query: 29079 ggtggtttagaacgatctggtcttaccctgctaccaactgttcatcggttattgttggag 29138
             ||||||  ||||| || |||||||||||||  ||||| || ||||| ||||  |  |||
Sbjct: 35273 ggtggtggagaac-atttggtcttaccctgaaaccaattgctcatcagtta--g-gggac 35328

Query: 29139 attgttctctgaaatgggaa 29158
             |||| |||||||||||||||
Sbjct: 35329 attggtctctgaaatgggaa 35348
```

Figure 3.8: BLASTn

sequences often highlights shared coding sequences. However, the correct frame of the encoded proteins can't be determined from a TBLASTX report.

3.4.6 Graphical representations of BLAST reports

BLAST reports are complex. They are long reports composed of statistical and biological data and as such are difficult to structure, organize and summarize in a meaningful simple view. Viewing them graphically can help the user to understand them better, especially when the reports are very long. At present just the web BLAST version includes graphical representation for the output [see figure 3.9].

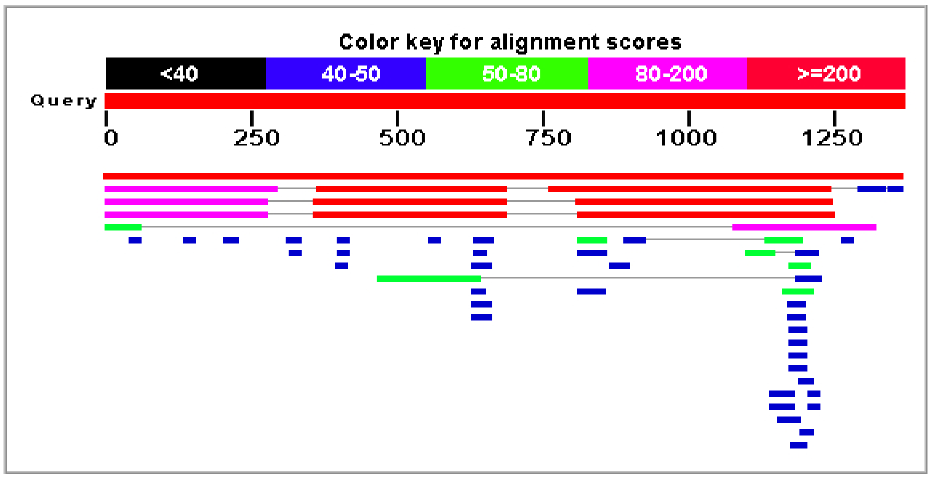


Figure 3.9: NCBI BLAST result

3.5 BioLibraries

Computational biology is a young science where data and methods to treat information, expand and change constantly. As such, pre-packaged commercial suites become outdated in a short period of time, often not addressing the demands of the scientist that continually needs to do better data analysis [28]. It was partly this frustration with commercial suites that drove the creation of the BioGroups for developing BioLibraries and other open source work [28].

The Open Bioinformatics Foundation or OBF is a non profit, volunteer organization which supports many open source projects (biolibraries among them) for computational biology. BioLibraries are based on common standard languages (Perl, Python, Java, Ruby, PHP, ...) that provide toolkits of functionality that facilitate the creation of pipelines or analytical modules.

To enhance performance many of them can be linked to compiled libraries, for instance graphic libraries in C++. All of them, in one way or another, facilitate access to relational database management systems. They are multiplatform and thus run on different versions of Unix, Linux, Windows and MacOS. All of them are freely available and open source.

These bioprojects are often well supported by wikipages and mailing lists. One impressive feature of these BioLibraries is the capacity they provide to generate graphics of analytical outputs as well as graphical user interfaces that make it easier for researches to access analytical resources [10] [8] [9].

OBF BioLibraries support several languages and can differ in structure and intended solutions. Examples of some of these libraries include BioJava, BioPerl, BioPython, BioRuby and BioPHP.

3.5.1 An overview of BioPython

The BioPython Project is an international association of developers to create freely available tools to address biological problems. The BioPython web site [10] provides an online resource for modules, scripts, and web links for developers of Python-based software for life science research.

The BioPython package (2.7 MB gzipped) contains about 60 classes. What it covers in comparison with its siblings is easy and straightforward to follow, and it provides a good base for further development. Installation of the complete package along with all the dependencies involves separately downloading and installing other components.

BioPython includes local and remote alignment resources as well as methods for biological and regular expression pattern matching, providing code fragments and scripts for doing this.

A big part of components included in BioPython have been conceived to parse the many different formats used in computational biology. Rather than having to write a completely new parser for each new format, BioPython relies on the Scanner/Consumer methods that are the core of the Martel regular expression parsing engine, which allows the user to make use of many current formats and easily define others using a SAX-like approach. BioPython also allows easy creation and manipulation of objects due to Python's support for object oriented programming [28].

3.6 Visualization

Visualization is any technique for creating images, diagrams, or animations to communicate a message. Visualization through visual imagery has historically been an effective way to communicate both abstract and concrete ideas [5]. In computer graphics, computers are utilized to generate visual images synthetically and to integrate or alter visual and spatial information sampled from the real world.

In computational biology the term visualization has become popular quite recently, meaning anything from visual starting points for navigation through biological data to transformations of data into graphical representations of biological results. There are an increasing number of tools being developed for both generic use (rule, tree, map and other graphing visualisers) and for computational biologists (genome browsers, 3D viewers, sequence searching filters, etc) [39].

Images or graphics have been the main way to represent, simplify and make understandable sequence annotation. Most genomes are considered as linear data with other information along the DNA sequence, such as exon, intron, gene, cytogenetic bands and other annotations [see figure 3.10 [17]]. Genomes are usually directly mapped to the linear representations in two dimensions of the visual space, and different types of annotated information are presented as parallel tracks along a DNA sequence [39]. In addition, various glyphs are utilized to give visual hints about the property of information on the tracks.

Bioinformatics programs often follow the three main steps for implementing a graphical representation, which include data acquisition, mapping and rendering [5]. The first step is to acquire data and apply preprocessing operations so that the data is in a format ready to be mapped to visual spaces. The second step is to map preprocessed data from the first step to visualisable geometric shapes with appropriate attributes, such as location, colour and size. The last step is to render final output images to a visible graphic view. For the purposes of this dissertation it is worth mentioning two different approaches to rendering the final output: bitmap graphics and vector graphics.

3.6.1 Bitmap graphics

Bitmap graphics are the most common graphic format in use on the web and, indeed, on the computer . With the exception of Flash and SVG (scalable vector graphic) format, every single graphic seen on the web is a bitmap [13]. As an alternative Java applets, and plugins like Chime [16] have also been used to a certain extent in bioinformatics to display vector graphics on the web. Bitmap

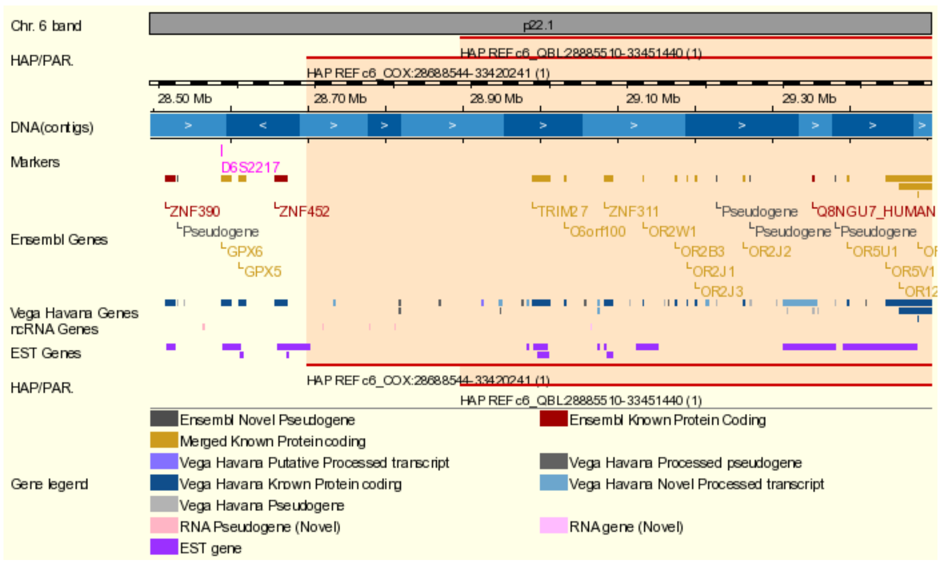


Figure 3.10: Linear representation of genome annotations by Ensembl [17] [17]

graphics are composed of pixels, each of which contains specific colour information. A pixel is minutely small; a single image may be composed of hundreds of thousands of individual pixels. Much like cells revealed from a piece of tissue when seen under a microscope, these pixels are only clearly and individually visible when the image is magnified. A graphic composed entirely of pixels each with its own colour properties is ideal for photographic images where there are thousands, even millions of different colours [13]. Complex fills, shading and gradient effects can easily be rendered.

In bitmap graphics, there is an immutable connection between pixels and the image they compose. When a bitmap graphic is saved, the computer is really saving an exact visual picture of the image: this pixel goes here and is this colour; that pixel goes there and is that colour, etc. This connection is responsible for the effects seen when resizing a bitmap graphic. Given three image sizes - the original, one smaller, and one larger - each will naturally contain a different number of pixels. Pixels do not change sizes, but the image does. It takes more pixels to fill the area of a larger space, fewer to fit into a smaller space. Making an existing bitmap graphic smaller is a process of reduction; pixels are removed from the image until it fits the new size [23]. Computers are well equipped to perform this task. An image can be sized smaller repeatedly and still maintain the same quality, up until the point where there are not enough pixels available to reproduce the image clearly. Increasing the size of a bitmap graphic is similar to pouring a drink from a smaller glass to a larger one. For the drink to occupy the full volume of the larger glass, we must add additional fluid. The original concoction is therefore diluted and the flavour weakened. In the same way, the computer must add additional information (pixels) to the original image to allow it to fill the new larger area.

Since there is no source for this information, it must be interpolated from what is currently available in the image. As such, bitmap images that have been scaled larger are frequently blurry. After extreme size increases, individual pixels "blocks" are more apparent and the image is said to be pixellated [13].

3.6.2 Vector graphics

Rather than being composed of pixels, vector graphics consist of points, lines, and curves which, when combined, can form complex objects [7]. These objects can be filled with solid colours, gradients, and even patterns.

Vector graphics are mathematical creations. For this reason, the programs that are used to create them save instructions on how the image should be drawn, rather than how it looks. This is the key difference between the two types of graphics. Because the computer has a description of how the image should look, it can be redrawn at any size, in any position, without losing any quality. A vector graphic resized to 5 times its original dimensions is simply reproduced, exactly, at the new size. It can also be freely manipulated without losing coherence, like a rubber band that can be stretched an infinite number of ways.

The price of this scaling flexibility is that vector images must remain relatively simple in comparison to Bitmap images. It is impossible to render the nuances of a photographic image in a vector editor; as a result, illustrative vector graphics have a distinct look and feel, even when produced in detail [7]. The colouredHowever, vector graphics are ideal for producing artwork which needs to be presented in different sizes or colours with no loss of quality [23].

Chapter 4

Review of the state of the art in BLAST graphic representation

4.1 NCBI-BLAST and its BLAST graphic representation

NCBI-BLAST is available from the National Center for Biotechnology Information (NCBI). Pre-compiled binaries and source code are available for free and without restrictions. The source code is in the public domain. This has made it easy for other researches to generate other versions of the package. NCBI-BLAST is currently available as a set of pre-compiled binaries for eleven operating system-hardware combinations. The currently-maintained implementation of BLAST is part of the NCBI "C++" Toolkit. The algorithm core is written in "c" and a documented "C++ " API is available. A "c" API that assures compatibility with the "c" toolkit is also available. The source code may be downloaded via anonymous FTP or by cvs [1]. Cvsweb and LXR source browsers are also freely accessible.

Three alternatives exist for BLAST searches:

BLAST web server: The BLAST web server [33], hosted by the NCBI, allows anyone with a web browser to perform similarity searches against constantly updated databases of proteins and DNA that include most of the newly sequenced organisms. There are two alternative ways to access the BLAST engine working within the web server:

BLASTcl3 network client: BLASTcl3 is a BLAST client software from NCBI [26]. This program runs on a local machine and uses the BLAST web server at NCBI to perform BLAST searches of the NCBI sequence databases.

Script driven URL API: Users can use URL encoded commands to interact with the NCBI BLAST server to perform searches through a browser interface or directly through a script without the browser.

BLAST standalone: This application is for users who wish to run stand-alone BLAST at their own computer or institution. One reason to do so might be the desire to use private databases not available at the NCBI.

wwwBLAST: This refers to a set of stand-alone BLAST programs/web forms, which, as well as performing BLAST alignments locally, uses web forms as a graphical user interface (GUI) and offers better security (within a protected environment) [22].

[1]More information at "http://www.ncbi.nlm.nih.gov/BLAST/"

Only the BLAST web server and the wwwBLAST program present graphics with their result sets. The source code of these two types of BLAST differ slightly from each other and thus there are some differences between the result sets that they produce. By comparing the BLAST web server and the wwwBLAST graphic representation we find that the former provides better and more elaborate graphics. Although the NCBI BLAST web service is publicly available, the graphics tool implemented in the BLAST server cannot be used outside of the NCBI servers due to the unavailability of the code for the graphics. As such, it can be only used from the NCBI servers.

When BLAST reports are long it is difficult to quickly grasp a general and comprehensive view of the results. Viewing them graphically can help a user to understand them better [Figure 3.9 shows the result from the NCBI web server]. The coloured bars in the graphic summarize the BLAST results. At the top is a linear map of the query. Each bar drawn below the map represents a nucleotide or protein sequence that matches the query sequence. The position of each bar relative to the linear map of the query, allows the user to instantly see the extent to which the database matches align with a single or multiple regions of the query. The most similar hits are shown at the top in red. Pink, green, blue and black bars follow, representing sequences in decreasing order of similarity. Hatched areas (when present) correspond to the non-similar sequence between two or more distinct regions of similarity found within the same database entry. Moving the mouse over the bars displays the name of the matching protein found in the textbox above.

As an example we show a BLAST result where an EST (Expressed sequence tags of cDNA clones) from Trichoderma virens [2] is compared against the protein Genbank database using BLASTx in the BLAST web server.

Pink bars represent alignments for the last 300 nucleotides of the EST. The similarity is measured by the score value. The values for the pink colour are narrowed down from 80 to 200 units showing high similarity alignments with "cell wall glucanase" and "hypothetical" proteins. Bars in black represent low similarity regions [see figure 4.1].

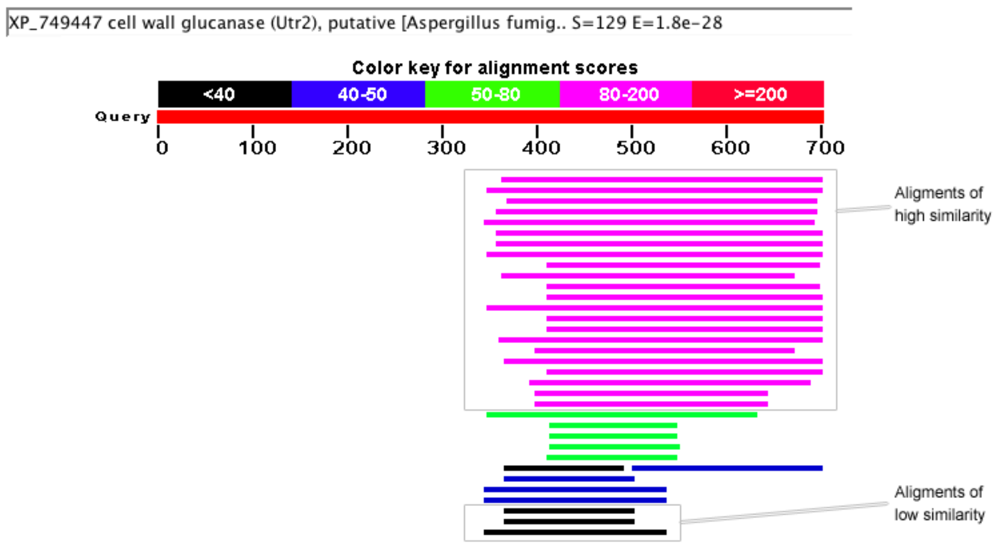

XP_749447 cell wall glucanase (Utr2), putative [Aspergillus fumig.. S=129 E=1.8e-28
Color key for alignment scores
<40
40-50
50-80
80-200
>=200
Query
0
100
200
300
400
500
600
700
Aligments of high similarity
Aligments of low similarity

Figure 4.1: EST gi:123226991. NCBI BLAST result

[2]gi—123226991—gb—EH628721.1—EH628721 EST00333 Trichoderma virens IMI 304061

4.2 GMOD and its BLAST graphic representation

The Generic Model Organism Database (GMOD) Project is an open source project that aims to develop a set of software tools for creating and administering a model organism database. Components of this project include genome visualization and editing tools, literature curation tools, a robust database schema, biological ontology tools, and a set of standard operating procedures.

The BLAST Graphic Viewer (BGV) tool is a component of the GMOD software for sequence alignment visualization. It provides a graphical visualization of the regions contained within High-scoring segment pairs (HSPs) relative to the query sequence to represent BLAST results [see figure 4.2]. This distribution includes sample databases and a BLAST Search form.

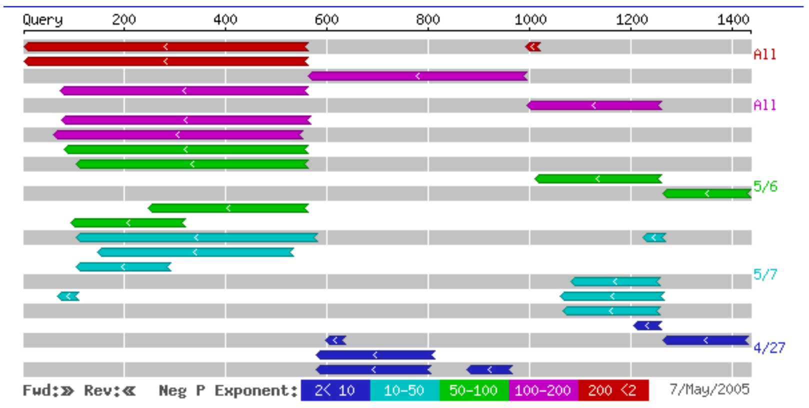


Figure 4.2: Example of a BGV graphic representation

The BGV tool is a Perl library which can be used as an independent library or as a part of the BioPerl library. It comes with two main packages supported by 18 small sub-packages. BGV makes use of the BioPerl library to parse and get results from the BLAST output. This information is then used to generate bitmap files making use of the GD graphic library [3]. BioPerl has different modules to work with BLAST locally as well as remotely.

BGV doesn't support most of the common BLAST programs mentioned in this dissertation. It just provides graphic visualization for a specific version of BLAST called WuBLAST.

[3]GD is an open source code library for the dynamic creation of bitmap images by programmers.

Chapter 5

Critical review of existing approaches

5.1 Visual design

NCBI BLAST graphics and BGV show an overview of all the HSP results in a compact space using positional rectangular shapes of different length and colours which depict the matching similarities. This overview makes the BLAST results easier to understand, navigate and access. Although the conceptual design and the graphics are good, they are not optimal. By better utilizing the rules of graphic design and usability, BLAST graphics could be improved and made more readable, interactive, appealing and usable.

Visualization is especially potent in promoting the intuition and insights that lead to breakthroughs in understanding the relevant connections among salient features [6] and effective presentation of results is critical to decision-making [5]. In graphical user interfaces visual representations help users to make sense of complex data [32].

Graphic design is the first and the last part of the user interface observed by the user [32]. Graphic design can significantly improve the communicative features of the interface leading to increased usability. System usability has many components including ease of learning, efficiency of use, memorability, reduced number of user errors, and subjective satisfaction. A good graphic design can improve all these attributes. [32].

Graphic design is the act of arranging text and visual elements to communicate a message to the viewer. Graphic design facilitates the delivery of visual logic and seeks an optimal balance between visual sensation and graphic information. Without the visual impact of shape, colour, and contrast, pages are graphically uninteresting and will not motivate the viewer. Visual and functional continuity in content organization, graphic design, and typography are essential to convince the audience that the system offers timely, accurate, and useful information [32]. A careful, systematic approach to page design can simplify navigation, reduce user errors, and make it easier for readers to take advantage of the information and features of the site. Nonetheless achieving maximum leverage for each element in a design requires a thorough understanding of the communication problem at hand in addition to the design elements at the viewer's disposal [32].

Mullet and Sano succeed in demonstrating that graphic design is not a black art but a very engineering-like discipline with its own rules [32]. Also as in other types of engineering, there are systematic steps that can be taken to improve the visual design characteristics of an interface. In this section we aim to analyse some graphic design concepts of the BLAST graphic representation applying some basic GUI visual design techniques [32].

5.1.1 Colour and contrast

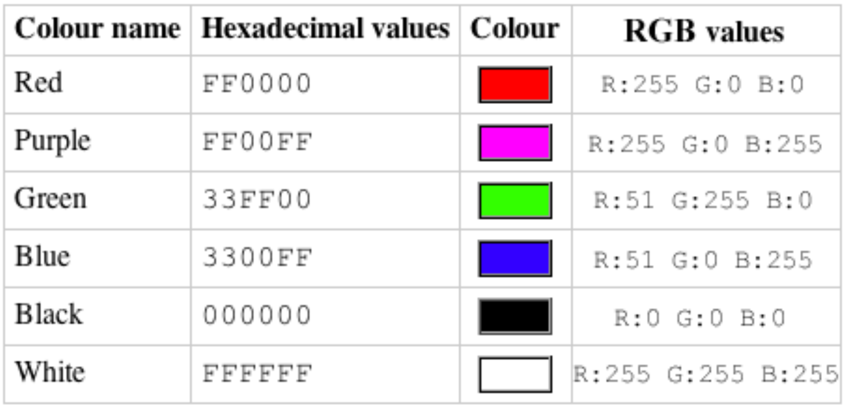
While the choice of colour is one of the most subjective decisions in graphic design, it is still important to understand the theory behind why some sets of colours should be chosen over others. The theory may not enable instant selection of a colour scheme, but it certainly leads a designer towards good choices. The colour theories of Chevreul, Bezold, Albers and Itten demonstrate how it affects design, so a designer can use them to advantage, moving a viewer's eye through a visual design [31]. Colour can enhance communication, but only if it is used correctly [34].

The limitation of a small colour palette have long been compounded by the tendency to fill the colour table with colours that are easy to describe digitally. This practice inevitably produces the familiar range of intense, over-saturated colours that still predominates in many computers displays [32].

NCBI uses a limited colour palette [see figure: 5.1] not taking advantage of higher standard colour resolutions in its interface. Five colours are used to represent different scores for the alignments, and, excluding the black, the rest of them use an extreme value in the RGB colour palette. Moreover the graphic presents absence of colour harmony which makes the graphic unappealing to the end user [32]. This neglects visual balance and visual contrast between colour combinations [32].

The relationship between a figure and its surrounding field portrays a level of contrast; the more an object contrasts with its surrounds, the more visible it becomes [38]. Contrast provides the basis for visual distinctions, which are the building blocks of meaning in a visual message [19]. In the NCBI graphics the use of green and purple figures on a white background yields a graphic with poor colour contrast [see figure 5.2]. Using a thin border around the shape would help to create contrast between the background and the shape. Less extreme values in the RGB range would help visibility and harmony on a light or white background. The graphic delivered by BVG considers these concepts of colour and contrast mentioned above, considerably improving the quality of colour harmony.

Colours, shades and lines in the background can create contrast to draw at-



Colour name	Hexadecimal values	Colour	RGB values
Red	FF0000	🟥	R:255 G:0 B:0
Purple	FF00FF	🟪	R:255 G:0 B:255
Green	33FF00	🟩	R:51 G:255 B:0
Blue	3300FF	🟦	R:51 G:0 B:255
Black	000000	⬛	R:0 G:0 B:0
White	FFFFFF	⬜	R:255 G:255 B:255

Figure 5.1: NCBI hexadecimal colour values

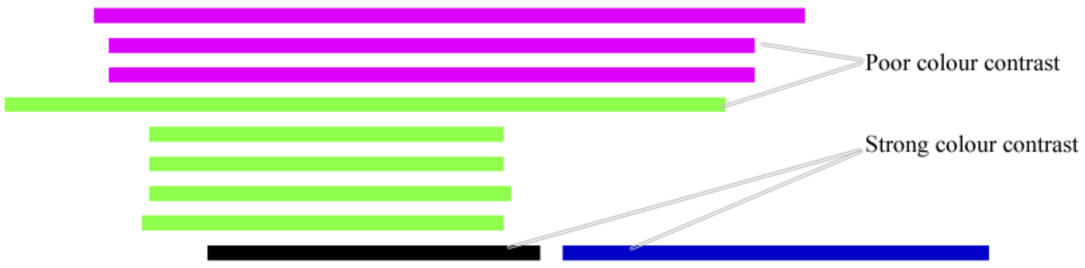


Figure 5.2: Example of poor colour contrast

tention to groups of related information as well as positioning them in the space. Regular positional and grouping elements allow the viewer to more easily spot areas of interest when making comparisons. In the NCBI display we cannot find any such elements of contrast.

5.1.2 Organization, simplicity and clarity

Effective graphic design for communication purposes is about organizing and classifying content in a clear and simple way so that it can be easily found and read or viewed [32]. The way the pieces are put together is usually as important as the pieces themselves. In a simplistic way critical information should be near the top of the site and information should be shown by level of importance [38].

In the case of NCBI graphics the layout of its components follows organizational rules showing the alignments with less relevance at the bottom of the graphic. However, as the legend contains additional and non critical information for the user, the legend (which describes the qualitative meaning of the colours) should be positioned at the bottom.

Simplicity is an issue of visual communication, considering how much detail it is necessary to convey [38]. The necessary information should be available without burdening the user with unnecessary information. This implies that redundant information should be avoided and all information related to one topic should be grouped together [38].

At first sight it seems NCBI graphics succeeds because it simply represents HSPs using quantitative and qualitative rectangular bars as part of the graphic. Nonetheless if the graphic is examined in depth and the meaning of alignments understood, it can be seen that this approach over-simplifies the representation. It does not make effective visual distinctions among HSPs, grouping all together in the same line and in the same figure if they overlap [see figure 5.3]. One sequence can be aligned with different fragments of another sequence producing several HSPs. It might confuse the user to see one rectangular bar when the display should show two or more. The graphic should not exclude non redundant information just because it overlaps. In this case HSP overlapping should be displayed in a different line. BVG makes this distinction among HSPs and thus improves the representation of BLAST alignments.

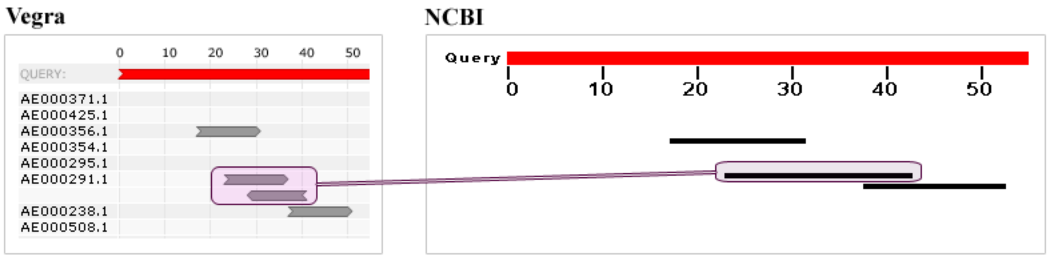


Figure 5.3: Two alignments found in different open reading frames for the same sequence overlaping partially in the same region. On the right the NCBI graphic representation doesn't show there are two HSPs, instead it groups them in the same line. On the left the same representation created by VEGRA shows two HSPs.

Clarity is another important goal of effective communication. Clarity is not the same as simplicity. Simple things are clear if the message is intended to be brief and small, but often the message is about a complex relationship that can only be presented with a necessarily large amount of data [32]. This complexity can be made clear by means of effective organization of the resulting repressentation.

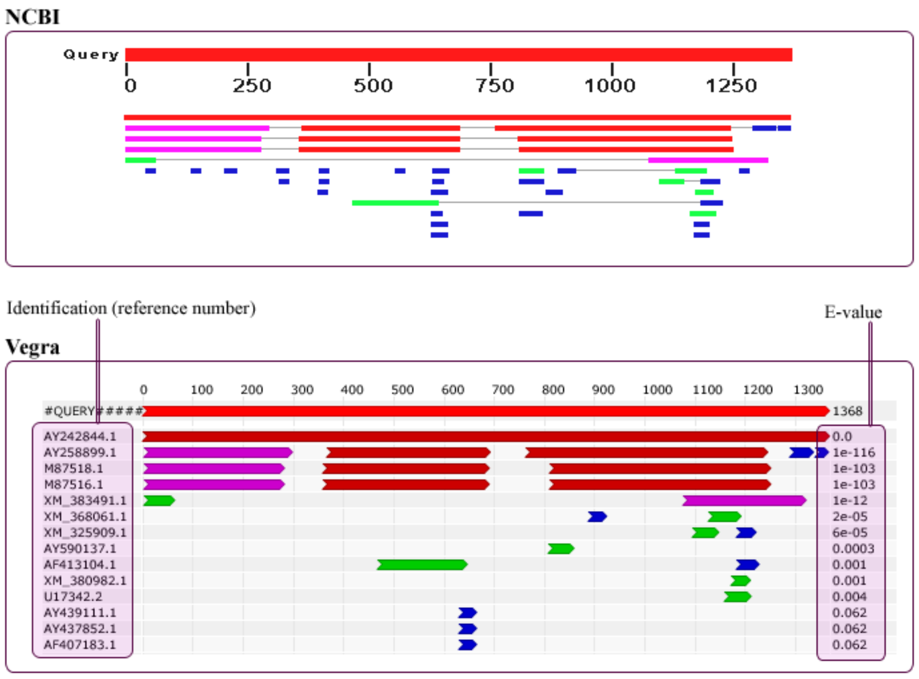
The NCBI aims for simplicity by avoiding text and references in its main display. Identification references for the sequence, the similarity value measuring the quality of the alignment and a brief description of the sequence can only be accessed in the NCBI tool by mousing over each sequence individually. At first sight there is no way to know which protein or nucleotide sequences the graphic displays. Lack of identifications for the linear representations makes the graphic difficult to understand since the user cannot identify and compare the protein or nucleotide alignments drawn in the main graphic [see figure 5.4]. Hence this approach does not clearly convey the message to the user. In terms of usability the user would save more time having an overview of all the important values and a short reference for each sequence representation at once. The simplified way adopted by the NCBI graphics requires the user to memorize and find out information about the alignments one by one.

5.2 Graphic format

Bitmap graphics are well differentiated from vector graphics. Vector graphics are better for design and layout, typography, logos, technical illustrations, and data representation while bitmap graphics are more suitable for retouching, photograph processing, and artistic illustrations [7].

Vector-oriented images can be resized and stretched without any loss in quality or distortion. Additionally, they look better on devices (monitors and printers) with higher resolution. Another important characteristic of vector graphics is that the representation of images often requires less memory than bit-mapped images.

The NCBI graphic representation has been using the bitmap graphic format since its first release [26]. The BLAST web server displays the text BLAST results at the bottom of the visible area with a graphical representation of the results at



AY242844.1	0.0
AY258899.1	1e-116
M87518.1	1e-103
M87516.1	1e-103
XM_383491.1	1e-12
XM_368061.1	2e-05
XM_325909.1	6e-05
AY590137.1	0.0003
AF413104.1	0.001
XM_380982.1	0.001
U17342.2	0.004
AY439111.1	0.062
AY437852.1	0.062
AF407183.1	0.062

Figure 5.4: The NCBI graphic representation on the top doesn't show identification references nor similarity values. On the bottom: an example of VEGRA showing these values for each line of alignments.

the top. The image file is a bitmap using the GIF format.

BLAST graphics were originally designed for web browsers which did not support vector format graphics, and GIFs were the most appropriate solution for this work. Developers were limited to a narrow set of web browser functionalities that did not support vector graphics. However, due to the increasing popularity of web-based applications during the last few years, improvements for browsers and the web have provided powerful scripting and event support thus enabling the use of vector graphics as a platform upon which to build graphically rich applications and user interfaces.

5.3 Dynamic actions

In general, bitmap graphics do not provide dynamic actions by themselves; however, within the context of web technologies it is possible to add dynamic effects to images using Dynamic HTML (DHTML). DHTML is a method for creating interactive web sites using HTML, JavaScript, Cascading Style Sheets (CSS), and the Document Object Model (DOM). BLAST graphics primarily use JavaScript to perform two basic graphic dynamic actions. By moving the mouse over specific colour bars the user can get direct access to information about the sequence such as ID reference, description and E-value. If the user clicks on the bar instead the graphic will redirect the user to the corresponding text alignment.

A disadvantage of DHTML is its difficulty to develop and debug; this is due to the different degrees of support across different web browsers. The main drawback of this technology when applied to BLAST graphics, is that the definition of dynamic actions are not in the graphic but in the code of the HTML page. This means that the image file by itself will not keep the interactive properties. The dynamic effects will only persist while the graphic is used in its HTML as part of the code.

In terms of usability the NCBI graphics do not show which parts are clickable to users. The shapes representing information which contain actions (e.g. links to other information) should be easy to notice when mouse-over actions are performed. These shapes should be defined as buttons where the colour changes upon mouse-over. Because BLAST graphics are bitmaps, those variations of colour in the graphic would require a new request to the server in order to create a new graphic. This would certainly slow down the communication with the user. BGV follows the same disadvantageous approach in therms of usability.

5.4 Customizable graphics

BLAST graphics do not allow users to change basic properties of the graphic such as the size and scale of the panel, colour of the background, the possibility to show or hide some specific information in the graphic, etc. There is no other way to achieve customization than by modifying the code itself. BGV similarly does not support changes in the basic properties of the graphic.

5.5 Availability

The BLAST web server does not allow users to access the source code used to generate graphics from BLAST results. The NCBI BLAST graphic representation can only be accessed on-the-fly via the NCBI servers. As the NCBI databases are being updated on a daily-basis this might be considered to be a disadvantage because it is not always possible to reproduce results and thus get the same graphic representation at different points in time. This is also a concern for researchers who want to work with private or local data since the only way to get this type of representation would be to make their data publicly available.

wwwBLAST supports graphics and makes its code available. However its performance is significantly worse than that of the NCBI BLAST server tool.

BGV improves visualization of BLAST results. However it only works with WuBLAST, and not other BLAST programmes.

5.6 Documentation and structure of the code

The source code available for BLAST graphics is not sufficient. There are no UML diagrams, guidelines or schemata to facilitate the understanding of the organization and content of the code. The code has no clear unified methodology and does not follow an object-oriented modelling approach. The same is true for BGV, however, its scripting language makes it much more understandable and easier to manipulate.

Chapter 6

Materials and Methods

6.1 Vector graphics on the web

The World Wide Web's enormous success has been achieved by marking Web documents with a simple set of HTML tags to denote basic structural and layout concepts such as bold headings, tables etc. Perhaps the most striking weakness of today's Web technology is the legacy of previous restrictions on the web. For instance, all graphic material had to be expressed in one of three raster-based formats (GIF, JPEG, PNG). Such formats distort badly at any scaling factor other than 1:1, due to the pixel-by-pixel way in which the image data is stored. Moreover, these formats are innately non-searchable and this is a severe limitation given that a common use of graphics on the Web is for representation of annotated maps and line diagrams, together with exotic characters and logotypes that lie outside conventional character sets.

In recent years Macromedia's Flash and its associated Shockwave format (SWF) have achieved a degree of acceptance as a standard for displaying vector graphics on the Web. More recently the World Wide Web Consortium (W3C) adopted a new standard named "Scalable Vector Graphics" (SVG) as a technology for deploying graphical content over the web.

Today there are three alternatives for visualizing vector graphics: SVG, Flash and PDF. Nonetheless, since PDF was not designed for the word wide web, SVG and Flash are the two main technologies used for vector graphics on the web.

6.1.1 Flash

Macromedia's Flash format is currently the most widely used vector graphics and animation format on the Web [35]. Flash graphics (and movies) are distributed as SWF files, a compact binary file format that requires an additional browser plug-in to be available in order for the graphic to be viewed. Objects in Flash graphics are described by standard vector graphics primitives such as lines and curves. Bitmap objects can be included in Flash files, all the objects may be animated, user interactions can be specified and sound can be streamed into movies [35].

Flash files are created by using the Flash commercial editor available from Macromedia, which generates files with a FLA extension. These files are editable by Flash tools and the objects, images and animations can be altered. Flash movies for Web distribution are turned into its final form by exporting them as SWF files from within the editor. This format is not editable and is a compact format suitable for Web distribution. The SWF format defines shapes, these are then placed onto a stage by using a transformation matrix to scale and position them. The transformation matrix can be altered from frame to frame to create animation effects.

6.1.2 SVG

Scalable Vector Graphics (SVG) is an XML markup language for describing two-dimensional vector graphics, both static and animated (either declarative or scripted) [7]. It is an open standard created by the World Wide Web Consortium, which is also responsible for standards such as HTML and XHTML. The use of SVG on the web gained support in recent years, but most browsers still require special plugins and its performance is not as efficient as is Flash [23]. In order to view SVG documents in a Web browser, a viewer needs to be installed. There are several viewers available but the Adobe SVG Viewer is probably the most commonly used and can be downloaded from the Adobe web site [24].

6.1.3 PDF

PDF was not primarily designed for the world wide web, yet it may be used with the help of plugins [1]. Its viewer, Acrobat Reader, free courtesy of Adobe, operates independently from the browser and it is available for multiple platforms.

6.1.4 Flash vs. SVG

Browser support

In most browsers a plugin is needed to view SVG content. The most widely available SVG plugin is from Adobe Systems (the Adobe SVG Viewer) which supports most of the SVG 1.0/1.1 specification. Curiously, Adobe Systems, the first supporter of the non-proprietary technology "SVG", acquired Macromedia and its product portfolio (including Flash) in 2005 [36]. Flash is widely supported by most browsers and firmly established after 10 years of deployment on the Internet [13].

Open standard formats

While SWF is not an open standard like SVG [35], Macromedia (now Adobe Systems) does make it available to the public, and many commercial and open source tools exist for generating and parsing SWF files. It can be easily created from data described using XML syntax, or any other syntactic structure for that matter, by using libSWF, Ming, FreeMovie, or even the freely available SWF C++ SDK from Macromedia [3].

Bandwidth efficiency

SWF can carry more encoded data than SVG using the same number of bytes [23]. Compared to SWF, SVG is bandwidth intensive which is a concern in limited bandwidth environments such as South Africa.

6.1.5 SWF and Ming

The Macromedia Flash File Format (SWF) has become the *de facto* standard for vector graphics on the Internet [3]. Although there is considerable interest in SVG, developers are still waiting for the tools to match the functionality offered by Flash [3]. The SWF file format has been tested on the Internet over an extensive period, while SVG is still a relatively young and less tested technology. Given these factors in addition to the previous review we have chosen SWF as the preferred vector format for this work.

There are some freely available libraries which can be used to write SWF files without the need to use any commercial tools like Macromedia Flash. The use of such a library is a significant help in the development of a method to automatically create SWF files from input data within a web server environment. We selected Ming as the SWF library for this work as it is widely supported [40] and provides complete documentation and many examples.

6.1.6 Ming

Ming is an open-source (LGPL) C library for generating SWF (”Flash”) format movies, as well as a set of wrappers for using the library from C++ and popular scripting languages such as PHP, Perl, Python and Ruby [40]. Ming was initially written in 2001 and has been under continuous development ever since. Version 0.3 was released in February 2006 [1]. The Ming project provides a set of tools and a library that has allowed the author to avoid using commercial authoring software during the development of this research work [29].

6.2 Working with BLAST results

Several versions of BLAST are available and the resulting formats for each version are slightly different. The BLAST graphic library presented here is independent of the BLAST version used. BioLibraries such as BioPython and BioPerl provide good solutions for result parsing as they have classes and functions that are updated frequently to work with different formats and versions of many programs used in computational biology, including BLAST.

[1]For more information ”http://ming.sourceforge.net/”

6.3 Programming a BLAST graphic library, why Python?

It is convenient to use the same programing language for retrieving information from BLAST outputs and for creating the graphic files by using a graphic library. Ming's provision of several different language wrappers is appealing as it allows for flexibility with a wide range of BioLibraries that are available in different languages. PHP may seem to be a good alternative when seeking to integrate the tool with a web application, however the Biolibrary in PHP is not as well supported as are BioPython, BioPerl or BioJava. Java is another alternative, as it is a consistent and reliable programming language, however the Java wrapper module for Ming is still relatively immature. BioPython was chosen over other BioLibraries due to its Object Oriented Model structure, BLAST support and principally due to the author's better understanding of the Python language.

Chapter 7

Results

7.1 The Vegra library

VEGRA BLAST stands for VEctor GRAphics for BLAST. It is an Object Oriented Python Library to create graphical representations from BLAST results. It takes the output of BLAST programs as input and produces SWF files as output. VEGRA uses BioPython to parse BLAST outputs and relies on Ming to create the SWF files. The SWF files generated by VEGRA can be integrated in the HTML body of the BLAST results together with the traditional BLAST bitmaps graphics [see figure 7.1].

The VEGRA library is composed of a collection of Python files (as well named modules). These files are organized into classes which provide modularity and structure in the object-oriented program. The classes have been organized into three groups according to level of abstraction and complexity. These groups are "Components", "Layers" and "Main class".

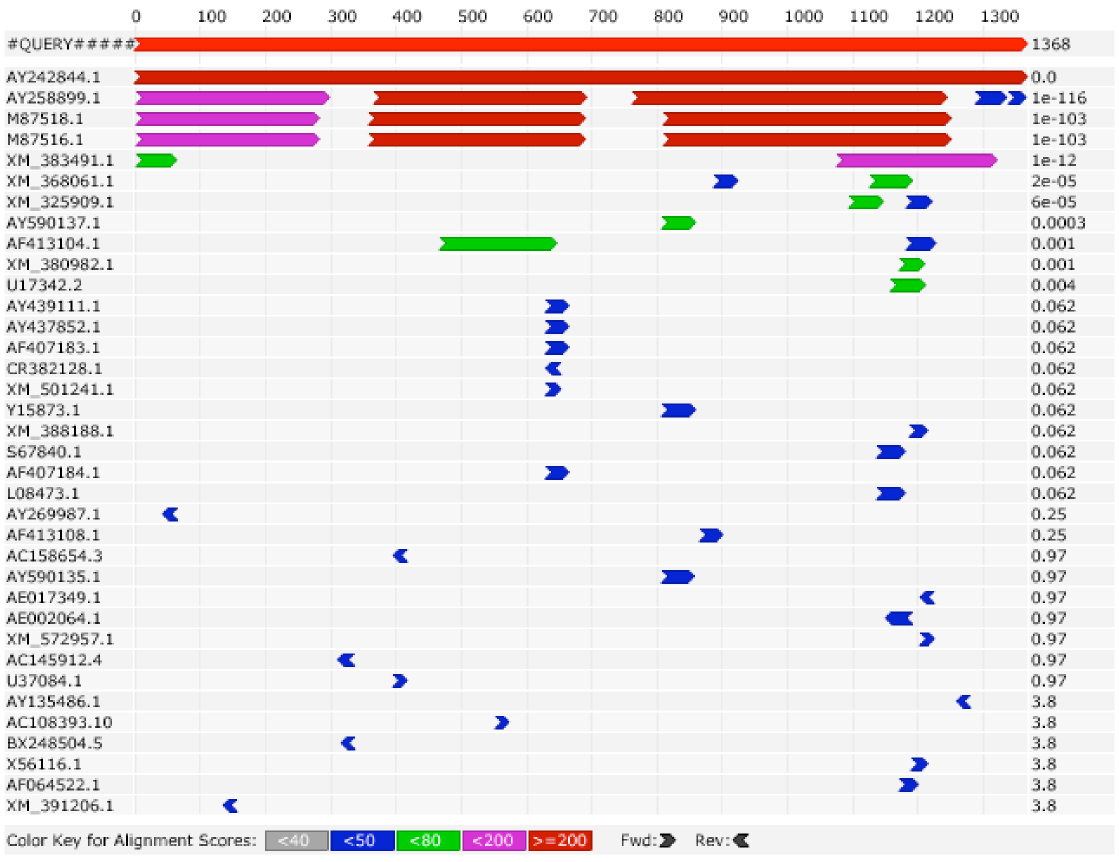


Figure 7.1: Vegra SWF graphic example

Groups of classes in VEGRA BLAST [see figure 7.2]:

- The components group consists of a group of unspecialized classes which help to define simple objects and perform basic functions that are often used in the library.
- The layers group contains classes that define specific parts of the graphic. These classes are responsible for creating the layout object of the graphic.
- The main class group consists of only one class which brings together all the layout objects to create the final SWF output file.
- There is an extra module which is not classified in these groups. This is the initialization file, responsible for starting VEGRA.

7.2 Groups of classes in the VEGRA library

7.2.1 Main class

BLASTGraph()

This is the only operational class capable of creating functional SWF files in VEGRA. It makes use of layer and component objects for setting the properties and structure of the final BLAST graphic. The code for this class defines and places the layer objects throughout the graphic panel distributing them from top to bottom in the following order: boxinfoLayer, queryLayer, alignmentLayer and legendLayer; and from left to right in the following order: ID frame, alignment frame and E-value frame [see figure 7.3].

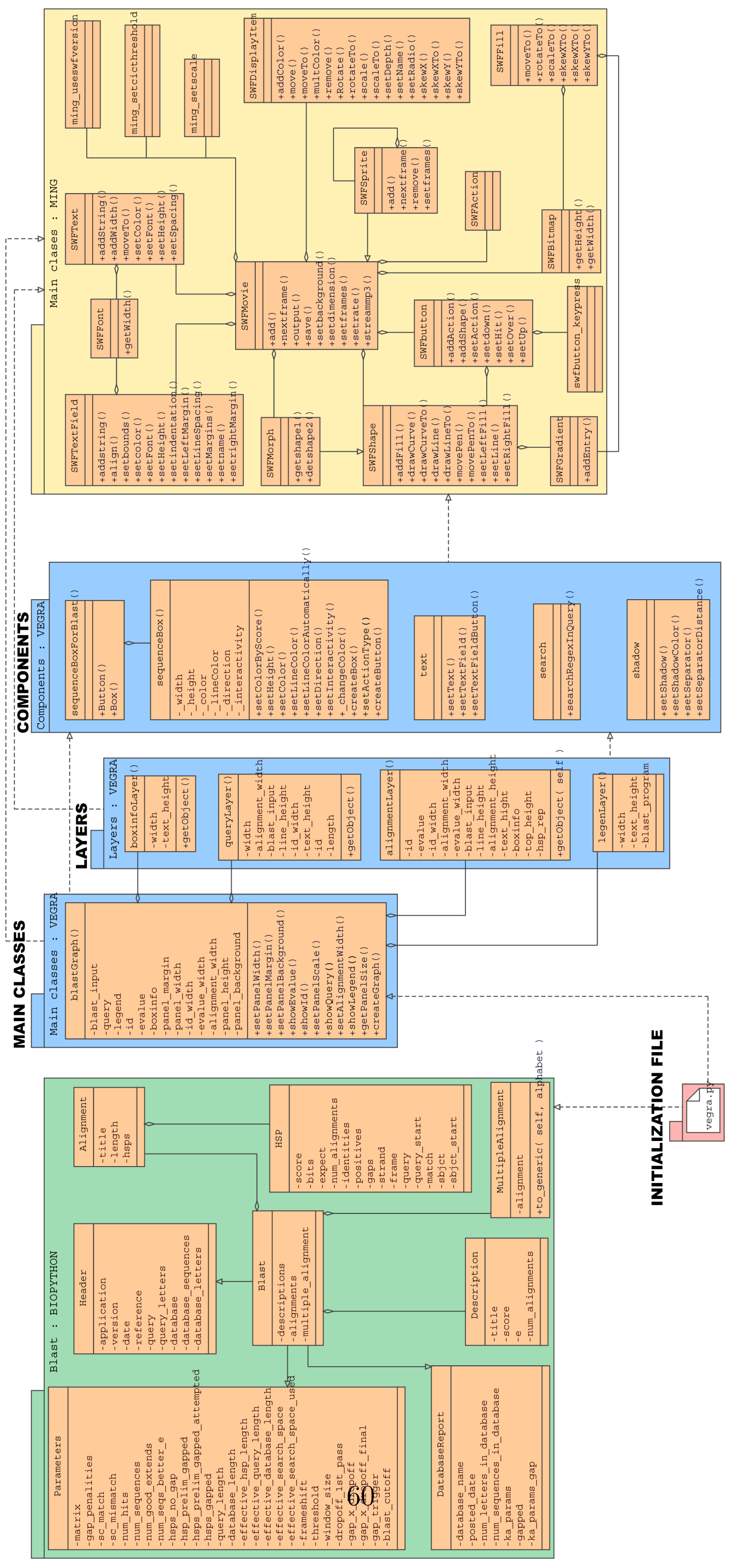


Figure 7.2: UML class schema of Vegra and its interaction with the Ming library and the BLAST classes from BioPython

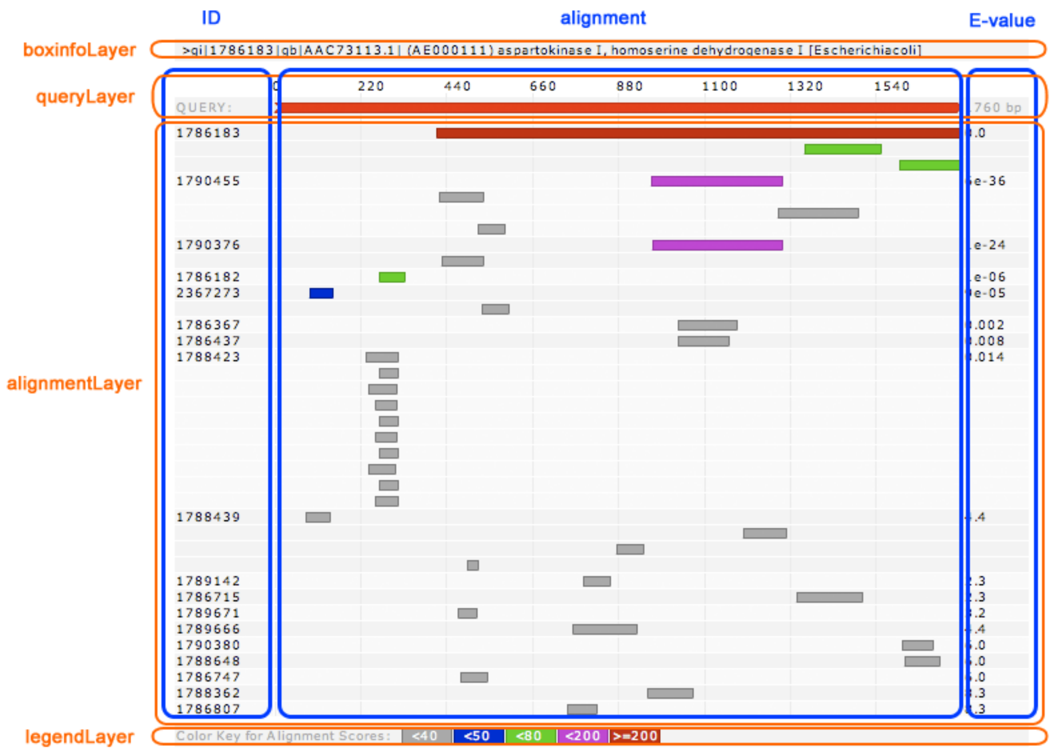


Figure 7.3: Distribution of graphic layers and frames in Vegra

This class contains functions that allow the hiding of different elements of the panel except the alignmentLayer. All of the frames can also be hidden, except for the alignment frame. When a layer is hidden, the BLASTGraph class skips the creation of the object, rearranging the position of the rest of the layer object. If a frame is hidden, the BLASTGraph class will create the layer objects giving instructions to the layer class to hide this frame. As such, the presence of a layer is dependent on the BLASTGraph class and the presence of the frames is dependent on the layer classes.

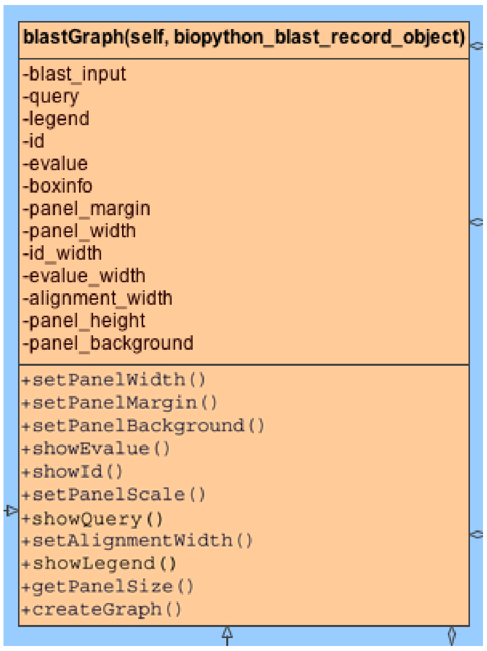


Figure 7.4: BLASTGraph class

It is possible to control the width of the panel which in turn will affect the width of all the components inside the panel. However, it is not possible to define its height as this value is dependent on the number of alignments in the graphic. While the width is known in advance the height cannot be known until all the information to create the graphic is generated.

Arguments given to the class:

- BioPython BLAST record object

Methods:

- The ”setPanelBackground” function sets the background of the panel by RGB format. Default: (R=0, G=0, B=0).
- The ”setPanelWidth” function sets the width of the panel in pixels. Default: 650.
- The ”setPanelMargin” function sets the panel margin in pixels. Default: 5.
- The ”showLegend” function determines if the Legend layer is going to be shown (True) or not (False). Default: True.
- The ”showQuery” function controls the visibility of the Query layer. Default: True.
- The ”showBoxinfo” function controls the boxinfoLayer visibility. Default: True.
- The ”showId” function determines whether the ID frame is going to be present or not. Default: True.
- The ”showEvalue” function determines whether the Evalue frame is going to be present or not. Default: True.
- The ”setAlignmentWidth” function is used in the BLASTGraph class to recalculate the width of the alignment layer considering the width of the panel, the width of the margin, and the presence or absence of the ID and Evalue frames.
- The ”getPanelSize” function helps to determine the size of the panel (width and height) after executing the function that creates the graphic. This func-

tion is useful for recovering size information of the SWF file which is necessary in order to accommodate the graphic in HTML pages.

- The "createGraph" function is the only compulsory function that must be invoked to be able to create a SWF object. This function requires a name for the SWF output file.

7.2.2 Layers

boxinfoLayer()

The boxinfoLayer class is responsible for creating a text field on the top of the graphic to display information about the HSPs. Once the graphic is displayed, mouse-over actions on the alignments representation cause the boxinfoLayer object to show information about the alignments selected.

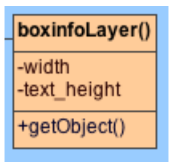


Figure 7.5: boxinfoLayer class

Class overview:

Arguments given to the class:

- width
- evalue width
- alignment width

Methods:

- The "getObject" must be invoked to create the object

queryLayer()

The queryLayer class is responsible for creating the information field and the positional values for the query sequence. It represents the query sequence with a red bar along the horizontal edge of the header. If the query is a nucleotide sequence it is represented by an arrow and if it is a protein it is represented by a box. The code in this class creates a scale of eight thin vertical lines under the horizontal bar showing position values along the sequence.

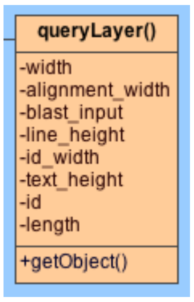


Figure 7.6: queryLayer class

The query information is retrieved through a BLAST record BioPython object given as one of the default arguments in this class. Other compulsory arguments are the presence or absence of the ID frame and the E-value frame, the width of these two frames and the width of the alignment frame. Like the previous class, it returns a sprite Ming object and getObject must be called to create it.

Arguments given to the class:

- BioPython BLAST record object
- id
- evalue
- id width
- evalue width
- alignment width

Methods:

- The "getObject" must be invoked to create the object

alignmentLayer()

The alignmentLayer class deals with the aligment representation. The alignments are represented by horizontal bars relative to the linear map of the query. If the alignments are nucleotide sequences the bars will be represented by a directional arrow; if they are protein sequences they will be represented by a non-directional box [see figure 7.8]. For a nucleotide alignment, if the beginning of the sequence aligned starts at the end of the alignment the arrow will point to the left, otherwise it will point to the right [see figure 7.8].

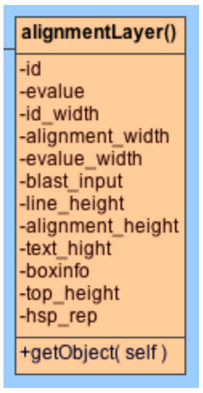


Figure 7.7: alignmentLayer class

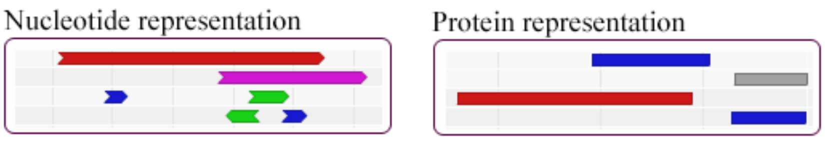


Figure 7.8: On the left an example of directional arrows representing nucleotide alignments. On the right an example of rectangular bars representing peptide alignments.

The query sequence can have more than one alignment with one sequence of the database. A background (shade) of the same colour will indicate that the alignments still belong to the same sequence. A change in the background colour will indicate a new sequence alignment or group of alignments.

The colours of the alignments indicate the degree of similarity based on the Score of each alignment. The range of colours and their correlation with the Score is shown at the bottom in the legend field. The most similar hits of each sequence are shown at the top in red, followed by purple, green, blue and grey representing sequences in decreasing order of similarity following the same colour pattern used by the NCBI BLAST graphics. Moving the mouse over the bars displays the name of the matching protein found in the boxinfo layer above the graphic.

If the E-value frame option is selected, the graphic will show the E-values for each sequence. If the ID frame option is selected, the graphic will show in this frame the ID number of every HSP match. A mouse click on the ID field will open a web browser window which will display the full database entry corresponding to this ID. The database queried in this case is the Genbank database [4].

Arguments given to the class:

- BioPython BLAST record object
- id
- evalue
- id width
- evalue width
- alignment width
- boxinfo
- top height

Methods:

- The "getObject" must be invoked to create the object

legendLayer()

The legendLayer is responsible for creating a horizontal layer of information which explains the significance of the colours and the directions of the alignments (just if the alignments are nucleotides) at the bottom of the graphic.

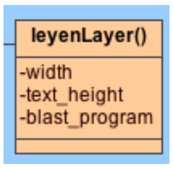


Figure 7.9: legend-Layer class

Arguments given to the class:

- id width
- evalue width
- alignment width
- BLAST program

Methods:

- The "getObject" must be invoked to create the object

7.2.3 Components

Functions

Some components of these libraries were not complex enough to write classes and the definition of a function was more than sufficient to perform the required task.

These functions are classified in three groups (text, shadow and search). ”Text” allows the definition of three kinds of Ming text object. ”Shadow” includes different functions to define the shades of the graphic as well as functions to define the scale. Search helps to parse text using regular expressions.

Text functions:

1. The ”setText” function returns a simple Ming object which provides the text content, the text height and, optionally, the colour in RGB format.

2. The ”setTextField” function is similar to the setText function. However, it returns a text field object with text content. It uses two arguments (shape and ID name).

3. The ”setTextFieldButton” function is the same as the setTextField function but with button effect. The button action causes text to change its colour on mouse-over and mouse-click actions. Moreover, it optionally allows the embedding of specific mouse events with ActionScript code.

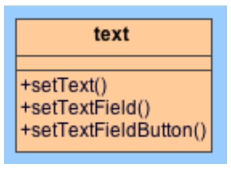


Figure 7.10: text module

Shadow functions:

1. The ”setShadow” function returns a Ming object (shape object). This function takes as arguments the width and height of the shape as well as its colour.

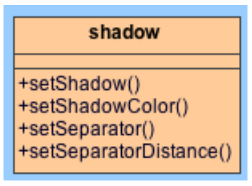


Figure 7.11: shadow module

2. The ”setShadowcolor” function is a loop function that controls the background colour for each alignment. A pattern is repeated by alternating two colours. By default these are grey and white.

3. The ”setSeparatorDistance” function calculates the scale distance. It works by dividing the sequence length among the numbers of bar lines shown. The function converts the separation to a multiple of 10 by adjusting the differences in the last segment. Multiples of 10 segments make it easier for the user to locate positions.

4. The ”setSeparator” function returns a Ming shape object representing a thin vertical line. It uses two attributes (high value and a RGB colour format).

Search functions:

1. The ”searchRegexInQuery” function enables search by regular expression. It returns an array or list of all the queries matched.

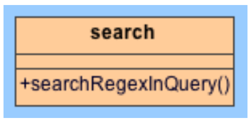


Figure 7.12: search module

Classes

The components package includes two classes related to the query and HSP representation.

sequenceBox(): The sequenceBox class includes a collection of functions used to define and create a Ming shape object to represent rectangular boxes.

There are two ways to define the colour of the box:

1. setcolourByScore: set the colour of the Box by score.
2. setcolour: sets the colour of the Box in RGB format.

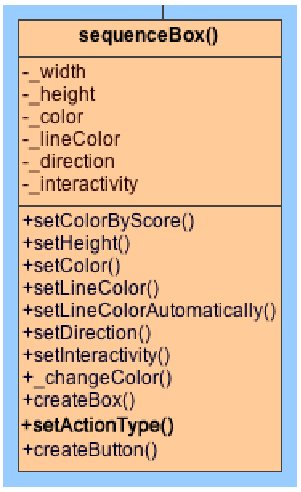


Figure 7.13: sequenceBox class

There are also two ways to define the border line of the box:

1. setLineColor: sets a specific RGB colour.
2. setLineColorAutomatically: automatically sets a RGB colour that is darker than the default box colour.

The rest of the functions are:

1. setHeight: defines the height of the Box.
2. setInteractivity: allows the addition of ActionScript code to add interactivity to the Box in case it is a button.
3. setActionType: sets the event triggering the interactive code. By default it is ”mousedown”.

4. changeColor: is used for the sequenceBox class for the setLineColorAutomatically function in order to help to get the new colour.

After all these functions above have been defined, one of the following functions must be instantiated to create the Box object:

1. createBox: creates a simple box.

2. createButton: creates a button that could include different kinds of specific actions.

sequenceBoxForBLAST(): This class facilitates the creation of the sequence alignment representation of BLAST results. It extends the sequenceBox class and allows the creation of button or box objects within itself.

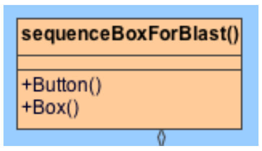


Figure 7.14: sequenceBox-ForBLAST class

Arguments given to the class:

- sequenceSize
- BLASTScore
- sequenceDirection
- actionScript
- actionType
- sequenceHeight

Methods:

- The ”Button” creates a button object
- The ”Box” creates a box object

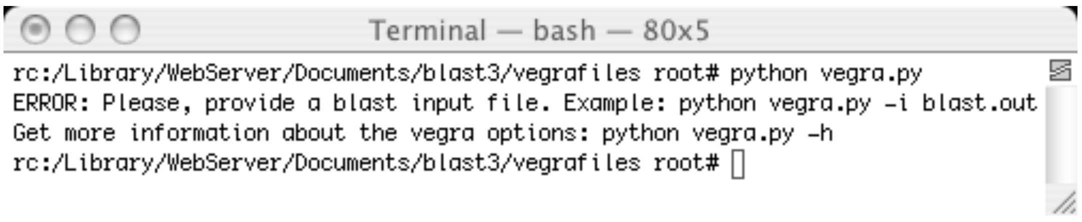
7.2.4 The initialization file (see vegra.py)

The initialization module starts the BLAST Modules from the BioPython Library and initializes VEGRA. It is also where the user defines the look and properties of the graphic by customizing the options of the VEGRA library. In ”vegra.py” the initialization file has been coded to define VEGRA options and create SWF files using a command line interface. The properties and behaviour of VEGRA are set by default, so user configuration is available but not necessary. The application needs the user to define the name of the BLAST result file and the path where it can be found.

Executing VEGRA with python

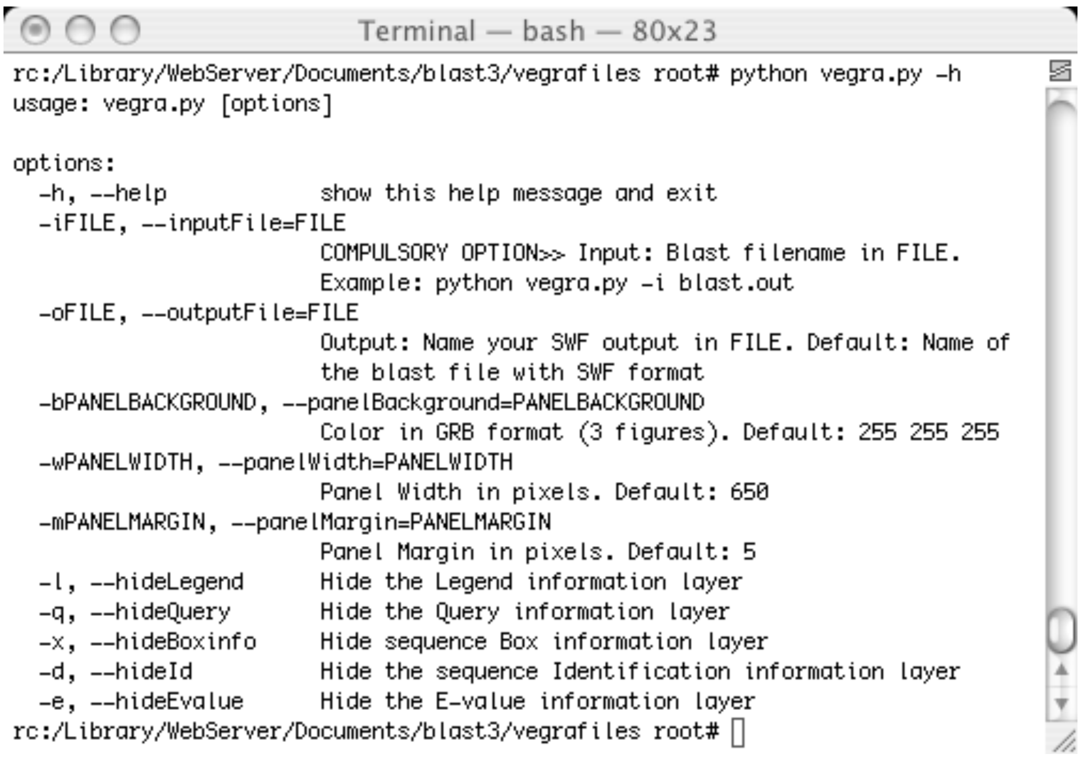
The VEGRA tool is executed from the command line using Python as follows: ”python vegra.py -i inputfile”. The command ”python vegra.py” line assists the user of how get help to start VEGRA with all the options available [see figure: 7.15].

An example of the command line help displayed by executing ”python vegra.py -h” [see figure: 7.16]



```
rc:/Library/WebServer/Documents/blast3/vegrafiles root# python vegra.py
ERROR: Please, provide a blast input file. Example: python vegra.py -i blast.out
Get more information about the vegra options: python vegra.py -h
rc:/Library/WebServer/Documents/blast3/vegrafiles root# []
```

Figure 7.15: Executing VEGRA on the command line



```
rc:/Library/WebServer/Documents/blast3/vegrafiles root# python vegra.py -h
usage: vegra.py [options]

options:
  -h, --help            show this help message and exit
  -iFILE, --inputFile=FILE
                        COMPULSORY OPTION>> Input: Blast filename in FILE.
                        Example: python vegra.py -i blast.out
  -oFILE, --outputFile=FILE
                        Output: Name your SWF output in FILE. Default: Name of
                        the blast file with SWF format
  -bPANELBACKGROUND, --panelBackground=PANELBACKGROUND
                        Color in GRB format (3 figures). Default: 255 255 255
  -wPANELWIDTH, --panelWidth=PANELWIDTH
                        Panel Width in pixels. Default: 650
  -mPANELMARGIN, --panelMargin=PANELMARGIN
                        Panel Margin in pixels. Default: 5
  -l, --hideLegend      Hide the Legend information layer
  -q, --hideQuery       Hide the Query information layer
  -x, --hideBoxinfo     Hide sequence Box information layer
  -d, --hideId          Hide the sequence Identification information layer
  -e, --hideEvalue      Hide the E-value information layer
rc:/Library/WebServer/Documents/blast3/vegrafiles root# []
```

Figure 7.16: Command Line help for VEGRA

Chapter 8

Conclusions

8.1 Achievements

VEGRA contributes by enabling a better visualization for BLAST outputs in four different ways:

- by replacing bitmaps with convenient vector graphics.
- by improving the visual design and usability of the graphical application.
- by providing a structured library that is easy to understand and hence easy to maintain and extend.
- by permitting easy personalization of the graphic configuration.

The problem stated in section one has been solved by developing VEGRA-BLAST. This allows the generation of SWF files from BLAST output files. By using vector graphics, BLAST results can be represented in a way that is scalable, zoomable, searchable and interactive.

The graphic design has been improved, following guidelines from graphic design; it is now possible to deliver graphics that are more appealing and with an optimal balance between visual sensation and graphic information.

VEGRA has been programmed following Object-Orientated paradigms, and UML methodologies. This makes it easier to share and reuse VEGRA's code. The library is quite versatile as classes could be easily extended to represent other types of biological results.

VEGRA BLAST allows the configuration of the layout within the graphic. It is also easy to adjust size and colour properties. It improves configuration usability by providing a straight-forward method with which to define graphic options via the command line interface.

8.2 Contributions

In general terms VEGRA is not just an improvement for BLAST for the management of the BLAST output, but also an alternative way to use Python to manage vector graphics when visualizing two-dimensional representations. VEGRA has demonstrated that this is possible without extensive coding. Code that is well structured and well documented facilitates reuse and fast improvements are possible.

8.3 Comments about the methodology

8.3.1 About vector graphics

Looking at the improvements of SVG over the last years there is no doubt that it will be as powerful and wide spread as flash and we hope that many other developers will begin to use this open standard to create professional visualization tools. In the meantime we have taken advantage of libraries like Ming to create suitable visualization tools in SWF Flash.

8.3.2 About Visual Design

As shown in this dissertation, visual design is essential for effective communication. Nonetheless, at the beginning of the process of developing graphical interfaces it is not easy to discern visual design problems. Thus, the conscious practice of applying visual design techniques would help developers to better understand and represent visual phenomena. As a GUI developer gains experience the task of mentally visualizing suitable designs for specific problems will likely improve. Software and GUI developers in bioinformatics should apply the concepts of visual design and become better trained in graphic design.

8.4 Further work

VEGRA BLAST fulfills its purpose by improving and presenting a tool to better represent BLAST results. Although most of the work has been done to fulfill this purpose we believe it can be improved:

- The VEGRA BLAST code has been developed and tested only for the most popular BLAST programs (BLASTn, BLASTp and BLASTx). When VEGRA was developed, BioPython was not capable of parsing outputs from other BLAST programs. Further developments in BioPython would allow the extension of the VEGRA functionality to other programs.

- VEGRA can create graphic representations from BLAST outputs and its results can be integrated with the text in an HTML page. However, this task has to be carried out manually by the user. A nice feature to develop for VEGRA would be a web application (similar to "wwwBLAST") which could deliver the BLAST text results with the VEGRA graphic file embedded in an HTML file.

- Because the VEGRA graphic image is not embedded in the BLAST text results there are no links between the graphic elements and their text equivalent. If VEGRA-generated graphics were embedded in the BLAST output, it would be an improvement that could allow the creation of links between the graphic and the HSP text results.

- VEGRA results are vector format files and as such they can be used by other computer vector graphic programs. However, currently the only way to export the SWF file to other readable vector format files is via the "FLASH editor", which is commercial software. Some work should be done to allow VEGRA to export its results to other formats. New ongoing research applications to convert SWF to SVG might provide a solution to this problem [35].

- It can be difficult to install VEGRA due to the difficulty of installing all its dependencies. Some work could be done to create a self-extracting package to easily install all of its necessary dependencies.

Chapter 9

How to install VEGRA BLAST

VEGRA BLAST can be used in Linux, Macintosh and Windows. However, it is much easier to install on one of the first two because its dependencies require much more effort to deploy in Windows. Since VEGRA is a Python library it needs Python and BioPython to be installed. To be able to create SWF files VEGRA requires the Ming graphic library to be installed. Ming is a C library, so in order to use it with python the user will have to install the Python wrapper which comes with Ming version 0.3.0. Most internet browsers include the "Flash player" plugin which allows viewing and playing of SWF files. In case a user does not have it or wants an independent viewer apart from the browser the free Macromedia "Flash Player" application from Adobe can be used.

VEGRA BLAST does not require any installation and can be executed anywhere on the file system where the user has permissions to create files inside the VEGRA folder. VEGRA is accessible from "http://www.nbn.ac.za/∼rafael/vegra/vegra.zip".

The VEGRA library is open source and freely available. All the dependencies utilized by VEGRA mentioned above are freely available.

VEGRA BLAST can create graphic representations from any BLAST output (BLASTn, BLASTp, BLASTx) whenever BioPython is capable of parsing the BLAST output. VEGRA BLAST does not require the user to have the BLAST stand-alone tool installed, however, we strongly recommend installing it for testing the library. In case a user want to test VEGRA directly we provide a BLAST output testing file at the following URL: "http://www.nbn.ac.za/∼rafael/vegra/seqaa.out

VEGRA has been tested and deployed successfully with:

- Mac OS X 10.3.9
- Python 2.3
- BioPython 1.41
- Ming 0.3.0
- Macromedia Flash Player 8
- BLAST 2.2.10

The user can get more information about the dependencies used by VEGRA as well as the downloading of files from the following URLs:

Python http://www.python.org/

BioPython http://biopython.org/

Ming http://ming.sourceforge.net

Flash Player http://www.adobe.com/shockwave/download/download.cgi

BLAST http://www.ncbi.nlm.nih.gov/Ftp/

VEGRA BLAST http://www.nbn.ac.za/∼rafael/vegra/vegra.zip

Bibliography

[1] J. Alspach. *PDF with Acrobat 5 (Visual QuickStart Guide).* Peachpit Press, 2nd edition, 2001.

[2] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman. Basic local alignment search tool. *J Mol Biol*, 215(3):403–410, Oct 1990.

[3] J. Artymiak. Swf is not flash (and other vectored thoughts). In *O'Reilly Articles*, http://www.oreillynet.com/pub/au/832, 2002.

[4] D.A. Benson, I. Karsch-Mizrachi, D.J. Lipman, J. Ostell, and D.L. Wheeler. Genbank. *Nucleic Acids Research*, 35(Database issue):16–20, 2007.

[5] S.K. Card. *Information Visualization, Using Vision to Think.* Morgan Kaufmann, London, 1st edition, 1999.

[6] C. Chen. *Information Visualization, Beyond the Horizont.* Springer, London, 2nd edition, 2004.

[7] M. Duignan and R. Biddle. Evaluating scalable vector graphics for use in software visualisation. *Australian Computer Society, Inc.*, 142:127–136, 2003.

[8] Open Bioinformatics Foundation. Biojava. http://www.biojava.org.

[9] Open Bioinformatics Foundation. Bioperl. http://www.bioperl.org.

[10] Open Bioinformatics Foundation. Biopython. http://www.biopython.org.

[11] M.Y. Galperin. The molecular biology database collection: 2007 update. *Nucleic Acids Res.*, 2007.

[12] J.H. Gillespie. *The Causes of Molecular Evolution.* Oxford University Press, New York, 1991.

[13] B. Gordon and M. Gordon. *The Complete Guide to Digital Graphic Design.* Thames Hudson, 2nd edition, 2005.

[14] A.J.F. Griffiths, J.H. Miller, D.T. Suzuki, R.C. Lewontin, and W.M. Gelbart. *Introduction to Genetic Analysis.* W. H. Freeman and Co., New York, 7th edition, 1999.

[15] D.L. Hartl. *Genetics, Principle and Analysis.* Jones and Bartlett, 1998.

[16] R.M. Horton. The molecular biology database collection: 2007 update. *Biotechniques*, pages 874–6, 1999.

[17] T.J. Hubbard. Ensembl 2007. *Nucleic Acids Res.*, 2007.

[18] National Human Genome Research Institute. Codon image. http://www.genome.gov/Pages/Hyperion//DIR/VIP/Glossary/Illustration/codon.shtml.

[19] R. Jacobson. *Information Design.* The MIT Press, first edition edition, 1999.

[20] R. Jimenez. Vector graphics to improve blast graphic representations. In *Sequence Analysis (Posters)*. National Bioinformatics Network of South Africa, ISMB, 2006.

[21] E.V. Koonin and M.Y. Galperin. *Sequence, Evolution, Function: Computational Approaches in Comparative Genomics.* Kluwer Academic Publishers, 2002.

[22] I. Korf, M. Yandell, and J. Bedell. *Blast.* O'Reilly, 2003.

[23] J. Krasner. *Motion Graphic Design and Fine Art Animation: Principles and Practice.* Springer, 2004.

[24] S. Kumar. *Developing a GIS-based Geo-Portal with Scalable Vector Graphics (SVG) for Accessing Environmental Information.* PhD thesis, Stuttgart University of Applied Sciences, 2003.

[25] K. Liolios, N. Tavernarakis, and P. Hugenholtz. The genomes on line database (gold) v.2: a monitor of genome projects worldwide. *NAR*, 34:332–334, 2006.

[26] T. Madden. *The NCBI Handbook. Part 3. Querying and Linking the Data.* NCBI, 2002.

[27] T. Mandel. *The Elements of User Interface Design.* Penguin, 1st edition, 1997.

[28] H. Mangalam. The bio* toolkits - a brief overview. *Briefings in Bioinformatics*, 3(3):1–7, 2002.

[29] J. Mangler and C. Bauer. A cewebs component for facilitating online cooperative brainstorming processes. In *International Conference on Interactive Computer-aided learning.* Department of Computer Science and Business Informatics, University of Vienna, 2004.

[30] E. Mayr. *What Evolution Is.* Basic Books, 2002.

[31] J. Morton. *Color Logic for Web Site Design.* Colorcom, 1998.

[32] K. Mullet and D. Sano. *Designing Visual Interfaces.* ACM Press, 1st edition, 1996.

[33] NCBI. Blast tutorial. http://www.ncbi.nlm.nih.gov/Education/BLASTinfo/tut1.html.

[34] D. Norman. *Emotional Design: Why We Love (or Hate) Everyday Things.* Basic Books, 2004.

[35] S. Probets. Vector graphics: from postscript and flash to svg. In *Proceedings of the 2001 ACM Symposium on Document engineering table of contents*, 2001.

[36] Adobe Press releases. Adobe to acquire macromedia. http://www.adobe.com/aboutadobe/invrelations/adobeandmacromedia.html, April 2005.

[37] T. Rhyne. Evolving visual metaphors and dynamic tools for bioinformatics visualization. North Carolina State University, 2002.

[38] N. Shedroff. *Experience Design 1.* Waite Group Press, 1st edition, 2001.

[39] Y. Tao, Y. Liu, C. Friedman, and Y.A.Lussier. Information visualization techniques in bioinformatics during the postgenomic era. *BIOSILICO*, 2:237–245, 2004.

[40] S. Wallace. *Perl Graphics Programming Creating SVG, SWF (Flash), JPEG and PNG files with Perl.* O'Reilly, first edition edition, 2002.