

Spring 5-31-2006

A data gathering toolkit for biological information integration

Munira Lokhandwala
New Jersey Institute of Technology

Follow this and additional works at: <https://digitalcommons.njit.edu/theses>



Part of the [Biostatistics Commons](#), and the [Computer Sciences Commons](#)

Recommended Citation

Lokhandwala, Munira, "A data gathering toolkit for biological information integration" (2006). *Theses*. 1713.

<https://digitalcommons.njit.edu/theses/1713>

This Thesis is brought to you for free and open access by the Electronic Theses and Dissertations at Digital Commons @ NJIT. It has been accepted for inclusion in Theses by an authorized administrator of Digital Commons @ NJIT. For more information, please contact digitalcommons@njit.edu.

Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be “used for any purpose other than private study, scholarship, or research.” If a user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of “fair use” that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select “Pages from: first page # to: last page #” on the print dialog screen

The Van Houten library has removed some of the personal information and all signatures from the approval page and biographical sketches of theses and dissertations in order to protect the identity of NJIT graduates and faculty.

ABSTRACT

A DATA GATHERING TOOLKIT FOR BIOLOGICAL INFORMATION INTEGRATION

**by
Munira Lokhandwala**

SYSTEMS is a biological information integration system containing protein sequences from many protein databases such as Swiss-Prot and TrEMBL and also protein sequences from complete genomes available at Ensembl, The Arabidopsis Information Resource, SGD and GeneDB. For some protein sequences their encoding nucleotide sequences can be found in their corresponding websites. However, for some protein sequences their encoding nucleotide sequences are missing.

The goal of this thesis is to collect all nucleotide sequences for the protein sequences in SYSTEMS and store them in a common database. There are two cases. The first case is that if the nucleotide sequences can be found, we collect them and put them in our database. The second case is that if the nucleotide sequences are missing, we use back-translation and use TBLASTN to search the nucleotide sequences and store them in our database.

**A DATA GATHERING TOOLKIT FOR
BIOLOGICAL INFORMATION INTEGRATION**

by
Munira Lokhandwala

**A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Bioinformatics**

Department of Computer Science

May 2006

APPROVAL PAGE

**A DATA GATHERING TOOLKIT FOR
BIOLOGICAL INFORMATION INTEGRATION**

Munira Lokhandwala

Dr. Jason Tsong Li Wang, Thesis Advisor Date
Professor, Department of Computer Science, NJIT

Dr. Marc Qun Ma, Committee Member Date
Assistant Professor, Department of Computer Science, NJIT

Dr. Vincent Oria, Committee Member Date
Assistant Professor, Department of Computer Science, NJIT

BIOGRAPHICAL SKETCH

Author: Munira Lokhandwala

Degree: Master of Science

Date: May 2006

Education:

- Master of Science in Bioinformatics,
New Jersey Institute of Technology, Newark, NJ, 2006
- Bachelor of Science in Computer Science,
Mumbai University, Mumbai, MH, India, 2003

Major: Bioinformatics

ACKNOWLEDGMENT

I would like to express my deepest appreciation to Dr. Jason Tsong-Li Wang, my advisor, for providing the initial spark that helped me get over the fear of learning and new challenging topics in the field of Bioinformatics. He not only served as my thesis advisor, but also constantly gave me support, encouragement, assistance and guidance in my research work. His recommendations and suggestions have been invaluable for the project development. Special thanks are given to Dr. Qun Ma and Dr. Vincent Oria for actively participating in my committee.

Finally, I wish to thank Ms. Junilda Spirollari, Ms. Maria Moutafis and Mr. Yang Song who helped me at various phases of the project.

*To my parents and my brother for their unconditional love and support and to my friends
who have stood by me all throughout my life.*

TABLE OF CONTENTS

Chapter	Page
1 INTRODUCTION	1
2 DESCRIPTION OF TOOLS	3
2.1 NCBI-BLAST	4
2.2 QBLAST	6
2.3 Biological Databases	9
2.3.1 SYSTERS	9
2.3.2 GenBank	11
2.3.3 UniProtKB/SwissProt and UniProtKB/TrEMBL	11
2.3.4 SGD	13
2.3.5 EBI and EMBL	13
2.4 PERL	14
2.4.1 Regular Expressions	15
2.4.2 LWP Modules	15
3 THE PROGRAM FOR DATA COLLECTION	18
3.1 Before The Program	18
3.2 Flow Of The Program	27
3.3 Program Description	31
3.4 Benefits and Limitations	60
3.5 Running Time Analysis	61
4 RESULTS AND SCREENSHOTS	62

TABLE OF CONTENTS
(Continued)

Chapter	Page
APPENDIX THE CODE	73
REFERENCES	87

LIST OF TABLES

Table		Page
3.1	Analysis of the Manual Extraction and the Program	61

LIST OF FIGURES

Figure	Page
2.1 Hierarchy resulting from the SYSTERS procedure	10
3.1 Providing the TBLASTN program with the tributyltin chloride resistance protein sequence in FASTA format	19
3.2 The TBLASTN intermediate result page: The request ID is mentioned in the textbox and the query sequence is given above it	20
3.3 The TBLASTN intermediate waiting page: This displays the search status which is 'searching' in this case and the time since the sequence was submitted	21
3.4 The NCBI BLAST graphic display: Red bars indicate the most similar sequences to the query sequence, pink bars indicate sequences which are less similar. Rolling over the mouse on the red bar displays D16369 Alteromonas sp. M-1 tbt A gene, tributyltin chloride re... S = 345 E = 3.9e-93 signifying that this is most similar sequence to the query sequence	22
3.5 The NCBI BLAST hit list: The hit list is ordered according to the Score and the E-value	23
3.6 The alignments reported by TBLASTN: The image displays the first hit of the results The score equals 345 bits and the E value is 4e-93. The percent identities equal 85% (178/207) indicating that is a very good hit. The positives field has the value of 182/207 (87%) indicating that 87% of the residues are identical or similar and the gaps field has the value 1/207 (0%) indicating that there are no gaps. The frame field refers to the Open Reading Frames (ORFs) and in this case it is the 3 rd ORF on the forward strand. The query sequence range is from 1...206 and the subject sequence range is from 990...1610	25
3.7 Back-translated DNA sequence: The figure shows the raw DNA sequence and the highlighted portion indicates the back-translated DNA sequence of the corresponding input protein (query) sequence	26
3.8 Flowchart to demonstrate the working of the program	27

LIST OF FIGURES
(Continued)

Figure	Page
3.9 Flowchart to demonstrate the working of the swissprot subroutine	28
3.10 Flowchart to demonstrate the working of the SGD subroutine	29
3.11 Flowchart to demonstrate the working of the tblastn subroutine	30
3.12 The Swissprot protein web page for the http://ca.expasy.org/uniprot/Q9UEB4 URL which is obtained as result when the URL is passed as a request to the HTTP::Request object. The highlighted portion is the primary accession number which is extracted from the web page	37
3.13 The Swissprot protein webpage when the page results into the '410 Gone' error	38
3.14 A part of the Swissprot protein web page which highlights the Genomic DNA Coding Sequence hyperlink, the URL of which is extracted from the HTML source code of the page and is used to obtain the nucleotide sequence web page	39
3.15 A part of the Swissprot protein web page which highlights the mRNA Coding Sequence hyperlink, the URL of which is extracted from the HTML source code of the page and is used to obtain the nucleotide sequence web page	39
3.16 The EBI-EMBL nucleotide sequence web page which is obtained as a result of passing the coding sequence URL as a request to the HTTP::Request object	41
3.17 The EMBL-EBI nucleotide sequence web page which highlights the nucleotide sequence that is extracted from the HTML source code of the page using the regular expression	42
3.18 The SGD protein web page for the http://db.yeastgenome.org/cgi-bin/locus.pl?sgdid=S0000433 URL which is obtained as result when the URL is passed as a request to the HTTP::Request object. The highlighted portion is the Systematic Name which is extracted from the HTML source code of the web page using regular expressions	44

LIST OF FIGURES
(Continued)

Figure	Page
3.19 The SGD nucleotide sequence web page which is obtained as a result of passing the coding sequence URL, http://db.yeastgenome.org/cgi-bin/getSeq?seq=YBR229C&flankl=0&flankr=0&map=n3map , as a request to the HTTP::Request object	46
3.20 The SGD nucleotide sequence web page which highlights the nucleotide sequence that is extracted from the HTML source code of the page using the regular expression mentioned above	47
3.21 Part of the HTML source code of the intermediate results page: It highlights the Request Id (RID) and the Remaining Time Of Execution (RTOE) which is extracted from the sources using regular expressions	49
3.22 The intermediate waiting page of TBLASTN: The status of the program is shown as 'Waiting' and also the RID is specified, which indicates that TBLASTN is searching for results for the RID mentioned	52
3.23 The results page obtained from the QBLAST system. It highlights the alignment part of the page that is extracted using regular expressions. Only the first hit is displayed as the hitlist_size parameter in the put query is set to one	54
3.24 The alignment section of the Blast results page highlighting the accession number of the nucleotide sequence as well as the subject range or the target sequence range which are extracted from the page using regular expressions and this information is used to obtain the back-translated DNA sequence	56
3.25 The final result obtained from the QBLAST system. It highlights the nucleotide sequence result which is obtained from the QBLAST system. The nucleotide sequence is extracted from the page using regular expressions and the final output result is printed with the protein sequence following the DNA sequence	58

LIST OF FIGURES
(Continued)

Figure	Page
4.1 (a) The start of the program. User inputs the file name to be executed and the database and the accession number are displayed. In this case the database shortcut matches the condition to the 'swissprot' subroutine and the merged URL is mentioned. Also the program checks if the primary accession number and the accession number from the insert statements have matched	63
4.1 (b) The figure displays the resulting back-translated DNA sequence for the first protein sequence in the file. It also shows the immediate execution of the next protein sequence	63
4.1 (c) This figure displays the database and accession number of the second protein sequence in the file along with the merged URL. It also displays the results for the second protein sequence	64
4.1 (d) The figure displays the second protein sequence along with its corresponding back-translated DNA sequence indicating that the DNA sequence has been obtained from the Swissprot/TrEMBL database. It also indicates that the file has completed execution and the program has terminated	64
4.2 This figure represents the output file 'result_out.txt' to which all the results obtained from the program are written to. The output file specifies how many protein sequences are present in the file along with the number of back-translated DNA sequences extracted from each of the databases. In this case it indicates that there are two protein sequences in the file and two back-translated DNA sequences have been obtained from the SWISSPROT/TrEMBL database	65
4.3 (a) The start of the program. User inputs the file name to be executed and the database and the accession number are displayed. In this case the database shortcut matches the condition to the 'SGD' subroutine and the first merged URL is mentioned. The systematic name and the URL formed after merging the systematic name is printed	67

LIST OF FIGURES
(Continued)

Figure	Page
4.3 (b) This figure is the continuation of the program for the Saccharomyces Cerevisiae protein sequence. It displays the protein sequence in FASTA format and the corresponding back-translated DNA sequence	67
4.3 (c) This figure shows the termination of the program for the Saccharomyces Cerevisiae protein sequence	68
4.4 (a) The start of the program. User inputs the file name to be executed and the database and the accession number are displayed. In this case the database shortcut does not match either to the condition of the 'swissprot' or of the 'SGD' subroutine and the tblastn subroutine is invoked	69
4.4 (b) Displays the execution of the tblastn program stating that the program is searching the nucleotide database for entries corresponding to the protein sequence and once the search is complete it retrieves the results. Along with this the accession number of the nucleotide sequence is displayed and even the information regarding the start and end positions of the subject or target sequence is printed	70
4.4 (c) This figure displays the protein sequence and the resulting DNA sequence for the first sequence in the file	70
4.4 (d) This figure shows the immediate execution of the second sequence in the file and shows all the corresponding information that is needed to extract the DNA sequence	71
4.4 (e) This figure partially displays the output of the second sequence in the file and the partial execution of the third sequence	71
4.4 (f) The last figure displays the results of the third sequence in the file and indicated that the file has completed execution and the program has terminated. All of these screenshots are used to demonstrate the flow of execution of tblastn	72

LIST OF TERMS

Alignment

Representation of two or more protein or nucleotide sequences where homologous amino acids or nucleotides are in the same columns while missing nucleotides area replaces with gaps.

Amino Acid

The fundamental block of building proteins. There are 20 naturally occurring amino acids in animals and around 100 more found only in plants.

Arabidopsis Thaliana

Known by its common name, thale cress, this mustard weed is a favorite organism for plant genetics and molecular biology. It was the first plant with a complete genomic sequence

Basepair (bp)

Any possible pairing between bases in opposing strands of DNA or RNA. Adenine pairs with thymine in DNA or with uracil in RNA; and guanine pairs with cytosine.

Bioinformatics

The application of computational techniques to the management and analysis of biological information.

BLAST

Basic Local Alignment Search Tool, or BLAST, is an algorithm for comparing biological sequences, such as the amino-acid sequences of different proteins or the DNA sequences. A BLAST search enables a researcher to compare a query sequence with a library or database of sequences, and identify library sequences that resemble the query sequence above a certain threshold.

Complement

The complement of a DNA sequence is the sequence on the other strand. For example, the complement of ACCCGT is TGGGCA.

DDBJ

DNA Data Bank of Japan.

DNA

Deoxyribonucleic acid; the genetic material of living things.

Drosophila melanogaster

The common fruit fly. This is one of the most famous organisms for genetic research and was one of the first animals whose complete genomic sequence was determined.

EBI

European Bioinformatics Institute. The European homologue of the NCBI in the US.

EMBL

European molecular biology laboratories. Maintain the EMBL database, one of the major public sequence databases.

Ensembl Project

A European project devoted to the annotation of the human genome.

Entrez

The search and retrieval system that integrates information from the National Center for Biotechnology (NCBI) databases. These databases include nucleotide sequences, protein sequences, macromolecular structures, whole genomes, and MEDLIN, through PubMed.

E-value

Expectation value. Given a database and the score of a hit, the E-value tells you how many times you could have expected such a result just by chance. In sequence analysis, good E-values must be very low (around 10^{-5} or even lower).

Exons

The protein-coding sequences DNA sequences of a gene.

Gene

The fundamental physical and functional unit of heredity. A gene is an ordered sequence of nucleotides located in a particular position on a particular chromosome that encodes a specific functional product (i.e. a protein or RNA molecule).

GeneBank

A population of organisms, each of which carries a DNA molecule that was inserted into a cloning vector. Ideally, all of the cloned DNA molecules represent the entire genome of another organism. Also called gene library, clone bank, bank library. This term is sometimes also used to denote all of the vector molecules, each carrying a piece of the chromosomal DNA of an organism, prior to the insertion of these molecules into a population of host cells.

Hit

Refers to a sequence similar to your query that you find while conducting a database search.

Introns

The sequence of DNA bases that interrupts the protein coding sequence of a gene; these sequences are transcribed into RNA but are edited out of the message before they are translated into protein.

Messenger RNA

An RNA molecule carrying the information that, during translation, specifies the amino acid sequence of a protein molecule.

NCBI

National Center for Biotechnology Information. A component of the U.S. national Institute of Health dedicated to bioinformatics research, software development, and the service and maintenance of leading public resources such as the GenBank (sequences) and PubMed (bibliography) databases. The United States' homologue of the EBI in Europe.

NIH

National Institute of Health (USA).

Non-coding Sequences

All sequence that do not encode protein. This includes introns, regulatory sequences (e.g. MARs, LCRs, enhancers, promoters) and genomic repeats.

NR

The Non Redundant Protein database, which contains all the putative protein sequences contained in the nucleotide databases. Its European equivalent is TrEMBL.

Nucleotide

A subunit of DNA or RNA consisting of a nitrogenous base (adenine, guanine, thymine, or cytosine in DNA; adenine, guanine, uracil, or cytosine in RNA), a phosphate molecule, and a sugar molecule (deoxyribose in DNA and ribose in RNA). Thousands of nucleotides are linked to form a DNA or RNA molecule.

Open Reading Frame

A series of DNA codons, including a 5' initiation codon and a termination codon, that encodes a putative or known gene. A part of a DNA sequences without Stop codons, thus allowing the (putative or real) translation of a protein sequence.

PDB

Brookhaven Protein Data Bank. A database and format of files which describe the 3D structure of a protein or nucleic acid, as determined by X-ray crystallography or nuclear magnetic resonance (NMR) imaging. The molecules described by the files are usually viewed locally by dedicated visualizing software, but can sometimes be visualized on the world wide web.

PIR

Protein Information Resources. An annotated protein database similar to SWISSPROT. PIR is also the name of a sequence format similar to FASTA.

Polymorphism

Alternative forms of genes and other sequences. They are often constituted of punctual mutations, similar to SNP.

Protein

A molecule composed of one or more chains of amino acids in a specific order; the order is determined by the base sequence of the nucleotides in the gene coding for the protein. Proteins are required for the structure, function and regulation of cells, tissues and organs, each protein having a specific role (e.g. hormones, enzymes, antibodies)

Protein Database

Contains protein sequence data from the translated coding region from DNA sequences in GeneBank, EMBL and DDBJ as well as protein sequences submitted to Protein Information Resources (PIR), SwissProt, Protein Research foundation and Protein Data Bank (PDB).

Protein Profile

A tool for visualizing a particular property (hydropathy, charge, local composition, and so on) along a sequence by using a sliding windows technique.

PSI-BLAST

Position-Specific Iterative Blast. An iterative search using the BLAST algorithm. A profile is built after the initial search, which is then used in subsequent searches. The processes may be repeated, if desired with new sequences found in each cycle, used to define the profile.

Query

Question asked when searching a database. If you make a similarity search, your query is a sequence. If you use SRS or Entrez, your query is a keyword. More complicated queries may involves several keywords, field restricted searches, and limits(such as dates).

RNA

A molecule chemically similar to DNA that plays a central role in protein synthesis. The structure if RNA is similar to that of DNA but it is less stable. It is a poly nucleotide that has Ribose Sugar and Uracil as one if its pyrimidines.

Similarity

Percent of similar amino acids in the alignment of two sequences. Two amino acids are similar if they have physicochemical properties.

Six-Frame Translation

Translation of a stretch of DNA taking into account three forward translations and three reverse translations, arising from the three possible reading frames of an uncharacteristic stretch of DNA.

SNP

Single Nucleotide Polymorphism. SNP's are point mutations observed when comparing different genomes of the same specie.

SRS

Sequence retrieval system. The system used at the EBI to search databases with keywords. It is similar to Entrez at NCBI.

Strand

A linear series of nucleotides that are linked to each other by phosphodiester bonds.

SWISS-PROT

A non-redundant protein sequence database thoroughly annotated and crossed referenced.

Transcript

An RNA molecule that has been synthesized from a specific DNA template.

Transcription

The process of RNA sequences that is catalyzed by RNA polymerase that uses a DNA strands as a template.

Translation

The process of protein synthesis in which the amino acid sequence of a protein is determined by mRNA mediated by tRNA molecules and carried out on ribosomes.

TrEMBL

Short for Translation EMBL, which contains all the putative protein sequences contained in the nucleotide databases. Its U.S. equivalent is NR.

CHAPTER 1

INTRODUCTION

“Bioinformatics is the study of information content and information flow in biological systems and processes. It has evolved to serve as the bridge between observations (data) in diverse biologically related disciplines and the deviations of understanding (information) about how the systems or processes function and subsequently the application (knowledge).”

-C.S.V. Murthy

In the emerging field of Bioinformatics it is become necessary to create various kinds of information and database resources which store all kinds of data pertaining to the needs of biologists, bioinformaticians, research scientists and students. There are many such data resources that store various kinds of bioinformatics related data like sequence information and whole genomes and also provide software tools for performing various tasks such as comparing sequences, building phylogenetic trees, performing alignments etc. which helps to simplify the work of biologists in handling and analyzing vast data. Technologies such as genome-sequencing, microarrays, proteomics and structural genomics have provided ‘parts lists’ for many living organisms, and researchers are now focusing on how the individual components fit together to build systems. Thus, there is an ongoing and growing need to collect, store and curate all this information in ways that allow its efficient retrieval and exploitation.

Also with the growth in the field of information technology, computer based tools now play an increasingly critical role in the advancement of biological research. With the advent of the World Wide Web and fast Internet connections, the data contained in the

databases and the many special-purpose tools and programs can be accessed quickly, easily and cheaply from any location in the world. The Internet has become a tool of critical importance to the biologist and scientist working in genomic and molecular biology. Via this mode the scientist and the biologist all around the world can share their findings, ideas and thoughts and thus, it has become a unifying force which helps to bind the biological community. But in any growing field there is always a need for different and new ideas.

This project aims to propose an innovative idea for data collection from Internet resources. It is an automated system which retrieves the back-translated DNA sequence given its corresponding protein sequence. The mechanism behind it involves connecting to various online data web servers and retrieving the genomic DNA coding sequences, if they are provided by the websites, or else connect to the TBLASTN program provided by the NCBI-BLAST at the NCBI website, which compares a protein sequence with a nucleotide database and obtains the back-translated DNA sequence. The protein sequences for which the DNA sequences are determined are obtained from the SYSTERS database which is a protein family database and performs large-scale protein clustering based on sequence similarity. The program which performs this task of data gathering has been coded using the PERL scripting language. PERL is favored by Bioinformaticians because it has a very good at handling String Operations and Regular Expressions. One of the most exciting things about being involved in computer programming and biology is that both fields are rich in new techniques and results.

CHAPTER 2

DESCRIPTION OF TOOLS

2.1 NCBI-BLAST

The **B**asic **L**ocal **A**lignment **S**earch **T**ool (BLAST) is a sophisticated software package, a service of the National Center for Biotechnology Information (NCBI), which finds regions of local similarity between sequences. The program compares nucleotide or protein sequences to sequence databases and calculates the statistical significance of matches. BLAST can be used to infer functional and evolutionary relationships between sequences as well as help identify members of gene families.

BLAST is fast, reliable and flexible and can be adapted to many sequence analysis scenarios. It comes in variations to help query different type of sequences (nucleotides and proteins) against different databases. The different programs that come under the BLAST family are blastn, blastp, blastx, tblastn and tblastx. Of all these programs, the TBLASTN program has been used for the efficient retrieval of back-translated DNA sequences as it compares a protein query sequence against a nucleotide sequence database dynamically translated in all six reading frames. The Qblast's URL API has been used for the purpose of submitting requests to the TBLASTN program, the description of which is given in the next section. The description of tblastn is as follows:

TBLASTN (Translating BLAST) is a program of the BLAST family of programs and is described as "protein vs. translated database". It is useful for finding protein homologs in unannotated nucleotide data. TBLASTN identifies transcripts, potentially from multiple organisms, similar to a given protein, mapping a protein to genomic DNA.

A TBLASTN search allows you to compare a protein sequence to the six-frame translations of a nucleotide database. There are three reading frames on the (+) strand also known as the direct strand and the other three reading frames are on the opposite strand known as the reverse strand. Each of these six possible translations yields a different protein. When tblastn compares a protein sequence with a DNA sequence it does everything and automatically turns any DNA sequence into six proteins. This way, TBLASTN takes care about the orientation or the frame of the DNA sequences in the databases and always outputs the sequence in the right orientation.

It can, thus, be a very productive way of finding homologous protein coding regions in unannotated nucleotide sequences such as expressed sequence tags (ESTs) and draft genome records (HTG).

2.2 QBLAST

The QBLAST system is a queuing system implemented for both basic and advanced BLAST which offers rapid reformatting of search results and enhances server performance by reducing the connection time with each user. QBLAST is not a new BLAST algorithm but it simply offers a modular approach that separates the search step from the output formatting step.

Before QBLAST was implemented an output format had to be specified prior to running a BLAST search. QBLAST saves the results under a Request ID number, which is then used to retrieve them in the desired formats. For secure data retrieval, Request IDs are unique numbers which are randomly generated and are not issued in sequential order.

Therefore, it is not possible for users to change any digit in their Request ID and receive the results of another person's search.

In order to initiate a search using the new QBLAST system a sequence query is entered in the Search text area and the BLAST button is pressed. A Formatting page reporting the Request ID and showing all the display options is then returned. At this point, one can wait for the search to finish, or jot down the Request ID and use it to call up the results later. In this way, one can view the results immediately by pressing the format button and the results will be displayed as soon as the search is completed, or, one can also view the results later by going to "Click here to retrieve results if you already have a Request ID" and entering the Request ID number. The QBLAST system allows the results of very big files to be stored for 30 minutes after which they are deleted. However, most of the results obtained are stored in for 24 hours. The URL API which has been used to perform all the above mentioned functions has been described in detail below.

The **QBlas**t's **URL API** is a standardized application program interface for accessing the NCBI QBLAST system. It uses direct HTTP-encoded requests to NCBI web server. These encoded requests are directed to the NCBI cgi-bin program:

`http://www.ncbi.nlm.nih.gov/blast/Blast.cgi`

In order to issue an URLAPI command the program needs to:

- Make a connection to port 80 to NCBI web server. Using telnet in UNIX the following command will have to be given

`telnet www.ncbi.nlm.nih.gov 80`

- And then send the following commands to the port:
 - POST /blast/Blast.cgi HTTP/1.0
 - User-Agent: Hi_there
 - Connection: Keep-Alive
 - Content-type: application/x-www-form-urlencoded
 - CMD=Put&QUERY=555... etc

These commands are sent to the NCBI web server using PERL modules in the program.

All the examples stated below are given in PERL.

Searching the NCBI QBLAST system consists of two major steps.

- The first step is called "Put", and it puts the query sequence with the appropriate search parameters into the QBLAST system.
- The second step is called "Get" and it formats the results with specified format parameters.

Listed below is an example how the put and the get steps work in the QBLAST system. For using the tblastn program to query the database for the back-translated DNA sequence pertaining to a particular protein sequence against the 'nr' database which is the default database for the tblastn program the put command for this part of the URL API will be as follows.

```
http://www.ncbi.nlm.nih.gov/blast/Blast.cgi?\  
CMD=Put&QUERY=$protein_sequence&DATABASE=nr&HITLIST_SIZE=1&FILTE  
R=L&PROGRAM=tblastn&SERVICE=plain
```

In the 'url-encoded' format the '?' means the start of a list of parameters, which is followed by a list of name-pairs separated with '&'. "CMD=Put" means that a new search is going to be put into the QBLAST system. The **query** parameter specifies the

sequence query that has to be sent to the QBLAST system. This parameter can take values such as, accession numbers, gi(s) or sequences in FASTA format like

```
“QUERY=MEDASAGPPPVDDGEVPAAPADSSPLNDAPASSGAEPGDGGYDEGEP
LDNEQAGPADVEG”.
```

The **database** parameter specifies the database name that is to be searched for against the input query sequence(s). The **hitlist_size** parameter specifies the number of hits that need to be retained from the result set. It takes a valid integer as input value and the default value for this parameter is 500. The next parameter that is the **filter** parameter is to specify the sequence filter identifier. It takes the values “L” for Low Complexity, “R” for Human Repeats and “m” for Mask for Lookup. There is no default value of this parameter that is this parameter has to be explicitly mentioned and it is possible to specify more than one filter in the URL request. The **program** parameter specifies the name of the blast program that is to be used. The values this parameter can take are blastn, blastp, blastx, tblastn and tblastx. The next parameter that is the **service** parameter specifies what type of blast service needs to be performed. The values taken by this parameter are plain, psi, phi, rpsblast, megablast. The default value of this parameter is plain.

The output of the ‘Put’ command will be a valid HTML page, the contents of which may be ignored except the following important section:

```
<!--QblastInfoBegin
  RID = 954517067-8610-1647
  RTOE = 207
QblastInfoEnd
-->
```

This portion of the output is special as it contains the Request Identifier (RID) and the estimated Request Time of Execution in seconds (RTOE) for the search. The RID is

different for every search, and is a mandatory parameter for the next step which is formatting the BLAST results.

In order to get the results for a given RID using the default format parameters the following URL has to be used:

```
http://www.ncbi.nlm.nih.gov/blast/Blast.cgi?\  
CMD=Get&RID=954517013-7639-11119&FORMAT_TYPE=HTML&NCBI_GI=yes
```

Where “CMD=Get” gets formatted results from the QBlast system which are retrieved for the query in the put command. The **rid** parameter is a mandatory field for the get command and specifies the request id of the request. The value of this parameter should be a valid ID and every time a request is given a unique request ID gets generated. The parameter **format_type** specifies the type of formatting that the result page would have and be displayed in. The **ncbi_gi** parameter specifies if the NCBI Gene Id should be displayed or no.

If the search is not yet complete, this will produce output in the following format with Status equal to "WAITING":

```
<!--QBlastInfoBegin  
  Status=WAITING  
QBlastInfoEnd  
-->
```

If the results are completed, the output will show the formatted results with status information like the following, with Status=READY:

```
<!--QBlastInfoBegin  
  Status=READY  
QBlastInfoEnd  
-->  
... <formatted output here>
```

In order to use the QBLAST system only one or two threads can be used to submit jobs. A new job can only be submitted once the RID is got back from the server of the previously submitted job. Initiating many threads at the same time could lead to the flooding of the server and in this case the server may block access to the scripts trying to do so.

2.3 Biological Databases

Biological databases are web sites that organize, store and disseminate files that contain information consisting of literature references, nucleic acid sequences, protein sequences and protein structures. To analyze sequence information is to assemble it into central and shareable resources such as databases which effectively are a convenient and efficient way of storing vast information. Some of the databases that are being extensively used for the purpose of this project are mentioned below.

2.3.1 SYSTERS

SYSTEMatic Re-Searching (SYSTERS) is a huge online resource of protein families. It uses a collection of graph-based algorithms to hierarchically partition a larger set of protein sequences into homologous families, superfamilies and subfamilies annotated with sequence information from various other resources.

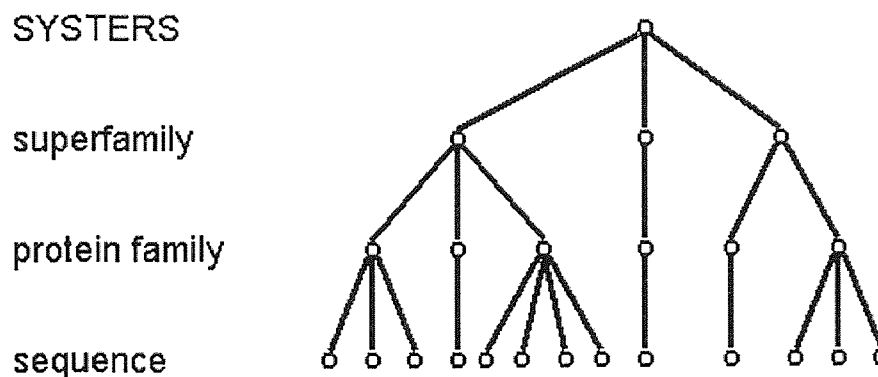


Figure 2.1 Hierarchy resulting from the SYSTERS procedure.

Clustering in SYSTERS is done in two ways. First, a single linkage tree is constructed based on the pairwise E-values obtained from the database searches. Superfamilies are derived from this tree based on a specific superfamily cutoff E-value for every superfamily. Second, a superfamily distance graph is constructed for every superfamily including only E-values equal to the superfamily cutoff. Splitting this graph recursively at weighted minimal cut sites, results in a collection of protein families (also called clusters).

The current SYSTERS cluster set which is SYSTERS release 4 contains 969,579 non-redundant sequences (and annotations of 1,168,542 redundant sequences) from the sequence databases which are Swiss-Prot and TrEMBL and the complete genome databases which are ENSEMBL (*Anopheles gambiae*, *Caenorhabditis briggsae*, *Caenorhabditis elegans*, *Drosophila melanogaster*, *Danio rerio*, *Fugu rubripes*, *Homo sapiens*, *Mus musculus*), The Arabidopsis Information Resource Database, SGD i.e. *Saccharomyces cerevisiae* and GeneDB i.e. *Schizosaccharomyces pombe*. These databases are sorted into 158,153 disjoint clusters.

2.3.2 GenBank

GenBank is the DNA database from the National Center for Biotechnology Information (NCBI); NCBI is a division of National Library of Medicines, located at National Institutes of Health (NIH) in Bethesda, Maryland. It incorporates sequences from publicly available sources, mainly from directed author submissions and large scale sequencing projects. In order to help ensure comprehensive coverage the resource exchanges data with both the European Molecular Biology Laboratory (EMBL) and DNA Data Bank of Japan (DDBJ). NCBI maintains sequence information from every organism, every source and all types of sequence related information such as DNA sequences – from mRNA to cDNA clones to expressed sequence tags, high throughput genome sequencing data and information about sequence polymorphisms.

As per the current statistics, there are approximately 59,750,386,305 bases in 54,584,635 sequence records in the traditional GenBank divisions and 63,183,065,091 bases in 12,465,546 sequence records in the WGS division as of February 2006.

In order to search a DNA sequence entry in the GenBank database, a text-based query to search the annotations associated with that DNA sequence can be used or else a search can also be performed by sequence analysis and database searching, BLAST, to compare a query DNA or protein sequence to a sequence database.

2.3.3 UniProtKB/SwissProt and UniProtKB/TrEMBL

UniProtKB/Swiss-Prot is a curated protein sequence database which strives to provide a high level of annotation (such as the description of the function of a protein, its domain structure, post-translational modifications, variants, etc.), a minimal level of redundancy and high level of integration with other databases. Swiss-Prot was produced

collaboratively by the department of Medical Biochemistry at the University of Geneva and EMBL. However, over a period of time the collaboration moved to the Swiss Institute of Bioinformatics (SIB) and the UniProtKB/Swiss-Prot database, as it is now called, is now been maintained by the UniProt Consortium, a collaboration between SIB and the Department of Bioinformatics and Structural Biology of the Geneva University, the European Bioinformatics Institute (EBI) and the Georgetown University Medical Center's Protein Information Resource (PIR).

The UniProtKB/TrEMBL database was created as a computer-annotated supplement of UniProtKB/Swiss-Prot that contains the translations of all coding sequences present in DDBJ/EMBL/GenBank nucleotide sequence database and also protein sequences extracted from the literature or submitted to UniProtKB/Swiss-Prot, which are not yet integrated into UniProtKB/Swiss-Prot.

UniProtKB/Swiss-Prot provides cross-references to external data collections such as DNA sequence entries in the DDBJ/EMBL/GenBank nucleotide sequence databases, 2D and 3D protein structure databases, various protein domain and family characterization databases, posttranslational modification (PTM) databases, species-specific data collections, variant databases and disease databases. UniProtKB/Swiss-Prot is regularly enhanced in its content and format to adequately mirror new findings. It is gradually being enhanced by the addition of a number of features that are specifically intended for researchers working on human genetic diseases, such as, links to human gene databases as well as to many gene-specific mutation databases.

The UniProtKB/Swiss-Prot protein knowledgebase contains 215741 entries and the UniProtKB/TrEMBL protein database contains 2737104 sequence entries comprising 880249930 amino acids.

2.3.4 SGD

The **Saccharomyces Genome Database (SGD)** project collects information and maintains a database of the molecular biology of the yeast *Saccharomyces cerevisiae*. This database includes a variety of genomic and biological information and is maintained and updated by SGD curators. The SGD also maintains the *S. cerevisiae* Gene Name Registry, a complete list of all gene names used in *S. cerevisiae*. The SGD is funded by the National Human Genome Research Institute at the US National Institutes of Health. The SGD is in the Department of Genetics at the School of Medicine, Stanford University. The wealth of information describing the genes and proteins of *S.cerevisiae* has both necessitated and made possible the creation of SGD's new Genome Snapshot, a constantly updated overview of the genome. By making accessible lists of Uncharacterized ORFs, it points researchers to some of the many intriguing questions that remain to be answered about the yeast genome and biological processes. Finally, Genome Snapshot documents the characterization of the genome, both by tracking annotation of ORFs to GO terms and by tracking increases in the number of Verified ORFs.

2.3.5 EBI and EMBL

The **European Bioinformatics Institute (EBI)** was established in 1994 with its headquarters in Heidelberg, Germany. It is a non-profit academic organization that forms part of the **European Molecular Biology Laboratory (EMBL)**. The EBI is a centre for

research and services in bioinformatics. The Institute manages databases of biological data including nucleic acid, protein sequences and macromolecular structures. The Campus also houses the Wellcome Trust Sanger Institute, making it one of the world's largest concentrations of expertise in genomics and bioinformatics. Accessibility to all the data and tools without any restrictions, development of standards to promote data sharing, maintaining comprehensive and up-to-date data sets and making the data and tools portable are some of the services provided by EBI-EMBL.

2.4 PERL

Practical Extraction and Reporting Language (PERL) is a popular programming language that is been extensively used in areas such as bioinformatics and web programming. It is ideally suited for writing programs that goes through mountains of data to just extract the information that is needed.

Perl can deal with information in ASCII text files or flat files which are exactly the kinds of files in which much important biological data appears like in the case of GenBank. Perl makes it easy to process and manipulate long sequences such as DNA and proteins. It also makes it convenient to write programs that controls one or more other programs. However, the only limitation to Perl programming is the speed with which the program executes. In the case of speed of execution, Perl is pretty good but not the best. Other programming languages such as C are preferred to Perl. A program written in C typically runs two or more times faster than the comparable Perl program.

The most common problem found in bioinformatics is parsing BLAST output. The result of a BLAST search is often a multimegabyte file full of raw data. By writing a

simple program in Perl the process of looking for critical information, which can help in obtaining other results, in the BLAST page, can become a fairly easy task.

2.4.1 Regular Expressions

Perl has many features that set it apart from other languages. Of all those features, one of the most important is its strong support for regular expressions. These allow fast, flexible and reliable string handling. Regular Expressions are tiny programs in their own special language, built inside Perl. This is because these programs have only one task: to look at a string and say if it matches or it doesn't match.

The regular expression, often called a pattern in Perl, is a template that matches or doesn't match a given string. The given pattern divides that infinite set into two groups: the ones that match and the ones that don't. A pattern may match one possible string; two or three, a hundred or an infinite number. It may match all strings except for one, except for some or again except for an infinite number.

2.4.2 LWP Modules

The **Library for World Wide Web in Perl (LWP)** is a set of Perl modules and object-oriented classes for getting data from the Web and for extracting information from HTML. It also helps to perform operations such as fetching the web pages, extracting information from them using regular expressions, submitting forms and authentication.

A URL is constructed for the page that needs to be fetched, an HTTP request is made for it and the HTTP response is decoded, then the HTML document is parsed to extract the information that is needed. LWP aids in doing this and makes this task much easier for the user.

2.4.2.1 URI::Escape. The URI::Escape module used as part of this program provides the `uri_escape()` function to help build URLs.

2.4.2.2 LWP::UserAgent. The LWP::UserAgent is a class implementing a web user agent. LWP::UserAgent objects can be used to directly dispatch web requests or it can be subclassed for application specific behavior. It brings together the HTTP::Request, HTTP::Response and the LWP::Protocol classes that form the rest of the core of libwww-perl library.

In normal use the application creates a LWP::UserAgent object, and then configures it with values for timeouts, proxies, name, etc. It then creates an instance of HTTP::Request for the request that needs to be performed. This request is then passed to one of the UserAgent's `request()` methods, which dispatches it using the relevant protocol, and returns a HTTP::Response object.

There are methods for sending the most common request types: `get()`, `head()` and `post()`. The `request()` method of the LWP::UserAgent class processes the content of the response in the 'in core' variant which stores the content in a scalar 'content' attribute of the response object and is suitable for small HTML replies that might need further parsing.

The LWP::UserAgent has a lot of functions associated with it. However, only those functions which have been used in the program are going to be discussed below.

`$ua = LWP::UserAgent->new()`: This class method constructs a new LWP::UserAgent object and returns a reference to it.

`$ua->request($request)`: This class method processes a request, including redirects and security. This method may send several different simple requests.

The **\$request** should be a reference to a HTTP::Request object with values defined for at least the method() and uri() attributes. The content is stored in the response object itself.

\$ua->agent([\$product_id]): This class method Gets/sets the product token that is used to identify the user agent on the network. The agent value is sent as the "User-Agent" header in the requests.

2.4.2.3 HTTP::Request::Common. The HTTP::Request::Common module constructs common HTTP:Request objects. This module provides functions that return newly created HTTP::Request objects. The HTTP::Request::Common has a lot of functions associated with it. However, only those functions which have been used in the program are going to be discussed below.

GET \$url: The get() function returns an HTTP::Request object initialized with the "GET" method and the specified URL. The get(...) method of LWP::UserAgent exists as a shortcut for \$ua->request(GET ...).

2.4.2.4 HTTP::Response. Response objects are returned by the request() method of the LWP::UserAgent. A response consists of a response line, some headers, and a content body. Instances of this class are usually created and returned by the request() method of an LWP::UserAgent object. The HTTP::Request::Common has a lot of functions associated with it. However, only those functions which have been used in the program are going to be discussed below.

\$r->content: This class method gets/sets the raw content and it is inherited from the HTTP::Message base class.

\$r->status_line: This class method returns the string "<code> <message>". If the message attribute is not set then the official name of <code> substituted.

CHAPTER 3

THE PROGRAM FOR DATA COLLECTION

3.1 Before the Program

The initial task of manually collecting the back-translated DNA sequences from TBLASTN was very time consuming and a very tedious job. It required extracting the protein sequences in FASTA format from the SYSTERS database, submitting them to the TBLASTN program on NCBI, making note of the subject range (nucleotide sequence range) from the results produced by TBLASTN, going to the nucleotide sequence web page by hitting on the link provided on the results page and extracting the back-translated DNA sequence from the entire nucleotide sequence mentioned. Only the non-redundant protein sequences from the SYSTERS database are being used for this purpose. One such sequence from the SYSTERS database pertaining to cluster ID 139621 will be used to state an example of the manual process using TBLASTN. This is the tributyltin chloride resistance protein sequence of the *Alteromonas* sp. and it is the first entry in the cluster. It belongs to the SWISSPROT protein database and the accession number of this protein is P32820 and its EMBL accession number is D16369. Its FASTA representation is given below.

```
>SPR|P32820|TBTA_ALTSM (207 AA) Possible tributyltin chloride resistance protein  
[Alteromonas sp. (strain M-1)  
MYNNALHGIYLTQITWMKSARAEPYLYYIVTEVEKRNLPIELALMPLIESDFNAS  
AYSHKHASGLWQLTPAIKQYFKVQISPWYDGRQDVIDSTRAALNFMEYLHKRF  
DGDWYHAI AALNLGEGRVLRAISNIKKNANPLIFQLKTAQNQSVRAKRTSCGTII  
KKPKNAFPAILNSPTIAVLPVDCAVILDNRKQWQQLEIFKPMV
```


The TBLASTN program is available on the NCBI web site at www.ncbi.nlm.nih.gov/BLAST. After connecting to the TBLASTN web page, via the link already mentioned, the protein sequence (query sequence) is pasted into the Search text area on the page. The TBLASTN Search box accepts a number of different types of input and automatically determines the format. It accepts sequences in FASTA format, bare sequences that is sequences without the FASTA definition line and identifiers or accession numbers.

NCBI Blast - Mozilla Firefox
 http://ncbi.nlm.nih.gov/BLAST/Blast.cgi?CMD=Web&LAYOUT=TwoWindows&AUTO_FORMAT=5&auto&ALIGNMENTS=50

NCBI *translating* **BLAST**
 Nucleotide Protein Translations Retrieve results for an RCL

[Search](#)

```
>SPR|P32820|TETA_ALTSM (207 AA) Possible tributyltin chloride
resistance protein [Alteromonas sp. (strain M-1)]
MYNNALHGIYLTQITWMKSARAEPYLYIIVTEVEKRNLP IELALMPLIESDFNASAYSXKHA
SGLWQLTPAIAKYFKVQISPWYDGRQDVIDSTRAALNFM EYHLKRFDDWYHAI AALNLGEG
RVLRAISNINKKANPLIFQLKTAQNQSVRAKRTSCGTI IKKPKNAFPAILNSPTIAVLPVDC
AVILDNRKQWQLEIFKPMV
```

[Choose a translation](#) PROTEIN query - TRANSLATED database [tblastn]

[Set subsequence](#) From: To:

[Choose database](#) nr

[Genetic codes](#) Disabled

Now: **BLAST!** or [Reset query](#) [Reset all](#)

Figure 3.1 Providing the TBLASTN program with the tributyltin chloride resistance protein sequence in FASTA format.

With all the parameters set to their default values and the database option set to 'nr' (protein database), the protein sequence is then submitted to the BLAST system by clicking on the BLAST button on the page.

After clicking on the BLAST button an intermediate page, that is, the formatting BLAST page opens which gives the Request ID (RID) of the query and allows the user to set the formatting options. A unique RID is generated for every request that is submitted to the BLAST system.

NCBI Blast Mozilla Firefox
http://ncbi.nlm.nih.gov/BLAST/Blast.cgi

Query = SPR[P32820]TBT_ALTSM (207 AA) Possible tributyltin chloride resistance protein [Alteromonas sp. (strain M-1)] (206 letters)

The request ID is 1145148604-26681-56387044476.BLASTQ4

Format! or **Resubmit**

The results are estimated to be ready in 2 seconds but may be done sooner.

Please press *FORMAT!* when you wish to check your results. You may change the formatting options for your result via the form below and press *FORMAT!* again. You request results of a different search by entering any other valid request ID to see other recent jobs.

Format

Show Graphical Overview Linkout Sequence Retrieval NCBI.gov Alignment in HTML format

CDS feature

Masking Character: Lower Case Masking Color: Gray

Number of: Descriptions: 100 Alignments: 50

Alignment view: Pairwise

Limit results by: entire query or select from: All organisms

Expect value: range

Figure 3.2 The TBLASTN intermediate result page: The request ID is mentioned in the textbox and the query sequence is given above it.

Upon setting the formatting options and clicking on the FORMAT button a new browser window opens which specifies the estimated time remaining for TBLASTN to compute and display the results.

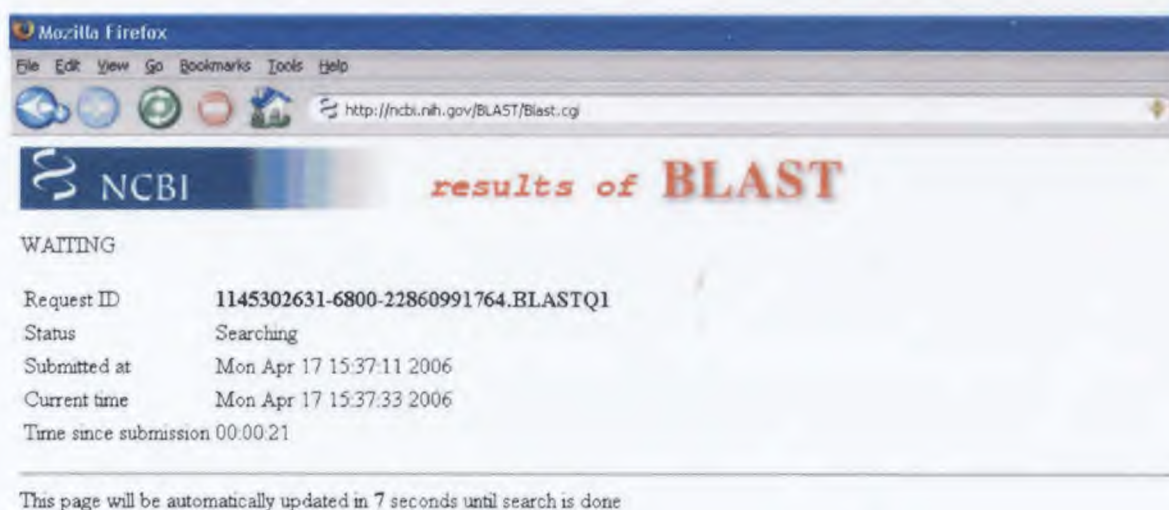


Figure 3.3 The TBLASTN intermediate waiting page: This displays the search status which is 'searching' in this case and the time since the sequence was submitted.

As soon as the search is complete, TBLASTN displays the results in this new window titled "results of BLAST" and contains all the sequences, also known as hits, which are similar to the input query sequence along with the E-value and the Score.

BLAST provides a graphic display on the results page which helps to visualize the results and specifies where the query is similar to the other sequences. Each bar represents the portion of another sequence which is similar to the query sequence and specifies the region where the similarity occurs. Red bars indicate the most similar sequences to the query sequence, pink bars indicate sequences which are less similar, green bars indicate matches that are not good at all and the black and blue bars indicate matches which have the worst scores (poor alignments that have nothing in common and are biologically insignificant). Red, pink and green are usually considered as good hits.

The good part of the graphic display is that it helps to see that some matches do not extend over the complete length of the sequence. It is good for discovering domains. If the mouse is rolled over on the bars then the name of the corresponding sequence appears in a window on top of the display and clicking on any bar would result in displaying the corresponding alignment. Depending on the server that is used this display changes a lot.

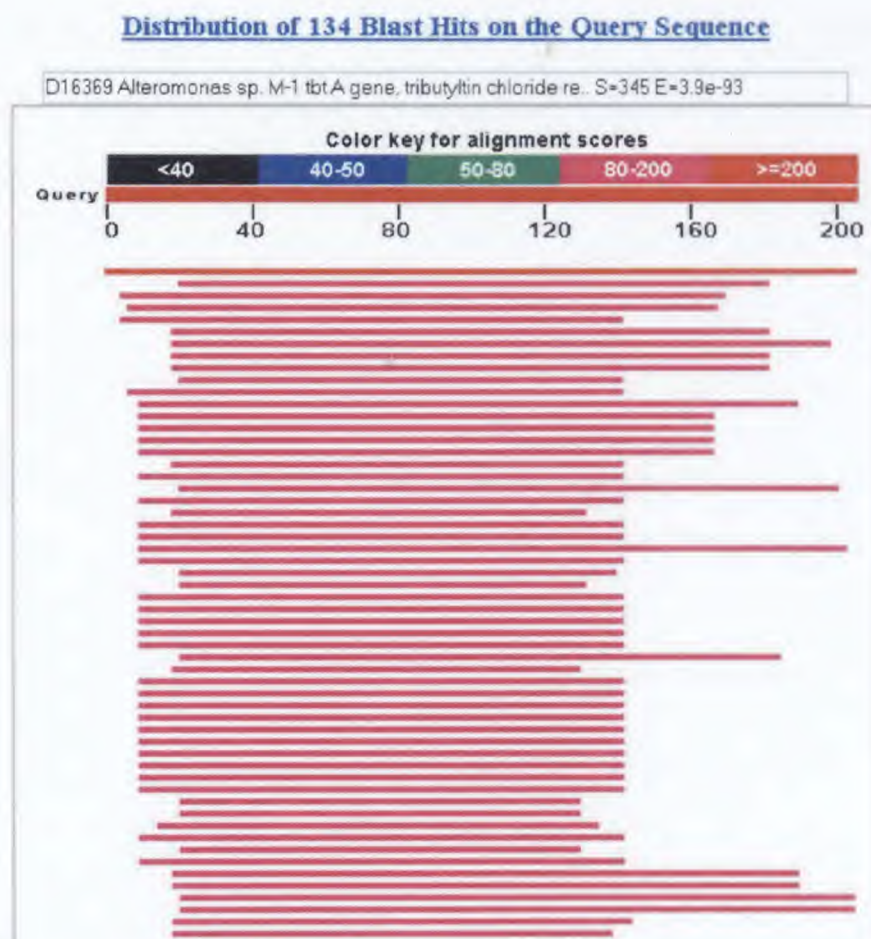


Figure 3.4 The NCBI BLAST graphic display: Red bars indicate the most similar sequences to the query sequence, pink bars indicate sequences which are less similar. Rolling over the mouse on the red bar displays D16369 Alteromonas sp. M-1 tbt A gene, tributyltin chloride re... S = 345 E = 3.9e-93 signifying that this is most similar sequence to the query sequence.

The hit list part of the output also provides useful information to the user. It tells whether the sequence looks like something already in the database and whether it is a good hit. A hyperlink of the sequence accession number and name is specified which links to the database entry that contains the sequence. A description of the sequence is mentioned which comes from the sequence annotations. The bit score is specified which measures the statistical significance of the alignment. The higher the bit score, the more similar are the two sequences. Bit scores below 50 are unreliable and are not considered as good hits. The E-value or the expectation value is the most important measure of statistical significance. The lower the E-value, the more similar are the two sequences.

Sequences producing significant alignments:	Score (Bits)	E Value
gi 303494 dbj D16369.1 ALTTBTA Alteromonas sp. M-1 tbt A gene, t	345	4e-93
gi 46914303 emb CR378672.1 Photobacterium profundum SS9; segmen	159	3e-37
gi 76873893 emb CR954246.1 Pseudoalteromonas haloplanktis st...	150	1e-34
gi 71143482 gb CP000083.1 Colwellia psychrerythraea 34H, comple	141	9e-32
gi 56178122 gb AE017340.1 Idiomarina loihiensis L2TR, complete	140	1e-31
gi 9656789 gb AE004295.1 Vibrio cholerae O1 biovar eltor str...	138	7e-31
gi 46913430 emb CR378669.1 Photobacterium profundum SS9; segmen	137	2e-30
gi 91983532 gb AE016795.2 Vibrio vulnificus MCHP6 chromosome I	137	2e-30
gi 37509034 dbj BA000037.2 Vibrio vulnificus YJ016 DNA, chromos	137	2e-30
gi 59478708 gb CP000020.1 Vibrio fischeri ES114 chromosome I, c	134	1e-29
gi 89949249 gb CP000282.1 Saccharophagus degradans 2-40, comple	132	5e-29
gi 24371479 gb AE014299.1 Shevanelia oneidensis MR-1, complete	132	5e-29
gi 45437263 gb AE017137.1 Yersinia pestis biovar Medievalis ...	132	7e-29
gi 21960007 gb AE013910.1 Yersinia pestis KIM section 310 of 41	132	7e-29
gi 51587641 emb BX936398.1 Yersinia pseudotuberculosis IP32953	132	7e-29
gi 15979072 emb AJ414146.1 Yersinia pestis strain CO92 complete	132	7e-29
gi 47118310 dbj BA000031.2 Vibrio parahaemolyticus RIMD 2210...	131	9e-29
gi 36784324 emb BX571862.1 Photorhabdus luminescens subsp. l...	130	2e-28
gi 83630956 gb CP000155.1 Halobacterium salinarum R1, complet	130	3e-28
gi 84778498 dbj AF008232.1 Sodalis glossinidius str. 'morsitans	129	3e-28
gi 91713371 gb CP000302.1 Shevanelia denitrificans OS217, compl	129	4e-28
gi 49609491 emb BX950851.1 Erwinia carotovora subsp. atroseptic	129	4e-28
gi 27479637 gb AF346500.2 Photorhabdus luminescens strain W1...	126	4e-27
gi 41292 emb X60739.1 ECDNIR E.coli dnaR gene, involved in he...	126	4e-27
gi 71066702 gb AE016828.2 Coxiella burnetii RSA 493, complete g	125	6e-27
gi 66270661 gb AE017282.2 Methylococcus capsulatus str. Bath, c	124	1e-26
gi 82913762 ref XM_723652.1 Plasmodium yoelii yoelii str. 17...	124	1e-26
gi 16418742 gb AE008706.1 Salmonella typhimurium LT2, sectio...	124	2e-26
gi 16501496 emb AL627266.1 Salmonella enterica serovar Typhi...	124	2e-26
gi 56126533 gb CP000026.1 Salmonella enterica subsp. enteric...	124	2e-26
gi 29140506 gb AE014613.1 Salmonella enterica subsp. enteric...	124	2e-26
gi 62126203 gb AE017220.1 Salmonella enterica subsp. enteric...	124	2e-26

Figure 3.5 The NCBI BLAST hit list: The hit list is ordered according to the Score and the E-value.

TBLASTN then displays the alignments below the hit list. The first line of the alignment contains a hyperlink of the accession number and the name of the corresponding nucleotide. The percent identity is a more concrete substitute for the E-value. The 'positives' field gives a measure of the fraction of residues that are either identical or similar and the 'gaps' field shows residues that are not aligned. The length specifies the length of the alignment, indicating the length of the two sequences that have been aligned by the BLAST system. The frame field specifies the reading frame that the sequence is translated in, on both the plus and minus strands.

In TBLASTN the query sequence is translated in three frames on both the plus and minus strands. 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 and similarly reading frame 3 starts at position 3. On the minus strand, the reading frame is calculated relative to the reverse complement of the plus strand, that is -1 corresponds to the last letter and -2 corresponds to the second-to-last position and similarly reading frame -3 starts at the third-to-last letter. 'Query' indicates the query sequence and 'Sbjct' indicates the hit or the subject sequence. The line between the two sequences suggests the alignment between both of them. It contains a (+) sign for similar amino acids, a letter for the identical residues and a space for mismatches. The XXXXX regions are known as low-complexity segments wherein the BLAST system automatically masks the region with of identical residues with Xs. This masking occurs only in the query sequence. The numbers at the sides of the sequences indicate the range of the match on the query sequence and on the hit sequence.

The range of the hit sequence is what is considered to get the back-translated DNA sequence from the nucleotide page. This range is noted and is very essential to the program. Below is the result that TBLASTN produces for the input query sequence. Here information about the raw DNA sequence is mentioned but there is no information about the coding regions, intron/exon and reading frame. The pairwise alignment of the query and the translated nucleotide sequence shows broad areas of similarity.

```
> gi|303494|dbj|D16369.1|ALTTETA Alteromonas sp. M-1 tbt A gene, tributyltin chloride resistance
Length=1602

Score = 345 bits (885), Expect = 4e-93
Identities = 178/207 (85%), Positives = 182/207 (87%), Gaps = 1/207 (0%)
Frame = +3

Query 1 MYNNALHGIYLTQITWMKSARAEPYLYYIVTEVEKRNLPIELALMPLIESDFNASAYSHK 50
        MYNNALHGIYLTQITWMKSARAEPYLYYIVTEVEKRNLPIELALMPLIESDFNASAYSHK
Sbjct 990 MYNNALHGIYLTQITWMKSARAEPYLYYIVTEVEKRNLPIELALMPLIESDFNASAYSHK 1169

Query 61 HASGLWQLTPAIAKYFKVQISPWYDGRQDVIDSTRAALNFMEYLHKRFDGDWYHAI AALN 120
        HASGLWQLTPAIAKYFKVQISPWYDGRQD + F + ++LN
Sbjct 1170 HASGLWQLTPAIAKYFKVQISPWYDGRQDRNRQYPCVEFYGIFTQL*W*LVSRYSSLN 1349

Query 121 LGEGRVLRRAISNIKNKANPLIFQLKTAQ-NQSVRAKRTSCGTIIKKPKNAFPAILNSPTI 179
        LGEGRVLRRAISNIKNKANPLIFQLKTAQ NQSVRAKRTSCGTIIKKPKNAFPAILNSPTI
Sbjct 1350 LGEGRVLRRAISNIKNKANPLIFQLKTAQTNQSVRAKRTSCGTIIKKPKNAFPAILNSPTI 1529

Query 180 AVL PVDCAVILDNRKQWQOLEIFKPMV 206
        AVL PVDCAVILDNRKQWQOLEIFKPMV
Sbjct 1530 AVL PVDCAVILDNRKQWQOLEIFKPMV 1610
```

Figure 3.6 The alignments reported by TBLASTN: The above image is the first hit of the results. The score equals 345 bits and the E value is $4e-93$. The percent identities equal 85% (178/207) indicating that is a very good hit. The positives field has the value of 182/207 (87%) indicating that 87% of the residues are identical or similar and the gaps field has the value 1/207 (0%) indicating that there are no gaps. The frame field refers to the Open Reading Frames (ORFs) and in this case it is the 3rd ORF on the forward strand. The query sequence range is from 1...206 and the subject sequence range is from 990...1610.

Clicking on the hyperlink in the alignment section of the results page opens the nucleotide web page which displays the raw DNA sequence or the mRNA sequence that is the sequence from where the back-translated DNA sequence can be obtained, of the corresponding protein. Of all the results that are displayed only the first hit is considered

as this usually represents the most similar sequence to the input query sequence. The first hit always has a higher score and a less expected value and is the result that is favored from all the other results.

From the figure mentioned above, it is noted that the translation of the protein sequence began at position 990 in the DNA sequence and it ended at position 1610. The DNA sequence between these two positions is the region that has got translated into the corresponding protein and is thus the back translated sequence. The [gi|303494|dbj|D16369.1|ALTTBTA](https://www.ncbi.nlm.nih.gov/nuccore/303494) hyperlink links to entry of the corresponding DNA sequence in the NCBI database and the result which is obtained is mentioned below.

ORIGIN

```

1 aagcttgatg ttcaaagtgg cagtattagg tttaggtgat tctagctatg aatttttctg
61 caaacctgtaa agactttgaa gagegttaac taaattcagg tgccagacgg ttatcatcac
121 ggcgctgacg tgattatgat gacgaagcag cgcagcttgg atygaggcg attaaatgag
181 tttgagccag atttaaaagc ccagcaagtt gcaactagcg gtcaagttgt atcaatgcca
241 tttggcgctc ctgcagcagc cggettagtcc agtaccacca aagccaaaat cccgtttgcc
301 ggctagtcca gtaccaccaa agccaaaatc cggtttgccg gctagagcct taagcttaga
361 caggtagctg taaaaaagat gggaaaagaa tttattgttc gcgtagagaa gttagattgg
421 cttgaatcaa atggcaatta tgtaaatcta cttattgccc tcgaatttcc cttttctcgtg
481 caaccatgac acaacttata actcagttag agccacaagg attttgccg attcaccgtt
541 ctcatgcggt tagattagac gcggtctgaa tcaatcacac cattagcaag tggcgagagt
601 gaagtaaac ttacaacagg aaaagcactt aacttatcgc gtcgctataa agatcaattt
661 aaggaaactgt taaatcaac tacaagtcac tcttaactca cgctatgta aagtaaccta
721 ctttaggtta gccgcattca ttttttcgga gtagttagt acgtgacct taaatcgatt
781 tatccgattt tgccattagt gttaatttta tccggttgtg aaacacacc gaacagccct
841 cttcacaacc tgatattaaa gctaaccaaa tattgttgtt aaagaacaaa acgaaaatat
901 tgagcaata gctcccttaa ggtccctcaga aactcgatga tgtgtggcac gatccgagcc
961 gagttacatt tgccaatcta gccaccaga tgtacaacaa cgcattgcat ggtatttacc
1021 tcaccceaat tacatggatg aaatcagcgc gtgctgagcc ttacctttat tatatagta
1081 cagaggttga aaagcggaac ttaccatag aattagcatt aatgccgcta attgaaagtg
1141 actttaacgc cagtgcctat tgcacaagc atgcactgg actttggcaa ttaacgcctg
1201 ccattgctaa atattttaa gtgcacaat atcccttgta tgaaggagct caggaccgta
1261 atagacagta cccgggctgc gttgaatttt atggaatatt tacacaaacg ctttgatggt
1321 gactggatc acgctatagc agccttaact taggtgaagg ccgtgtaact agagcaatta
1381 gtaataaaa aaacaaggca aacccaactg tttttcaact caaaaactgcc caaacaaacc
1441 agtcagtacg tgccaaaagg actagctgcg gcacaattat caaaaagcca aaaaatgctt
1501 ttccctgcaat tttaaaacagc ccaacaattg cagtattgac tgttgactgc cctgttattt
1561 tagataaccg aaagcaatgg cagcaacttg aaatctttaa accaatggtg tgactcgctt
1621 tggcccaggc aatatgatgc gccccacact gtgttccagt gtgaacaaac acaatttaaa
1681 gacatgctcg ctaatcttga ttccaatgat tatagtcagt ggcaacacta cagtaaaacg
1741 tggtgatagt taagtgttat agcgaaacgc tacaagtag gtattagcca agctcaaacg
1801 tt

```

Figure 3.7 Back-translated DNA sequence: The above figure shows the raw DNA sequence and the highlighted portion indicates the back-translated DNA sequence of the corresponding input protein (query) sequence.

3.2 Flow of the Program

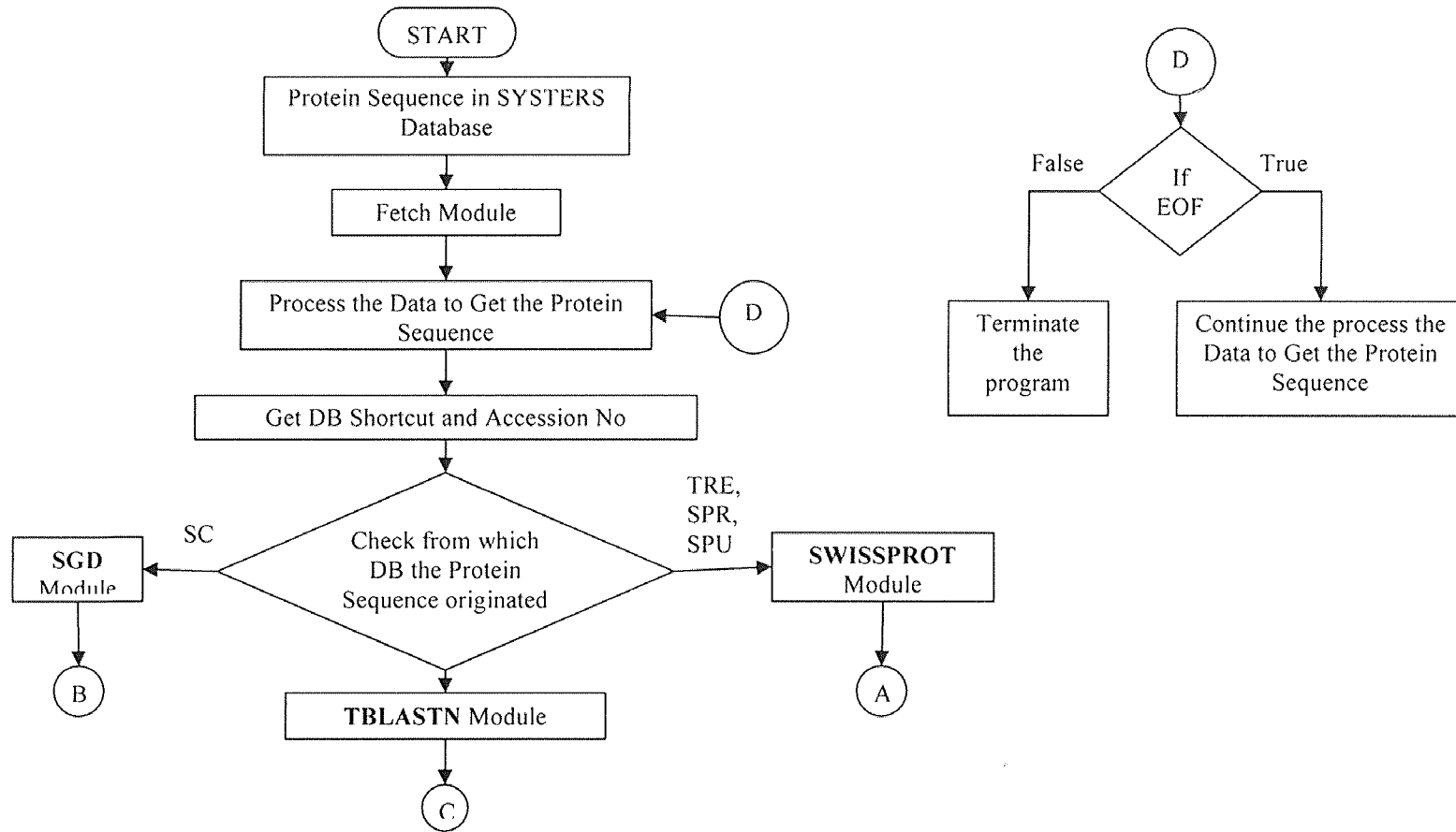


Figure 3.8 Flowchart to demonstrate the working of the program.

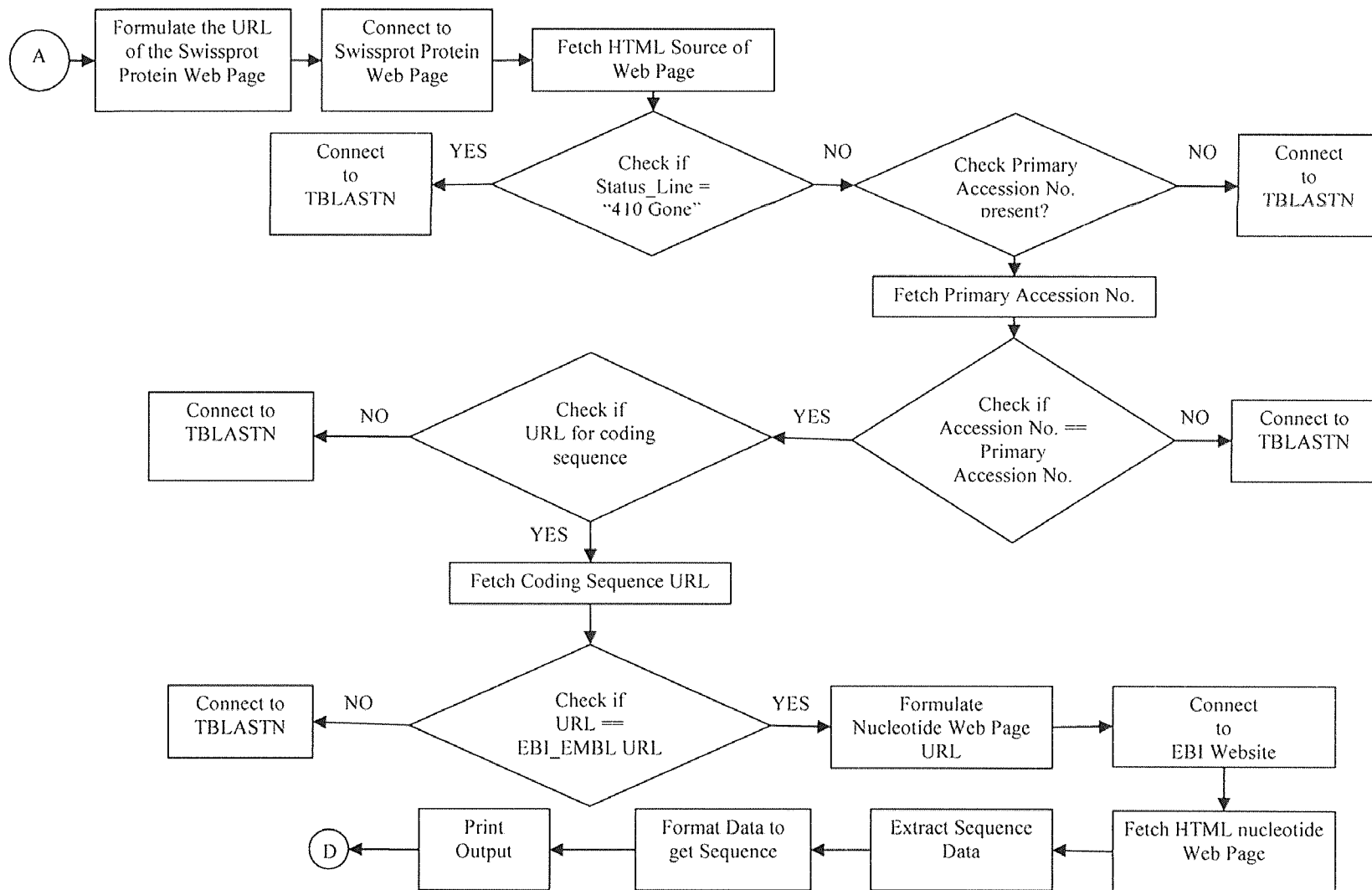


Figure 3.9 Flowchart to demonstrate the working of the swissprot subroutine.

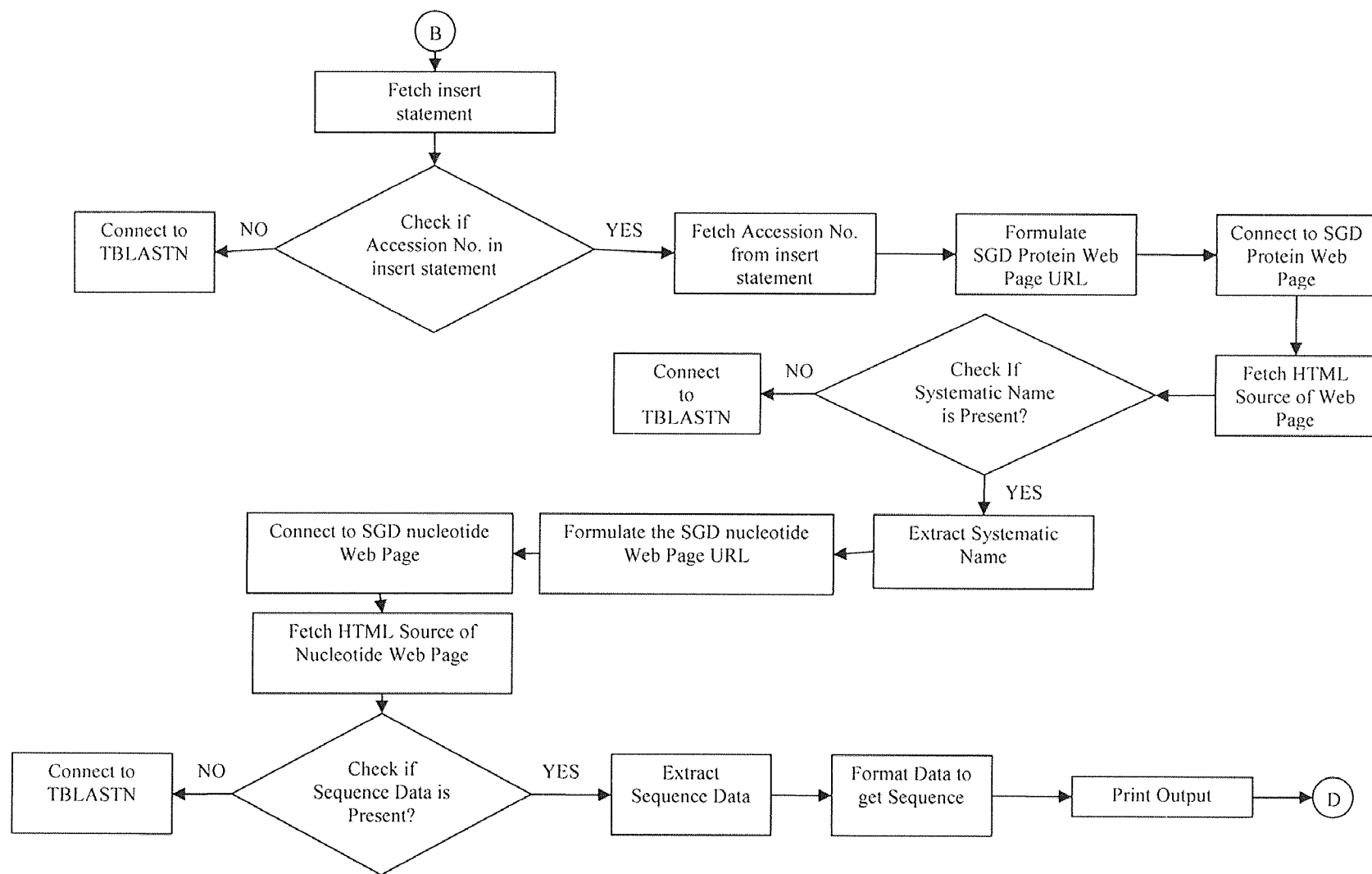


Figure 3.10 Flowchart to demonstrate the working of the SGD subroutine.

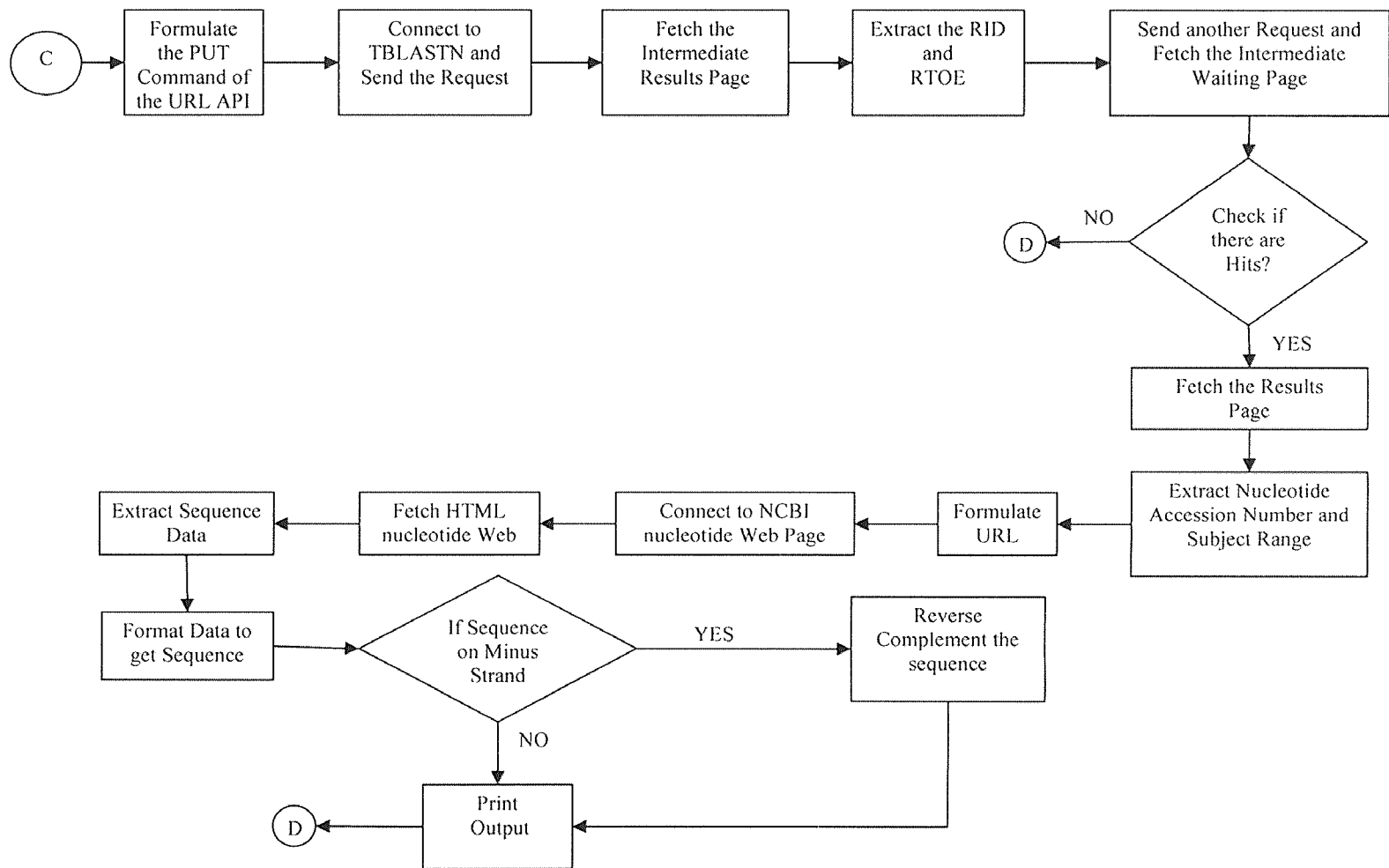


Figure 3.11 Flowchart to demonstrate the working of the tblastn subroutine.

3.3 Program Description

This program performs the task of retrieving and collecting back-translated DNA sequences for proteins which are obtained from the SYSTERS database. It overrides the manual extraction of the back-translated DNA sequences from TBLASTN as mentioned above. Besides TBLASTN it also has the added feature of collecting the back-translated DNA sequences (coding sequences) from other databases and the various web resources available. The program has been coded using the PERL scripting language.

The program begins by asking the user to input the file name in .sql format of the file in which the protein sequences are stored. These protein sequences have been obtained from the SYSTERS database and are stored in the file in the form of insert statements. An example of the insert statements is given below:

```
INSERT INTO `protein` (`pid`, `accno`, `length`, `sequence`, `organism`, `taxon_id`)
VALUES (510187,'ENSCBRP00000012998',208,'>CB|ENSCBRP00000012998 (208
AA) Gene:ENSCBRG00000010527 Clone:c010001328 Contig:c010001328.1.46180
Chr:cb25.fpc1570 Basepair:10448 Status:known
IIVVTPTYKRMTRIADMLRMANTLSHVKDLHWIVIEDGNKTIPAVQDILDRTGLP
YTYQAHKTALGYPRRGWYQRTMALKLIRSNTSQILGQDHQEGVVYFGDDDNSY
DIRLFTDYIRNVKTLGIWAVGLVGGTVVEAPKVVDGKVTAFNVKWNPKRRFAV
DMAGFAVNKVVVLSDAVFGTSCKRGGGAPETCLEDMGLEREDIEP','Caenorha
bditis briggsae',6238);
```

The filename is collected from the user through the command line and is assigned to a variable 'protein_file_name'. Any newline character at the end of the file name is then removed and the file is opened. The newline character is removed with the use of the following regular expression:

```
s/\s//gs
```

A filehandle, with the same name as that of the file, is associated with the file for readability and the file is read into another variable as one single string. A filehandle is a

nickname for the file that is used in the program and is a temporary name assigned to a file. If the file does not open due to any errors then the program exits printing the corresponding information on to the screen. After the file is read it is closed. Another file, 'results_out.txt', is created as the output file to which the results of the program are written to. This file also has a filehandle associated with it. Again if this file does not open due to any errors then program terminates execution printing the corresponding message on to the screen.

A split operation is then performed on the string that stores the entire file. The split operation is performed at the index of the ';' to separate the insert statements which are then stored in an array, 'seq_data'. The length of this array is then determined and is assigned to the variable 'len_seq_data' which is written to the output file to specify the number of sequences in a given file.

The insert statements stored in the 'seq_data' array are then accepted one at a time by a 'for' loop and are executed by the program. Once the 'for' loop executes, the first step is to obtain sequence part the insert statement. Regular Expressions have been used to perform this operation. The two regular expressions that have been used are given below.

```
/'(>.*[\n]((( [A-Z a-z].*\n)+).*) )', '/mg
```

```
/'(>.*\n(([A-Z a-z]).*).*)', '/mg
```

These two regular expressions are used depending on what pattern matches to the insert statements. The first regular expression is used to match an insert statement with two or more lines of the sequence part which is as follows:

```
INSERT INTO `protein` (`pid`, `accno`, `length`, `sequence`, `organism`, `taxon_id`)
VALUES (510206,'Q8JHI9',199,'>TRE|Q8JHI9|Q8JHI9 (199 AA) Beta-1,3-
glucuronyltransferase-3-like protein (Fragment) [Brachydanio rerio (Zebrafish) (Danio
rerio)]
MRLKLTVFVLYFMVSLFGLLYALMQLGQRCDHCRDHEQSKDQQISQLKGELQK
LQEHIKTSELSKKTDPRIYVITPTYARLVQKAELTRLSHTFLHVPQLHWIVVEDA
PQQTQLVSDFLSASGLTYTHLNKLTPKERKLQEGDPNWLKPRGAEQRNEGLRWL
RWMGSTVHGKEAAALEEAVVYFADDDNTYSLQLFEE','Danio rerio',7955);
```

The second regular expression is used to match insert statements which only have one line of the sequence part which is as follows:

```
INSERT INTO `protein` (`pid`, `accno`, `length`, `sequence`, `organism`, `taxon_id`)
VALUES (510506,'Q9QVII',18,'>TRE|Q9QVII|Q9QVII (18 AA) Sucrase-alpha-
dextrinase subunit beta, S-D subunit beta (Fragment) [Rattus sp]
IKLPSNPISLRVEVKYH','Rattus sp.',10118);
```

Both these regular expressions are used to extract the protein sequence in FASTA format (highlighted in grey + yellow) which is written to the 'results_out.txt' file as output, along with the back-translated DNA sequence, to the user and the raw protein sequence (highlighted in yellow), without the FASTA definition line, to submit it as query to the QBLAST system. Both these sequences are then assigned to their respective variables and are stored as strings. Any whitespace characters in the strings are then removed.

After this step, the database shortcut and the accession number from the protein sequence in FASTA format are obtained and are assigned to the variables 'db' and 'accno' respectively in order to determine which database was the sequence originally obtained from and stored and clustered in SYSTERS. Three regular expressions are mentioned depending upon which pattern matches to the protein sequence in FASTA format.

If the protein sequence in FASTA format is as the one given below:

```
>TRE|Q9QVII|Q9QVII (18 AA) Sucrase-alpha-dextrinase subunit beta, S-D subunit
beta (Fragment) [Rattus sp]
IKLPSNPISLRVEVKYH
```

Then the regular expression,

$$/> (.*) \\ | (.*) \\ | .*/ \mathbf{m g}$$

is used to extract the database shortcut (highlighted in grey) and the accession number (highlighted in yellow) from the sequence.

If the protein sequence in FASTA format is as the one given below:

```
>MM|ENSMUSP00000036910 (202 AA) Gene:ENSMUSG00000036479
Clone:9.26000001-27000000 Contig:9.26000001-27000000 Chr:9 Basepair:26824291
Status:known
MPKRRDILAIVLIVLPWTLITVWHQSSLAPLLAVHKDEGSDPRHEAPPGADPRE
YCMSDRDIVEVVRTEYVYTRPPPWSDTLPTIHVVPTPTYSRPVQKAELTRMANTL
LHVPNLHVLVVEDAPRRTPLTARLLRDTGLNYTHLHVETPRNYKLRGDARDPRI
PRGTMQRNLALRWLRETFPRNSTQPGVVVFADDDNTYSL
```

Then the regular expression,

$$/> (.*) \\ | (.*) \\ (\ \mathbf{d} / \mathbf{m g}$$

is used to extract the database shortcut (highlighted in grey) and the accession number (highlighted in yellow) from the sequence.

And if the protein sequence in FASTA format is as the one given below:

```
>CE|C47F8.4 (248 AA) Gene:C47F8.4 Clone:C47F8 Contig:AL009246.1.1.21816 Chr:I
Basepair:12325371 Status:known
MVIVVTPTYKRITRIPDMTRLANTLAHVENLHVLVVEDGYGIVPEVRQMLERTN
LSYTYMAHKTAGYPSRGWYQRTMALRYIRSSSAKILGKQRNGAVVVFADDDN
AYDVRLFTDYIRNVNTLGVWAVGLVGGVVVEAPKVVNQKVTA FNVRWALSRR
FAVDMAGFAINLKLILNSDAVFGTDCKRGE GAPETCLEDMGLKMEDIEPFGYD
ATKVRDIMVWHTKTSPEIEQTDQPIDSLGYFVEY
```


Then the regular expression,

$$/>(.*)\|(.*\.\d+)\ (\d/mg$$

is used to extract the database shortcut (highlighted in grey) and the accession number (highlighted in yellow) from the sequence.

Once the database shortcut and the accession numbers are obtained, the database shortcut is then matched against the SWISSPROT and SGD databases, to access the database in which the protein sequence is stored and accordingly the subroutines are invoked. Only these two databases have been coded for as the coding sequences were readily available on their websites for the corresponding protein sequences. So, if the database shortcut matches to 'TRE', 'SPR' or 'SPU' then the 'swissprot' subroutine is invoked and the accession number of the sequence, the protein sequence in FASTA format and the raw protein sequence are passed as parameters to it. If the database shortcut matches to 'SC' then the 'SGD' subroutine is invoked and the insert statement, stored in the .sql file, along with the protein sequence in FASTA format and the raw protein sequence are passed as parameters to it. Else, if both these matches don't take place then the subroutine 'tblastn' is invoked and the protein sequence in FASTA format along with the raw sequence, are passed as parameters to it. Two counters, one each for the 'swissprot' subroutine and the 'SGD' subroutine, are also mentioned to indicate how many back-translated DNA sequences have been obtained from each of these databases.

Now moving on to the description of the subroutines, we first have the 'swissprot' subroutine. This subroutine, as mentioned above, takes the accession number, the protein sequence in FASTA format and the raw protein sequence as parameters. The protein sequence in FASTA format and the raw protein sequence are moreover used to be passed

as parameters to the 'tblastn' subroutine in case the 'swissprot' subroutine fails at any step. The most important part is the accession number that is passed to this subroutine. This accession number is assigned to the variable 'acc_no'.

A new LWP::UserAgent object is then created and a reference to it is made and assigned to 'ua_swissprot' and the UserAgent object is assigned an identity, Agent03. The URL for swissprot is then formed in order to connect to it and fetch the web page of the corresponding protein sequence. The URL to link to swissprot is 'http://ca.expasy.org/uniprot/' at the end of which the accession number has to be appended in order to complete the URL. This is achieved by merging the accession number parameter that is passed to the subroutine to the URL to get the complete merged URL of the webpage that needs to be fetched. For example, the merged URL would look something like this, 'http://ca.expasy.org/uniprot/Q9UEB4'. The merged URL is then assigned to the 'merge_url' variable. After this step, a new HTTP::Request object, 'req_swissprot', is created where in the URL is passed to the GET command of the HTTP::Request object. The UserAgent request method is then called and the HTTP::Request object is passed to it as a request. This is assigned to the HTTP::Response object, 'res_swissprot'.

UniProtKB/TrEMBL entry
Q9UEB4

[Entry info](#) [Name and origin](#) [References](#) [Comments](#) [Cross-references](#) [Keywords](#) [Features](#) [Sequence](#) [Tools](#)

Note: most headings are clickable, even if they don't appear as links. They link to the user manual or other documents.

Entry information

Entry name	Q9UEB4_HUMAN
Primary accession number	Q9UEB4
Secondary accession numbers	None
Integrated into TrEMBL on	May 1, 2000
Sequence was last modified on	May 1, 2000 (Sequence version 1)
Annotations were last modified on	April 18, 2006 (Entry version 14)

Name and origin of the protein

Protein name	WUGSC:H_RG158K20.1 protein [Fragment]
Synonyms	None

Figure 3.12 The Swissprot protein web page for the <http://ca.expasy.org/uniprot/Q9UEB4> URL which is obtained as result when the URL is passed as a request to the HTTP::Request object. The highlighted portion is the primary accession number which is extracted from the web page.

The UNIPROTKB/SWISSPROT AND UNIPROTKB/TrEMBL website at times results into error when the sequence is not found, that is it no longer exists in the database. It results into the '410 gone' error and is caught using the `status_line()` method of the HTTP::Response object. In case of this error the `tblastn` subroutine is invoked and once the `tblastn` subroutine gets executed the next iteration is performed.

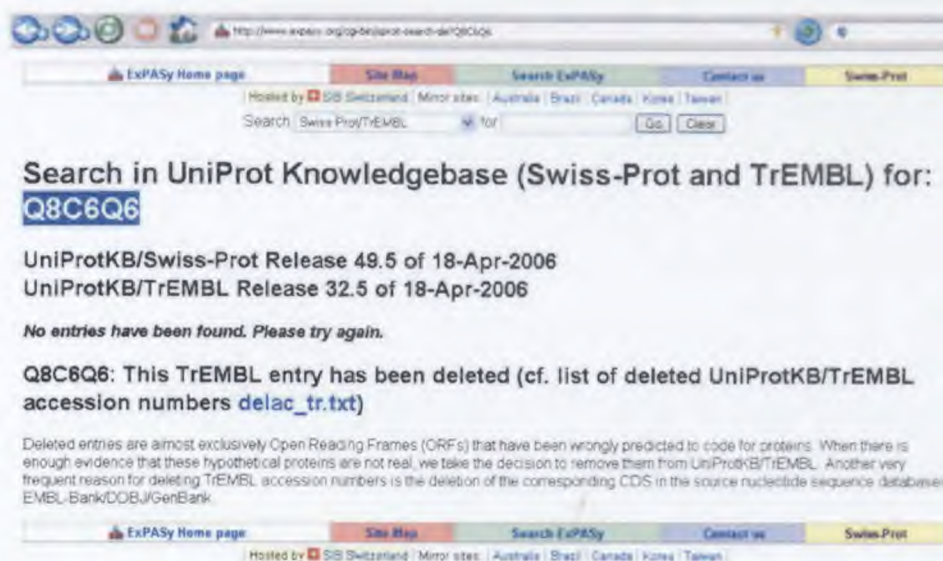


Figure 3.13 The Swissprot protein webpage when the page results into the '410 Gone' error.

If there is no error, then the content of the webpage is checked to see if there is a primary accession number (highlighted in blue in Figure 3.8) present on the page. This is done by using the following regular expression.

`/Primary accession number<.*\n.*(.*?)/m`

If the regular expression does not get matched to the content of the web page then the `tblastn` subroutine is invoked. However, if the primary accession number is present then it is extracted and assigned to the variable `'a_no'`. This accession number is then matched to the accession number of the protein sequence which is passed as a parameter to the `'swissprot'` subroutine. This is done because at times a protein sequence has one primary accession number and a number of secondary accession numbers indicating that parts of this protein also encode for other genes besides the gene encoded by the whole protein. If the two accession numbers do not match, it means that the wrong page has been fetched and the `tblastn` subroutine is invoked. If the two accession numbers match

the content of the web page is further parsed to check if there is a corresponding coding sequence present for the protein sequence. The coding sequence is given as a Genomic_DNA or an mRNA entry on the protein sequence web page and it is hyperlinked to the nucleotide web page as shown in the Figures 3.10 & 3.11 below.

Cross-references	
Sequence databases	
EMBL	AE011700; AAM35598 1; -, Genomic_DNA [EMBL / GenBank / DDBJ] [CoDingSequence]
3D structure databases	
ModBase	Q8PPH7
2D gel databases	
SWISS-2DPAGE	Get region on 2D PAGE
Organism-specific gene databases	

Figure 3.14 A part of the Swissprot protein web page which highlights the Genomic DNA Coding Sequence hyperlink, the URL of which is extracted from the HTML source code of the page and is used to obtain the nucleotide sequence web page.

Cross-references	
Sequence databases	
EMBL	AF288821; AAG16228 1; -, mRNA [EMBL / GenBank / DDBJ] [CoDingSequence]
3D structure databases	
HSSP	P02791; 1DAT [HSSP ENTRY / PDB]
SMR	Q9GLE7; 2-160
ModBase	Q9GLE7
2D gel databases	

Figure 3.15 A part of the Swissprot protein web page which highlights the mRNA Coding Sequence hyperlink, the URL of which is extracted from the HTML source code of the page and is used to obtain the nucleotide sequence web page.

The coding sequences for the mRNA hyperlink as well as for the Genomic_DNA hyperlinks are the same when compared to each other. Therefore, the regular expressions in the 'if' condition can either match to the Genomic_DNA coding sequence or the mRNA coding sequence on the protein web page. The two regular expressions are given as follows:

```
^/d; -; Genomic_DNA.*\[<a href="(.)">CoDingSequence</a>\]/mg
```

```
^/d; -; mRNA.*\[<a href="(.)">CoDingSequence</a>\]/mg
```

If the patterns do not match either entry then the `tblastn` subroutine is invoked. If the match occurs then the regular expressions extracts the URL of the website where the coding sequence is located.

After visiting many protein sequence web pages and their corresponding coding sequences it has been concluded that the URL mainly links to the EBI-EMBL website and the program has been coded to get the nucleotide sequence from this website. If the URL links to any other website then the `tblastn` subroutine is invoked and executed.

The URL is stored in the variable 'temp_swissprot1'. The URL to the EBI-EMBL website is given as follows in the HTML source of the protein sequence web page.

```
http://www.ebi.ac.uk/cgi-bin/dbfetch?db=emblcds&id=AAC04618
```

The URL address is then edited and corrected by the substitute operation to substitute the '&' by '&' to give the URL,

```
http://www.ebi.ac.uk/cgi-bin/dbfetch?db=emblcds&id=AAC04618
```

which is the URL that links to the EBI-EMBL nucleotide web page specifying the sequence ID and the database as the EMBL coding sequence database.

Once the URL has been determined that it links to the EBI-EMBL website a new `LWP::UserAgent` object is created and a reference to it is made and assigned to 'ua_ebi' and the `UserAgent` object is assigned an identity, `Agent04`. The URL which links to EBI-EMBL nucleotide web page is assigned to 'ebi_url'. After this step a new `HTTP::Request` object, 'req_ebi', is created and the URL is passed to the `GET` command of the `HTTP::Request` object. The `UserAgent` request method is then called and the

HTTP::Request object is passed to it as a request. This is assigned to the HTTP::Response object, 'res_ebi'. The content of the response object which is obtained as a result from the request is assigned to 'temp_ebi'. The content which is received in the response object is the nucleotide web page from EBI-EMBL.

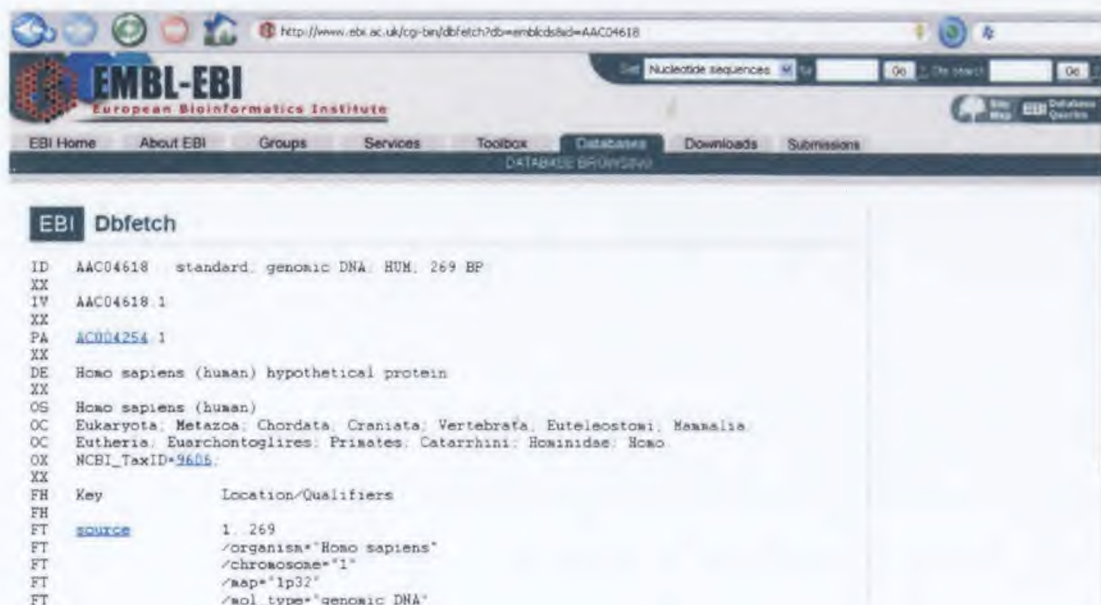


Figure 3.16 The EBI-EMBL nucleotide sequence web page which is obtained as a result of passing the coding sequence URL as a request to the HTTP::Request object.

This HTML page is then parsed to obtain the coding sequence. The sequence part of the page is extracted using the following regular expression and is shown in Figure 3.16:

$$/(SQ.*\n+ (. *\n+)+)\n+mg$$

```

FT  CDS          join(complement(AC004254 1:12979..>13095),
FT              complement(AC004254 1:6516..6575),
FT              complement(AC004254 1:(1675..1766)))
FT              /codon_start=1
FT              /gene="WUGSC:H_RG158K20.1"
FT              /note="similar to C. elegans hypothetical protein, similar
FT              to AF038615 (PID_g2736329), H_RG158K20.1"
FT              /db_xref="GOA:Q9JEB4"
FT              /db_xref="InterPro:IPR006055"
FT              /db_xref="InterPro:IPR012337"
FT              /db_xref="UniProtKB/TrEMBL:Q9JEB4"
FT              /inference="non-experimental evidence, no additional
FT              details recorded"
FT              /protein_id="AAC04618.1"
FT              /translation="EIIIEFPILKLNRTMEIESTFHMVQPVVHPQLTPFCTELTGIIQ
FT              AMVDGQPSLQQVLERVDEVMMAKEGLLDPNVKSIFVTCGDWDLKV"
XX
SQ  Sequence 269 BP: 72 A, 65 C, 67 G, 65 T; 0 other; 2563293887 CRC32;
    gaaatcatcg agttccocat cctaaagata aatggcogga ccatggagat tgagtctacc 60
    ttccacatgt atgtccagcc tgtagtccat cccacagctta cccattctg tacagagctc 120
    acogggatta ttcaagccat ggtggatggt cagcccaagc tgcagcaagt gatggagagg 180
    gtogatgat ggtggcgaa ggaagggcctc ttgatccaa acgtcaagtc aatttttctc 240
    acctgtggag actgggactt aaaagtcac 269

```

Figure 3.17 The EMBL-EBI nucleotide sequence web page which highlights the nucleotide sequence that is extracted from the HTML source code of the page using the regular expression mentioned above.

This intermediate result is stored in the variable 'temp_ebi1'. Once the sequence is stored in the 'temp_ebi1' variable substitute operations are performed on the variable removing any whitespace characters, line number and any other text besides a, c, t, g. The coding sequence obtained is then stored as a single string 'dna_seq' which forms the final back-translated DNA sequence from the swissprot subroutine. The back-translated DNA sequence along with its corresponding protein sequence in FASTA format are then written to the output file 'results_out.txt' as well as printed on the console as the final result. A flag is mentioned that the back-translated DNA sequence is obtained from Swissprot/TrEMBL.

The second subroutine is the 'SGD' subroutine which is written to extract the coding sequences for the protein sequence from the Saccharomyces Genome Database. To this subroutine the insert statement, the protein sequence in FASTA format and the raw protein sequence are passed as parameters. The protein sequence in FASTA format

and the raw protein sequence are moreover used to be passed as parameters to the 'tblastn' subroutine in case the 'swissprot' subroutine fails at any step.

The insert statement for the SC entry is used to extract the accession number as it is not mentioned in the FASTA definition line of the protein sequence. The insert statement for an SC entry is as follows and is assigned to 'prot_seq'

```
INSERT INTO `protein` (`pid`, `accno`, `length`, `sequence`, `organism`, `taxon_id`)
VALUES (516549, 'S0004650(NR_SC:SW-N116_YEAST)', 1113, '>SC|NR_SC:SW-
N116_YEAST (1113 AA) SW:N116_YEAST Q02630 saccharomyces cerevisiae
(baker\'s yeast). nucleoporin nup116/nsp116 (nuclear pore protein nup116/nsp116).
11/1997; PIR:S28925 nuclear pore complex protein NSP116 -
MFGVSRGAFPSATTQPFGSTGSTFGGQQQQQQPVANTSAFGLSQQTNTTQAPAF
GNFGNQTSNSPFGMSGSTTANGTPFGQSQLTNNNASGSIFGGMGNNNTALSAGSA
SVVPNSTAGTSIKPFTTFEEKDPTTGVINVFQSITCMPEYRNFSFEELRFQDYQAG
RKFGTSQNGTGTTFNNIERLKKNPNSKFESYDADSGTYVVFIVNHAAEQT', 'Sacchar
omyces cerevisiae', 4932);
```

The accession number of the protein sequence (highlighted in yellow) is extracted from the insert statement using the following regular expression,

$$/\backslash(\backslash d + , '(. *) \backslash (. * \backslash)' / m$$

and is assigned to the variable 'prot_acc_no'. After this step a new LWP::UserAgent object is created and a reference to it is made and assigned to 'ua_ebi' and the UserAgent object is assigned an identity, Agent05. The URL of the SGD website to link to it is then formed in order to connect to it and fetch the web page of the corresponding protein sequence. The URL to link to SGD is 'http://db.yeastgenome.org/cgi-bin/locus.pl?sgdid=' at the end of which the sgdid parameter takes the accession number and it has to be appended in order to complete the URL. This is achieved by merging the accession number stored in 'prot_acc_no' to the URL to get the completed merged URL of the webpage that needs to be fetched. For example, the merged URL would look something like this, 'http://db.yeastgenome.org/cgi-bin/locus.pl?sgdid= S0000433'. The

merged URL is then assigned to the 'merge_url' variable. After this step a new HTTP::Request object, 'req_yeastgenome', is created where in the URL is passed to the GET command of the HTTP::Request object. The HTTP::Request object is then passed to the request method of the UserAgent object to send a request to the desired URL and get a response. The response which is obtained from the request sent is stored in the HTTP::Response object 'res_yeastgenome'.

The content stored in the response object is assigned to the variable 'temp_yeastgenome'. The content of 'temp_yeastgenome' is the SGD HTML page which contains the protein sequence.

The screenshot shows the SGD protein web page for ROT2/YBR229C. The page is titled "ROT2/YBR229C Summary" and includes a navigation bar with "Quick Search" and "Submit" buttons. Below the navigation bar, there are tabs for "Summary", "Locus History", "Literature", "Gene Ontology", "Phenotype", "Interactions", "Expression", and "Protein". The "Summary" tab is selected, displaying "ROT2 BASIC INFORMATION" and "ROT2 RESOURCES".

ROT2 BASIC INFORMATION [View References]

Standard Name	ROT2
Systematic Name	YBR229C
Alias	GLS2
Feature Type	ORF, Verified
Description	Glucosidase II catalytic subunit required for normal cell wall synthesis, mutations in rot2 suppress tor2 mutations, and are synthetically lethal with rot1 mutations (1, 2, 3 and see Summary Paragraph)
GO Annotations	ROT2 GO evidence and references
Molecular Function	<ul style="list-style-type: none"> alpha-glucosidase activity (IDA)
Biological Process	<ul style="list-style-type: none"> cell wall biosynthesis (sensu Fungi) (IMP)
Cellular Component	<ul style="list-style-type: none"> endoplasmic reticulum (ISS)

ROT2 RESOURCES

Click on map for expanded view

SGD ORF map GBrowse

• Literature

Literature Guide [View]

• Retrieve Sequences

Genomic DNA [View]

Figure 3.18 The SGD protein web page for the `http://db.yeastgenome.org/cgi-bin/locus.pl?sgdid= S0000433` URL which is obtained as result when the URL is passed as a request to the HTTP::Request object. The highlighted portion is the Systematic Name which is extracted from the HTML source code of the web page using regular expressions.

The HTML page is then parsed to get the Systematic Name (highlighted in purple in figure 15) of the protein which is used in the formation of the URL to the web page of the coding sequence. Some of the protein pages have the alias feature after the systematic name and some of the pages have the feature type entry after the systematic name specifies. Thus, two regular expressions have been used to match the two conditions and if the patterns match then systematic name of the protein sequence is extracted from the content. The two regular expressions that have been used for this purpose are as follows:

`/Systematic Name.*"top">(.*?)<\/td><\/tr><\/table>.*Alias/m`

`/Systematic Name.*"top">(.*?)<\/td><\/tr><\/table>.*Feature Type/m`

The systematic name is assigned to the variable 'temp1_yeastgenome'. If the patterns do not match the regular expressions then the tblastn subroutine is invoked.

Once the systematic name is obtained then another LWP::UserAgent object is created and the UserAgent is given an identity as Agent 06. The URL to connect to and fetch the corresponding nucleotide\coding sequence is then formed. The URL to link to the nucleotide web page is 'http://db.yeastgenome.org/cgi-bin/getSeq?seq=YBR229C&flankl=0&flankr=0&map=n3map'. This URL is then broken into three parts. The first part 'http://db.yeastgenome.org/cgi-bin/getSeq?seq=' is assigned to variable 'yeastgen_url' and the third part 'flankl=0&flankr=0&map=n3map' is assigned to the variable 'yeastgen_url_merge'. The middle part is the systematic name which is obtained from the regular expressions mentioned above. These three parts are then merged to form the complete URL and it is assigned to the variable 'yeastgen_merged_url'.

After this step, a new HTTP::Request object, 'req_yeastgen', is created where in the URL is passed to the GET command of the HTTP::Request object. The HTTP::Request object is then passed to the request method of the UserAgent object to send a request to the desired URL and get a response. The response, which is obtained from the request sent, is stored in the HTTP::Response object 'res_yeastgen'.

The content of the 'res_yeastgen' object is an HTML source code of the SGD nucleotide/coding sequence web page.

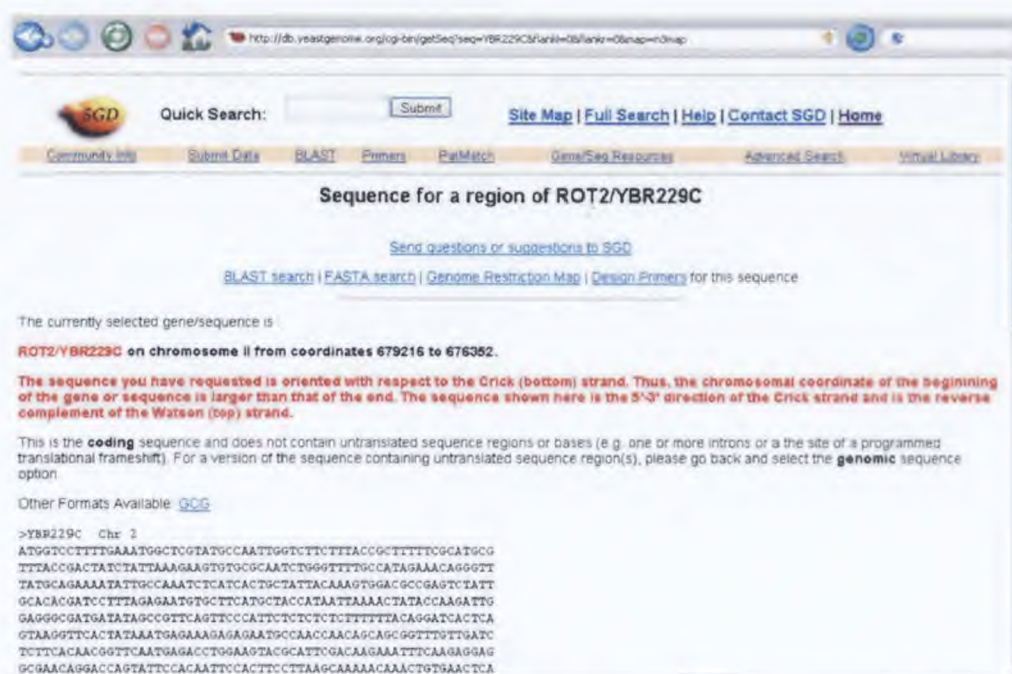


Figure 3.19 The SGD nucleotide sequence web page which is obtained as a result of passing the coding sequence URL, `http://db.yeastgenome.org/cgi-bin/getSeq?seq=YBR229C&flankl=0&flankr=0&map=n3map`, as a request to the HTTP::Request object.

From this content the sequence part of the page is extracted using regular expressions. Firstly, it uses an 'if' condition to determine if the web page has a nucleotide sequence contained in it. If the sequence is not found or the webpage results into any

errors then the `tblastn` subroutine is invoked. However, if the regular expression is matched to the pattern on the web page then the sequence part of the web page is extracted. The following regular expression is used to perform this operation.

```
/<pre>>.*\n(( [ACTG] .* \n )+)<\/pre>/m
```



Figure 3.20 The SGD nucleotide sequence web page which highlights the nucleotide sequence that is extracted from the HTML source code of the page using the regular expression mentioned above.

Any whitespace character is removed using the substitution operation and the final back-translated DNA sequence is obtained as a single string. The DNA sequence is usually in uppercase letters and is converted to lowercase letters and assigned to the variable `'dna_sequence'`. The value stored in `'dna_sequence'` along with its corresponding sequence in FASTA format is then written to the file and also printed out onto the console as the final output. A flag is mentioned that the back-translated DNA sequence is obtained from yeastgenome.

Finally, the program description ends with the description of the `tblastn` subroutine. This subroutine is called a couple of times throughout the program. The `tblastn` subroutine is called at any time the other routines fail to execute a certain step. This is the most important part of the program. This is the subroutine which connects to the NCBI Blast program `TBLASTN`, sends the query request to it, processes the request and returns the final result. The `tblastn` subroutine as mentioned before takes in the protein sequence in FASTA format and the raw protein sequence as parameters. One more thing to note is that before every `tblastn` routine call is made a wait period of two minutes is mentioned. This is because the QBLAST system requires that between every two request there should be a minimum gap of at least one minute.

In the beginning of the sub routine a flag variable is initialized. The use of this variable is described later. A new `LWP::UserAgent` object, `'ua_tblastn'`, is created and the `UserAgent` is given an identity as `Agent 01`. The `put` command of the URL API is formed and is assigned to the variable `'args'`. The query parameter of the command specifies the raw protein sequence which is passed to the subroutine. The database parameter specifies the nucleotide database with which the query sequence is compared. The `'hitlist_size'` parameter indicates that only the first hit is to be returned from the result set and the program parameter specifies that the `tblastn` program of the BLAST family is to be executed. The `put` command is as follows

```
CMD=Put&QUERY=${_[0]}&DATABASE=nr&HITLIST_SIZE=1&FILTER=L&EXPECT=1&FORMAT_TYPE=HTML&PROGRAM=tblastn&CLIENT=web&SERVICE=plain&NCBI_GI=on&PAGE=nucleotides";
```

After the `put` command is formed, a new `HTTP::Request` object, `'req_tblastn'` is created and the `put` command stored in `args` is passed to the `get` method of the class. The

HTTP::Request object is then passed to the request method of the UserAgent object which in turn passes the command to the QBLAST system and gets a response. The response is stored in the HTTP::Response object, 'res_tblastn'. The URL API put command when sent via the request object invokes the blast.cgi script and the tblastn program begins execution.

The response that is obtained and stored in the response object is the intermediate result page which allows the user to set the formatting options. However, the result of this page is not displayed on the output screen. All the formatting options that the user needs to set can be set in the put command of the QBlast URL API. From the content of this page, which is the HTML source code of the intermediate results page stored in the 'res_tblastn', the request ID and the remaining time of execution are extracted using the following regular expressions:

```
/^   RID = (. * $) / m
```

```
/^   RTOE = (. * $) / m
```

```
</tr>
<tr><td align="left" bgcolor="#FF6600" colspan="2" height="4"><img align="midd
</table>
<input name="RID" type="hidden" value="1145994353-15448-140490310741.BLASTQ1">
<!--QBlastInfoBegin
   RID = 1145994353-15448-140490310741.BLASTQ1
   RTOE = 14
QBlastInfoEnd
--></form>

</body>
</html>
```

Figure 3.21 Part of the HTML source code of the intermediate results page: It highlights the Request Id (RID) and the Remaining Time Of Execution (RTOE) which is extracted from the sources using the above mentioned regular expressions.

The `tblastn` program issues a unique RID for every request that is passed to it. It calculates the remaining time of execution (RTOE) which is an estimate amount of time that the program needs to produce the results. During this time the input query is being processed and the `tblastn` program searches for the results against the nucleotide database GENBANK. These values are stored in the corresponding variables, 'rid' and 'rtoe'. The program is then made to wait for the time period value stored in 'rtoe'. While the intermediate page state is true the program creates a new `HTTP::Request` object, 'req_tblastn', which overrides the previous 'req_tblastn' request object. Through this object a URL is passed as a request to the QBLAST system. This URL is the link to the intermediate waiting page of TBLASTN. The request ID is passed as a parameter and the output pertaining to that request ID is stored in the `HTTP::Response` object. This response is then parsed in order to obtain the status of the query. If the status on the page is shown as waiting then the program prints the statement 'Searching'. This page is refreshed after sometime and each time it is refreshed, that is as long as the while statement is true, the search takes place. If the status results in FAILED then the protein sequence along with the error message, 'Search failed;please report to blast-help@ncbi.nlm.nih.gov.\n' is written to the output file and the program exits the subroutine. If the status results in UNKNOWN then the protein sequence along with the error message, 'Search expired.\n' is written to the output file and the program exits the subroutine. However, if the status displays ready, it means that the search has ended and TBLASTN is now ready to display the results and the if condition also checks to see if there are any hits present for the input query sequence or there are no hits present.

If there are hits present then message, 'Search complete, retrieving results...', is printed on the console and the program execution is forced out of the while loop with the 'last' statement. At this point the flag is turned to 1. However, if no hits are found then the protein sequence in FASTA format along with the message, 'No hits found', is written to the output file. The program control then exits the 'while' loop and the next iteration begins. If none of the above conditions mentioned are fulfilled then the program reaches the end of the while loop where the message 'if we get here, something unexpected happened' is printed on the output screen. The program reaches this point if there is no RID specified for the input query sequence, that is, when an empty string is (no input query sequence) is passed to the program and the tblastn program cannot specify an RID, or when the NCBI BLAST program blocks access to user.

```

Telnet afs13.njit.edu
<tr><td>Time since submission</td>
<td>00:00:19</td>
</tr></table>
<p><hr>This page will be automatically updated in <b>5</b> seconds until search
is done<br>
</form>

Searching...

<HTML>
<HEAD>
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
</HEAD>
<BODY BGCOLOR="#FFFFFF" LINK="#0000FF" ULINK="#660099" ALINK="#660099">
<IMG SRC="images/head_results.gif" ALT="Header of the page" BORDER="0" NAME="Blast
HeaderGif" WIDTH="600" HEIGHT="45" ALIGN="middle">
<form action="Blast.cgi" enctype="application/x-www-form-urlencoded" method="POST">
<input name="FORMAT_OBJECT" type="hidden" value="SearchInfo"><input name="RID"
type="hidden" value="1145995794-23707-12720328832.BLASTQ4"><input name="SEARCH
_DB_STATUS" type="hidden" value="31"><input name="_PGR" type="hidden" value="0">
<input name="_PGR" type="hidden" value="0"><input name="CMD" type="hidden" value
="Get"></form>
<form action="Blast.cgi" enctype="application/x-www-form-urlencoded" method="POST"
TARGET=""><input name="FORMAT_OBJECT" type="hidden" value="SearchInfo"><input
name="RID" type="hidden" value="1145995794-23707-12720328832.BLASTQ4"><input name="SEARCH_DB_STATUS" type="hidden" value="31"><input name="_PGR" type="hidden" value="0"><input name="CMD" type="hidden" value="Get">WAITING
</form>
<HTML>
<p></p>
<!--
QBlastInfoBegin
      Status=WAITING
QBlastInfoEnd
--><p></p>
<SCRIPT LANGUAGE="JavaScript"><!--
setTimeout('document.forms[0].submit();',5000);
--></SCRIPT>
</table>
<tr><td>Request ID</td><td> <b>1145995794-23707-12720328832.BLASTQ4</b></td></tr>
<tr><td>Status</td><td>Searching</td></tr>
<tr><td>Submitted at</td><td>Tue Apr 25 16:09:54 2006</td></tr>
<tr><td>Current time</td><td>Tue Apr 25 16:10:19 2006</td></tr>
<tr><td>Time since submission</td>
<td>00:00:24</td>
</tr></table>
<p><hr>This page will be automatically updated in <b>5</b> seconds until search
is done<br>
</form>

Searching...

```

Figure 3.22 The intermediate waiting page of TBLASTN: The status of the program is shown as Waiting and also the RID is specified, which indicates that TBLASTN is searching for results for the RID mentioned.

Therefore, if there are results returned by the tblastn program then the flag is set to 1 and the program continues execution to retrieve the back-translated DNA sequence for the input protein sequence. A new HTTP::Request object, 'req_tblastn', is created and via the request object the get command of the URL API is passed to the QBLAST system specifying the URL and the Request ID to get the results pertaining to that RID. The URL sent is as follows,

`http://www.ncbi.nlm.nih.gov/blast/Blast.cgi?CMD=Get&FORMAT_TYPE=Text&RID=$rid`

where '\$rid' is substituted by the value of the Request ID stored in it.

The HTTP::Request object is then passed to the request method of the UserAgent object to send a request to the desired URL and get a response. The response which is obtained from the request is stored in the HTTP::Response object, 'res_tblastn'.

The response content of 'res_tblastn' is the blast results page which gives the results pertaining to the input query sequence. This is the web page which contains the NCBI BLAST graphic display along with the hits and alignments sections. However, the page content obtained from the Qblast system differs from the actual web page (Figure 20). The content is assigned to a variable 'tblastn_results_page'.

The response that is obtained from the QBLAST system and stored in 'tblastn_results_page' is an HTML web page which is parsed to obtain the accession number of the raw nucleotide sequence. Also the start and end positions of the subject sequence or the nucleotide sequence are extracted. In order to obtain this, only the alignment section of the result page is extracted using the regular expressions

```
s /. * ^ A L I G N M E N T S // m s
```

```
s / ^   D a t a b a s e : . * //
```

```

Telnet afs13.njit.edu
Database: All GenBank+EMBL+DDBJ+PDB sequences (but no EST, STS,
GSS, environmental samples or phase 0, 1 or 2 HTGS sequences)
3,867,695 sequences; 17,091,186,980 total letters
Query=
Length=253

Sequences producing significant alignments:
Score      E
(Bits)    Value
dbj!AK175910.1: Arabidopsis thaliana mRNA for putative ferrit... 407 2e-11
1

ALIGNMENTS
>dbj!AK175910.1: Arabidopsis thaliana mRNA for putative ferritin subunit precursor,
complete cds, clone: RAFL22-57-B22
Length=1039

Score = 407 bits (1045), Expect = 2e-111
Identities = 253/253 (100%), Positives = 253/253 (100%), Gaps = 0/253 (0%)
Frame = +1

Query 1  MLAKAS PAXXXXXXXXXXXXXXXXXXFFPSRNSNLLFSPSGSRFSUQAAKGINTKSLTGXXXX 60
Sbjct 55  MLAKAS PALSLLSSGYTGGGNLFPPSRNSNLLFSPSGSRFSUQAAKGINTKSLTGUUFE 234

Query 61  XXXXXXXXXXXXLUPTTTPFUSLARHKFSDDESAINDQINUENUSYUYHALYVYFDRDNUG 120
Sbjct 235  PFEEVKKEMELUPTTTPFUSLARHKFSDDESAINDQINUENUSYUYHALYVYFDRDNUG 414

Query 121 LKGFAKFFNDSSLEERGHAEFMEYQNKRGGRUKLQSI LMPUSEFDHEEKGDALHAMELA 180
Sbjct 415  LKGFAKFFNDSSLEERGHAEFMEYQNKRGGRUKLQSI LMPUSEFDHEEKGDALHAMELA 594

Query 181  XXXXXXXXXXXXXXXXXXXXQUGUKNNDUQLUDFUESEPLGEQEA I KKI SEYUQLRRI GKGHG 240
Sbjct 595  LSLEKLTNEKLLKLSUGUKNNDUQLUDFUESEPLGEQEA I KKI SEYUQLRRI GKGHG 774

Query 241  UVHFDQMLLNDEV 253
Sbjct 775  UVHFDQMLLNDEV 813

Database: All GenBank+EMBL+DDBJ+PDB sequences (but no EST, STS, GSS, environmen
tal
samples or phase 0, 1 or 2 HTGS sequences)
Posted date: Apr 23, 2006 9:58 PM
Number of letters in database: 17,091,186,980
Number of sequences in database: 3,867,695
Lambda K H
0.316 0.132 0.376
Gapped
Lambda K H
0.267 0.0410 0.140
Matrix: BLOSUM62
Gap Penalties: Existence: 11, Extension: 1
Number of Sequences: 3867695

```

Figure 3.23 The results page obtained from the QBLAST system. It highlights the alignment part of the page that is extracted using the above mentioned regular expressions. Only the first hit is displayed as the `hitlist_size` parameter in the put query is set to one.

The above two regular expressions indicate that the page material till the word alignments and below the word database is removed and only the section between these two words is kept. This part of the page is then assigned to the variable 'temp4'.

If there is more than one alignment suggested for a particular input query sequence, then only the first alignment is taken into consideration. This is obtained by performing a split operation at the index of the word 'Score' and the alignments are stored in an array, 'arr'. From the array, only the first alignment is considered and further operations are performed on the string using regular expressions to obtain the accession number of the raw nucleotide sequence and the subject range. The regular expression that is used to extract the accession number is:

```
/^>.*\|(.*)\.\d+\|/gm
```

The subject range is obtained by first performing a join operation on the subject lines of the alignment and assigning it to the variable 'protein'.

```
join(' ', ($seq_tblastn =~ /^Sbjct.*\n/gm) )
```

After this step, another join operation is performed where the first number (start position) and the last number (end position) are extracted and joined together by '..'.

```
join('..', ($protein =~ /(\d+).*\D(\d+)/s))
```

This string is assigned to the variable 'subject_range'. A split operation is then performed on this variable at the index of '..' to separate the start and end positions and store the values in the array, 'start_end'.

```

>dbj|AK175910.1| Arabidopsis thaliana mRNA for putative ferritin subunit precursor,
complete cds, clone: RAFL22-57-B22
Length=1039

Score = 407 bits (1045), Expect = 2e-111
Identities = 253/253 (100%), Positives = 253/253 (100%), Gaps = 0/253 (0%)
Frame = +1

Query  1      MLHKASPAXXXXXXXXXXXXXXXXFPPSRNSNLLFSPSGSRFSUQAAKGTNTKSLIGXXXX 60
Sbjct  55      MLHKASPALSLLSSGYTGGGNLFPPSRNSNLLFSPSGSRFSUQAAKGTNTKSLIGUUFFE 234

Query  61      XXXXXXXXXXXXLUPTTFFUSLARHKFSDDESAINDQINUEYNUSYUYHALYAYFDRDNUG 120
Sbjct  235      PFEEUKKEMELUPTTFFUSLARHKFSDDESAINDQINUEYNUSYUYHALYAYFDRDNUG 414

Query  121     LKGFAPFFNDSSLEERGHAEFMFMEYQNKRGGRUQLQSI LMPUSEFDHEEKGDALHAMELA 180
Sbjct  415     LKGFAPFFNDSSLEERGHAEFMFMEYQNKRGGRUQLQSI LMPUSEFDHEEKGDALHAMELA 594

Query  181     XXXXXXXXXXXXXXXQSUGUKNNDUQLUDFUESEFLGEQVEAIKKISEYUACLRRIGKGGH 240
Sbjct  595     LSLEKLTNEKLLKLSUGUKNNDUQLUDFUESEFLGEQVEAIKKISEYUACLRRIGKGGH 774

Query  241     UWHFDQMLLNDEU 253
Sbjct  775     UWHFDQMLLNDEU 813

```

Figure 3.24 The alignment section of the Blast results page highlighting the accession number of the nucleotide sequence as well as the subject range or the target sequence range which are extracted from the page using the above mentioned regular expressions and this information is used to obtain the back-translated DNA sequence.

The array values are then stored in the individual variables 'temp_start' and 'temp_end'. These values are then compared to each other. If the value of the 'temp_start' is smaller than the value of 'temp_end' then the back-translated DNA sequence is on the plus strand and the 'temp_start' value is assigned to 'start_position' and the 'temp_end' value is assigned to the 'end_position' variables. However, if the value of 'temp_start' is greater than the value of 'temp_end' then the nucleotide sequence is located on the minus strand and the 'temp_start' value is assigned to 'end_position' and the 'temp_end' value is assigned to the 'start_position' variables. The values in the variables are reversed in order to read the nucleotide sequence from the NCBI nucleotide web page.

After all the information that is needed to obtain the back-translated DNA sequence is obtained, a new LWP::UserAgent object, 'ua_genbank', is created and the

UserAgent is given an identity as Agent 02. A new HTTP::Request object, 'req_genbank' is created and a connection is made to the GenBank nucleotide web page by passing the URL information to the get method of the class. The URL that is used to connect to GenBank is as follows:

```
http://www.ncbi.nlm.nih.gov/entrez/viewer.fcgi?txt=yes\&list_uids=$acc_no\&db=n.
```

The '\$acc_no' is substituted by the accession number value stored in it, which is obtained from the BLAST results page.

The HTTP::Request object is then passed to the request method of the userAgent object to send a request to the URL and get a response. The response which is obtained is stored in the HTTP::Response object, 'res_genbank'.

The content of the 'res_genbank' object is the HTML nucleotide webpage. This HTML page is parsed to extract the nucleotide sequence. This is the entire nucleotide sequence from which the back-translated DNA sequence needs to be extracted. This is done by using the regular expressions given below:

```
s /. * ORIGIN // m s
```

```

/gene="At3g11050"
/codon_start=1
/product="putative ferritin subunit precursor"
/protein_id="BAD43673.1"
/db_xref="GI:51969962"
/translation="MLHKASPALSLLSSGYTGGNLFPPSRNSNLLFSPSGSRFSUQ
AAKGTINKSLTGVUPEPFEUKKEMELUPTTFFUSLARHKFSDDESAINDQINUVEYN
USYUYHALYAYFDRDNUGLKGFAPFNDSLEERGHAEFMFQYQNKRCGRUKLQSLM
PUSFEDHEEKGDALHAMELALSLEKLTNEKLLKLSUGUKNNDUQLUDFUSEFLGEQ
VEAIKKISEYVAQLRRIKGGHGUWHFDQMLLNDEU"
ORIGIN
  1 gttttacaga gactatgctc ttctgtctt tctctctcta gatcttaagc ccaaatgtty
  61 cacaaggctt ctcccgcctt ctctctctt agctccggct acaccggcgg tggaaatctg
  121 tttctcctcg egagaaatc gtegaatctt ctgttttctc cgagtggatc caggtttctt
  181 gttcaggcgg cgaaggaaac gaacacgaag tegttaaccg gagttgtatt cgaacctttt
  241 yaggagggtg agaaagaaat ygagctctgt cccactaccc cttttgtttc tctctctctc
  301 cacaagttct cggacgattc tgaatctgoc atcaacgac agatcaactg ggagtacaac
  361 gtctcgtatg tctaccatgc cctgtatgcc tactttgaca gagacaatgt cggcttgaaa
  421 ggtttcggca agttttttta cgattcagat cttgaagaac gaggtcatgc tggagtgttt
  481 atggagtatc agaacaagcg tgggtggaga gtgaagctgc agtctatttt gatgcccgtc
  541 tctgagtttg atcacgagga gaaggagat gcattgcatg cgtatggagc tgcattgtct
  601 ttggagaaac ttacaaatga aaagcttcty aagttacaaa gtgttgggtg gaagaacaat
  661 gatgttcagc tggttgattt tgtagaatct gattttctag gcgagcaggt cgaagctatc
  721 aagaaaatct cagagtactg tgcacagcta agaagaatag gaaagggcca tggagtgtgg
  781 cattttgate aaatgcttct caatgatgag gtttaaggaa ggagagtcca gcttctgagt
  841 ttgatgacaa tcttctctgt gctataggg accgttctat ctctataaac acgtctctaa
  901 gtttggctcg agaaaagtgt tcttgcctgt tctttctttt ctttgttttt ggttaaccga
  961 atgcttgtga gtgtgtactt aataatgtaa ctctgagctt gataataaat gcaagtccca
  1021 ctgtttcaga aaaaaaaaaa
//
The protein sequence is:
>AT1At3g11050.1 (253 AA) ferritin subunit, putative / similar to ferritin sub
unit cowpea2 precursor GI:2970654 [Vigna unguiculata]; supported by full-length
cDNA: Ceres: 5710.
MLHKASPALSLLSSGYTGGNLFPPSRNSNLLFSPSGSRFSUQAAKGTINKSLTGVUPE
PFEUKKEMELUPTTFFUSLARHKFSDDESAINDQINUVEYNUSYUYHALYAYFDRDNUG
LKGFAPFNDSLEERGHAEFMFQYQNKRCGRUKLQSLM PUSFEDHEEKGDALHAMELA
LSLEKLTNEKLLKLSUGUKNNDUQLUDFUSEFLGEQVEAIKKISEYVAQLRRIKGGH
GUWHFDQMLLNDEU
And the corresponding DNA sequence from TBLASTN is:
atgttgcaacaaggcttctcccgcctctctctctcttgagctccggctacaccggcgggtggaaatctgtttctccgctcgag
aaatctgctgcaatctctgtttctccgagtgatccagggtttctctgttcaggcggcgaaggaaacgaacacgaagctgt
taaccggagttgtatcgaaccttttgaggaggtgaaagaaagaaatggagctcgttcccactacccttttgtttctctc
gctcggcacaagttctccgacgatctgaatctgccatcaacgacagatcaactggagtaaacgctctctgtatgtcta
ccatgcccctgtatgctactttgacagagacaatgtcggcttgaaagggttccgccaagtttttaacgattcagatcttg
aagaacgaggtcatgctgagatgtttatggagtatcagaacaagcgtgggtgggagagtgaagctgcagctatatttgatg
ccgctctctgagtttgatcacgaggagaaggagatgcatgcatgcatggagcttgcattgtctttggagaaacttac
aaatgaaaagcttctgaagttacaaggtgttgggtgtaagaacaatgatgtcagctgggttgattttgtagaatctgag
ttctagggcagcaggtcgaagctatcaagaaaatctcagagtagcttgcacagctagaagaataggaaagggtcatgga
gtgtggcattttgatcaaatgcttctcaatgatgaggtt

```

Figure 3.25 This is the final result obtained from the QBLast system. It highlights the nucleotide sequence result which is obtained from the QBLast system. The nucleotide sequence is extracted from the page using regular expressions and the final output result is printed with the protein sequence following the DNA sequence.

First, everything on the web page till the word 'origin' is removed and the resulting string is stored in the variable 'temp5'. The string value stored in 'temp5' is then assigned to an array 'seq_genbank'. Each line from the array is then taken and any whitespace characters or line numbers are removed. In this fashion, all the lines from the array are taken one at a time; they are refined and are merged to the previous line to form one long continuous string of the DNA sequence. This value is then assigned to the variable 'line_seq'.

Once the DNA sequence is refined, the PERL program then makes use of the substr() function in order to get the back-translated DNA sequence.

substr(\$line_seq, \$start_position-1, \$end_position-(\$start_position-1))

The DNA sequence, stored in the variable 'line_seq', is passed to the substr() function along with the start and end positions obtained from the BLAST results page. The substr() function takes in these three parameters and chops the 'line_seq' string from the index of the 'start_position' value to the 'end_position' value. The nucleotide sequence between these two values is the final back-translated DNA sequence and is assigned to the variable 'final_dna'.

The value stored in 'final_dna' along with its corresponding sequence in FASTA format is then written to the output file and also printed out onto the console as the final output. However if the nucleotide sequence is located on the minus strand then the reverse complement of the DNA strand is calculated and that sequence along with the FASTA protein sequence is given as output. A flag is mentioned that the back-translated DNA sequence is obtained from TBLASTN.

3.4 Benefits and Limitations

As it is seen from the above descriptions that before the automated system was implemented in order to gather the back-translated DNA sequences from the various web servers and databases it was a very tedious job to manually search and extract the back-translated DNA sequences. In this case, the program does all the dirty work of connecting to various web servers and detecting if the DNA sequences for the protein sequences are available or not. It first searches in the databases and if the coding sequences are not present then it connects to TBLASTN, executes the program and gets the results. A lot of time is saved at the user end to extract the same information manually. It saves the efforts of browsing web pages and look for coding sequences and if the DNA sequences are not found in the databases then it saves the efforts of connecting to TBLASTN and executing the protein sequence. Therefore, the results are obtained more quickly.

There are some limitations to the program too. The program does not code for all databases. That is the program has been coded only to search in the UniProtKB/SwissProt and the UniProtKB/TrEMBL databases and the SGD database. There are a lot of other data resources from which the protein sequences are obtained and clustered in SYSTEMS, like Ensembl, but these databases do not provide easy access to the coding sequence. Thus, more modules can be added to the program to incorporate more data resources to fetch back-translated DNA sequences from.

3.5 Running Time Analysis

Table 3.1 Running Time Analysis of the Manual Extraction and the Program

	Sequences	Execution Time
Manual Extraction	10 sequences (4 from tblastn, 4 from swissprot & 2 from SGD)	~20 mins @ 11 Mbps
Extraction using the Perl Script	10 sequences (4 from tblastn, 4 from swissprot & 2 from SGD)	~8 mins @ 11 Mbps with a one min gap between two TBLASTN requests

The table above provides a rough estimate of the amount of time it would take to extract the sequences manually by visiting the web sites and putting them together in a file, and the amount of time it would take to gather the same data by running the Perl script. The sequences that have been used for this analysis are chosen so that they are found in the web databases and the subroutines do not fail at any stage. In case, the subroutines fail and the tblastn subroutine is invoked then the program will take a longer time to execute.

For a larger number of sequences, it takes more time for execution. It also depends on the sequences that are being searched for. If more of the protein sequences being searched for are from Swissprot or SGD and there are a few sequences that are executed by TBLASTN, then the program takes less time to compute the results as, only the sequences obtained from TBLASTN take time to execute. On the other hand, if there are more number of sequences to be executed by TBLASTN and few sequences that come from the other databases, then the program will take a longer time to compute the results. However, in any case the program is a faster alternative to manual extraction.

CHAPTER 4

RESULTS AND SCREEN SHOTS

In this section the results that are obtained from the program are going to be discussed. From the initial insert statement how the accession number and the database shortcut is extracted is discussed in the program description section. Also how the protein sequence in FASTA format and the raw protein sequence are obtained is also discussed in the program description section.

If the database shortcut matches to 'TRE', 'SPR' or 'SPU' then the 'swissprot' subroutine is executed which connects to the SWISSPROT/TrEMBL website and gets the corresponding back-translated DNA sequence. As an example, the following protein sequence entries are taken to demonstrate the output results. The protein sequences are stored in the file called seq.sql

```
INSERT INTO `protein` (`pid`, `accno`, `length`, `sequence`, `organism`, `taxon_id`)
VALUES (510708,'Q64660',85,'>TRE|Q64660|Q64660 (85 AA) Ferritin H subunit
(Fragment) [Cavia (guinea pigs)]
ASYVYLSMSYFDRDDVALKNFAKYNLHQSHEEREHAEKLMKLNQRGGRIFL
QDIKKPDRDDWENGLNAMECALHLEKSVNQSL','Cavia',10140);
INSERT INTO `protein` (`pid`, `accno`, `length`, `sequence`, `organism`, `taxon_id`)
VALUES (510706,'P29389',185,'>SPR|P29389|FRIH_CRIGR (185 AA) Ferritin heavy
chain (Ferritin H subunit) [Cricetulus griseus (Chinese hamster)]
TTTALTTASPSQVRQNYHQDSEAAINRQINLELYASYVYLSMSCYFDRDDVALK
NFAKYFLHQSHEEREHAEKLMKLNQRGGRIFLQDIKKPDRDDWESGLNAMEC
ALHLEKSVNQSLELHKLATDKNDPHLCDFIETHYLNEQVKSIELGDHVTNLRK
MGAPEAGMAEYLFDKHTLGHSES','Cricetulus griseus',10029);
```

The command `/usr/local/bin/Perl tblastn.pl` is used to execute the program.

Screenshots:

```

Telnet afs13.njit.edu
Please input the Sequence Filename in .sql Format:
seq.sql

The Database Is: TRE
The Accession Number Is: Q64660

The Merged URL is: http://ca.expasy.org/uniprot/Q64660

The Accession Numbers Have Matched

```

Figure 4.1 (a) The start of the program. User inputs the file name to be executed and the database and the accession number are displayed. In this case the database shortcut matches the condition to the 'swissprot' subroutine and the merged URL is mentioned. Also the program checks if the primary accession number and the accession number from the insert statements have matched.

```

Telnet afs13.njit.edu
The Merged URL is: http://ca.expasy.org/uniprot/Q64660

The Accession Numbers Have Matched

The protein sequence is:
>TRE:Q64660:Q64660 (85 AA) Ferritin H subunit (Fragment) ICavia (guinea pigs)
ASVUYLSMSYYFDRDDUALKNFAKYNLHQSHEREHAERLMKQLQNRGGRI FLQDIKKPD
RDDWENGLNAMECALHLEKSUNQSL

And the corresponding DNA sequence from Swissprot/TrEMBL is:
gcctcctacgtctacctgtccatgtcttactactttgaccgcgatgatgtggcttgaagaacttcgcccatacaact
tcaccaatctcatgaggagaggaacatgcagagaagctgatgaagctgcagaaccagcgcggtggcgc atctttcttc
aggatatcaagaaccagaccgtgatgactgggagaatgggctgaatgccatggagtgcccttcacttggagaagagt
gtgaatcagtcacta

The Database Is: SPR
The Accession Number Is: P29389

```

Figure 4.1 (b) The figure displays the resulting back-translated DNA sequence for the first protein sequence in the file. It also shows the immediate execution of the next protein sequence.

```

Telnet afs13.njit.edu
The Database Is: SPR
The Accession Number Is: P29389
The Merged URL is: http://ca.expasy.org/uniprot/P29389
The Accession Numbers Have Matched
The protein sequence is:
>SPR:P29389:FRH_CRIGR (185 AA) Ferritin heavy chain (Ferritin H subunit) [Crice
tulus griseus (Chinese hamster)]
TTTALTASPSQURQNYHQDSEAA INRQINLELYASYUYLSMSCYFDRDDUALKNEAKYF
LHQSHEEREHA EKLMKLNQNRGGRI FLQDI KKPDRDDVESGLNAMECALHLEKSUNQSL
ELHKLATDKNDPHLCDFIETHYLNEQVKS I KELGDHUTNLRKMGAP EAGMAEYLFDKHTL
GHSES
And the corresponding DNA sequence from Swissprot/TrEMBL is:
atgaccaccaccgacctgaccaccgctctcctcgcaggtgcgccagaactaccaccaggactcggaggctgccatcaa
ccgccagatcaacctggaactgtacgcctctacgtctatctgtctatgtcttgcactttgaccgggatgatgtggctc
tgaagaactttgccaaatactttctccaccaatctcatgaggagagggaaacacgctgagaactgatgaagctgcagaac

```

Figure 4.1 (c) This figure displays the database and accession number of the second protein sequence in the file along with the merged URL. It also displays the results for the second protein sequence.

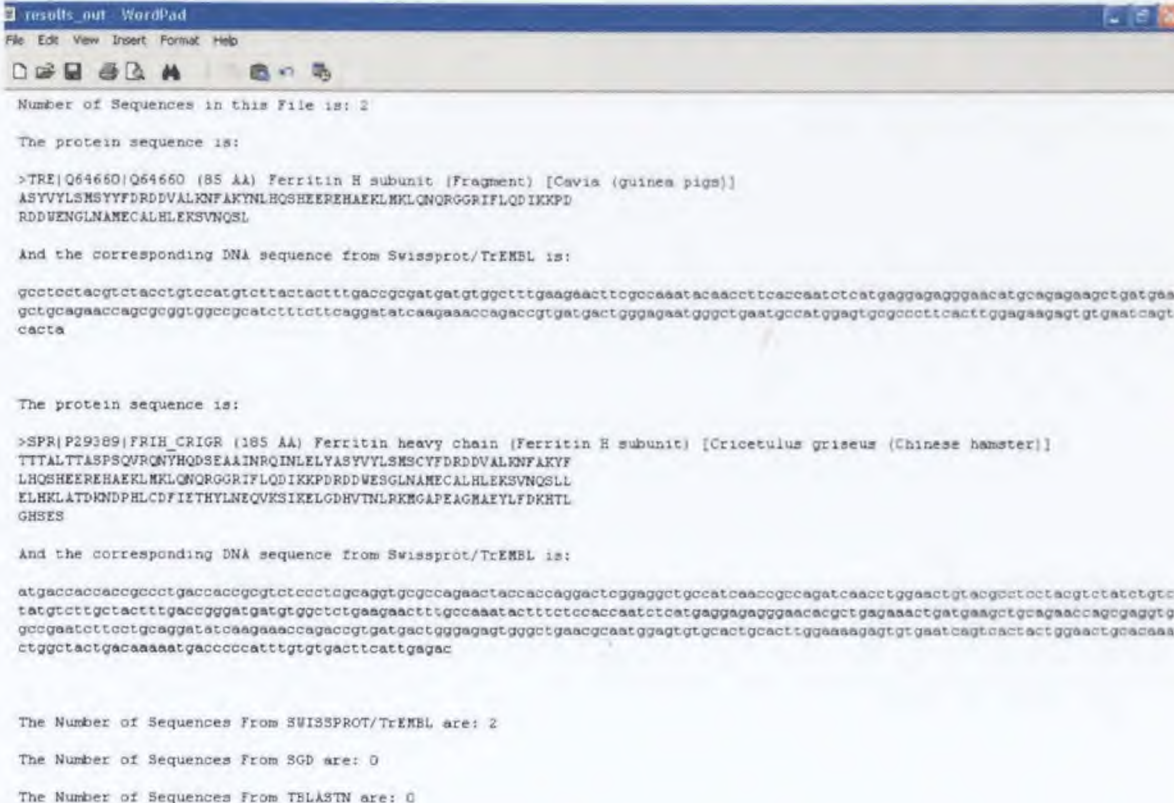
```

Telnet afs13.njit.edu
The protein sequence is:
>SPR:P29389:FRH_CRIGR (185 AA) Ferritin heavy chain (Ferritin H subunit) [Crice
tulus griseus (Chinese hamster)]
TTTALTASPSQURQNYHQDSEAA INRQINLELYASYUYLSMSCYFDRDDUALKNEAKYF
LHQSHEEREHA EKLMKLNQNRGGRI FLQDI KKPDRDDVESGLNAMECALHLEKSUNQSL
ELHKLATDKNDPHLCDFIETHYLNEQVKS I KELGDHUTNLRKMGAP EAGMAEYLFDKHTL
GHSES
And the corresponding DNA sequence from Swissprot/TrEMBL is:
atgaccaccaccgacctgaccaccgctctcctcgcaggtgcgccagaactaccaccaggactcggaggctgccatcaa
ccgccagatcaacctggaactgtacgcctctacgtctatctgtctatgtcttgcactttgaccgggatgatgtggctc
tgaagaactttgccaaatactttctccaccaatctcatgaggagagggaaacacgctgagaactgatgaagctgcagaac
cagcgaggtggccgaatcttctcgcaggatataaagaaccagaccgtgatgactgggagagtggctgaacgcaatgga
gtgtgcactgcacttggaaaagtgatgaatcagtcactactggaactgcacaaactggctactgacaaaaatgaccccc
atctgtgtgacttcattgagac
The File Has Completed Execution And This Is The End Of The Program
feinberg-52 mm16>:

```

Figure 4.1 (d) The figure displays the second protein sequence along with its corresponding back-translated DNA sequence indicating that the DNA sequence has been obtained from the Swissprot/TrEMBL database. It also indicates that the file has completed execution and the program has terminated.

The results are also written to the 'results_out.txt' file. The screenshot for which is given below



```

results_out WordPad
File Edit View Insert Format Help
Number of Sequences in this File is: 2

The protein sequence is:
>TRE|Q64660|Q64660 (85 AA) Ferritin H subunit (Fragment) [Cavia (guinea pig)]
ASYVYLSMSYFFDRDDVALKNFAKYNLHQSHHEEHAERLKLQNRGGRIFLQD IKKPD
RDDUENGLNAMECALHLEKSVNQSLL

And the corresponding DNA sequence from Swissprot/TrEMBL is:
gctctactcgtctaccctgctccatgctcttactactcttggaccggatgatgtggctttgaagaacttcgccaatcacaccctcccaatctcatgaggagaggggaacatgcagagaagctgatgaa
gctgcagaaccagcggctggcggcctcttctcaggatatacaagaaccagaccgtgatgactgggagaatggctgaatgccatggagtggccctcactcggagaagatgtgaatcagt
cacta

The protein sequence is:
>SPR|P29389|FRIH_CRIGR (185 AA) Ferritin heavy chain (Ferritin H subunit) [Cricetulus griseus (Chinese hamster)]
TTTALTASPSQVRQNYHQDSEAAINRQINLELYASYVYLSMSYFFDRDDVALKNFAKYF
LHQSHHEEHAERLKLQNRGGRIFLQD IKKPD RDDUESGLNAMECALHLEKSVNQSLL
ELHKLATDKNDP HLCDFIETHYLINEQVKSIRKELGDHVITLKKMGAPFAGMAEYLFDKHTL
GHSES

And the corresponding DNA sequence from Swissprot/TrEMBL is:
atgaccaccaccggcctgaccaccggctctccctcgcaggctggccagaaactaccaccaggactcggaggctgccatccaccgcccagatccaccctggaactgtacggctctcactctatctgtc
tatgctctgctactttgaccgggatgatgtggctctgaagaacttcgccaatacttctccaccaatctcatgaggagaggggaacacgctgagaaactgatgaagctgcagaaccagcggctg
ggcgaatctctcctgcaggatatacaagaaccagaccgtgatgactgggagaatggctgaacgcaatggagtgtgcactgcactcggaaagagtgtgaatcagtcactactggaaactgcacaa
ctgctactgcacaaatgacccccatttggctgacttcattgagac

The Number of Sequences From SWISSPROT/TrEMBL are: 2
The Number of Sequences From SGD are: 0
The Number of Sequences From TBLASTN are: 0

```

Figure 4.2 This figure represents the output file 'results_out.txt' to which all the results obtained from the program are written to. The output file specifies how many protein sequences are present in the file along with the number of back-translated DNA sequences extracted from each of the databases. In this case it indicates that there are two protein sequences in the file and two back-translated DNA sequences have been obtained from the SWISSPROT/TrEMBL database.

Now, if the database shortcut matches to 'SC' then the 'SGD' subroutine is executed which connects to the SGD website and gets the corresponding back-translated DNA sequence. As an example, the following protein sequence entry is taken to demonstrate the output results. The protein sequence is stored in the file called seq.sql

```

INSERT INTO `protein` (`pid`, `accno`, `length`, `sequence`, `organism`, `taxon_id`)
VALUES (516538,'S0003060(NR_SC:SW-N145_YEAST)',1317,'>SC|NR_SC:SW-
N145_YEAST (1317 AA) SW:N145_YEAST P49687 saccharomyces cerevisiae
(baker's yeast). nucleoporin nup145 (nuclear pore protein nup145). 2/2003; PIR:A54831
nuclear pore complex protein NUP145 - yeast (Saccharo
MFNKSVNSGFTFGNQNTSTPTSTPAQPSSSLQFPQKSTGLFGNVNVNANTSTPSPS
GGLFNANSNANSISQQPANNSLFGNKPAQPSGGLFGATNNTTSKSAGSLFGNNN
ATANSTGSTGLFSGSNNIASSTQNGGLFGNSNNNNITSTTQNGGLFGKPTTTPAG
AGGLFGNSSSTNSTTGLFGSNNTQSSTGIFGQKPGASTTGGLFGNNGASFPRSGET
TGTMSTNPYGINISNVPMAVADMPRSITSSLSDVNGKSDAEPKPIENRRTYSFSSS
VSGNAPLPLASQSSLVSRLLSTRLLKATQKSTSPNEIFSPSYSKPWLNGAGSAPLVDD
FFSSKMTSLAPNENSIFPQNGFNFLSSQRADLTELRKLIKIDSNRSAAKKLLKLLSGT
PAITKKHMQDEQDSSENEPIANADSVTNIDRKENRDNNLDNTYLNGKEQSNNLN
KQDGENTLQHEKSSSFGYWCSPEQLERLSLKQLAAVSNFVIGRRGYGCITFQH
DVDLTAFTKSFREELFGKIVIFRSSKTVEVYPDEATKPMIGHGLNVPAITLENVYP
VDKKTCKKPMKDTTKFAEFQVFDRKLRSMREMNYISYNPFGGTWTFKVNHFISIW
GLVNEEDAEIDEDDLKQEDGGEQPLRKVRTLAQSKPSDKEVILKTDGTFGTLSG
KDDSIVEEKA YEPDLSADDFEGIEASPKLDVSKDWVEQLILAGSSLRVVFATSKEF
DGPCQNEIDLLFSECNDEIDNAKLIMKERRFTASYTFKSTGSMMLTKDIVGKSG
VSIKRLPTLQQRKFLFDDVYLDKEIEKVTEIARKSNPYPQISESSLFKDALDYME
KTSSDYNLWKLSSILFDPVSYYPYKTDNDQVKMALLKKERHCRLTSWIVSQIGPEI
EEKIRNSSNEIEQIFLYLLLNDVVRASKLAIESKNGHLSVLISYLGSDNDRIRDLAE
LQLQKWSTGGCSIDKNISKIYKLLSGSPFEGFLSLKELESEFSWLCLLNLTLCYGQI
DEYSLESLVQSHLDKFSPLPYDDPIGVIFQLYAANENTEKLYKEVRQRTNALDVQF
CWYLIQTLRFNGTRVFSKETSDEATFAFAAQLEFAQLHGHSLFVSCFLNDDKAAE
DTIKRLVMREITLLRASTNDHILNRLKIPSQLIFNAQALKDRYEGNYLSEVQNLLL
GSSYDLAEMAIVTSLGPRLLLSNNPVQNNELKTLREILNEFPDSERDKWSVSINVF
EVYLKLVLDNVETQETIDSLISGMKIFYDQYKHCREVAACCNVMSQEIVSKILEK
NNPSIGDSKAKLLELPLGQPEKAYLRGEFAQDLMKCTYKI','Saccharomyces
cerevisiae',4932);

```

The command `/usr/local/bin/Perl tblastn.pl` is used to execute the program.

Screenshots:

```

ca Telnet afs13.njit.edu
feinberg-55 mm16> /usr/local/bin/perl tblastn.pl
Please input the Sequence Filename in .sql Format:
seq.sql

The Database Is: SC

The Accession Number Is: S0003060

The Merged URL Is: http://db.yeastgenome.org/cgi-bin/locus.pl?sgdid=S0003060

The Systematic Name Is: YGL092W

The Merged URL Is: http://db.yeastgenome.org/cgi-bin/getSeq?seq=YGL092W&flankl=0
&flankr=0&map=n3map

The protein sequence is:
>SC:NR_SC:SW-N145_YEAST (1317 AA) SW:N145_YEAST P49687 saccharomyces cerevisiae
(baker's yeast). nucleoporin nup145 (nuclear pore protein nup145). 2/2003; PIR:
054831 nuclear pore complex protein NUP145 - yeast (Saccharo

```

Figure 4.3 (a) The start of the program. User inputs the file name to be executed and the database and the accession number are displayed. In this case the database shortcut matches the condition to the 'SGD' subroutine and the first merged URL is mentioned. The systematic name and the URL formed after merging the systematic name is printed.

```

ca Telnet afs13.njit.edu
USNFUIGRRYGCIFQHDUDLTAFTKSFREELFGKIUIFRSSKTUVEYVDEATKPMIGH
GLNUPAIIITLNUYVUDKKTKKPKMDITTKFAEFQFDRKLRSMREMYISYNPFGGTWTF
KUNHFSIUGLUNEEDEAIEDDLSKQEDGGEQPLRKRRTLAQSKPSDKEVILKTDGTFGT
LSGKDDSIUEEKAYEPDLSADDFEGLI EASPKLDUSKDWUEQLILAGSSLRSUFATSKFED
GPCQNEIDLDFSECNDEIDNAKLIMKERRFTASYTFAKFSTGSMMLTKDIUGKSGUSIKR
LPTLQKRFDFDDUYLDKEIEKUTIEARKSNPYPQISESSLFKDALDYMEKTSDDYNLW
KLSSILFDPUYPPYKTDNDQURKMLLQKERHCRILTWSIUQIGPEIEEKIRNSSNEIEQI
FLYLLNDUURASKLAI ESKMGHLSULISYLGSDNPRI RDLAELQLQKQVSTGGCSIDKNI
SKIYKLLSGSPFEGFLFSLKELES EFSWLCLLNLTLCYGQIDEYSLESLUQSHLDKPSLPY
DDPIGUILFQLYAAMENTEKLYKEURQRTNALDUQFCWYLIQTLRFNGTRUFSEKETSDEAT
PAPAAQLEFAQLHGHSLFVSCFLNDDKAAEDIKRLUMREITLLRASTNDHILWRLKIPS
QLIFNAQALKDRYEGNYLSVUQNLLGSSYDLAEMAIUTSLGPRLLSNPNPUQNNELKTL
REILNEFPDSERDKWSUSINUFEUYLKLULDNUETQETIDSLISGMKIFYDQYKHCREVA
ACCNUMSQEISUKILEKNNPSIGDSKAKLLELPLGQPERAYLRGEFAQDLMKCTYKI

And the corresponding DNA sequence from YeastGenome is:

atggtttaataaaagtgtaaatagtggtttcaccttttggaaacaaaacacatccacaccgacatccactccggcagagcc
ttccagttcccttcaattccctcaaaaatctactggyttatttgggaacggttaatggttaatgcaataacttcaaacctta
gcccacctgggtggattattcaatgccaattcaaacgctaattctataagtcacacaccggccataaattcattatttggga
aataaacagcacaaccatcagtggtcttatttggcctacaacaataactacttctaaaagtccgggaagcttattcggg
aaataaatagccacagcaaatctacgggctctacaggttattcagtggtagcaataatagcctctagtagcccaaaa
atggttggtttggttaatagcaacaataacaataaacttctaccagcgaataatgggtggtctggttcggttaagcctact
acaacccctgctggtgctggcggcctggttggaaattctcctacagcaaacctcaacgacaggtattatttggatcaataaa
tacacaaaattctacaggtatttttgggtcaaaaagccgggagcttcaacaactgggggattggttggaaataatggtgctt

```

Figure 4.3 (b) This figure is the continuation of the program for the *Saccharomyces Cerevisiae* protein sequence. It displays the protein sequence in FASTA format and the corresponding back-translated DNA sequence.

```

Telnet afs13.njit.edu
agattatattggaacacccctagtgattacaatctatggaaccttcatctattttatccgaccccgtttccatccat
acaagacggacaatgatcaagtaaaaatggcattatataaaaaaagaaagggcattgtcgatttaacttcgtggattgttlayc
cagatagggtccggaattgaagagaaaaataggaattcttctaacgaaatagaaCaatatattctatatttgcctttgaa
tgatgtcgttaagggcatccaagttagcaattgagtcataagaatggtcatttgagcgttttlyatatacctatttgggttcaa
acgacccaaggatacgtgattagcgggaattacaattgcaaaagtggtaactgggggtttagtatcgacaagaatatac
tcgaagatttacaatataaagcgggttcaccttttgaaggtttatcttcttaaggaacttgaagtgaaatlcagttg
gttgtgctttttgaaattgacactgtgctacggccagatagatgagatattccttagaattctcttgycaatcgcalttag
acaagttttctttaccctatgatgatcctattggagttattttccaaactatatactgctaatgaaaatacyggagaagctc
tataaggaggttagacaagaaccaagccttagatgttcagttttgctggtaacttgattcaaacattgagatttaattgg
aacacgcgttttttcgaaggaaactagcagatgaggtacattcgcatttggcgcctcaacttgaatttgcacaattacatg
gccactctcttttttcttcttcttcttctttaaataatgatgacaagggcagcagaagataccataaaaaaggcttgttatgctgaa
ataacattgttaagagcctctaccaatgatcacattttaaatagattaaagatccttcgcaattgatattcaatgctca
agctttgaaagatagatataagggcaattacctttctgaaatcaaaaatctactctgggttcgtcttatgatttggcag
aaatggctattgttacgtcattaggaccaaggctgttgttatcaacaatcctgtacagaataatgaactgaaaactttg
agggaaattcttaacgaatttccagactctgagagggacaatggagtgtaagtataaacgtatttgaagtatactaaa
gcttgtgttggataaattgtgaaacgcaggagactatcgattcattaaatagtggtatgaagatattttatgatcagata
agcacgtcgtgaaagtggccgcattgttgaattgttatgtcccaagagattgtatcaaaaatattggagaaaaaacatcct
tcaataggcgatttcgaaggccaatttgccttgagctcccttagggcagccagaaaaggcataacttgaggggtgagtttgc
ccagatttaatgaagtgatataagatataa

```

The File Has Completed Execution And This Is The End Of The Program

```

feinberg-56 mm16>

```

Figure 4.3 (c) This figure shows the termination of the program for the *Saccharomyces Cerevisiae* protein sequence.

The results are also written to the 'results_out.txt' file similar to the ones shown above for the swissprot example. The screenshot of the text file in this case is not shown as it is similar Figure 4.2 with the only exception that the number of sequences in the file is one and there are zero protein sequences from SWISSPROT/TrEMBL and TBLASTN and there is one sequence obtained from the SGD database.

If the database shortcut do not match to any of the shortcuts mentioned above then the 'tblastn' subroutine is executed which connects to the NCBI-BLAST program TBLASTN to get the corresponding back-translated DNA sequence. As an example, the following protein sequence entries are taken to demonstrate the output results. The protein sequences are stored in the file called seq.sql

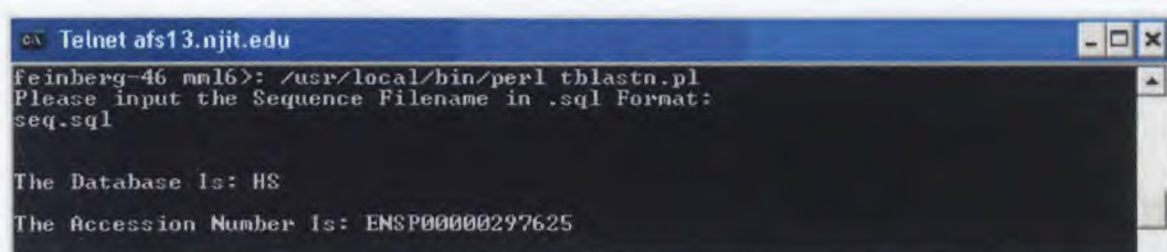
```

INSERT INTO `protein` (`pid`, `accno`, `length`, `sequence`, `organism`, `taxon_id`)
VALUES (510590,'ENSP00000297625',217,'>HS|ENSP00000297625 (217 AA)
Gene:ENSG00000164976 Clone:AL356494 Contig:AL356494.13.1.83750 Chr:9
Basepair:34540243 Status:known
FSLAEVRVGYQSQNISCFRLLVDRDSVWGYDLGLRSLIPAVLTVSMLGYPFILPD
MVGGNVAVPQRTAGGDVPERELYIRWLEVAAFMPAMQFSIPPWRYDAEVVAIAQ
KFAALRASLVAPLLELAGEVTDGDPVIRPLWWIAPGDETAHRIDSQFLIGDTLL
VAPVLEPGKQERDVYLPAGKWRSYKGFELDFDKTPVLLTDYPVDLDEIAYFTWAS',
Homo sapiens',9606);
INSERT INTO `protein` (`pid`, `accno`, `length`, `sequence`, `organism`, `taxon_id`)
VALUES (510688,'ENSMUSP00000025563',182,'>MM|ENSMUSP00000025563 (182
AA) Gene:ENSMUSG00000024661 Clone:19.9000001-10000000 Contig:19.9000001-
10000000 Chr:19 Basepair:9320883 Status:known
MTTASPSQVRQNYHQDAEAANRQINLELYASYVYLSMSCYFDRDDVALKNFA
KYFLHQSHREHAELMKLQNRGGRIFLQDIKKPDRDDWESGLNAMECALH
LEKSVNQSLLELHKLATDKNDPHLCDFIETYLSEQVKSIELGDHVTNLRKMG
APEAGMAEYLFDKHTLGHGDES','Mus musculus',10090);
INSERT INTO `protein` (`pid`, `accno`, `length`, `sequence`, `organism`, `taxon_id`)
VALUES (510719,'SINFRUP00000151380',177,'>FR|SINFRUP00000151380 (177 AA)
Gene:SINFRUG00000142595 Clone:scaffold_1408 Contig:scaffold_1408
Chr:Chr_scaffold_1408 Basepair:11700 Status:known
MSSQVRQNFHQDCEAANRQINLELYASYVYLSMSYFDRDDQALHNFQAKFFRH
QSHEEREHAELMKMQNRGGRIFLQDVRKPERDEWGSMEALECALQLEKSV
NQSLDLHKMCSHDNDPHLCDFIETHFLDEQVKSIELADWVTNLRMRMGAPQN
GMAEYLFDKHTLGKESS','Takifugu rubripes',31033);

```

The command `/usr/local/bin/Perl tblastn.pl` is used to execute the program.

Screenshots:



```

Telnet afs13.njit.edu
feinberg-46 nml6>: /usr/local/bin/perl tblastn.pl
Please input the Sequence Filename in .sql Format:
seq.sql

The Database Is: HS
The Accession Number Is: ENSP00000297625

```

Figure 4.4 (a) The start of the program. User inputs the file name to be executed and the database and the accession number are displayed. In this case the database shortcut does not match either to the condition of the 'swissprot' or of the 'SGD' subroutine and the tblastn subroutine is invoked.

```

Telnet afs13.njit.edu
The Database Is: HS
The Accession Number Is: ENSP00000297625
Searching...
Searching...
Searching...
Searching...
Searching...
Searching...
Searching...
Searching...
Search complete, retrieving results...
The Accession Number Is: BC110493
The Start Position = 1390
The End Position = 2040
The protein sequence is:
>HS:ENSP00000297625 (217 AA) Gene:ENSG00000164976 Clone:AL356494 Contig:AL356494
.13.1.83750 Chr:9 Basepair:34540243 Status:known

```

Figure 4.4 (b) Displays the execution of the tblastn program stating that the program is searching the nucleotide database for entries corresponding to the protein sequence and once the search is complete it retrieves the results. Along with this the accession number of the nucleotide sequence is displayed and even the information regarding the start and end positions of the subject or target sequence is printed.

```

Telnet afs13.njit.edu
The protein sequence is:
>HS:ENSP00000297625 (217 AA) Gene:ENSG00000164976 Clone:AL356494 Contig:AL356494
.13.1.83750 Chr:9 Basepair:34540243 Status:known
FSLAEVRUGYQSQNI SCFFRLUDRDSUUGYDLGLRSLI PAULTUSMLGYPFILPDMUGGN
AUPQRTAGGDUPERELY IRVLEUAAEMPAMQFSIPPUVRYDAEUVAIAQKFAALRASLUAP
LLELAGEVTDTGDP IURPLWIIAPGDETAHRI DSQFLIGDTLLUAPULEPCKQERDUYL
PAGKWRSYKGELFDKTPULLTDYPUDLDEIAYFTWAS
And the corresponding DNA sequence from TBLASTN is:
ttctcgctggcggagggtgcgcgtaggctaccagtcacagaacatctctctgcttcttccgcctgggtggatcgcgactctgt
gtggggctacgacctggggttgcgctcactc atccccgcgggtgctc accgtcagc atgctgggct acccatc atctac
ccgatatggtgggcggcaacgcccgtgcccc agcggacagccggcgcgatgtgccgagcgcgagctctacatcgctgg
ctggaagtggcgcgctttatgcccgcacatgcagttctctatcccgcctggcgcctacgacgcggaagtgggtggccatcgc
gcagaagttcgcgcctgcggcctcgcttgtggcaccgctgttgc ttagctggcgggcgaggtcaccgacacggggtg
accctatcgctgcgccccctttggtggattgcgccggcgacgagacagctcaccgtatcgactcgcagttccttattggg
gacacgctgcttgtggccccggtgctggagccaggcaagcaggagcgcgacgctcatttggcccggcaggtaagtgccag
ctacaaggggtgagcttttcgac aagacgcccgtgctgctc accgattaccgggtcgacctggatgagatcgctacttta
cctgggcgtcc

```

Figure 4.4 (c) This figure displays the protein sequence and the resulting DNA sequence for the first sequence in the file.

```

Telnet afs13.njit.edu
Searching...
Searching...
Searching...
Searching...
Search complete, retrieving results...

The Accession Number Is: BC026454
The Start Position = 843
The End Position = 1388

The protein sequence is:
>MM:ENSMUSP00000025563 (182 AA) Gene:ENSMUSG00000024661 Clone:19.9000001-1000000
0 Contig:19.9000001-1000000 Chr:19 Basepair:9320883 Status:known
MTTASPSQRQNYHQDAEAAINRQINLELYASYUYLSMSCYFDRDDUALKNFAKYFLHQS
HEEREHAEKLMKLNQQRGGRI FLQDI KKPDRDDVESGLNAMECALHLEKSUNQS LLELHK
LATDKNDPHLCDFI ETYYLSEVQKSI KELGDHUTNLRKMGAPeAGMAEYLFDKHTLGHG
ES

And the corresponding DNA sequence from TBLASTN is:
atgaccaccgcgtctcctctgcgaagtgcgccagaactaccaccaggacgcggagcctgccatcaaccgccagatcaact

```

Figure 4.4 (d) This figure shows the immediate execution of the second sequence in the file and shows all the corresponding information that is needed to extract the DNA sequence.

```

Telnet afs13.njit.edu
MTTASPSQRQNYHQDAEAAINRQINLELYASYUYLSMSCYFDRDDUALKNFAKYFLHQS
HEEREHAEKLMKLNQQRGGRI FLQDI KKPDRDDVESGLNAMECALHLEKSUNQS LLELHK
LATDKNDPHLCDFI ETYYLSEVQKSI KELGDHUTNLRKMGAPeAGMAEYLFDKHTLGHG
ES

And the corresponding DNA sequence from TBLASTN is:

atgaccaccgcgtctcctctgcgaagtgcgccagaactaccaccaggacgcggagcctgccatcaaccgccagatcaact
ggagttgtatgcctcctacgtctatctgtctatgtcttgttattttgaccgagatgatgtggctctgaagaactttgcca
aatactttctccaccaatctcatgaggagagggagcctgcgagaaactgatgaagctgcagaaccagcaggtggccga
atcttctctcaggatataaagaaccagaccgtgatgactgggagagcggcctgaatgcaatgagtgctgactgcactt
ggaaaagagtgatgaatcagtcactactggaactgcacaaactggctactgacaagaatgatccccacttatgtgactta
ttgagacgtattatctgagtgaaacaggtgaaatccatataagaactgggtgaccacgtgaccaacttacgcaagatgggt
gcccctgaagctggcctggcagaatctctcttggacaagcacaccctgggacacgggtgatgagagc

The Database Is: FR
The Accession Number Is: SINFRUP00000151380

Searching...
Searching...
Searching...

```

Figure 4.4 (e) This figure partially displays the output of the second sequence in the file and the partial execution of the third sequence.

```

Telnet afs13.njit.edu
The End Position = 791

The protein sequence is:
>FR1SINFRUP00000151380 (177 AA) Gene:SINFRUC00000142595 Clone:scaffold_1408 Cont
ig:scaffold_1408 Chr:Chr_scaffold_1408 Basepair:11700 Status:known
MSSQURQNFHQDCEAAINRQINLELYASYVYLSMSVYFDRDDQALHNFQKFFRHQSHEER
EHA EKLMKMQNQRGCRIFLDURKPERDEMGSGMEALECALQLEKSUNQSLDLAKMCS
D
HNDPHLCDFIETHFLDEQVKS IKELADVVTLRRMGAPQNGMAEYLFDIHTLGKSS

And the corresponding DNA sequence from TBLASTN is:
atgagttcccaggtgagacagaacttccaccaggactcgcaggctgcaatcaacaggcagatcaacctggaactatacgc
ctcctatgtctacctgtccatggcctattactttgaccgggacgaccaggcattgcacaacttgccaagttttccgce
atcagtcacacgaggagcgcgagcatgcggaaaagtgatgaaaatgcagaaccagaggggaggaaggatcttctcgc
aatgatgtcaggaaacagaacgggacgagtgggcagtggaaccgaagccctgaaatgcgccctgcagcttgagaagagcgt
gaaccagtcctcgc tggacatgcacaagatgtgctctgatcacaacgaccacacatgtgtgatttcattgagacacatt
tccctggacgagcaagtgaagtcacaaagaactggcagactgggtgaccaacctgcggcgc atgggagc tctcagaac
ggcatggccgaatacctgtttgacaacaacatacccttggcaagagagcagc

The File Has Completed Execution And This Is The End Of The Program

```

Figure 4.4 (f) The last figure in this group displays the results of the third sequence in the file and indicated that the file has completed execution and the program has terminated. All of these screenshots are used to demonstrate the flow of execution of tblastn.

The results are also written to the 'results_out.txt' file similar to the ones shown above for the swissprot and the SGD example. The screenshot of the text file in this case is not shown as it is similar Figure 4.2 with the only exception that the file contains three sequences and there are zero protein sequences from SWISSPROT/TrEMBL and SGD and the number of sequences obtained from the execution of the TBLASTN program is three.

APPENDIX

THE CODE

```
#####  
#                                                                                               #  
#           A Data Gathering Toolkit for Biological Information Integration                       #  
#                               Munira Lokhandwala                                           #  
#           This program helps to gather back-translated DNA sequences for given           #  
#           protein sequences using PERL as the programming language                         #  
#                                                                                               #  
#####  
  
#!/usr/local/bin/perl  
  
#Include all the modules  
use URI::Escape;  
use LWP::UserAgent;  
use HTTP::Request::Common qw(POST);  
use IO::Handle;  
  
#To flush the IO handle  
autoflush STDOUT 1;  
  
#Initializing variables  
my $dna_seq;  
my $swiss_ctr = 0;  
my $yeast_ctr = 0;  
my $tblastn_ctr = 0;  
  
#Ask the user to provide the file name in .sql format of the file in which the protein sequences are stored.  
#These protein sequences are obtained from the SYSTERS database and are stored in the form of insert  
#statements. The filename is collected from the command line and is assigned to the variable  
#protein_file_name.  
  
print STDOUT "Please input the Sequence Filename in .sql Format:\n";  
my $protein_file_name = <STDIN>;  
  
#Remove any new line character at the end of the file name  
chomp ($protein_file_name);  
  
#Open the file and associate the file handle PROTEIN_FILE_NAME with it for readability. If the file does  
#not open due to any errors print the information given below and exit  
  
unless (open(PROTEIN_FILE_NAME, $protein_file_name)) #Begin unless  
{  
    print STDOUT "Cannot open file \"$protein_file_name\"\n\n";  
    exit;  
} #End unless  
  
#Read from the file and remove any white space characters such as a space, tab or newline character to  
#make it a one long continuous string of characters and assign it to the variable prot_seq
```

```

while (my $text = <PROTEIN_FILE_NAME>) #Begin while
{
    $text =~ ~s/\s//gs;
    $prot_seq = $prot_seq.$text;
} #End while

#After the file has been read this command is given to close the file
close PROTEIN_FILE_NAME;

#The split operation is performed at the index of ';' to separate the insert statements and store them in an
#array seq_data
my @seq_data = split (/, \d+\/, $prot_seq);

#The length of the array is determined and is assigned to the variable len_seq_data
my $len_seq_data = @seq_data;

#Create a file to store the output. Assign the file name to the variable output_file
my $output_file = "results_out.txt";

#Open the file and associate the file handle OUTPUT_FILE with it. Also if the file does not open due to
#any errors print the information given below and exit
unless (open(OUTPUT_FILE, ">>$output_file")) #Begin unless
{
    print "Cannot open file \"$output_file\" to write to!!\n\n";
    exit;
} #End unless

#To autoflush the file handle
autoflush OUTPUT_FILE 1;

#Print out the length of the seq_data array to determine the number of sequences in a #given file
print OUTPUT_FILE "\n\nNumber of Sequences in this File is: $len_seq_data\n\n";

#Take in one insert statement at a time from the seq_data array
foreach my $i (@seq_data) #Begin foreach
{
    #Initializing variables
    my $seq = "";
    my $sequence = "";

    #Regular Expression to obtain the sequence part of the insert statement the sequence is obtained in
    #FASTA format to print it along with the back-translated DNA sequence as output. Also the raw
    #protein sequence without the FASTA definition line is also obtained to submit it as query to the
    #QBLAST system. The #regular expressions given below are used to obtain both these sequences.
    #The two if conditions are given depending on what pattern matches to the insert statement

    if ($i =~ /(>.*[\n]((([A-Z].*\n)+).*)'.'/mg) # Begin if
    {
        $seq = $2;           # Raw protein sequence
        $sequence = $1;     # Protein sequence in FASTA format

        #Any whitespace character is removed from the raw protein sequence
        $seq =~ s/\s//gs;
    } # End if
    if ($i =~ /(>.*\n(([A-Z]).*).*)'.'/mg) # Begin if
    {

```



```

        $seq = $2;
        $sequence = $1;
    } # End if

#The Raw protein sequence is assigned to the variable protein_sequence
my $protein_sequence = $seq;

#Initializing variables
my $db = "";
my $accno = "";

#The database shortcut and the accession number from the protein sequence in FASTA format are
#obtained and are assigned to the variables db and accno respectively in order to determine which
#database was the sequence originally obtained from and stored in SYSTERS. Three if conditions
#are mentioned depending upon which pattern matches to the protein sequence in FASTA format.

if ($sequence =~ />(.*)(.*)\./mg) # Begin if
{
    $db = $1;      # Shortcut to database
    $accno = $2;  # Accession number
} # End if
if ($sequence =~ />(.*)(.*) \d/mg) # Begin if
{
    $db = $1;
    $accno = $2;
} # End if
if ($sequence =~ />(.*)(.*\d+) \d/mg) # Begin if
{
    $db = $1;
    $accno = $2;
} # End if

print STDOUT "\n\nThe Database Is: $db\n";
print STDOUT "\n\nThe Accession Number Is: $accno\n\n";

#The below if conditions match the database name and according to the match invoke the
#corresponding subroutines. if the database shortcut matches to TRE, SPR, or SPU then the
#swissprot subroutine is invoked and the accession number of the sequence, the protein sequence
#in FASTA format and the raw protein sequence are passed as parameters. if the database shortcut
#matches to SC then the SGD subroutine is invoked and the insert statement along with the
#protein sequence in FASTA format and the raw protein sequence are passed as parameters. else
#the subroutine tblastn is invoked and the protein sequence in FASTA format along with the raw
#sequence are passed as parameters.

if (($db eq "TRE") || ($db eq "SPR") || ($db eq "SPU")) # Begin if
{
    &swissprot($accno, $sequence, $protein_sequence);

    #This is the counter to indicate how many back-translated DNA sequences are obtained
    #from swissprot/TrEMBL database
    $swiss_ctr = $swiss_ctr + 1;
} # End if
elsif ($db eq "SC") # Begin elsif
{
    &SGD($i, $sequence, $protein_sequence);
}

```

```

        #This is the counter to indicate how many back-translated DNA sequences are obtained
        #from SGD database
        $yeast_ctr = $yeast_ctr + 1;
    } # End elsif
    else
    {
        sleep 180;
        &tblastn($sequence, $protein_sequence);
    } # End else
} # End foreach

#This is the counter to indicate how many back-translated DNA sequences are obtained from the NCBI
#TBLASTN program
$tblastn_ctr = $len_seq_data - ($swiss_ctr + $yeast_ctr);

print OUTPUT_FILE "\n\nThe Number of Sequences From SWISSPROT/TrEMBL are: ".$swiss_ctr;
print OUTPUT_FILE "\n\nThe Number of Sequences From SGD are: ".$yeast_ctr;
print OUTPUT_FILE "\n\nThe Number of Sequences From TBLASTN are: ".$tblastn_ctr;

close OUTPUT_FILE;

# End of program
print "\n\nThe File Has Completed Execution And This Is The End Of The Program\n\n";

# Exit the program
exit;

#####
# Subroutines
#####

#This is where the swissprot subroutine begins. It takes in the accession number, the protein sequence in
#FASTA format and the raw protein sequence as parameters.

sub swissprot
{
    #Initializing variables
    my $acc_no = $_[0];          # Accession number
    my $prot_seq = $_[1];       # Protein sequence in FASTA format
    my $protein_seq = $_[2];    # Raw protein sequence

    #A new LWP::UserAgent object is created and the UserAgent is given an identity as Agent 03
    #my $ua_swissprot = LWP::UserAgent->new;
    $ua_swissprot->agent("Agent03");

    #This is the URL of the UniProtKB/Swiss-Prot and the UniProtKB/TrEMBL website. This URL
    #when merged with the accession number of the protein sequence displays the web page of the
    #protein sequence from where the link to the coding sequence is obtained

    my $swissprot_url = "http://ca.expasy.org/uniprot/";
    my $merged_url = $swissprot_url.$acc_no;    # The merged URL

    #A new HTTP::Request object is created and the merged URL is passed to the get method of the
    #class. The HTTP::Request object is then passed to the request method of the UserAgent object to

```

#send a request to the desired URL and get a response. The response which is obtained from the #request is stored in a HTTP::Response object

```
my $req_swissprot = HTTP::Request->new(GET => $merge_url);
my $res_swissprot = $ua_swissprot->request($req_swissprot);
```

#if the page results into a 410 gone error then that is caught by the status_line method of the #HTTP::Response class and the tblastn subroutine is invoked

```
if($res_swissprot->status_line eq "410 Gone") # Begin if
{
    sleep 120;
    &tblastn($pro_seq, $protein_seq);
    next;
} # End if
```

#The content of the HTTP::Response object is obtained by the content method of the #HTTP::Response class and it is assigned to a variable & is stored as one continuous string. The #content in this case is the HTML source of the swissprot protein web page

```
my $temp_swissprot = $res_swissprot->content;
```

#if there is a primary accession number present on the page then the code is further executed after #the if statement else the tblastn subroutine is invoked

```
#Begin if
if($temp_swissprot =~ /Primary accession number<.*\n.*<b>(.*?)</b>/m)
{
    my $a_no = $1;
```

#if the primary accession number matches the secondary accession number then the code #is further executed after the if statement else the tblastn subroutine is invoked

```
if($a_no eq $acc_no) # Begin if
{
    #if the coding sequence is present on the webpage then the HTML content of the
    #page is parsed and the link to the web page of the nucleotide sequence is
    #extracted using regular expressions. if the coding sequence is not present then
    #the tblastn subroutine is invoked to get the back-translated DNA sequence
```

```
if(($temp_swissprot =~ /\d; -; Genomic_DNA.*\[<a
href="(.*?)>CoDingSequence</a>\]/mg) || ($temp_swissprot =~ /\d; -; mRNA.*\[<a
href="(.*?)>CoDingSequence</a>\]/mg)) # Begin if
{
```

```
    my $temp_swissprot1 = $1;
```

```
    $temp_swissprot1 =~ s/&\/&/g;
    my $temp_swissprot2 = $temp_swissprot1;
```

```
    # if the link to the coding sequence is determined then another
    #connection is made to nucleotide web page using the coding sequence
    #URL. if the link to the coding sequence is not determined then the
    #tblastn subroutine is invoked and executed
```

```

if ($temp_swissprot =~ m/http://www.ebi.ac.uk/g)
{
    #Another LWP::UserAgent object is created with identity
    #Agent04.
    my $ua_ebi = LWP::UserAgent->new;
    $ua_ebi->agent("Agent04");

    #The URL in this case is the URL of the coding sequence
    #obtained from the swissprot protein web page
    my $ebi_url = $temp_swissprot2;

    #A new HTTP::Request object is created and the coding
    #sequence URL is passed to the get method of the class. The
    #HTTP::Request object is then passed to the request method
    #of the UserAgent object to send a request to the desired URL
    #and get a response. The response which is stored in the
    #HTTP::Response object

    my $req_ebi = HTTP::Request->new(GET => $ebi_url);
    my $res_ebi = $ua_ebi->request($req_ebi);
    #The content of the HTTP::Response object is obtained and
    #it is assigned to a variable. The content in this case is the
    #HTML source of the EBI #nucleotide/coding sequence
    #webpage
    my $temp_ebi = $res_ebi->content;

    #From the HTML source the sequence part of the page is then
    #extracted using regular expressions. Any whitespace
    #characters and numbers are removed and the sequence is
    #assigned to a variable.
    $temp_ebi =~ /(SQ.*\n+ (.*\n+))\n/mg;
    my $temp_ebi1 = $1;

    $temp_ebi1 =~ s/SQ.*//gm;
    $temp_ebi1 =~ s/[^actg]//gm;

    #This is the final back-translated DNA sequence obtained
    #from swissprot and EBI
    $dna_seq = $temp_ebi1;

    #print statements are then executed to print the protein
    #sequence in FASTA format and the corresponding back-
    #translated DNA sequence on the console as well as write it to
    #the output file, results_out

    print "\n\nThe protein sequence is:\n\n".$prot_seq."\n\n";
    print OUTPUT_FILE "\n\nThe protein sequence
    is:\n\n".$prot_seq."\n\n";

    print "And the corresponding DNA sequence from
    Swissprot/TrEMBL is:\n\n".$dna_seq."\n\n";
    print OUTPUT_FILE "And the corresponding DNA sequence
    from Swissprot/TrEMBL is:\n\n".$dna_seq."\n\n";
} # End if
else # Begin else
{

```

```

                sleep 180;
                &tblastn($protein_seq, $prot_seq);
            } # End else
        } # End if
    else # Begin else
    {
        sleep 180;
        &tblastn($protein_seq, $prot_seq);
    } # End else
    } # End if
else # Begin else
{
    sleep 180;
    &tblastn($protein_seq, $prot_seq);
} # End else
} # End sub

```

This is where the SGD subroutine begins. It takes in the insert statement, the protein sequence in FASTA #format and the raw protein sequence as parameters.

```

sub SGD
{
    my $prot_seq = $_[0];           # insert statment
    my $seq = $_[1];               # protein sequence in FASTA format
    my $p_seq = $_[2];            # raw protein sequence
    my $dna_sequence;

    #The if condition checks to see if there is an accession number mentioned in the insert statment
    #and if is present it extracts the accession number assigns it to a variable using regular
    #expressions. However, if the accession number is not mentioned then the tblastn subroutine is
    #invoked

    if ($prot_seq =~ /\(d+,'(.*)'\)/m) # Begin if
    {
        my $prot_acc_no = $1;

        #A new LWP::UserAgent object is created and the UserAgent is given an identity as
        #Agent 05
        my $ua_yeastgenome = LWP::UserAgent->new;
        $ua_yeastgenome->agent("Agent05");

        #This is the URL of the SGD website to link to it. This URL when merged with the
        #accession number of the protein, which is obtained as above from the insert statment,
        #sequence displays the web page of the protein sequence. From this page the systematic
        #name of the protein is obtained which helps to determine the coding sequence
        my $yeastgenome_url = "http://db.yeastgenome.org/cgi-bin/locus.pl?sgdid=";

        #This is URL merged with the accession number
        my $merge_url = $yeastgenome_url.$prot_acc_no;
    }
}

```

```
#A new HTTP::Request object is created and the merged URL is passed to get method of
#the class. The HTTP::Request object is then passed to the request method of the
#UserAgent object to send a request to the desired URL and get a response. The response
#which is obtained from the request is stored in the HTTP::Response object
```

```
my $req_yeastgenome = HTTP::Request->new(GET => $merge_url);
my $res_yeastgenome = $ua_yeastgenome->request($req_yeastgenome);
```

```
#The content of the HTTP::Response object is obtained and it is assigned to a variable.
#The content in this case is the HTML source of the SGD protein web page
my $stemp_yeastgenome = $res_yeastgenome->content;
```

```
#There are two pattern matches in order to obtain the systematic name of the portein. if
#either of them match to the HTML source content then the systematic name is extracted
#from the content and is assigned to a variable. else if neither of the patterns match then
#the tblastn subroutine is invoked
```

```
if ($stemp_yeastgenome =~ /Systematic Name.*"top">(.*<\td><\tr><\table>.*Alias/m)
{
    $stemp1_yeastgenome = $1;
}
elsif ($stemp_yeastgenome =~ /Systematic
Name.*"top">(.*<\td><\tr><\table>.*Feature Type/m)
{
    $stemp1_yeastgenome = $1;
}
else
{
    sleep 120;
    &tblastn($seq, $p_seq);
}
}
```

```
#Once the systematic name is extracted A new LWP::UserAgent object is created and the
#UserAgent is given an identity as Agent 06
```

```
my $ua_yeastgen = LWP::UserAgent->new;
$ua_yeastgen->agent("Agent06");
```

```
#This is the URL of the SGD website where the coding sequence for the corresponding
#protein sequence can be obtained. However the systematic name has to be merged with
#the URL in order to complete it. This URL when merged with the systematic name of
#the protein sequence displays the web page of the coding sequence.
```

```
my $yeastgen_url = "http://db.yeastgenome.org/cgi-bin/getSeq?seq=";
my $yeastgen_url_merge = "&flankl=0&flankr=0&map=n3map";
my $yeastgen_merged_url = $yeastgen_url.$stemp1_yeastgenome.$yeastgen_url_merge;
```

```
#A new HTTP::Request object is created and the merged URL is passed to the get
#method of the class. The HTTP::Request object is then passed to the request method of
#the UserAgent object to send a request to the desired URL and get a response. The
#response which is obtained from the request is stored in the HTTP::Response object
```

```
my $req_yeastgen = HTTP::Request->new(GET => $yeastgen_merged_url);
my $res_yeastgen = $ua_yeastgen->request($req_yeastgen);
```

```

#The content of the HTTP::Response object is obtained and it is assigned to a variable.
#The content in this case is the HTML source of the SGD nucleotide/coding sequence
#webpage
my $temp_yeastgen = $res_yeastgen->content;

#From the HTML source the nucleotide sequence part of the web page is then extracted
#using regular expressions. if the nucleotide sequence is not found on the page then the
#tblastn subroutine is invoked. Any whitespace characters and numbers are removed and
#the sequence is assigned to a variable.
if ($temp_yeastgen =~ /<pre>>.*\n(([ACTG].*\n)+)<\pre>/m) # Begin if
{
    $temp1_yeastgen = $1;
    $temp1_yeastgen =~ s/\n//g;
    $dna_sequence = $temp1_yeastgen;

    #This is the final back-translated DNA sequence obtained from SGD
    $dna_sequence =~ tr/ATCG/atcg/;

    #print statements are then executed to print the protein sequence in FASTA
    #format and the corresponding back-translated DNA sequence on the console as
    #well as write it to the output file, results_out

    print STDOUT "\n\nThe protein sequence is:\n\n".$seq."\n\n";
    print OUTPUT_FILE "\n\nThe protein sequence is:\n\n".$seq."\n\n";

    print STDOUT "And the corresponding DNA sequence from YeastGenome
    is:\n\n".$dna_sequence."\n\n";
    print OUTPUT_FILE "And the corresponding DNA sequence from
    YeastGenome is:\n\n".$dna_sequence."\n\n";
} # End if
else # Begin else
{
    sleep 120;
    &tblastn($seq, $p_seq);
} # End else
} # End if
else # Begin else
{
    sleep 120;
    &tblastn($seq, $p_seq);
} # End else
} # End sub

#This is from where the tblastn routine begins. It takes in the protein sequence in FASTA format and the
#raw protein sequence as parameters.

sub tblastn
{
    my $flag = 0;                # Flag is set
    my $prot_seq = $_[1];        # Protein sequence in FASTA format

    #A new LWP::UserAgent object is created and the UserAgent is given an identity as Agent 01
    my $ua_tblastn = LWP::UserAgent->new;
    $ua_tblastn->agent("Agent01");

```

#The put command of the URL API is formed and is assigned to the variable args. The query
#parameter specifies the raw protein sequence. Database specifies the nucleotide database with
#which the query sequence is compared. The hitlist_size indicates that only the first hit is to be
#returned from the result set and the program specifies that tblastn is to be executed.

```
my $args =
"CMD=Put&QUERY=$_[0]&DATABASE=nr&HITLIST_SIZE=1&FILTER=L&EXPECT=1&FORMAT
_TYPE=HTML&PROGRAM=tblastn&CLIENT=web&SERVICE=plain&NCBI_GI=on&PAGE=nucleoti
des";
```

#A new HTTP::Request object is created and put command that is formed and assigned to args is
#passed to the get method of the class. The HTTP::Request object is then passed to the request
#method of the UserAgent object which in turn passes the command to the the QBLAST system
#and get a response. The response is stored in the HTTP::Response object

```
my $req_tblastn = HTTP::Request->new(POST => http://www.ncbi.nlm.nih.gov/blast/Blast.cgi");
$req_tblastn->content_type ("application/x-www-form-urlencoded");
$req_tblastn->content($args);
my $res_tblastn = $ua_tblastn->request($req_tblastn);
```

#From the response that is obtained from the QBLAST system, the RID and the remaining time of
#execution are determined and are assigned to their corresponding variables. The content of this
#HTTP:Response object is the tblastn intermediate result page where the formatting options can
#be set

```
$res_tblastn->content =~ /^  RID = (.*$)/m;
my $srid=$1;
```

```
$res_tblastn->content =~ /^  RTOE = (.*$)/m;
my $rtoe=$1;
```

#The program is made to halt in an intermediate state till tblastn does not complete execution and
#send the results
sleep \$rtoe;

#while the program is in the intermediate state till the time of execution is completed a new
#HTTP::Request object is created and the get command of the URL API is passed to the QBLAST
#system specifying the request ID of the input query request The HTTP::Request object is then
#passed to the request method of the UserAgent object to send a request to the desired URL and
#get a response. The response which is obtained from the request is stored in the HTTP::Response
#object

```
while (true) # Begin while
{
    sleep 5;          # Another 5 seconds delay is specified
    $req_tblastn = HTTP::Request->new(GET =>
"http://www.ncbi.nlm.nih.gov/blast/Blast.cgi?CMD=Get&FORMAT_OBJECT=SearchInfo&RID=$srid");
    $res_tblastn = $ua_tblastn->request($req_tblastn);
```

#The content of this HTTP:Response object is the tblastn intermediate waiting page
#which specifies the status of the current request being processed, the Request ID for
#which is passed. From this response that is obtained from the QBLAST system, the
#status of execution is determined and depending upon the status the different if
#conditions are executed.

```
$response = $res_tblastn->content;
```



```

#if the status is shown as WAITING then the print statement 'Searching...'
#is execute to display to the user that the search is taking place
if ($res_tblastn->content =~ /tStatus=WAITING/m) # Begin if
{
    print STDOUT "Searching...\n";
    next;
} # End if

#if the status results in FAILED then the protein sequence along with the error message is
#written to the output file and the program exits the subroutine
if ($res_tblastn->content =~ /tStatus=FAILED/m) # Begin if
{
    print OUTPUT_FILE "\n\nThe protein sequence is:\n\n".$prot_seq."\n\n";
    print OUTPUT_FILE "Search $rid failed;please report to blast-
help\@ncbi.nlm.nih.gov.\n";
    $flag = 0;
    last;
} # End if

#if the status results in UNKNOWN then the protein sequence along with the error
#message is written to the output file and the program exits the subroutine
if ($res_tblastn->content =~ /tStatus=UNKNOWN/m) # Begin if
{
    print OUTPUT_FILE "\n\nThe protein sequence is:\n\n".$prot_seq."\n\n";
    print OUTPUT_FILE "Search $rid expired.\n";
    $flag = 0;
    last;
} # End if

#and if the status results in READY, that is the search is complete, then it checks to see if
#any hits are returned or there are no hits pertaining to the query sequence. if there are
#hits then the flag is set to 1 and the program control leaves while loop and if there are no
#hits found then the protein sequence along with the error message is written to the
#output file and the program exits the subroutine
if ($res_tblastn->content =~ /tStatus=READY/m) # Begin if
{
    if ($res_tblastn->content =~ /tThereAreHits=yes/m) # Begin if
    {
        print STDOUT "Search complete, retrieving results...\n";
        $flag = 1;
        last;
    } # End if
    else # Begin else
    {
        print OUTPUT_FILE "\n\nThe protein sequence
is:\n\n".$prot_seq."\n\n";
        print OUTPUT_FILE "No hits found.\n";
        $flag = 0;
        last;
    } # End else
} # End if

#if none of the above conditions are fulfilled the program control reaches this part and it
#indicates that there is something unexpected happened with the execution of tblastn.
#When NCBI blocks access to the code this part of the code is executed

```

```

        print "\n\nif we get here, something unexpected happened.\n\n";
    } # End while

    #if there are any hits to the input query sequence then the flag is set to 1 and the program begins
    #executing at this point

    if ($flag == 1) # Begin if
    {
        #A new HTTP::Request object is created and the get command of the URL API is passed
        #to the QBLAST system specifying the request ID to get the results pertaining to the
        #RID. The HTTP::Request object is then passed to the request method of the UserAgent
        #object to send a request to the desired URL and get a response. The response which is
        #obtained from the request is stored in the HTTP::Response object

        $req_tblastn = HTTP::Request->new(GET =>
"http://www.ncbi.nlm.nih.gov/blast/Blast.cgi?CMD=Get&FORMAT_TYPE=Text&RID=$rid");
        $res_tblastn = $ua_tblastn->request($req_tblastn);

        #The content of this HTTP:Response object is the blast results page which gives the
        #results pertaining to the input query sequence. The NCBI BLAST graphic display along
        #with the hits and alignments sections are mentioned on this web page. The content is
        #assigned to a variable as a string

        my $tblastn_results_page = $res_tblastn->content;

        #From the response that is obtained from the QBLAST system, the accession number of
        #the nucleotide sequence is extracted and assigned to a variable. Also the start and end
        #positions of the subject sequence or the nucleotide sequence is extracted. To do this
        #only the alignment section of the result page is taken and further operations are
        #performed using regular expressions to obtain what is needed to get the back-translated
        #DNA sequence from the nucleotide web page

        my $temp0 = $tblastn_results_page;
        $temp0 =~ /^>.*\.(.*)\d+\/gm;
        $acc_no = $1;

        my $temp1 = $tblastn_results_page;
        $temp1 =~ s/.*^ALIGNMENTS//ms;
        my $temp2 = $temp1;

        my $temp3 = $temp2;
        $temp3 =~ s/^ Database:.*//ms;
        my $temp4 = $temp3;

        my @arr = split (/Score/, $temp4);
        my $len = @arr;
        my $seq_tblastn = @arr[1];

        my $protein = join (" ", ($seq_tblastn =~ /^Sbjct.*\n/gm) );
        my $subject_range = join('.', ($protein =~ /(\d+).*\D(\d+)/s));
        my @start_end = split(/\./, $subject_range);

        my $temp_start = $start_end[0];
        my $temp_end = $start_end[1];

```

```

my $start_position;
my $end_position;

#if the value of the start position is lesser than the value of the end position then the back-
#translated DNA sequence is on the plus strand. However if the value of the start position
#is greater than the value of the end position then the nucleotide sequence is located on
#the minus strand. In this case, in order to read the nucleotide sequence from the web
#page the values of the start and end positions are reversed

if ($temp_start < $temp_end) # Begin if
{
    $start_position = $temp_start;
    $end_position = $temp_end;
} # End if
else # Begin else to reverse the values
{
    $start_position = $temp_end;
    $end_position = $temp_start;
} # End else

#A new LWP::UserAgent object is created and the UserAgent is given an identity as
#Agent 02
my $ua_genbank = LWP::UserAgent->new;
$ua_genbank->agent("Agent02");

#A new HTTP::Request object is created and the URL to the GenBank nucleotide web
#page is passed to the get method of the class. The HTTP::Request object is then passed
#to the request method of the UserAgent object to send a request to the desired URL and
#get a response. The response which is stored in the HTTP::Response object

my $req_genbank = HTTP::Request->new(GET =>
"http://www.ncbi.nlm.nih.gov/entrez/viewer.fcgi?txt=yes&list_uids=$acc_no&db=n");
my $res_genbank = $ua_genbank->request($req_genbank);

#The content of the HTTP::Response object is obtained and it is assigned to a variable.
#The content in this case is the HTML source of the NCBI #nucleotide webpage
my $temp5 = $res_genbank->content;

#From the source of the webpage the nucleotide sequence is obtained using the regular
#expressions given below. This is the entire nucleotide sequence from which the back-
#translated DNA sequence needs to be extracted. This part of the page is stored in an
#array

$temp5 =~ s/.*ORIGIN//ms;
my @seq_genbank = $temp5;

#Initializing variables
my $d_seq;
my $line_seq = "";

#each line from the array is then taken, the whitespace characters are removed along with
#any line numbers and the all the lines are merged together and stores it as a one long
#continuous string. This string is #assigned to the variable line_seq

foreach my $line2 (@seq_genbank)
{

```

```

    chomp($line2);

    $d_seq = $line2;
    $d_seq =~ s/[^acgt]//g;
    $line_seq .= $d_seq;
}

#To obtain the back-translated DNA sequence ranging from the start position to the end
#position the substr function is used to extract the part of the DNA sequence between the
#start and end positions from the string stored in line_seq and assigns the back-translated
#DNA sequence to the final_DNA variable

my $final_DNA = substr($line_seq, $start_position-1, $end_position-($start_position-1));

#In case the back-translated DNA sequence lies on the minus strand
if ($start_position > $end_position) # Begin if
{
    #Print the protein sequence in FASTA format from the .sql file

    print "\n\nThe protein sequence is:\n\n".$prot_seq."\n\n";
    print OUTPUT_FILE "\n\nThe protein sequence is:\n\n".$prot_seq."\n\n";

    #Calculating the reverse complement
    #Copying the intermediate DNA result into a new variable
    #Substitute all the bases by their complement using the tr function
    #A -> T, T -> A, C -> G, G -> C

    my $revcom = reverse $final_DNA;
    $revcom =~ tr/ACGTacgt/TGCAtgca/;

    #Print the reverse complement back-translated DNA sequence obtained from
    #TBLASTN
    print "And the corresponding DNA sequence from TBLASTN
    is:\n\n".$revcom."\n\n";
    print OUTPUT_FILE "And the corresponding DNA sequence from TBLASTN
    is:\n\n".$revcom."\n\n";
}
else
{
    #Print the protein sequence in FASTA format from the .sql file
    print "\n\nThe protein sequence is:\n\n".$prot_seq."\n\n";
    print OUTPUT_FILE "\n\nThe protein sequence is:\n\n".$prot_seq."\n\n";

    #Print the back-translated DNA sequence obtained from tblastn
    print "And the corresponding DNA sequence from TBLASTN
    is:\n\n".$final_DNA."\n\n";
    print OUTPUT_FILE "And the corresponding DNA sequence from TBLASTN
    is:\n\n".$final_DNA."\n\n";
}
}
}

```

REFERENCES

- [1] DW. (1999, May). NCBI News: QBLAST Provides Quick Reformat Feature. NCBI, MD. [Online]. Available: <http://www.ncbi.nlm.nih.gov/Web/Newsltr/Summer99/qblast.html>.
- [2] NCBI. (2002, Oct.). BLAST: QBLast's URL API User's Guide. NCBI, MD. [Online] Available: <http://www.ncbi.nlm.nih.gov/blast/Doc/urlapi.html>.
- [3] Jean-Michael Claverie and Cedric Notredame, *Bioinformatics for Dummies*. Hoboken, New Jersey: Wiley, John & Sons, Incorporated, 2003.
- [4] C.S.V Murthy, *Bioinformatics*. Mumbai, India: Himalaya Publishing House, 2003.
- [5] A. Krause, J. Stoye and M. Vingron, "The SYSTERS protein sequence cluster set," *Nucleic Acids Research*, vol. 28(1), pp. 270-272, Jan. 2000.
- [6] Antje Krause, Hannes Luz, Eike Staub, Jens Stoye, Heiko Schmidt, Martin Vingron, Thomas Meinel, Pierre Nicodème and Marc Rehmsmeier. (2006, Feb.). SYSTERS Database Searching and Clustering. MPI for Molecular Genetics Computational Molecular Biology, Germany. [Online]. Available: <http://system.s.molgen.mpg.de/cgi-bin/info.pl#systeminfo>.
- [7] Genetic Sequence Data Bank. (2006, Apr.). NCBI-GenBank Flat File Release 153.0. GenBank, NCBI, MD. [Online]. Available: <ftp://ftp.ncbi.nih.gov/genbank/gbrel.txt>.
- [8] Brigitte Boeckmann, Amos Bairoch, Rolf Apweiler, Marie-Claude Blatter, Anne Estreicher, Elisabeth Gasteiger, Maria J. Martin, Karine Michoud, Claire O'Donovan, Isabelle Phan, Sandrine Pilbout and Michel Schneider, "The SWISS-PROT protein knowledgebase and its supplement TrEMBL in 2003," *Nucleic Acids Research*, vol. 31(1), pp. 365-370, Jan. 2003.
- [9] Swiss-Prot Protein Knowledgebase and TrEMBL Computer-Annotated Supplement to Swiss-Prot. (2006, May). Swiss-Prot Protein Knowledgebase Release 49.6 Statistics. SwissProt/TrEMBL, Geneva. [Online]. Available: <http://ca.expasy.org/sprot/relnotes/relstat.html>.
- [10] Swiss-Prot Protein Knowledgebase and TrEMBL Computer-Annotated Supplement to Swiss-Prot. (2006, May). UniProtKB/TrEMBL Protein Database Release 32.6 Statistics. SwissProt/TrEMBL, Geneva. [Online]. Available: http://www.ebi.ac.uk/swissprot/sptr_stats/index.html.
- [11] ELG. (2005 Dec.). UniProtKB/Swiss-Prot. SwissProt/TrEMBL, Geneva. [Online]. Available: http://www.expasy.org/sprot/sprot_details.html.

- [12] Amos Bairoch, Rolf Apweiler, Cathy H. Wu, Winona C. Barker, Brigitte Boeckmann, Serenella Ferro, Elisabeth Gasteiger, Hongzhan Huang and Rodrigo Lopez. (2005, Jan.). The Universal Protein Resource (UniProt). *Nucleic Acids Research*. [Online]. 33, Database Issue: D154-D159. Available: http://nar.oxfordjournals.org/cgi/reprint/33/suppl_1/D154.
- [13] Jodi E. Hirschman, Rama Balakrishnan, Karen R. Christie, Maria C. Costanzo, Selina S. Dwight, Stacia R. Engel, Dianna G. Fisk, Eurie L. Hong, Michael S. Livstone, Robert Nash, Julie Park, Rose Oughtred, Marek Skrzypek, Barry Starr, Chandra L. Theesfeld, Jennifer Williams, Rey Andrada, Gail Binkley and Qing Dong. (2006, Jan.). Genome Snapshot: a new resource at the *Saccharomyces Genome Database (SGD)* presenting an overview of the *Saccharomyces cerevisiae* genome. *Nucleic Acids Research*. [Online]. 34, Database Issue: D442-D445. Available: http://nar.oxfordjournals.org/cgi/reprint/34/suppl_1/D442.
- [14] Scott McGinnis and Thomas L. Madden. (2004, Jul.). BLAST: at the core of a powerful and diverse set of sequence analysis tool. *Nucleic Acids Research*. [Online]. 32, Web Server Issue: W20-W25. Available: <http://www.pubmedcentral.gov/picrender.fcgi?artid=441573&blobtype=pdf>.
- [15] Ian Korf, Mark Yandell and Joseph Bedell, *BLAST*. Sebastopol, CA: O'Reilly, 2000.
- [16] James Tisdall, *Beginning Perl For Bioinformatics*. Sebastopol, CA: O'Reilly, 2001.
- [17] Randal L. Schwartz, Tom Phoenix and Brian D Foy, *Learning Perl*, 3rd ed. Sebastopol, CA: O'Reilly, 2001.
- [18] EBI Support. (2006, Feb.). EMBL-EBI: About the EMBL-EBI. EMBL-EBI, UK. [Online]. Available: http://www.ebi.ac.uk/Information/AboutEBI/about_ebi.html.