# Information Alert in Biological Sequence Databases

by

Georgia Adamopoulou

A thesis submitted in fulfillment of the
requirements for the degree of

Master of Computer Engineering

Technical University of Crete
Department of Electronic and Computer Engineering

Laboratory for Programming and
Intelligent Systems Engineering

# Contents

# List of Figures

# List of Tables

To Mum and Dad

# Acknowledgements

I would like to thank everyone who contributed to the fulfillment of this thesis and first of all my supervisor Dr. Manolis Koubarakis for his precious advice and his guidance. I would also like to thank the group of Bridgemap and especially Dr. George Kotoulas for his explanations about biological processes and terms and his patience to my continuous questions when I was trying to figure out the needs of molecular biology related to informatics. Without his help molecular biology would be so confusing to me...

I owe many thanks to my friends and the research group of the Intelligence lab Christos, Paraskeui, Thodoris and Stamatis for their solidarity. The moments we spent altogether except that made my living in Chania a memorable experience gave me the courage to continue.

I am also very grateful to Athena and her family for their support. I will remember plenty of wonderful moments I spent with them.

Finally, I would like to thank Petros for his presence as a friend and good listener especially at these last months before the presentation of this work.

# Chapter 1

# Introduction

An unprecedented wealth of data is being generated by genome sequencing projects and other experimental efforts to determine the structure and function of biological molecules. The demands and opportunities for interpreting this data are expanding more than ever. *Bioinformatics* is the development and application of computer methods for management, analysis, interpretation and prediction, as well as for the design of experiments [30]. This is the area where this work belongs to.

## 1.1 Overview

The latest decades we have noticed a blast in molecular biology. This great development is sealed by sequencing the complete human genome. This is only one of the genomes of organisms that have been sequenced all these years.

The genome of more and more organisms is under study. This means that large-scale sequencing is taking place everyday in genomic laboratories all over the world. Consequently, a huge amount of information is produced very rapidly. Moreover, this quantity of information becomes available in the web, leading to an unpredictable and rapid increase of available information. This information is crucial for the scientific research of biologists but if they want to use this information, they have to cope with both the problem of *information overload* and the problem of being *continually informed* of new information.

One solution to this problem is the use of techniques from *selective dissemination of information.* The idea of selective information dissemination is that users express their desire and preferences for information by posting *profiles* or *long-standing queries* to a computer system. The system then informs the user about any incoming information matching his/her profile.

The same idea is implemented by an *information alert* scenario where users actually receive alerts whenever any incoming information matching to their interests. Thus, in this work we will use the terms *information dissemination* and *information alert* interchangeably.

A high level view of such a system is shown in Figure 1.1 as it is presented in [37]. We notice that the figure presents an information dissemination scenario in the context of a *distributed peer-to-peer (P2P) agent architecture*. In this scenario users utilize their *end-agents* to post *profiles* or *documents* (expressed in some appropriate language) to some *middle-agents*. End-agents play a dual role: they can be information producers and information consumers at the same time. The P2P network of middle-agents is the "glue" that makes sure that published documents arrive at interested subscribers. To achieve this, middle-agents forward posted profiles to other middle-agents using an appropriate P2P protocol. In this way, matching of a profile with a document can take place at a middle-agent that is as close as possible to the origin of the incoming document. Profile forwarding can be done in a sophisticated way to minimize network traffic e.g., no profiles that are less general than the one that has already been processed are actually forwarded.

Although there is plenty of research on distributed agent architecture [36, 14, 13] as well as implemented systems [25, 38], at the first step the emphasis was given on systems making use of a single central middle-agent. Such an architecture we propose for the alert system that we describe in Chapter 5.

This dissertation concentrates on the problem of information dissemination focusing mainly on the view of molecular biology. We were motivated by the study of both the algorithms of sequence comparison and the information dissemination systems to end up to this work with the following contributions:

- We developed a variation of the BLAST algorithm that trades space for time. The main idea is the use of alternative techniques in order to reduce the running time of the algorithm. Specifically, we make use of indexing methods in one part of the algorithm to achieve a faster and more efficient way of matching. After the implementation of the variation we propose, we evaluate it experimentally using real genomic data part of which come from of the European project Bridgemap [2]. Apart from the evaluation we also compare it to the original BLAST algorithm.

- We designed and implemented BioAlert, an information alert system for biologists that allows the specification of long-standing queries on textual as well as sequence data of protein databases. Such a service is provided by the Swiss-Prot database which is named Swiss-Shop [8]. Although our system works using the weekly updates of Swiss-Prot, as Swiss-Shop does, it differs from that service in the way that offers more functionality to users. Users can specify long-standing queries either on textual or on sequence data, as they did using Swiss-Shop, but now they can also specify queries on both type of data. This kind of conjunctive query functionality is not offered in Swiss-Shop. In our system we provide this functionality by combining algorithms implementing the two different types of search. For the textual

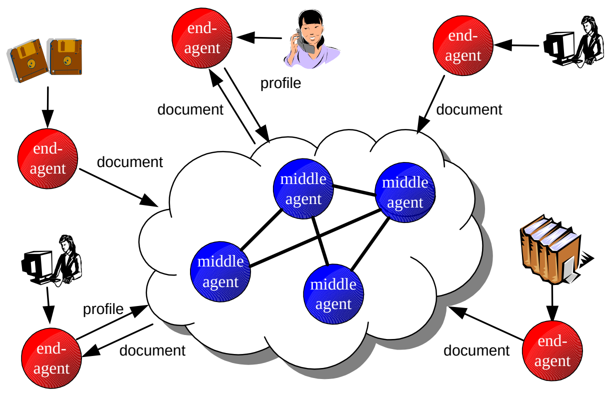


Figure 1.1: A high level view of an information dissemination system

search we use the data model and algorithms presented in [38, 36] that rely on the Boolean model from Information Retrieval.

## 1.2 Organization of the dissertation

This dissertation is organized as follows. In Chapter 2 we briefly present two well-known to the biologists and representative for our work services and two textual information dissemination systems. In Chapter 3 we discuss some fundamental concepts and techniques of computational biology that we use in this thesis. In the same chapter we also present some algorithms for sequence comparison, the most basic operation in computational biology. In Chapter 4 we study an algorithm that we propose for sequence comparing and which is a modification of the popular algorithm used in BLAST. In Chapter 5 we present the BioAlert system we implemented. Finally, in Chapter 6 we present our conclusions and the future work possibilities.

# Chapter 2

# Related Work

This chapter is a discussion of information dissemination systems (IDS) and a presentation of some representative ones. We distinguish two categories of systems according to the kind of data they handle. In the first category we have systems that handle textual information while in the second category we include systems that handle biological information. In the following sections we present two representative systems from each category.

## 2.1 IDS and Textual information

We present the SIFT and DIAS information dissemination systems explaining in a few words their main characteristics.

### 2.1.1 The SIFT Information Dissemination System

The Stanford Information Filtering Tool (SIFT) is an information dissemination system that was originally presented in [38]. It evolved to become a commercial system and was used for the dissemination of Usenet articles. It was daily serving up to 18400 users and 80000 profiles.

Let us see a short description of how it works. A user submits his queries or profiles via a web interface or via email and receives notifications by email. Since it was based on a client-server model a main server was responsible to store the profile database and to alert the users for the documents matching their profiles. The matching procedure was done under either the Boolean model or under the Vector Space model (VSM) but not under both of them. In order to improve efficiency, SIFT employs various methods using inverted indices while the overall design and implementation targets efficiency and matching speed. An interesting characteristic of SIFT is the incorporation of a *similarity threshold* for the matching procedure under VSM. Thus, users are notified only for those documents whose similarity under VSM is above the specified similarity threshold. Under Boolean model, the matching procedure is straightforward since the similarity threshold is not applied.

Let us give an example for a better understanding. Although SIFT supports both the Boolean and the VSM model for queries, its query language is not as expressive as others [23, 12, 25] since the documents and queries are represented by free text. Under both models the user specifies words or set of words that he wants to be included or to be excluded from the documents he will receive. The following example makes it more understandable.

**Example 1** *Let us consider the Swiss-Prot entry shown in Figure 2.1 to be our document.*

- *A query in SIFT under the Boolean model would be "membrane protein* **not** *homo-sapiens". That query matches our document since the word "membrane" and the word "protein" are included in the document and the word "homo-sapiens" is not included.*

- *The queries under the VSM model are also written in natural language and are represented as vectors, where each entry indicates the importance of the term in the search. For example we have the query:*

$$Q = < (\text{“}meningitidis\text{”}, 0.97), (\text{“}Neisseria\text{”}, 0.9), (\text{“}genome\text{”}, 0.56) >$$

  *In this query, term "meningitidis" has weight 0.97, "Neisseria" has 0.9, "genome" has 0.56 and all others that are not listed have a zero weight. The document is also represented as vector containing the weight assigning to each term. The weight of a term indicates how statistically important it is. In our case part of the vector for our document could be:*

$$D = < 0.65, 0.78, 0, 0.45, \dots >$$

  *In this document the first term has weight 0.65 while the third term is not contained in this document. Apart from the query, the user specifies a similarity threshold. We measure the similarity between the document and the query using a formula which is based on the weights of the corresponding terms and is presented in [38]. The user will be notified only if the computed similarity is above the specified similarity threshold.*

Although SIFT was originally based on a client-server model, distributed versions have also been studied.

### 2.1.2 The Distributed Information Alert System DIAS

DIAS [25] is an information dissemination system that targets, just like Hermes [14], the dissemination of digital library information.

The architecture of DIAS is based on a peer-to-peer network of agents like the one shown in Figure 1.1. An information provider publishes documents through a so called resource agent. A user of the system utilizes an end-agent through which he can connect to the middle agent network and send his profiles or receive notifications about documents. The middle agents of the main P2P network communicate with each other and with resource agents as well as end-agents. A resource agent has the mission to collect the information from an information provider and upload it to the rest of the network. Middle agents are the ones that store the profiles posted by end-agents agents and perform the matching of the profiles and documents posted by resource agents. After the process of matching, middle agents send notifications to the end-agents representing the users that submitted the matching profiles.

DIAS supports a subset of the expressive data language AWP and AWPS presented in [23, 37]. Moreover, specialized filtering algorithms are used to perform matching. Some of them are presented in [37] and some latest ones in [12]. In the following lines we try to explain the the AWP data model and the document schema that are used in DIAS giving examples. According to the AWP data model a document schema is consisted by a set of attribute-value pairs. The attributes are used to encode information such as title, author, description and so on. They differ relatively to the document's context. The value of each attribute is a finite-length string of a vocabulary $V$. The context of our examples comes from biology so the attributes and their value will use terms with biological interpretation.

**Example 2** *Let us consider the Swiss-Prot entry shown in Figure 2.1 to be our document over the set of attributes that are presented in Table 2.1.*

The query language under the AWP schema uses the operator $=$, having the known interpretation and $\sqsupseteq$, with the interpretation "contains".

**Example 3** *The following are some queries in AWP using the schema of document of Example 2:*

$$\begin{gathered} KW = \text{“}Serine\ protease\text{”}, \\ CC \sqsupseteq (Belongs\ \wedge (UPF0324\ \prec_{[0,2]} family\text{”})), \\ (RA \sqsupseteq (Nelson \prec_{[0,4]} Peterson))\wedge \\ (RT = \text{“}Complete\ genome\ of\ Neisseria\text{”}) \end{gathered}$$

The first query is satisfied or matches a document if and only if the KW attribute has the exact value "Serine protease". The second query is satisfied a document is and only if the CC attribute contains the word "Belongs" and the words “UPF0324”, “family” with at least zero and at most 2 words between them. Finally the third query matches a document if and only if the attribute OC contains the names “Nelson” and “Peterson” with either zero or at most 4 words in between and the attribute RT has the exact value “Complete genome of Neisseria”. Thus, the first query is not satisfied by the document of Figure 2.1 while the other two are satisfied by the same document.

A newer system called P2P-DIET that offers more functionality than DIAS has been developed by our group and is available at `http://www.intelligence.tuc.gr/p2pdiet/`. Recent papers on P2P-DIET include [19, 20, 18].

## 2.2 IDS and Biological information

We discuss *Swiss-Shop*, which is an existing system closely related to the one we implemented as a fulfilment of this thesis, and *The Sequence Alerting Server*, which is currently not operational.

### 2.2.1 Swiss-Shop

Swiss-Shop [8] is an alert service that is developed and is hosted by the ExPASy proteomics[1] server of the Swiss Institute of Bioinformatics (SIB) [5]. The purpose of its development is to allow a biologist to automatically obtain new sequence entries of the Swiss-Prot database relevant to his fields of interest before the actual database release.

Swiss-Prot [7] is a protein sequence database which provides a high level of annotation, a minimal level of redundancy and high level of integration with other databases. It is part of UniProt[2] Knowledgebase [9] with TrEMBL database[3] to be the other part. Swiss-Prot database consists of sequence entries which are composed of different line types, each with their own format. Each entry is structured in such a way so as to be usable by both the human readers and computer programs. A sample sequence entry is shown in Figure 2.1. As we see, each line begins with a two-character line code, which indicates the type of data contained in the line, followed by three blanks. The current line types and line codes and the order in which they appear in an entry are shown in Table 2.1. We notice some line types occur many times in a single entry, while others are optional. Each entry must begin with an identification line and end with a terminator line. Moreover, the explanations, descriptions, comments are in ordinary English and wherever needed, there are symbols familiar to molecular

[1]The qualitative and quantitative comparison of proteomes under different conditions to further unravel biological processes [3].

[2]Universal Protein Resource.

[3]A computer-annotated supplement of Swiss-Prot that contains all the translations of EMBL nucleotide sequence entries not yet integrated in Swiss-Prot.

```
ID   YJ79_NEIMB     STANDARD;      PRT;   338 AA.
AC   Q9JXM0;
DT   01-OCT-2004 (Rel. 45, Created)
DT   01-OCT-2004 (Rel. 45, Last sequence update)
DT   01-OCT-2004 (Rel. 45, Last annotation update)
DE   Hypothetical UPF0324 membrane protein NMB1979.
GN   OrderedLocusNames=NMB1979;
OS   Neisseria meningitidis (serogroup B).
OC   Bacteria; Proteobacteria; Betaproteobacteria; Neisseriales;
OC   Neisseriaceae; Neisseria.
OX   NCBI_TaxID=491;
RN   [1]
RP   SEQUENCE FROM N.A.
RC   STRAIN=MC58 / Serogroup B;
RX   MEDLINE=20175755; PubMed=10710307;
RA   Tettelin H., Saunders N.J., Heidelberg J., Jeffries A.C., Nelson K.E.,
RA   Eisen J.A., Ketchum K.A., Hood D.W., Peden J.F., Dodson R.J.,
RA   Nelson W.C., Gwinn M.L., DeBoy R., Peterson J.D., Hickey E.K.,
RA   Haft D.H., Salzberg S.L., White O., Fleischmann R.D., Dougherty B.A.,
RA   Mason T., Ciecko A., Parksey D.S., Blair E., Cittone H., Clark E.B.,
RA   Cotton M.D., Utterback T.R., Khouri H., Qin H., Vamathevan J.,
RA   Gill J., Scarlato V., Masignani V., Pizza M., Grandi G., Sun L.,
RA   Smith H.O., Fraser C.M., Moxon E.R., Rappuoli R., Venter J.C.;
RT   "Complete genome sequence of Neisseria meningitidis serogroup B strain
RT   MC58.";
RL   Science 287:1809-1815(2000).
CC   -!- SUBCELLULAR LOCATION: Integral membrane protein (Potential).
CC   -!- SIMILARITY: Belongs to the UPF0324 family.
CC   -----------------------------------------------------------------------
CC   This SWISS-PROT entry is copyright. It is produced through a collaboration
CC   between  the Swiss Institute of Bioinformatics  and the  EMBL outstation -
CC   the European Bioinformatics Institute.  There are no  restrictions on  its
CC   use  by  non-profit  institutions as long  as its content  is  in  no  way
CC   modified and this statement is not removed.  Usage  by  and for commercial
CC   entities requires a license agreement (See http://www.isb-sib.ch/announce/
CC   or send an email to license@isb-sib.ch).
CC   -----------------------------------------------------------------------
DR   EMBL; AE002546; AAF42307.1; -.
DR   PIR; D81020; D81020.
DR   TIGR; NMB1979; -.
DR   InterPro; IPR004630; Cons_hypoth698.
DR   Pfam; PF03601; Cons_hypoth698; 1.
DR   TIGRFAMs; TIGR00698; Cons_hypoth698; 1.
KW   Complete proteome; Hypothetical protein; Transmembrane.
FT   TRANSMEM      5     23       Potential.
FT   TRANSMEM     33     55       Potential.
FT   TRANSMEM     62     84       Potential.
FT   TRANSMEM     94    116       Potential.
FT   TRANSMEM    123    145       Potential.
FT   TRANSMEM    155    177       Potential.
FT   TRANSMEM    222    239       Potential.
FT   TRANSMEM    254    273       Potential.
FT   TRANSMEM    280    302       Potential.
FT   TRANSMEM    312    334       Potential.
SQ   SEQUENCE   338 AA;  36806 MW;  BC160B75713BEEA5 CRC64;
     MNTRPFYFGL IFIAIIAILA NYLGNTDFSH HYHISALIIA ILLGMAIGNT IYPQFSTQVE
     KGVLFAKGAL LRTGIVLYGF RLTFGDIADV GLNAVVTDAI MLISTFFFTA LLGIRYLKMD
     KQLVYLTGAG CSICGAAAVM AAEPVTKAES HKVSVAIAVV VIFGTLAIFT YPLFYTWSQH
     LINAHQFGIY VGSSVHEVAQ VYAIGENIDP IVANTAVISK MIRVMMLAPF LLMLSWLLTR
     SNGVSENTSH KITIPWFAVL FIGVAIFNSF DLLPKELVKL FVEIDSFLLI SSMAALGLTT
     QASAIKKAGL KPFVLGILTY LWLVVGGFLV NYGISKLI
//
```

Figure 2.1: A Swiss-Prot entry

| **Line code** | **Content** | **Occurrence in an entry** |
|---|---|---|
| ID | Identification | Once; starts the entry |
| AC | Accession number(s) | Once or more |
| DT | Date | Three times |
| DE | Description | Once or more |
| GN | Gene name(s) | Optional |
| OS | Organism species | Once or more |
| OG | Organelle | Optional |
| OC | Organism classification | Once or more |
| OX | Taxonomy cross-reference(s) | Once or more |
| RN | Reference number | Once or more |
| RP | Reference position | Once or more |
| RC | Reference comment(s) | Optional |
| RX | Reference cross-reference(s) | Optional |
| RG | Reference group | Once or more (Optional if RA line) |
| RA | Reference authors | Once or more (Optional if RG line) |
| RT | Reference title | Optional |
| RL | Reference location | Once or more |
| CC | Comments or notes | Optional |
| DR | Database cross-references | Optional |
| KW | Keywords | Optional |
| FT | Feature table data | Optional |
| SQ | Sequence header | Once |
| (blanks) | Sequence data | Once or more |
| // | Termination line | Once; ends the entry |

Table 2.1: Line types and codes of a Swiss-Prot entry

biologists. Finally, we must refer that for standarization purposes the format of Swiss-Prot follows as closely as possible that of the EMBL Nucleotide Sequence Database.

Swiss-Shop service performs searches on the current biweekly updates of Swiss-Prot. The searches can be either sequence/pattern-based or keyword-based.

### Keyword-based search

A *keyword-based search* searches the database entries for the user defined keywords on the fields with the line codes AC, OS, OG, OC, RA, DE, CC, KW, DR and FT of Swiss-Prot. The user specifies the fields that are interesting to him and give the keywords for each of them. An entry satisfies the keyword-based search if and only if that entry contains the user-specified keywords of all the fields, meaning that the fields are conjunctive to each other. Moreover, the user may define more than one keywords for one filed. These keywords are interpreted as a disjunction. We now give an example.

**Example 4** *Let us consider the database entry that is shown in Figure 2.1 and the following keyword-based searches:*

$$\begin{gathered} RA \sqsupseteq \text{“}Venter\text{”}, \\ (DR \sqsupseteq \text{“}PIR\text{”}) \wedge (OS \sqsupseteq \text{“}meningitidis\text{”}), \\ (DR \sqsupseteq (\text{“}PIR\text{”} \vee \text{“}TIGR\text{”}) \wedge (AC = \text{“}Q9JXMO\text{”}), \\ (DR \sqsupseteq \text{“}InterPro\text{”}) \wedge (OC \sqsupseteq \text{“}Bacteria\text{”}) \end{gathered}$$

*We use the operators* equals (=) *and* contains ($\sqsupseteq$) *with the same interpretation as in the AWP model of DIAS. The first three searches are satisfied by that entry while the last one is not because the keyword “Bacteria” is not contained to the field OC.*

### Sequence/pattern-based search

A *sequence* or *pattern-based search* is also supported. The new protein sequences of the biweekly update are searched against the sequence or the pattern which the user provides. The search is either a BLAST search or a scan with the provided pattern. Instead of the raw protein sequence itself you can enter the Swiss-Prot identification (ID) or the Swiss-Prot accession number (AC) of the sequence if it has been submitted to Swiss-Prot. Similarly, instead of your own pattern in PROSITE[4] format you can provide the AC number or the PROSITE ID. In both cases, the user receives a list of the new sequences.

Swiss-Shop informs the users by sending an email to them. The email is sent either every time a search is run even if there are no Swiss-Prot entries matching the query or only when there are matching the query entries. The user has also the option to set the date until when his query will be active. Moreover, the service provides the users with the capability to set the degree of detail that they want to receive the results. Thus, they can receive the complete Swiss-prot entries or a short report containing information from some Swiss-Prot lines or just a list of Swiss-prot accession numbers.

## 2.2.2 The Sequence Alerting Server

The Sequence Alerting Server [16] is an alerting server with a WWW interface which was developed by EMBL. It informs users with query sequences in database searches about new entries in protein databases related to their query.

[4]PROSITE [6] is a database of protein families and domains. It consists of biologically significant sites, patterns and profiles that help to reliably identify to which known protein family a new sequence belongs.

Let us describe how it works. The server regularly compares users' subscribed sequences to the new entries in all the public protein databases by using BLAST programs, as they are shown in Table 3.5. The non-redundant protein database, called *nrdb*, is used by the server against to which the users' queries are searched. Since *nrdb* database contains every publicly available protein sequence once, the user does not worry for the multiple appearances of the same sequence in the different protein databases. The *nrdb* is built up every day at EMBL from the SWISSPROT, PDB, TREMBL, PIR, GENPEPT and WORMPEPT databases. Users submit their query sequences to the server via a WWW interface and these are searched daily against a new subset of *nrdb*, so as users to be kept informed about new entries in protein databases related to their queries.

This service is a reliable tool which is user friendly and does not require any software to be installed locally except a web browser. Its main advantage is that the searches are performed against the most complete database *nrdb*. It can be found in the web address `http://www.bork.embl-heidelberg.de/alerting/` but it is currently not working and the users are suggested to use the *Swiss-Shop* service.

## 2.3 Summary

In this chapter we presented four different information dissemination systems. Two of them were designed to handle textual data while the other two are alert services focused on molecular biologists' interests. An extensive description was given to the *Swiss-Shop* service from which our system, *BioAlert*, was inspired. We started with general information about this service and continued with a description of the functionalities it provides through a web interface. We also discussed the format of the entries of *Swiss-Prot* database and focused on the two different types of search that are supported.

# Chapter 3

# Querying Sequence Databases

This chapter is an introduction to some basic ideas of computational biology as they will be explored in this thesis. The chapter is important to be understood by readers that have no biological background. At first, we present the most basic operation in computational biology and our work, *sequence comparison*, while the rest of the chapter deals specifically with searching sequence databases.

We explain sequence comparison in much detail, giving definitions, examples and the basic algorithms. We focus on querying sequence databases that is an important issue in this thesis and present two most popular algorithms that are used by biologists all over the world.

## 3.1 Sequence Comparison

The major operation in computational biology is sequence comparison. It is the most important manipulation and is the basis for many other, more complex manipulations, such as large DNA sequencing. We can define it informally as finding which parts of the sequences are alike and which parts differ. Regarding sequence comparison, we deal with two terms that are often used in computational biology. One is sequence *similarity* and the other sequence *alignment*. The first one gives a measure of how similar the sequences are while the second one is a way of placing one sequence above the other in order to make clear correspondence between similar characters or substrings from the sequences [34].

We start presenting some basic and formal definitions for sequence comparison in computational biology. Our definitions are from the book [15] (in some cases verbatim!).

An *alphabet* is a finite set of letters. An alphabet will usually be denoted by the letter $\Sigma$ possibly with a subscript. In this section we will deal with two alphabets: $\Sigma_{DNA}$ and $\Sigma_{amino}$. $\Sigma_{DNA}$ is a set of four letters $(A, T, C, G)$ representing the kinds of bases found in DNA molecules. Table 3.1 gives the names of these bases. $\Sigma_{amino}$ is a set of twenty letters representing the twenty

| **Letter code** | **Name** |
|---|---|
| A | Adenine |
| T | Thymine |
| C | Cytosine |
| G | Guanine |

Table 3.1: The four kinds of of bases found in DNA molecules

amino acids typically found in proteins. The elements of set $\Sigma_{amino}$ are given in Table 3.2 together with their 3-letter codes and names.

Let us now define the concept of sequence which is fundamental in our work.

**Definition 1** *A* sequence *s of length n over alphabet* $\Sigma$ *is a total function s:* $\{1,2,...,n\} \rightarrow \Sigma$. *A* DNA sequence *is a sequence from the alphabet* $\Sigma_{DNA}$. *A* protein sequence *is a sequence from the alphabet* $\Sigma_{amino}$.

In other words, a sequence is an ordered succession of characters drawn from an alphabet $\Sigma$. For each sequence s, $s(i)$ denotes the $i$-th character of s while its length will be denoted by $|s|$. If the length of a sequence is 0, then we say that we have an *empty sequence* that will be denoted by $\varepsilon$.

**Example 5** *The succession of nucleotides*

$$GATCGGAATAG$$

*is a DNA sequence of length 11 with s(1)=G, s(2)=A, s(3)=T, and so on while this succession of amino acids*

$$GDVEKGKKIFVQKCAQCHTV$$

*represents a protein sequence of length 20.*

According to the definition and the examples a sequence is actually a string. Thus, these two terms are often used as synonyms in biological literature. But although, the sequence and the string are synonyms, the terms *substring* and *subsequence* have different meanings. The following definitions and the example make the difference clear.

**Definition 2** *Let s be a string over an alphabet* $\Sigma$. *A* substring *t of s is a string s[i..j] that starts at position i and ends at position j of s.*

**Definition 3** *Let s be a sequence of length n over an alphabet* $\Sigma$. *A* subsequence *t of s is specified by a list of indices* $i_1 < i_2 < i_3 < ... < i_k$, *for some* $k \leq n$. *The subsequence specified by this list of indices is the sequence* $s(i_1)s(i_2)s(i_3)...s(i_k)$.

| **One letter code** | **Three-letter code** | **Name** |
|---|---|---|
| A | Ala | Alanine |
| C | Cys | Cysteine |
| D | Asp | Aspartic Acid |
| E | Glu | Glutamic Acid |
| F | Phe | Phenylalanine |
| G | Gly | Glycine |
| H | His | Histidine |
| I | Ile | Isoleucine |
| K | Lys | Lysine |
| L | Leu | Leucine |
| M | Met | Methionine |
| N | Asn | Asparagine |
| P | Pro | Proline |
| Q | Gln | Glutamine |
| R | Arg | Arginine |
| S | Ser | Serine |
| T | Thr | Threonine |
| V | Val | Valine |
| W | Trp | Tryptophan |
| Y | Tyr | Tyrosine |

Table 3.2: The twenty amino acids commonly found in proteins

In other words, a subsequence can be obtained from sequence $s$ by removal of some characters. So, a subsequence need not consist of contiguous characters of sequence $s$ whereas a substring of string $s$ is formed by consecutive characters of $s$. Thus, a substring is always a subsequence, but not all subsequences of a sequence $s$ are substrings of $s$. Moreover, one sequence has many subsequences while the empty sequence $\varepsilon$ is a subsequence of every sequence. Let us see an example.

**Example 6** *The succession of nucleotides*

$$CGAGTTCCCCGAGGTGTACG$$

*represents a DNA sequence $s$ of length 20.*

$CGAGTTCCCCGGGGG$ *is a subsequence of the previous sequence*

*while*

$CGAGTTCCCCGAG$ *is both a substring and a subsequence of $s$.*

Let us now define the concepts of *sequence alignment* and *sequence similarity* which are fundamental in computational biology and this work.

**Definition 4** *Let $s_1$ and $s_2$ be two sequences. An* alignment *$\alpha$ between $s_1$ and $s_2$ is obtained by first inserting spaces (or gaps) in chosen positions, either into or at the ends of $s_1$ and $s_2$ so that they become equal in size, and then placing the two resulting sequences one above the other so that every character in either sequence is opposite a unique character or a unique space in the other sequence and every space in either sequence is opposite a unique character in the other sequence. A* global alignment *is an alignment that involves the entire sequences. A* local alignment *is an alignment that involves substrings of the sequences.*

In other words, "it is a way of placing one sequence above the other in order to make clear correspondence between similar characters or substrings from the sequences"[34]. The alignment between two sequences makes sense only when the two sequences have the same length, otherwise you can not have a correspondence between similar characters or not. Due to the different length of sequences, we insert spaces in arbitrary locations along the sequences so that they end up with the same size. Spaces can be inserted in both the beginning and the end of the sequences. The restriction, according to definition 4, is that each space in one sequence must be aligned with a character in the other and not with a space. Let us see an example.

**Example 7** *The following are global alignments between two sequences.*

*For the two DNA sequences $GACGGATTAG$ and $GATCGGAATAG$ an alignment is*

*GA-CGGATTAG*
*GATCGGAATAG*

*For the DNA sequences GCTGAACG and CTATAATC an alignment is*

*GCTGA-A–CG*
*–CT-ATAATC*

*a second one is*

*- - - - - - GCTGAACG*
*CTATAATC - - - - - -*

*and another one is*

*GCTG-AA-CG*
*-CTATAATC-*

**Definition 5** *Let $s_1$, $s_2$ be two sequences over an alphabet $\Sigma$, and $\Sigma'$ be $\Sigma$ with the added character '-' denoting a space. Then, for any two characters x, y in $\Sigma'$, s(x,y) denotes the* score *(or* value*) obtained by aligning character x against character y.*

**Definition 6** *Let $\alpha$ be an alignment between two sequences $s_1$, $s_2$. Now, let $s'_1$, $s'_2$ denote the sequences after the chosen insertion of spaces, and l the equal length of the two sequences $s'_1$, $s'_2$ in $\alpha$. The* score *(*value*) of alignment $\alpha$ is defined as $\sum_{i=1}^{l} s(s'_1(i), s'_2(i))$.*

According to the above definition, the score of an alignment $\alpha$ is obtained by summing the score contributed by each pair. The score of each pair is defined beforehand in scoring matrices.

**Definition 7** *A* pairwise scoring matrix *over an alphabet $\Sigma$, is a matrix that defines the pairwise scores between the characters of the alphabet $\Sigma$.*

Let us now see an example that illustrates the computation of the score of an alignment between two sequences.

**Example 8** *We consider the DNA alphabet $\Sigma_{DNA}$ and let the pairwise scores over this alphabet be defined in the following matrix:*

| s | A | T | C | G | - |
|---|---|---|---|---|---|
| A | 1 | -1 | -1 | -1 | -2 |
| T | -1 | 1 | -1 | -1 | -2 |
| C | -1 | -1 | 1 | -1 | -2 |
| G | -1 | -1 | -1 | 1 | -2 |
| - | -2 | -2 | -2 | -2 | -2 |

*Now, we compute the score for each of the four alignments of Example 7 and we have:*
*1+1+(-2)+1+1+1+1+(-1)+1+1+1 = 6*
*-2+1+1+(-2)+1+(-2)+1+(-2)+(-1)+(-1) = -6*
*-2+(-2)+(-2)+(-2)+(-2)+(-2)+(-1)+1+(-2)+(-2)+(-2)+(-2)+(-2)+(-2) = -24*
*-2+1+1+(-1)+(-2)+1+1+(-2)+1+(-2) = -4*

The scoring matrix used in this example and many others penalize mismatches and inserted spaces by assigning a negative value to them while assign greater than or equal to zero values to characters of the alphabet that match. In this way, they emphasize the matches. So, using such scoring systems, the alignment with as large a score as possible, has as many characters as possible that match. Consequently, the similarity between the two sequences is high enough. This is a way to measure the similarity between the two sequences. We continue with the formal definition of the sequence similarity.

**Definition 8** *Given a pairwise scoring matrix over the alphabet $\Sigma'$, the* global similarity *of two sequences $s_1$ and $s_2$ is defined as the score of the alignment $\alpha$ of $s_1$ and $s_2$ that is maximum over the space of all possible alignments. This is also called the* optimal global alignment score *of $s_1$ and $s_2$.*

**Definition 9** *Given two sequences $s_1$ and $s_2$ and a scoring scheme $\rho$, the* optimal global alignment *between $s_1$ and $s_2$ is the alignment with the optimal global alignment score over the scoring scheme $\rho$.*

It is possible that more than one alignment has the same maximum score. So, the optimal alignment between two sequences may not be unique. Moreover, the optimal alignment is not the same for different scoring schemes. Each scoring scheme gives a different ranking of the alignments, depending on how match scores compare to mismatch and space scores. The following example illustrates how the score of an alignment depends on the used scoring scheme.

**Example 9** *We assume the following scoring scheme:*

| *s* | *A* | *T* | *C* | *G* | - |
|---|---|---|---|---|---|
| *A* | *5* | *-1* | *-3* | *-3* | *0* |
| *T* | *-1* | *5* | *-3* | *-3* | *0* |
| *C* | *-3* | *-3* | *5* | *-1* | *0* |
| *G* | *-3* | *-3* | *-1* | *5* | *0* |
| - | *0* | *0* | *0* | *0* | *-5* |

*Now, let us compute the score of the different alignments of the DNA sequences of Example 7 and compare it with the score that was computed using* *another scoring scheme in Example 8.*

*The score for the first alignment is 0+5+5+0+5+0+5+0+(-3)+(-1) = 16 while it was -6. For the second one it is 0+0+0+0+0+0+(-3)+5+0+0+0+0+0+0 = 2 while it was -24. For the third one it is 0+5+5+(-3)+0+5+5+0+5+0 = 22 while it was -4. The last alignment has the maximum total score in both cases and this is the optimal alignment for the two sequences. So, the two sequences have similarity of -4 over the first scoring scheme and 22 over the second one. The second scoring scheme resulted to a higher value because the score of matches and the inserted spaces is higher in this scoring scheme than in the other, 5 and 0 is greater than 1 and -2 respectively, although the mismatches contribute to the sum with a greater negative value.*

Many scoring matrices have been suggested for both protein and DNA sequences [11, 17, 33]. Which scoring scheme is used depends on the application.

## 3.2 Sequence Comparison and Molecular Biology

In this section we discuss the importance of sequence comparison in molecular biology and compare the global to local alignment.

In the first subsection we explain why sequence comparison is a fundamental process in molecular biology by presenting the first fact of biological sequence analysis and several examples as well. In the second subsection we compare the two kinds of alignment and since each kind is suitable for different applications, we present some examples that illustrate this difference.

### 3.2.1 A fundamental process in molecular biology

Sequence comparison is fundamental in molecular biology because in biomolecular sequences (DNA, RNA, amino acid sequences), high sequence similarity usually implies significant functional or structural similarity. This is called the *first fact of biological sequence analysis* [15].

According to the first fact of biology, comparing sequences and finding high similarity implies that there is functional similarity between them and if one of them is a sequence with identified functionality then we extract precious information for the other sequence. This is crucial in molecular biology and genetics since life is based on duplication and modification of molecular structures. The same or related molecular structures just like proteins, DNA regulatory sequences and others, are shared among the organisms. Not only are the molecular structures shared among the organisms but this is so common that there is enormous redundancy and as Doolittle says in [32] “The vast majority of extant proteins are the result of a continuous series of genetic duplications and subsequent modifications. As a result, redundancy is a built-in characteristic of protein sequences, and we should not be surprised that so many new sequences resemble already known sequences." Thus, it is not surprising that the same genes that work in flies work in humans as well. So, the genomes of two very different organisms might have regions that are quite similar. Consequently, due to the first fact of biological analysis and the duplication and modification of molecular structures among the species, sequence comparison is extremely indispensable for biologists. it is essential to compare a new sequence against all the existing sequences in search of similarities.

Let us, now, describe the standard practice that is globally applied by the biologists when they study the genome of an organism that have not been studied yet. At the first step, the gene is cloned and sequenced. After its DNA sequence is obtained, it is translated into an amino acid sequence. In the next step, they search for similarities between the translated protein sequence and a protein database. The resulted sequence similarities imply functional or structural similarities, thus the researchers gain useful knowledge that guides their work afterwards.

Let us now present a simple categorization of some common problems in sequence comparison and their correspondence in the computational biology field, as they are presented in [34].

1. Let us say that we have two sequences over the same alphabet, both about the same length (tens of thousands of characters). We know that the sequences are almost equal, with only a few isolated differences such as insertions, deletions, and substitutions of characters. The average frequency of these differences is low, say, one each hundred characters. We want to find the places where the differences occur.

   This problem may appear when the same gene is sequenced by two different labs and they want to compare the results, or when we are looking for typing errors.

2. We have two sequences over the same alphabet with a few hundred characters each. We want to know whether there is a prefix of one which is similar to a suffix of the other. If the answer is yes, the prefix and the suffix involved must be produced. Another case of the same problem is to have several hundred sequences that must be compared each one against all). In addition, we know that the great majority of sequence pairs are unrelated, that is, they will not have the required degree of similarity.

   These kind of problems appear in the context of fragment assembly in programs to help large-scale DNA sequencing.

3. We have two sequences over the same alphabet with a few hundred characters each. We want to know whether there are two substrings, one from

each sequence, that are similar. Another case of the same problem is to have one sequence that must be compared to thousands of others, instead of only two sequences.

This problem occurs in the context of searches for local similarities using large sequence databases.

To be more specific, we give a real case example from [15]. Considering the above three cases of problems, this belongs to the third one. It actually proves why database search is so essential for biologists. The problem is referred to as the first complete DNA sequence of a free-living organism [31]. The sequencing revealed a total of 1734 coding regions. Each of those sequences was translated into one or more amino acid sequences and used to search for similar sequences in the protein sequence database Swiss-Prot. The search resulted to high similarity for 1007 of the genes. The match was so unambiguous that the specific biochemical function deduced for each one. These deductions were made possible by linking Swiss-Prot, the database under search, to Riley database where protein sequences are stored divided up into 102 biochemical roles. So, the biochemical function for 1007 of 1734 genes was specified via comparisons between the derived protein sequences and the protein sequences of a database whose biochemical roles were already specified in detail.

### 3.2.2 Global vs. Local Alignment

As it was discussed in the previous section, duplications and modifications are common in proteins and DNA sequences. There are many regions that are quite similar and others that are repeated not only in the genome of a distinct organism but also among the organisms. Thus, organisms share pieces of the same information while they modify others through evolution. So, when comparing two related biosequences (let us say two proteins that belong to the same family) we find that the alignment between the entire sequences is very poor although there are regions that are very similar. In other words, while there may be little global similarity between related sequences, there are usually strong local similarities. So, there are cases where local similarity is meaningful whereas global similarity is even misleading. For example, let us consider long DNA sequences. When we want to compare long fragments of DNA sequences that are unknown, global similarity will give misleading results since only some internal sections may be related. Local alignment is appropriate in locating these sections. Another application that illustrates the meaningfulness of global alignments is the comparison of proteins which belong to very different protein families. Proteins are made up of the same structural or functional subunits (*motifs*). These subunits are common in proteins even if the proteins belong to different families. Searching for these subunits local alignment must be used since global alignment involves the entire length of the sequences being compared. An example is the proteins that are encoded by *homeobox* genes[1]. These genes show up in a wide variety of species, from frogs to humans and flies. The proteins that they encode are very different in species except a region, called the *homeodomain*, where there is identity in alignments up to 95%. Local alignment is also useful in detection of conserved characters of related proteins. A global alignment is not helpful since we are searching only some particular isolated characters in high similarity and not the entire sequence. An example is the family of *serine proteases* where a few isolated, conserved amino acids characterize the family [15].

Although local alignment is the most appropriate method for comparing proteins from different protein families, in [39], is pointed out that there is extensive global similarity between protein sequences that are related by strong local similarity. Thus, global alignment can be used in protein comparisons as well as in exposing important biological commonalities [26]. For example, the protein *cytochrome c* has almost the same length in most organisms that produce it. So, a relationship between two cytochromes from any two different species over the entire length is expected. Another case that illustrates the effectiveness of global alignments is when we are trying to deduce evolutionary history by examining protein sequence similarities and differences. In this case again, proteins of the same family are compared and differences or similarities in their entire length must be located so an alignment that involves their entire length is required thus the global alignment is the appropriate method to be used.

## 3.3 Algorithms for Sequence Alignment

Let us now discuss some algorithms for sequence alignment. The simplest way to compute the similarity between two sequences is to generate all the possible alignments between the two sequences and to choose the one with the highest score. The best alignment will also have been found. This idea results in an algorithm that is so slow that is forbidden for implementation. Another idea is to use *dynamic programming*. We are going to present some basic points of this technique, that is widely applicable in computational biological problems. Much more details you can find in many places e.g., [27].

Dynamic programming is an algorithmic technique that has been used in many algorithms that solve optimization problems. It finds the best solution by breaking the original problem into smaller instances of the same problem (subproblems). It solves all the subproblems storing at the same time the intermediate solutions in a table and then it chooses the sequence of the best solutions among all the solutions from the table.

Let us now see how dynamic programming is used to obtain optimal alignment between two sequences. Firstly, we consider the case of global alignment.

[1]http://homeobox.biosci.ki.se/

| j | | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|---|
| i | | | C | A | T | G | T |
| 0 | | 0 | -1 | -2 | -3 | -4 | -5 |
| 1 | A | -1 | -1 | 1 | 0 | -1 | -2 |
| 2 | C | -2 | 1 | 0 | 0 | -1 | -2 |
| 3 | G | -3 | 0 | | | | |
| 4 | C | -4 | | | | | |
| 5 | T | -5 | | | | | |
| 6 | G | -6 | | | | | |

Figure 3.1: Snapshot of computing the dynamic programming table for two sequences. A match costs +2 whereas a gap and a mismatch cost -1.

**Global Sequence Alignment** Let $S = s_1 ... s_n$ and $T = t_1 ... t_m$ be two sequences with length $n$ and $m$, respectively. We want to find an optimal global alignment between them. We recall that $s(x, y)$ is the score of the alignment of character $x$ with character $y$ according to the Definition 5, while $s(x, -)$ and $s(-, y)$ are the scores of the alignment of one character of each sequence with a space from the other. We will also use $V(i, j)$ to denote the optimal alignment score of prefixes $S[1...i]$ and $T[1...j]$ ($0 \le i \le n$, $0 \le j \le m$).

Dynamic programming has three steps: the recurrence relation, the tabular computation and the traceback. We will explain each one in turn.

1. The recurrence relation is a recursive relationship between the value of *V(i,j)* and values of V with index pairs smaller than *i, j*. So, for the score $V$ of the optimal global alignment we have the recurrence relation:

$$V(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ and } j = 0 \\ V(i-1,j) + s(S(i), -) & \text{if } i > 0 \text{ and } j = 0 \\ V(i, j-1) + s(-, T(j)) & \text{if } i = 0 \text{ and } j > 0 \\ \max \begin{cases} V(i-1, j-1) + s(S(i), T(j)) \\ V(i-1, j) + s(S(i), -) \\ V(i, j-1) + s(-, T(j)) \end{cases} & \text{if } i > 0 \text{ and } j > 0 \end{cases} \tag{3.1}$$

Considering the optimal alignment of $S_{1...i}$ and $T_{1...j}$, the recurrence relation describes the three possibilities that there are in any alignment:

- *Aligning $S_i$ with $T_j$.* In this case, the score is equal to the score of aligning $i-1$ characters of S with $j-1$ characters of $T$ plus the score $s(S_i, T_j)$ of aligning $S_i$ with $T_j$. Thus, the score is *V(i-1,j-1)+s($S_i$,$T_j$)*.
- *Aligning $S_i$ with a space character in sequence* T. In this case, the score is equal to the score of aligning the $i-1$ characters of $S$ with $j$ characters of $T$ plus the score *s($S_i$,-)* of aligning $S_i$ with a space. Thus, the score is *V(i-1,j)+s($S_i$,-)*.

| j | | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|---|
| i | | | C | A | T | G | T |
| 0 | | 0 | ←-1 | -2 | -3 | -4 | -5 |
| 1 | A | ↑-1 | -1 | ↖1 | 0 | -1 | -2 |
| 2 | C | -2 | ↖ 1 | 0 | ↖ 0 | -1 | -2 |
| 3 | G | -3 | ↑ 0 | ↖ 0 | -1 | 2 | 1 |
| 4 | C | -4 | -1 | ↖ -1 ↑ | -1 | ↑ 1 | 1 |
| 5 | T | -5 | -2 | -2 | ↖1 | 0 | ↖ 3 |
| 6 | G | -6 | -3 | -3 | 0 | ↖ 3 | ← 2↑ |

There are three possible paths that represent alignments that give the highest score.

These are:

1. -C-ATGT
   ACGCTG-
   with score: -1+2-1-1+2+2-1 = 2

2. -CA-TGT
   ACGCTG-
   with score: -1+2-1-1+2+2-1 = 2

3. CATG-T-
   -ACGCTG
   with score: -1+2-1+2-1+2-1 = 2

Figure 3.2: Computing the alignment

   - *Aligning $T_j$ with a space character in sequence* S. This case is similar to the previous one. Thus, the score is $V(i,j-1)+s(-,T_j)$.

2. The second step of dynamic programming is to use the recurrence relations to compute the score of the optimal alignment. This is done using a tabular bottom-up computation. According to this, we compute the score *V(i,j)* for all possible values of *i* and *j*. We start from smaller values of *i*, *j* and increase them. We store these values in a dynamic programming table of size *(n+1)*(m+1)*. Figure 3.1 depicts the snapshot of the dynamic programming table for the sequences S=ACGCTG and T=CATGT. The score of the optimal global alignment will be the value *V(n,m)*, since the entire length of the sequences is involved.

3. Once the values have been computed, the alignment has to be extracted and this is the third step. In order to obtain the alignment, we follow the pointers that have been established in the cells of the table as values are computed. When the value of the cell $(i, j)$ is computed, a pointer is set for this cell that points to cell that its value was used to the computation of the value of the current cell. It is possible more than one pointer (and up to three, since three are the possibilities) to be set from a cell. If we follow the path of pointers from cell *(n,m)* to *(0,0)*, then we will get the optimal global alignment. Figure 3.2 depicts the dynamic programming table of Figure 3.1. It is totally filled in with the values and the pointers have been set. As you can notice, there are three ways to reach cell *(0,0)* starting from cell *(n,m)* and following the pointers. Thus, there are three paths, each of which represents an alignment that gives the highest score. These three alignments and their score are depicted in the same figure. One can notice that each horizontal and vertical edge in the path specifies a space inserted into one of the sequences while a diagonal edge specifies either a match or a mismatch.

Except tables, weighted graphs are useful to represent dynamic programming solutions. They are called *alignment graphs* and they are weighted directed acyclic graphs. Each cell of the dynamic programming table corresponds to a node of the directed graph. The weight of each edge is the score computed applying the recurrence relation 3.1, as depicted in Figure 3.3. The alignment is again a path from node *(0,0)* to node *(n,m)* while the alignment score is the total weight of the edges along the path. The goal is to find the path from node *(0,0)* to node *(n,m)* with the highest weight (heaviest path).

In Figures 3.4 and 3.5, the dynamic programming algorithm is presented for the global and local alignment problem, respectively. In both cases it is taken from [40].

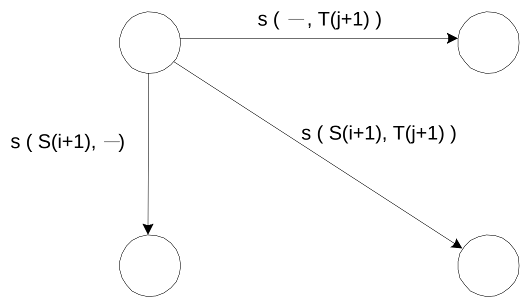


Figure 3.3: A segment of the alignment graph

**Local Sequence Alignment** In local alignment the task is to find and extract a pair of regions one from each of the two sequences, that exhibit high similarity. So, we search for alignment that begin and end at any position of the sequences. A formal definition of the problem follows.

**Definition 10** *Given two sequences S and T, find substrings a and b of S and T, respectively, whose similarity (optimal global alignment value) is maximum over all pairs of substrings from S and T. The value of an optimal local alignment is denoted by V'. [15]*

Comparing the pseudocode of the two algorithms of Figures 3.4 and 3.5, we notice that there are small differences between them although they solve different problems. The recurrence relation is modified to contain a zero-term and whenever the score becomes negative, it is initialized to zero. This allows the alignments to start at any position of the sequences when the score becomes negative. In local alignment the common scoring systems mark negatively the mismatches and the gaps whereas the matches are given positive value. Thus, a positive score means more matches and consequently higher local similarity. Since the local alignment can end at any position, as the values are computed and saved in the table the best score, that is actually the highest, is saved at each step. At the end when the table is filled up, the best score is reported as the local similarity score. The cell that corresponds to best score, let say $(i_2, j_2)$, is the one at which the local alignment ends. The local alignment of the two sequences is found by backtracing the pointers, that are set again, from the cell $(i_2, j_2)$ until a cell $(i_1, j_1)$ is reached that has score zero.

**Time Complexity** Let us see now how much time these dynamic programming algorithms take. The time required in both kind of alignments is the same. The dynamic programming table for two sequences of length n and m can be constructed in $O(nm)$ time, in both cases. The $V(i, j)$ needs four comparisons and three arithmetic operations per cell to be computed. Hence, each cell needs a constant number of comparisons and arithmetic operations to be filled in and

```
$V(0,0) \leftarrow 0$
**for** $j \leftarrow 1$ **to** m **do**
    $V(0,j) \leftarrow V(0,j-1) + s(-,T(j))$
**for** $i \leftarrow 1$ **to** n **do** {
    $V(i,0) \leftarrow V(i-1,0) + s(S(i),-)$
    **for** $j \leftarrow 1$ **to** m **do**
        $V(i,j) \leftarrow \max[V(i-1,j-1) + s(S(i),T(j)), V(i-1,j)+$
        $s(S(i),-), V(i,j-1) + s(-,T(j))]$
}
**write** 'Global similarity score is' *V(n,m)*
```

Figure 3.4: Pseudo-code for dynamic programming algorithms for computing global alignments

```
$best \leftarrow 0$
**for** $j \leftarrow 1$ **to** m **do**
    $V(0,j) \leftarrow 0$
**for** $i \leftarrow 1$ **to** n **do** {
    $V(i,0) \leftarrow 0$
    **for** $j \leftarrow 1$ **to** m **do**
        $V(i,j) \leftarrow \max[0, V(i-1,j-1) + s(S(i),T(j)), V(i-1,j)+$
        $s(S(i),-), V(i,j-1) + s(-,T(j))]$
        $best \leftarrow \max(V(i,j), best)$
    }
}
**write** 'Local similarity score is' *best*
```

Figure 3.5: Pseudo-code for dynamic programming algorithms for computing local alignments

since there are $O(nm)$ cells in the table the total time required is $O(nm)$. Once the table has been computed, the search for the optimal alignment can also be found in $O(nm)$ time, in both cases.

## 3.4 Efficient Algorithms for Sequence Comparison in Large Sequence Databanks

The sequencing of nucleic acids and proteins has started for decades but only in the last years, that the technology made it possible, large-scale sequencing started to take place. Up to now, the genome of many organisms has been sequenced and this remains to be done for many others. Among the results that this great scale of sequencing caused is the creation of large centralized databanks. The importance of sequence similarity searching that is concluded by the first fact of biological sequence analysis, that is explained in Section 3.2.1, in combination with the existence of these databanks led to the need of efficient algorithms for querying these data banks that store large quantities of genomic data produced by labs all over the world. In this section we give a categorization of the existing databases and right after this we present the most popular program for database search giving details of its algorithm.

### 3.4.1 Publicly available sequence databases

We distinguish two categories of sequence databases depending on what type of sequence, DNA or protein, they store. Thus, we speak about DNA archives and protein sequence archives. GenBank is one of the DNA archives, that stores all DNA sequences that are made public. It was maintained at the Los Alamos National Labs (LANL), but it is now maintained by the National Center for Biotechnology Information (NCBI)[2] at the National Library of Medicine (NLM), which is part of the National Institutes of Health (NIH) in the USA. It is split into smaller, discrete divisions, that can be seen in Table 3.3. The European Molecular Biology Library (EMBL) is another large DNA archive. It is maintained by EBI (European Bioinformatics Institute)[3]. Like GenBank it is also spilt into several divisions, each of which differs by the amount of sequences and by the quality of the data. More information about EMBL can be obtained from its home page[4]. The DNA DataBase of Japan (DDBJ)[5] and the Genome Sequence DataBase (GSDB)[6] also exist. The GSDB is a new DNA database, which was created by a project of the National Center for Genome Resources (NCGR)

[2]http://www.ncbi.nlm.nih.gov/
[3]http://www.ebi.ac.uk/
[4]http://www.ebi.ac.uk/embl/
[5]http://www.ddbj.nig.ac.jp/
[6]http://scop.wehi.edu.au/gsdb/gsdb.html

| Division Code | Description |
|---|---|
| PRI | Primate sequences |
| ROD | rodent sequences |
| MAM | other mammalian sequences |
| VRT | other vertebrate sequences |
| INV | invertebrate sequences |
| PLN | plant, fungal, and algal sequences |
| BCT | bacterial sequences |
| RNA | structural RNA sequences |
| VRL | viral sequences |
| PHG | bacteriophage sequences |
| UNA | unannotated sequences |
| EST | EST sequences (expressed sequence tags) |
| PAT | patent sequences |
| STS | STS sequences (sequence tagged sites) |
| GSS | GSS sequences (genome survey sequences) |
| HTG | HTGS sequences (high throughput genomic sequences) |

Table 3.3: GenBank divisions

supported by the Department of Energy (DOE). As a result of the International Nucleotide Sequence Database Collaboration, all the DNA archives, mentioned above, share information between them updating one another periodically. Regarding the protein sequence archives, the Protein Information Resource (PIR)[7] in the United states and the Swiss-Prot[8] in Europe are the major ones. Unlike PIR, Swiss-Prot provides high level annotations including description of protein function. Recently, PIR-International[9] together with EBI and SIB (Swiss Institute of Bioinformatics)[10] have joined in a common effort to establish the UniProt (United Protein Databases), the central resource of protein sequence and function. Except the aforementioned databases, there are also plenty of others, specialized databases. Each of them is specialized in a particular area. For example, some store sequences about a particular organism or a cell type, others record all mutations and differences (polymorphisms) that have been discovered in a set of genes, some store information on particular biological functions while others follow specialized terminology and taxonomic style that are particular to a subfield of biology. Moreover, other differ in the way the data are stored and the kind of retrieval service they offer. Table 3.4 gives the names, the website and a small description of some of these most popular databases. Pointers to a

[7]http://pir.georgetown.edu/

[8]http://www.ebi.ac.uk/swissprot/

[9]http://pir.georgetown.edu/pirwww/aboutpir/collaborate.html

[10]http://www.isb.sib.ch

| Database Name | Characterization |
|---|---|
| Prosite | Database of protein families and domains http://www.expasy.ch/prosite |
| FLYBASE | A Database of the Drosophila Genome http://flybase.bio.indiana.edu/ |
| RHdb | A collection of raw data used in constructing radiation hybrid maps http://www.ebi.ac.uk/RHdb/ |
| CD40Lbase | A collection of clinical and molecular data on CD40 ligand defects leading to X-linked Hyper-IgM syndrome http://us.expasy.org/cd40lbase/ |
| SBASE | A collection of protein domain sequences collected from the literature, from protein sequence databases and from genomic databases http://hydra.icgeb.trieste.it/ kristian/SBASE/ |
| ENZYME | Repository of information relative to the nomenclature of enzymes http://us.expasy.org/enzyme/ |
| EPD | Eukaryotic Promoter Database. It provides information about eukaryotic promoters available in the EMBL Data Library http://www.epd.isb-sib.ch/ |
| REBASE | Restriction Enzyme Database. It is a collection of information about restriction enzymes, methylases, the microorganisms from which they have been isolated, recognition sequences, cleavage sites, methylation specificity, the commercial availability of the enzymes and references http://www.neb.com/rebase |

Table 3.4: A catalogue with some of the existing specialized databases

large number of databanks organized in categories can be found in [4].

### 3.4.2 BLAST: Efficient similarity searches in large sequence databanks

The dynamic programming algorithms for computing the similarity and the optimal alignment between two sequences, that we have presented so far, are too slow for searching large databases hence their use is forbidden. To overcome this problem, one of the three following approaches are applicable.

1. The implementation of the dynamic programming algorithms in hardware. In this case they will be executed faster but the high cost of this approach makes it unaffordable.

2. Using parallel hardware, the distribution of the problem to a number of processors and the integration of the results. This method is also expensive and not widely acceptable.

3. The implementation of algorithms using heuristic methods that work much faster than the original dynamic programming.

One of the two most popular heuristic methods for detecting sequence similarities, which are widely applicable, is presented in this section. The first method that was developed is FASTA (short for "fast-all") [29, 24] and after this BLAST(short for Basic Local Alignment Search Tool) [10].

Both are based on the following observations:

- The nucleotide or residue substitutions appear much more often that the insertions and deletions between two sequences.

- There is the ability to preprocess the database.

- Homologous sequences[11] contain a lot of segments with matches and substitutions. These segments can be the starting points for further searching that will probably conclude in high similarity.

**BLAST** The ideas in BLAST were developed by S. Altschul, W. Gish, W. Miller, E. Myers and D. Lipman in 1990 [10] with purpose to increase the speed of the FASTA program [24]. The acronym BLAST refers not to a single program but to a family of programs. Each of them is suitable for a different problem domain but they all use the BLAST algorithm. Table 3.5 presents the name and the description of all the programs that are members of BLAST family.

What BLAST actually does is to find regions of high similarity in alignments without gaps between a query sequence and the database sequences, evaluated by a scoring matrix. Before we go on to present the algorithm in detail, we define some fundamental terms.

**Definition 11** *Given two strings $S_1$ and $S_2$, a* segment pair *is a pair of equal-length substrings of $S_1$ and $S_2$, aligned without spaces (gaps). A* locally maximal segment pair *is a segment pair whose alignment score (without gaps) would decrease either by expanding or shortening the segments on either side. A* maximal segment pair (MSP) *in $S_1$, $S_2$ is a segment pair with the maximum score over all segment pairs in $S_1$,$S_2$. A* high scoring segment pair (HSP) *in $S_1$, $S_2$ is a locally maximal segment pair with score over a threshold.*

Figure 3.6 graphically represents the terms just defined.

Let $Q$ be the query sequence which will be compared against a database in search of similarities. In the following paragraphs when we use the term *database*,

[11]The sequences that are very similar because they have a common evolution origin.

| Program Name | Description |
|---|---|
| BLASTN | compares a nucleotide query sequence against a nucleotide sequence database |
| BLASTP | compares an amino acid query sequence against a protein sequence database |
| BLASTX | compares the six-frame conceptual translation products of a nucleotide query sequence (both strands) against a protein sequence database |
| TBLASTN | compares a protein query sequence against a nucleotide sequence database dynamically translated in all six reading frames (both strands) |
| TBLASTX | compares the six-frame translations of a nucleotide query sequence against the six-frame translations of a nucleotide sequence database |

Table 3.5: The BLAST Family Programs

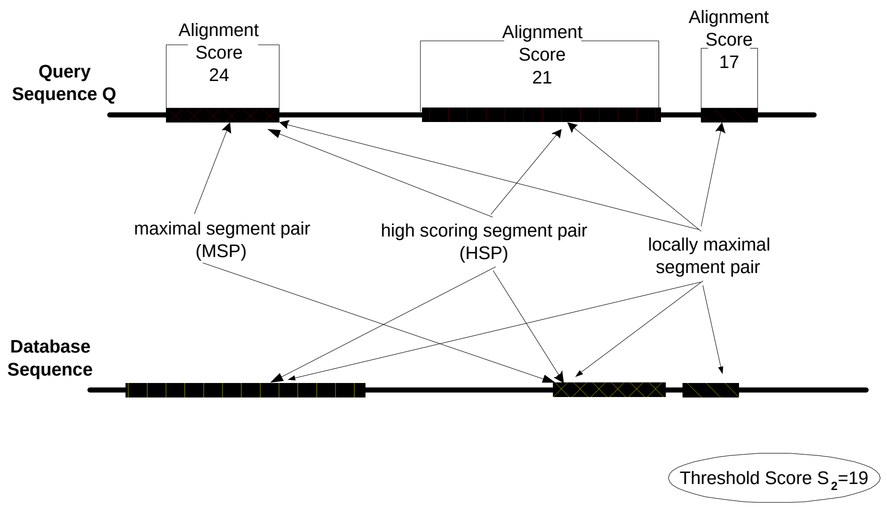


Figure 3.6: BLAST terms

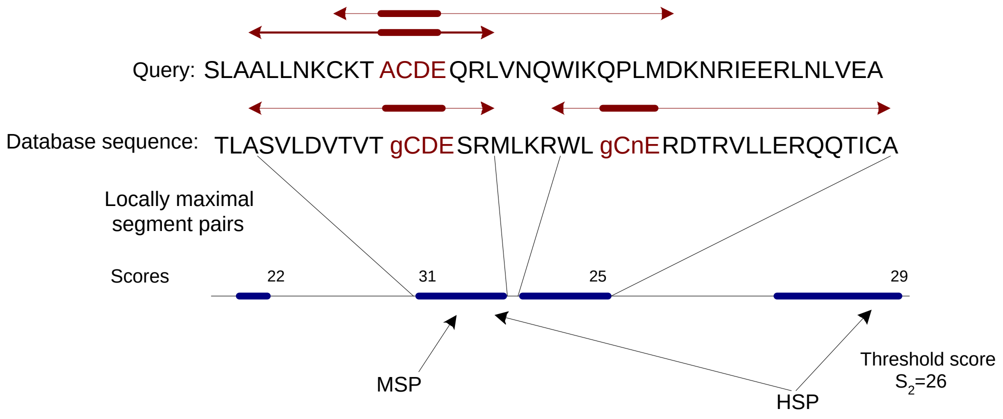

Query: SLAALLNKCKT ACDE QRLVNQWIKQPLMDKNRIEERLNLVEA
Database sequence: TLASVLDVTVT gCDE SRMLKRWL gCnE RDTRVLLERQQTICA
Locally maximal segment pairs
Scores
22
31
25
29
MSP
HSP
Threshold score
S2=26

Figure 3.7: An illustration of the BLAST search algorithm

we refer just to a large set of catalogued sequences. In other words, database for us will be a collection of sequences and nothing more. BLAST first finds all the database sequences that when compared with the query sequence $Q$ contain HSPs (locally maximal segment pairs with score over a `threshold`), then evaluates the statistical significance of any matches that were found and finally reports only those that satisfy a user-selectable threshold of significance $E$. The *statistical significance threshold* $E$ for reporting database sequence matches should be thought as the upper bound on the expected frequency of chance occurrence of an HSP within the context of the entire database search [35]. For the evaluation of the statistical significance of the HSPs the algorithm uses the statistical method of Karlin and Altschul [21, 22]. The score value that corresponds to the value of the statistical significance of the threshold $E$ represents the score at which a single HSP would by itself satisfy the significance threshold $E$ and is denoted by the capital letter $S$. Figure 3.8 represents the BLAST strategy visually and as one can notice, multiple HSPs can be detected between the query sequence $Q$ and a database sequence.

Before we start describing the steps of the algorithm, let us give a list of the program's inputs. We present only the parameters that are accepted by all programs of BLAST family and characterize the algorithm itself. Thus, BLAST take as inputs:

- A query sequence Q User's query is accepted in FASTA format. However, the BLAST web pages of NCBI also accept input sequences which are defined by their NCBI Accession number, or their GI in GenBank.

- A database DB The user can select one of the existing databases found at ftp sites of several institutes and laboratories, as they presented in subsection 3.4.1, or can use his own database. The sequences of the database must located in a single text file and follow the FASTA format and before their use they must be preprocessed using the suitable program.

- E The statistical significance threshold that must be satisfied by the matches (HSPs) that were found in order the database sequences corresponding to these matches are reported. It is a Real valued parameter having range $0 < E \leq 1000$ and default value 10.

- S The score at which a HSP would satisfy $E$. Its default value is calculated from the value of $E$, either the default one or the user-defined one, if it has not been set on the command line.

- $E_2$ The The statistical significance threshold that must be satisfied by a MSP to be considered a HSP. It should be thought as the expected number of HSPs that will be found when comparing two sequences of the same length. Its default value is 0.15.

- $S_2$ The threshold score at which a MSP is defined as HSP. It should be thought as the expected score for MSP between two sequences. Its default value is calculated from the value of $E_2$ if it has not been set on the command line.

- W The length of the short words that are identified in the query sequence $Q$ that either match[12] or satisfy some positive-valued threshold score $T$[13] when aligned with a word of the same length in a database sequence.

- T The word score threshold for generating all words of length $W$ that yield a score of at least $T$ when aligned with some word of length $W$ from the query sequence $Q$.

- X The quantity by which the alignment score must fall off from its maximum achieved value so as the extension stops. It is a positive integer and its default value depends on the type of BLAST program.

  Parameters related with sequence lengths and computed in purpose of obtaining more accurate statistical significance estimates:

- Y The effective length of the query sequence $Q$, measured in residues. Its default value is the actual length of the query sequence.

- Z The effective length of the database, measured in residues. Its default value is the actual length of the query sequence.

[12]in case of DNA sequences
[13]in case of protein sequences

More about how the effective lengths are computed and how the affect the statistical significance of the matches can be found in the program's manual [35].

Parameters related with the output:

- H It regulates the display of a histogram of the expected frequency of chance occurrence of the database matches found. In its default value, which is zero, no histogram is displayed. On the contrary, if $H$ is set to a non-zero value then a histogram will be displayed.
- V The maximum number of databases sequences for which one-line descriptions will be reported. Only positive values are valid and its default value is 500.
- B The maximum number of database sequences for which high-scoring segment pairs will be reported. Only positive values are valid and its default value is 250.

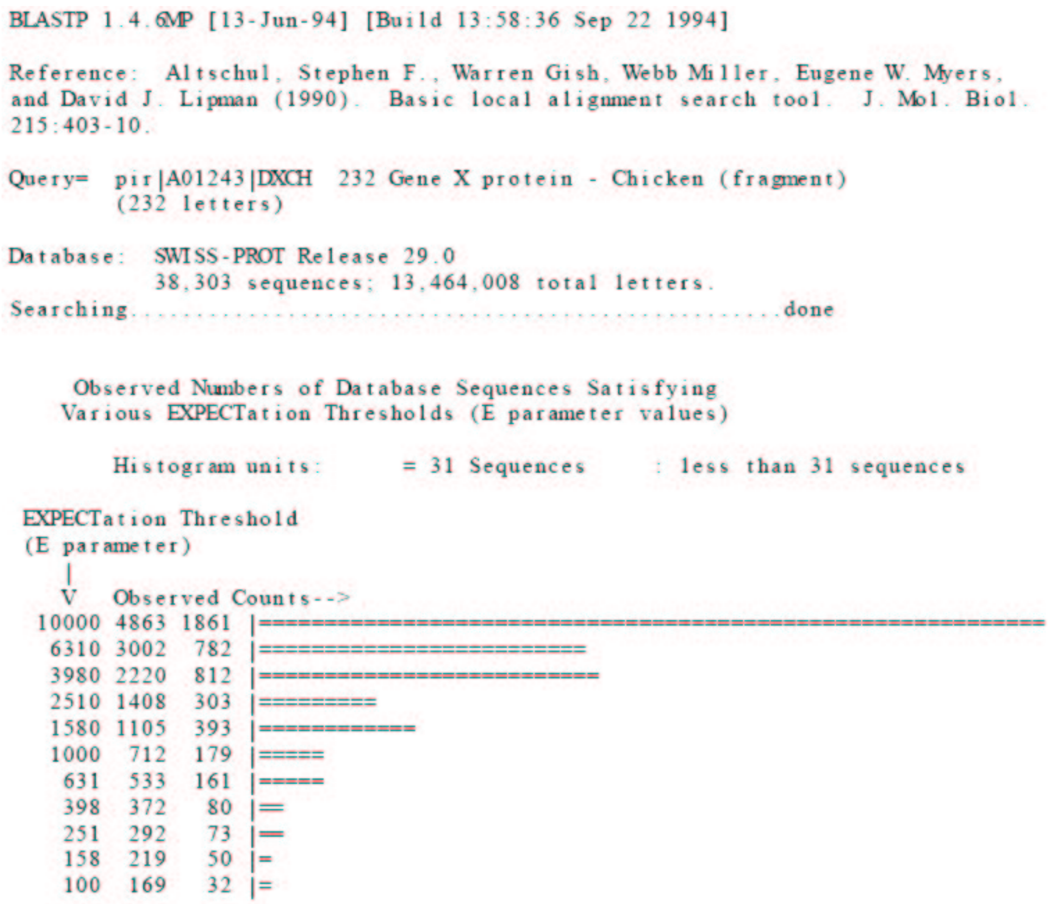
There tens of other parameters more of which are highly depended on the type of program executed. We just refer two of them that are used by the `blastn` program; the one that we study in this work. These are:

- M The score for a pair of matching residues. It must be a positive integer.
- N The penalty score for a pair of mismatching residues. It must be a negative integer.

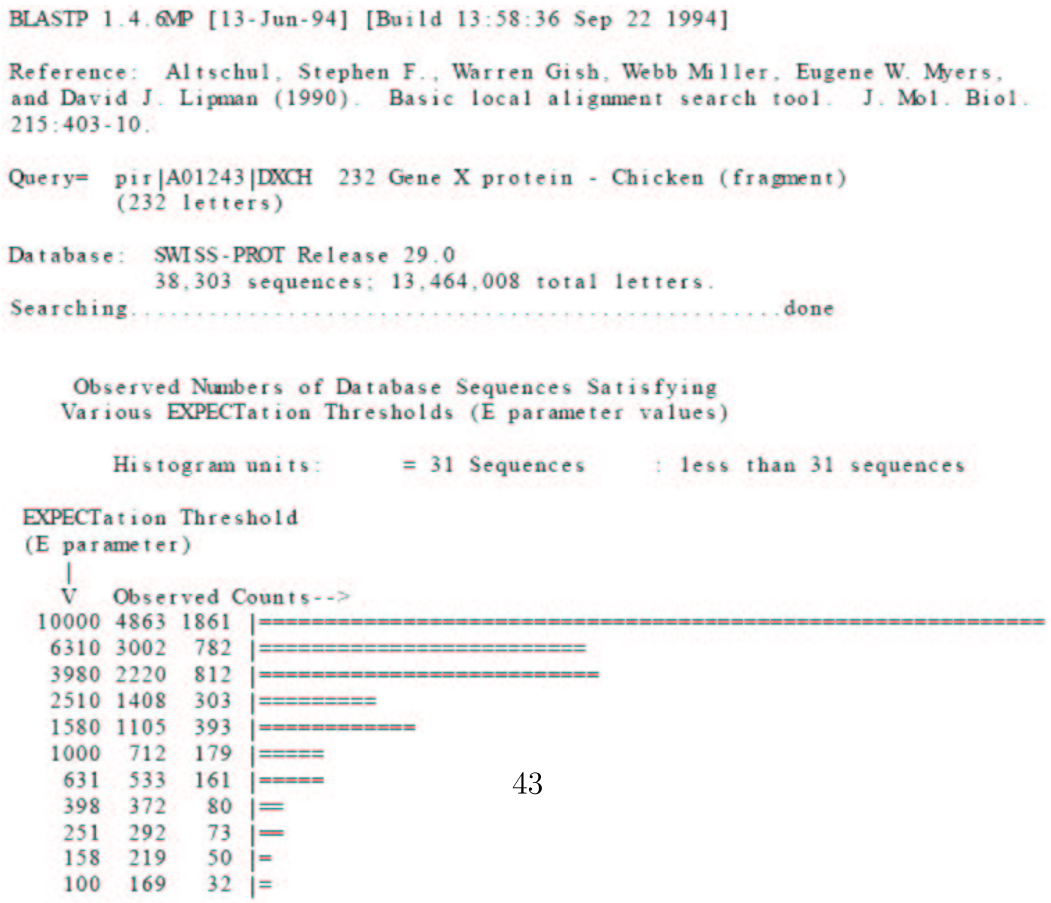
Let us now describe the steps of the algorithm in more detail. In order to find locally maximal segment pairs the three following steps have to be executed:

1. Compilation of a word list using the query sequence $Q$ only
2. Search for "hits" in the database according to the word list
3. Extension of the hits

Depending on what type of sequences are compared (DNA or protein) the particular steps of the algorithm have differences, as it is described in the following lines. In the case of protein sequences the construction of the word list differs from the case of DNA sequences. The other two steps are the same in either case (protein or DNA).



```
BLASTP 1.4.6MP [13-Jun-94] [Build 13:58:36 Sep 22 1994]

Reference:  Altschul, Stephen F., Warren Gish, Webb Miller, Eugene W. Myers,
and David J. Lipman (1990).  Basic local alignment search tool.  J. Mol. Biol.
215:403-10.

Query=  pir|A01243|DXCH  232 Gene X protein - Chicken (fragment)
        (232 letters)

Database:  SWISS-PROT Release 29.0
           38,303 sequences; 13,464,008 total letters.
Searching.................................................done


     Observed Numbers of Database Sequences Satisfying
    Various EXPECTation Thresholds (E parameter values)

        Histogram units:      = 31 Sequences     : less than 31 sequences

 EXPECTation Threshold
 (E parameter)
    |
    V   Observed Counts-->
  10000 4863 1861 |============================================================
   6310 3002  782 |=========================
   3980 2220  812 |==========================
   2510 1408  303 |=========
   1580 1105  393 |============
   1000  712  179 |=====
    631  533  161 |=====
    398  372   80 |==
    251  292   73 |==
    158  219   50 |=
    100  169   32 |=
```



```
BLASTP 1.4.6MP [13-Jun-94] [Build 13:58:36 Sep 22 1994]

Reference:  Altschul, Stephen F., Warren Gish, Webb Miller, Eugene W. Myers,
and David J. Lipman (1990).  Basic local alignment search tool.  J. Mol. Biol.
215:403-10.

Query=  pir|A01243|DXCH  232 Gene X protein - Chicken (fragment)
        (232 letters)

Database:  SWISS-PROT Release 29.0
           38,303 sequences; 13,464,008 total letters.
Searching.................................................done


     Observed Numbers of Database Sequences Satisfying
    Various EXPECTation Thresholds (E parameter values)

        Histogram units:      = 31 Sequences     : less than 31 sequences

 EXPECTation Threshold
 (E parameter)
    |
    V   Observed Counts-->
  10000 4863 1861 |============================================================
   6310 3002  782 |=========================
   3980 2220  812 |==========================
   2510 1408  303 |=========
   1580 1105  393 |============
   1000  712  179 |=====
    631  533  161 |=====
    398  372   80 |==
    251  292   73 |==
    158  219   50 |=
    100  169   32 |=
```

Figure 3.8: An illustration of the BLAST search algorithm

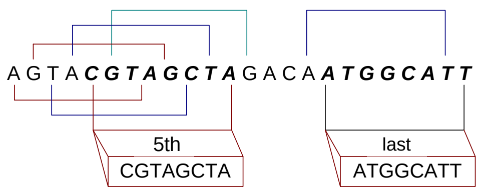


Figure 3.9: Word list generation for DNA sequence

**First step: compilation of a word list** In this first step a list of length-$w$ substrings of query sequence $Q$ is generated. These length-$w$ substrings are called *$w$-mers* or *words* in BLAST. In the case of a DNA sequence, the list contains all the contiguous substrings of length $w$ of the query sequence $Q$. The length of the word list depends on both the length of the sequence and the length $w$ of each substring (word). If $L$ is the length of the sequence and $w$ the length of the substrings, then the generated word list will contain $L - w + 1$ words. So, increasing $w$ decreases the word list size and conversely decreasing $w$ increases it. The following example illustrates the word list computation of a DNA sequence while Figure 3.10 presents the pseudo-code for it.

**Example 10** *Let*

$$AGTACGTAGCTAGACAATGGCATT$$

*be the query DNA sequence with length $L = 24$ and $w = 8$ the length of each substring (word). The word list contains all the $L - w + 1 = 24 - 8 + 1 = 17$ contiguous substrings of length 8 with*
*$AGTACGTA$ to be the first word,*
*$GTACGTAG$ the second,*
*$TACGTAGC$ the third etc. and*
*$ATGGCATT$ the last one.*
*Figure 3.9 represents the whole procedure visually.*

In the previous example, the value of parameter $w$ was 8. This value is not standard but it varies between 8 to 12 in case of DNA sequences. The default value of $w$ used by the program `blastn` that perform nucleotide comparisons is 11. This means that in this case the program is restricted to find sequences that share at least an 11-*mer* stretch of 100% identity with the query. Under the random sequence model proposed by Karlin and Altschul [21], stretches of 11 consecutive matching residues are unlikely to occur merely by chance even between only moderately diverged homologs. If better sensitivity is needed, one should use a smaller value for $w$.

A more complex strategy[14] is applied for the compilation of the word list in case of protein sequences. In this case a threshold value $t$ is set, a scoring

[14]The same idea is used in Myers's sublinear matching method [28].

```
function WordList ( query sequence $Q$, length $w$) {
    if ($Q$ is a DNA sequence) then
        index=1;
        while $index \leq length(Q) - w + 1$ do
            read the $w$ next characters
            store them as an entry to the list
            index++;
        end while
    end if
}
```

Figure 3.10: Pseudo-code for the compilation of the word list

matrix is chosen and the value of $w$ is set as before. In this case the value of $w$ is set to a lesser value than in case of DNA sequences. In particular, $w$ is set to 3 or 4 for proteins sequences. Its default value is 3 for the programs that perform amino acid comparisons (`blastp`, `blastx`, `tblastn` and `tblastx`). Let us now describe the way the word list is generated for protein sequences. For each $w$-length substring $a$ in query sequence $Q$, all possible $w$-length strings $b$ of the alphabet $\Sigma_{amino}$ are constructed. Each of them is aligned to current substring $a$ and its similarity score is computed. A chosen scoring matrix is utilized for the computation of the similarity score. If the score is at least $t$, then the string $b$ is included in the word list, otherwise it is excluded and we go on with the next. At the end, the word list contains all the $w$-length words from alphabet $\Sigma_{amino}$ whose alignment score with $a$ is greater than or equal to threshold $t$. This step is repeated for each $w$-length word of the query. The final word list is the union of all word lists corresponding to $w$-length words. The following example illustrates the whole procedure.

**Example 11** *Let Q be the protein query sequence*

$$KNRIERACDEKQPLMD$$

*with length $L = 16$ and $w = 4$ the length of each word. We also set the threshold $t$ to be 17 and the scoring matrix we use is PAM120*[15]*. For each 4-length substring of Q, we construct the words that align to the current 4-length substring with an alignment score above 17.*

*Let us consider ACDE, the 4-length substring of Q, which is located at the seventh place of the protein sequence. The construction of the words can be done either by directed substitutions of the letters or the exhaustive try of all the possibilities. Since trying all the possibilities is slow, a more efficient way can be used*

[15]PAM120 belongs to the family of PAM amino acid scoring matrices and is provided in the BLAST software distribution.

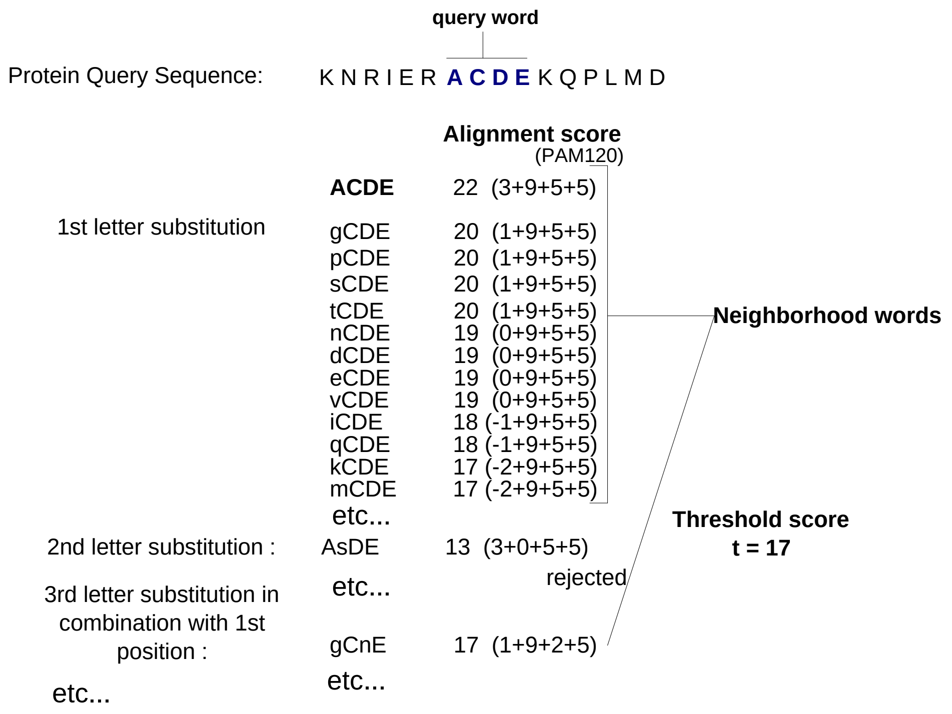


Figure 3.11: Word list generation for protein sequence

*and this is the directed letter substitutions. According to this, we substitute each character of w-length substring of Q to all acceptable amino acids, that result to an alignment score above the threshold t. In order to save time in substitutions, we must have the scoring matrix ordered in descendent order. If we do so, we stop substituting the rest of amino acids whenever the alignment score falls below the threshold. Since the matrix is ordered, the rest of the amino acids will result to a lower score. Having the substitutions of each character ordered, we now construct the w-length string directed by the score of these in respect to threshold 4. A portion of the word list for the substring ACDE is presented in Figure 3.11. Following the same way, similar lists are constructed for all the 4-length substrings of the query sequence Q. The total word list consists of all the words of these lists.*

In this way, short words of length $w$ are identified in the query sequence that either match or satisfy some positive-valued threshold score $t$ when aligned with a word of the same length in the database sequence. The score threshold $t$ is referred to [10] as the *neighborhood word score threshold* and the list of words generated as *neighborhood*. Let us now see how the neighborhood word score threshold $t$ influences the sensitivity and the speed of the algorithm. If the value of $t$ is low, the alignment score of the words is low, then more words are contained in the list since the alignment score is low, thus the word list size is increased and consequently the search needs more time to be completed hence the program runs slower. On the other hand, the sensitivity of the search is improved since there are more words in the list and consequently there are more candidates that result in locally maximal segment pairs and HSPs. Conversely, increasing the value of $t$ decreases the word list size and the word hits number and finally the likelihood of detecting HSPs but increases the speed of the algorithm. The value of $t$ that is used in applications is calculated at run-time from the composition and the length of the query sequence and the scoring matrix employed on which is highly depended. Experience tell us that for "best" values of $w$ and $t$ there are typically about 50 words in the list for every residue in the query sequence.

More about the calculation of $t$ for particular applications can be found in [35], a detailed manual about BLAST and on the website of NCBI [1].

**Second step: search the database for "hits" according to the word list**
After the word list generation, the database sequences are searched for all the exact occurrences of the words in the word list. Any word in the list found to be located in a database sequence is called a "hit" and it is possible to be part of a HSP between the query sequence and the database sequence that was found in. Thus, these *word hits* act as seeds for initiating searches to find HSPs containing them. In the BLAST programs the search for word hits is actually implemented by the use of a deterministic finite state automaton (DFA) but an index can also be used as it is also mentioned in the original paper for BLAST [10]. We recall that this step is common in either case (DNA or protein) and describe these two approaches.

Using a deterministic finite state automaton (DFA)

The DFA takes as input the database sequences and recognizes the words of the word list. Such a DFA works as follows. It reads the characters of the sequence and makes a proper transition according to the character that was read and its current state. A state transition table has been calculated which tells what state to go to based on the next character in the sequence. If a transition lead us to a state that is a final state then a word has been recognized.

Using a hash table

In this case a hash table is constructed with as many entries as the number of all possible words of $w$ letters in either the alphabet $\Sigma_{DNA}$ or the alphabet $\Sigma_{amino}$. In its *ith* entry this table contains a list of all the occurrences in the query sequence of the *ith* word. Consequently, as we scan the database, each database word leads us to the corresponding hits. Because the size of this hash table is typically large, the original paper of BLAST [10] prefers to use the DFA solution.

**Third step: extension of the hits** Once the database has been scanned and the hits have been found, each one of the hits is extended to a locally maximal segment pair. The extension of each hit is done in both directions along each sequence and is terminated when the accumulated score of the hit under extension stops increasing and falls a certain amount $X$ below the maximum score achieved for shorter extensions, or when the score goes to zero or below, or when we reach the end of either sequence. The score of a hit may increase or decrease since the scoring matrices include negative values so the score may go to zero or below. The aforementioned strategy for the extensions has been followed in earlier versions of BLAST [10] and it is also followed by us in this work. In more recent versions the extensions are terminated as long as the score continues to increase. After the end of the extension, the score of the (locally) maximal segment pair is checked against a predefined cut-off score $S_2$. If it is above $S_2$, then it is a HSP. After that the statistical significance evaluation of its score follows and only if the score satisfies the user-selectable threshold $E$, the match is reported. In the following lines we explain the extension procedure in more detail while the pseudocode for it is presented in Figure 3.12.

After the hit has been found and the starting positions in both the query sequence and the database sequence have been located, the first thing we compute is the *diagonal* to which this hit belongs. The diagonal is defined as the difference of the starting position $x$ of the word in the database sequence and the starting position $y$ in the query sequence.

**Example 12** *As an example let us consider the word "ATT" that has been recognized to be part of the query sequence*

*11 . . . C**ATT**GAGTTC. . . 20*

*starting at position* 12 *and part of the database sequence*

*131 . . . GGAC**ATT**AAC. . . 140*

*starting at position* 135*. In this case, the diagonal of this hit will be* $135-12 = 123$*.*

All words starting at positions $x$, $y$ which have the same difference $x - y$ of their starting positions belong to the same diagonal. The diagonal and the position where the hit extension ends in the query sequence are stored in an array. After the computation of the diagonal, we search the data structure for an entry that stores the same value with the one just computed. Finding one means that there is at least one hit that has been extended previously and belongs to this specific diagonal. Since, a locally maximal segment pair is typically much longer than a single hit, it is likely that multiple hits have the same diagonal. It is also possible that the starting position of a hit is part of another hit already extended. That is why, after we have found that the hit under examination belongs to a diagonal for which an extension has been done, we must compare the starting position of the hit, that is about to be extended, to the ending position of the previous extended hit for that diagonal. If the position of the hit under examination comes before the other's, meaning that the hit is part of the already extended one, it is rejected and no extension takes place for that hit. If it does not, the extension starts immediately and after its completion the position of the query sequence where the extension ends, is saved overwriting the position of the previous extended hit for this specific diagonal. If we had not found any entry of the data structure which stores the same value of diagonal with the one just computed, we do start the extension, but now, when the extension ends, we store a new entry to the structure since there is no other for this diagonal. This entry stores the values of the diagonal and the position at the query sequence where the extension ends. When the extension comes to an end, the new locally maximal segment pair is stored. Part of the information stored for an *HSP* contains:

- The score of the hit after the extension.
- The length of the extended hit.
- The position of the query sequence where the extension started.
- The position of the database sequence where the extension started.
- A pointer to the next entry.

Due to the consecutive words that may be hits, a *HSP* may include other *HSPs* or may be included by other *HSPs*. Thus, when a new *HSP* is stored, we search among the list of the existing *HSPs* looking for overlaps. If we do not find one, we just add the current *HSP* in the list. If we find one or more overlaps, we overwrite the first overlap with the new *HSP* and discard all the others from the list. The discarded *HSPs* are saved in a different list.

Regarding the scoring schemes exploited to compute the similarity scores, in the case of protein sequences the same scoring matrix that was utilized in word list generation is utilized again. As for the case of DNA sequences, the scoring scheme that is commonly used is depicted in Table 3.6 and gives a positive value to matches and a negative one to mismatches. This table has been set to be the default one for the application program `blastn` while BLOSUM62 matrix is the default for the application programs that compare protein sequences (`blastp`, `blastx`, `tblastn`, `tblastx`).

Let us now see how the speed and the sensitivity of the algorithm are affected when the parameter $X$ varies. We remind the reader that it represents the maximum decay of the alignment score under which the extension of a hit stops. Increasing $X$, results to lower alignment scores for the segment pairs increasing the number of MSPs that are detected so the possibility an HSP to be overlooked decreases. But consequently, the run time of the algorithm also increases since

|  | **A** | **T** | **G** | **C** |
|---|---|---|---|---|
| **A** | 5 | -4 | -4 | -4 |
| **T** | -4 | 5 | -4 | -4 |
| **G** | -4 | -4 | 5 | -4 |
| **C** | -4 | -4 | -4 | 5 |

Table 3.6: Similarity matrix

now the extensions take more time to stop. On the contrary there is no or little variation to the sensitivity of the algorithm.

We now give details and present the relation between the parameters $E$,$S$,$E_2$,$S_2$ of the algorithm. We remind to the reader that $E$ is the statistical significance threshold for reporting the sequence matches while $S$ is the score of an HSP which satisfies the statistical significance threshold $E$. Respectively, $S_2$ is the threshold score at which a MSP is considered to be a HSP while $E_2$ is the statistical significance threshold that must be satisfied by a MSP to be considered a HSP. In both cases, the statistical significance $E$ or $E_2$ that corresponds to the score $S$ or $S_2$ is calculated according to the method proposed by Karlin and Altschul [21, 41] and it is given by the equation

$$E = KNe^{-\lambda S} \tag{3.2}$$

where $E$ is the statistical significance of either a HSP or a MSP having score $S$; $K$ and $\lambda$ are Karlin-Altschul parameters; $N$ is the product of the query and database sequence lengths. More details can be found at BLAST manual [35]. $S$ and $E$ are both another measure of how sensitive a BLAST search is. If a search is sensitive, even the less significant HSPs will be reported, these that have higher probability of chance occurrence. Since the lower statistical significance HSPs correspond to low values for $S$, a sensitive search supposes low values for $S$. Conversely, an insensitive search make use of high value for $S$. If a search is insensitive then only the highly significant HSPs are reported.

## 3.5 Summary

In this chapter we presented some basic ideas of computational biology and we concentrated on presenting the most basic operation, *sequence comparison*. We presented definitions and algorithms while we focused on the algorithm of most popular program's family of searching similarities in genomic databases.

In the next chapter we present, implement and evaluate an algorithm for the second part of algorithm presented in Section 3.4.2 of Chapter 3.

```
function HitExtension ( HitList $hit$ ) {
        for each $hit$ do
                $diag$ = computediagonal($hit$);
                if ( $diag$ exists ) then
                  if (diag.ending_position $\leq$ $hit.starting\_position$ )then
                    do_extension();
                    $diag.ending\_position$ = $ending\_position$;
                    SaveHSP();
                  end if
                  else
                    return, no extension takes place;
                end if
                else
                  do_extension();
                  $diag.ending\_position$ = $ending\_position$;
                  SaveHSP();
                end else
        end for
}

function do_extension ( Distance $X$, database sequence $DBs$, query sequence $Q$) {
        for both directions do
                while (we have not reach the end of either of the sequences) do
                      ch1 = read the next character $\in DBs$
                      ch2 = read the next character $\in Q$
                      if (ch1 != ch2) then
                        current-score = computescore(ch1,ch2);
                        if (current-score $\geq X$) then
                          continue;
                        else
                          break;
                      end if
                      current-score = computescore(ch1,ch2);
                      if (current-score > 0) then
                        Score = Score + current-score;
                        current-score = 0;
                        update the starting_position and the ending_position
                      end if
                end while
        end for
}
```

Figure 3.12: Pseudo-code for the hit extension

# Chapter 4

# Trading Space for Time in BLAST

After Chapter 3 where fundamental biological concepts were introduced and the most popular sequence comparison algorithms were presented, in this chapter we propose a variation of BLAST algorithm which can also be used in an Information Alert scenario, as it is discussed in Section 4.4.

## 4.1 Introduction

In Section 3.4.2 we studied in detail the most popular similarity search algorithm in molecular biology, BLAST. In this chapter we propose a variation of it. We were motivated by the use of indexing methods to reduce the running time of the algorithm while accepting an increase in the memory usage.

As we discussed in Section 3.4.2, the BLAST algorithm is divided in three distinct steps. We adopt the algorithmic idea of the first and third step, that is the compiling of the word list and the extension of the hits, while we modify the algorithm of the second step, the way of finding the words' exact matches, in order to apply it to our requirements. Now we explain how our proposal differs from the BLAST algorithm. In both cases we have a query sequence $Q$ that must be compared against a set of sequences $S$. According to BLAST, $Q$ is compared to each sequence separately, meaning that each one of the three steps of BLAST are applied sequentially. On the other hand, in our case, the first step is applied to all sequences of $S$ beforehand. Thus, the second step, the word matching, is executed for the query sequence $Q$ and all the sequences of the set $S$ and not for one by one sequence of $S$. The third step, the extension, takes place after the end of the second one and is executed sequentially as before except if we make use of parallel processing. In other words, we organize the word lists of all the sequences of set $S$ in such a way so as to find efficiently the matches between all of them and the query sequence $Q$.

In the rest of the chapter, we present our proposed method, giving details about the data structures and the algorithm, the experimental evaluation of it and finally we discuss how our algorithm could be used in an Information Alert scenario.

## 4.2 The Method SeedsIndex

### 4.2.1 Description

The strategy of SeedsIndex method is the preprocessing of a database of sequences, so as when the query sequence comes, the search will not be time consuming. The method's preprocessing utilizes an index over the word lists of the database sequences and not over the word list of the query sequence, as a brute force method would do. When the query sequence comes, its own word list is compiled. There is no need for storage space for this, since the matching process starts as soon as each word is generated. During the matching process, as it is presented in Figure 4.4, a Hash Table (HT) is probed for all the words of the query sequence one by one. If the slot of HT is empty, then we do not have a match, meaning that this word is not contained in any of the database sequences, otherwise, we have a match. In case of a match, the location of the word in the query sequence is saved in an auxiliary structure attached to the HT. When the matching procedure is over, the extension of the hits takes place. The extension algorithm is identical to the one that is used by the first version of `blastn` and was presented in Figure 3.12. All the aforementioned data structures are described in detail in the following paragraphs.

### 4.2.2 Database Sequences Data Structures

To facilitate the matching process, SeedsIndex uses a hash table (HT) to store the compiled words of all the database sequences. This hash table uses words as keys and its size is constant and equal to the number of all possible words of $w$ letters. Thus, the hash table will have either $4^w$ entries for DNA sequences[1] or $20^w$ entries for protein sequences[2], $w$ is the length of the compiled words. The usual value of $w$ varies between 8 to 12 for nucleotides and between 3 to 4 for amino acids, thus the hash table will have up to $4^{12}$ ($4^{12}$=$67,108,864$) entries or $20^4$(=$160,000$) depending on the type of sequence. The slots of HT can be either empty or point to a linked list depending on the existence of the words in the word lists of the database sequences. This linked list, called PSL (ProfileSequenceLocations) list,

[1]There are 4 different options since we have 4 kinds of nucleotides.

[2]There are 20 different options since we have 20 kinds of amino acids which construct the proteins.

saves the locations of the word in the sequence for each database sequence. A detailed description of it follows in the next paragraphs.

The hash function, we use, assigns a unique number to each entry. We describe such a function assuming that we have a DNA sequence. In this case, we assume that each one of the 4 nucleotides (A,C,T,G) corresponds to a number from 0 to 3. We also assume that $w = 4$ and $a_1a_2a_3a_4$ is a word of the word list. Then, the hash function is given by the following equation:

$$f'(v_1, v_2, v_3, v_4) = [(v_1 \cdot 4 + v_2) \cdot 4 + v_3] \cdot 4 + v_4 \tag{4.1}$$

where $v_1$, $v_2$, $v_3$, $v_4$ is the assigned number to each letter. This hash function constructs a number by shifting the 2 bits of the number, corresponding to each letter, to the left and adding the next one until the number of $2w$ bits is formed. This number corresponds to an entry of the hash table; it is actually the index of the hash table. Consequently, that hash function assigns each word to a distinct entry of the hash table, meaning that we have one to one correspondence. An example that illustrates the use of the hash function and the hash table follows.

**Example 13** *Let us assume that we have the following DNA sequence:*

*CCCTAACGTACCTTGGAA*

*We compile the word list of this sequence and calculate the hash value of each word in the word list, according to the pre-described hash function. The hash function corresponds each nucleotide to an integer from 0 to 3. In the example, we assume that A corresponds to 0, C to 1, G to 2 and T to 3. We also consider that word length is 3. The word list and the calculated hash values according to equation 4.1 are shown in Table 4.1. The hash table for this sequence is shown in Figure 4.1. All the words of the word list are stored in the entries of the hash table that are indicated by their hash value. For instance, the fifth word of the word list AAC has hash value 1 and it is found at the entry of the hash table that has index 1. Each slot points to a linked list where the locations of the words in the sequence are stored. So, the AAC word starts at location 5 of the query sequence while the word CCT is found twice in the sequence in locations 2 and 11. Thus, the list for them has two elements.*

As it is described in Section 4.2.1, in case of a match, the location of the matched word of the query sequence is saved in an auxiliary structure attached to the HT. This data structure is a linked list, called NSL (NewSequenceLocations)list. A detailed description of it follows in the next paragraphs.

### The PSL and NSL lists

Given a word $w'$ of the word lists of the database sequences, the element of PSL list saves:

| **position** | **word** | **corresponding ints** | **hash value** |
|---|---|---|---|
| **1** | CCC | 1 1 1 | 21 |
| **2** | CCT | 1 1 3 | 23 |
| **3** | CTA | 1 3 0 | 28 |
| **4** | TAA | 3 0 0 | 48 |
| **5** | AAC | 0 0 1 | 1 |
| **6** | ACG | 0 1 2 | 6 |
| **7** | CGT | 1 2 3 | 27 |
| **8** | GTA | 2 3 0 | 44 |
| **9** | TAC | 3 0 1 | 49 |
| **10** | ACC | 0 1 1 | 5 |
| **11** | CCT | 1 1 3 | 23 |
| **12** | CTT | 1 3 3 | 31 |
| **13** | TTG | 3 3 2 | 62 |
| **14** | TGG | 3 2 2 | 58 |
| **15** | GGA | 2 2 0 | 40 |
| **16** | GAA | 2 0 0 | 32 |

Table 4.1: Calculation of the hash values for the DNA sequence CCCTAACG-TACCTTGGAA

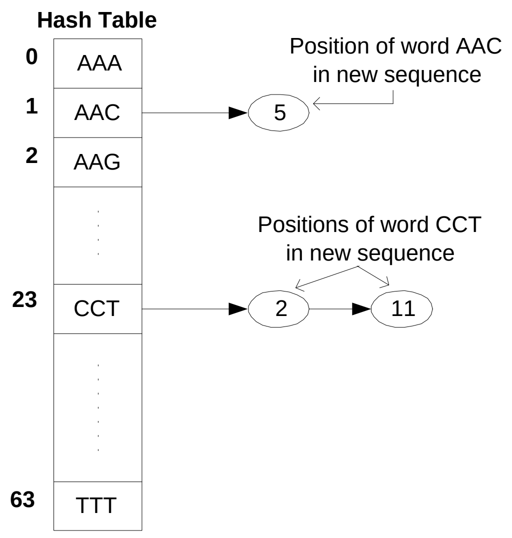

Hash Table
0
AAA
1
AAC
2
AAG
23
CCT
63
TTT
5
Position of word AAC in new sequence
2
11
Positions of word CCT in new sequence

Figure 4.1: The Hash Table for the Brute Force algorithm of the example

1. the number of the database sequence to which word $w'$ belongs
2. a list of all the locations in the database sequence where word $w'$ is located, because the same word may appear multiple times in a sequence.

Each element of PSL list saves the locations for a specified database sequence. So, the PSL list will have as many elements as the number of the database sequences to which the word appears. Moreover, each slot of HT points to a PSL list if and only if the correspondent to that slot word belongs to one or more word lists. For example, if a word appears in four different database sequences in six different locations in the three first sequences and in just one location in the fourth, then the PSL list will have 4 nodes, one for each database sequence. The three nodes will have a list of six nodes for the six locations of the word in the sequences while the last one will have a list of one node.

The element of NSL list saves:

1. all the locations in the query sequence where word $w'$ is located.

So, the NSL list will have as many elements as the number of locations where the word appears in the query sequence.

Now, we give some details about the insertion of a new node for each of the two lists. Because PSL list is an auxiliary data structure of the HT, its construction is part of the HT's construction. Figure 4.3 presents the pseudo-code for both the construction of the hash Table HT and the PSL lists. For each word in the word list of each database sequence, a node of PSL list is inserted or an existing one is updated storing the new location. According to the pseudo-code, the word list of each database sequence is built and then the HT is probed for each word of the list. If the word, let us say, $w$ is found in the HT, then the function *UpdatePSLlist()* is called. Finding the word $w$ in HT means that $w$ is either part to more than one sequence or appears to the same sequence more than once. The function *UpdatePSLlist()* checks the PSL list to find if a node for that database sequence has already been saved. Finding one means that word $w$ is located more than once in that database sequence. In this case, a new node is inserted to the list that saves the locations. Finding no node for that database sequence in the PSL list means that word $w$ belongs to some of the previous sequences except the current one. In this case, a new node of PSL list is created and inserted to the list. If the word $w$ is not found in the HT, then the function *InsertNewPSLNode()* is called. Not finding the word $w$ in HT means that $w$ does not belong to any of the previous examined sequences, so there is not a PSL list linked to that slot of HT. In this case, function *InsertNewPSLNode()* creates the first node of the PSL list and links the list to the HT slot. Thus, if a slot of HT does not point to any structure, then we infer that the word that corresponds to that slot does not belong to any database sequence.

The NSL list is constructed during the matching process as it is shown in the pseudo-code of the matching process presented by Figure 4.4. The HT is probed

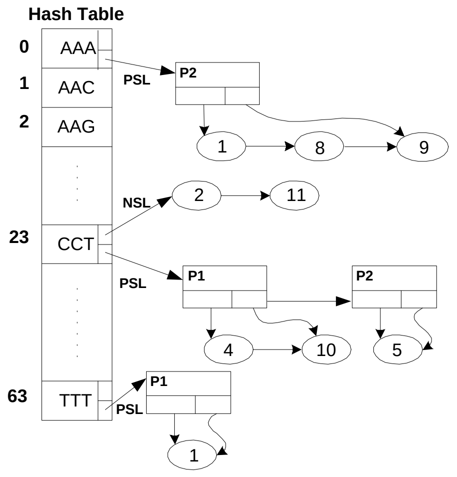


Figure 4.2: The Hash Table for the SeedsIndex algorithm

for each word of the word list of the query sequence . If the word is found, we have a match and function *InsertNewNSLNode()* is called. This function creates and inserts a new node to the NSL list that stores the words' locations in the query sequence. Thus, there is no need to use a separate data structure that will store the information of the matches, since these are stored in the NSL list. When the execution of the matching procedure finishes, some of the slots of HT will point to both lists. The words that correspond to that slots are the matched ones. Having stored all the information needed for the next step of the algorithm in the PSL and NSL lists we are ready to go on with the execution of it.

### 4.2.3 An Example

Now that we have described the data structures, an example that illustrates how the SeedsIndex proceeds follows. Let us assume that we have the three following DNA sequences:

- $p_1$: TTTCCTGGTCCTA
- $p_2$: AAATCCTAAAAC
- Q : CCCTAACGTACCTTGGAA

The first two sequences are database sequences with database identifiers $p_1$ and $p_2$, respectively. The third one is our query sequence $Q$.

According to SeedsIndex method, we create a hash table for the database sequences. For both sequences we compile their word list and calculate the hash value of each word in the word list, according to the pre-described hash function

```
function constructHT ( number of database sequences num ) {
        for each  database sequence $q$ do
                  wordlist = BuildWordList($w$)
                  for each word $w \in$ wordlist do
                          probe HT with $w$
                          if found then
                             UpdatePSLlist($w$, location)
                          else
                             InsertNewPSLNode($w$, location)
                          end if
                  end for
        end for
}
```

Figure 4.3: Pseudo-code for the construction of the HT in SeedsIndex method

```
function Match ( query sequence Q, hash table HT ) {
        for each word $w \in$ word list($Q$) do
                probe HT with $w$
                if found then
                   InserNewNSLNode(location, $w$)
                end if
        end for
}
```

Figure 4.4: Pseudo-code for the matching process in SeedsIndex method

| position | word | correspondent ints | hash value |
|---|---|---|---|
| **1** | TTT | 3 3 3 | 63 |
| **2** | TTC | 3 3 1 | 61 |
| **3** | TCC | 3 1 1 | 53 |
| **4** | CCT | 1 1 3 | 23 |
| **5** | CTG | 1 3 2 | 30 |
| **6** | TGG | 3 2 2 | 58 |
| **7** | GGT | 2 2 3 | 43 |
| **8** | GTC | 2 3 1 | 45 |
| **9** | TCC | 3 1 1 | 53 |
| **10** | CCT | 1 1 3 | 23 |
| **11** | CTA | 1 3 0 | 28 |

Table 4.2: Calculation of the hash values for database sequence with database identifier $p_1$

| **position** | **word** | **correspondent ints** | **hash value** |
|---|---|---|---|
| **1** | AAA | 0 0 0 | 0 |
| **2** | AAT | 0 0 3 | 3 |
| **3** | ATC | 0 3 1 | 13 |
| **4** | TCC | 3 1 1 | 53 |
| **5** | CCT | 1 1 3 | 23 |
| **6** | CTA | 1 3 0 | 28 |
| **7** | TAA | 3 0 0 | 12 |
| **8** | AAA | 0 0 0 | 0 |
| **9** | AAA | 0 0 0 | 0 |
| **10** | AAC | 0 0 1 | 1 |

Table 4.3: Calculation of the hash values for database sequence with database identifier $p_2$

4.1. As in example 13, we consider that each word has 3 letters. The word lists of each sequence and the calculated hash values are shown in Table 4.2 and Table 4.3. After we have calculated the hash value of each word of the word list, as described above, we create the hash table. The hash table of our example is shown in Figure 4.2. For each word in the word lists of the two database sequences, we store the *database id* and the *position* of each one to the correspondent entry of the hash table. For example, we consider the first word of the word list of the first sequence, the `TTT`. This has the hash value 63. This means that we store its position and the database identifier in the sixty third entry of the hash table. As it is shown in Figure 4.2 a node of PSL list stores the database identifier $p_1$ and a list storing the position of TTT in the $p_1$ sequence, in the example the position 1. It is possible the same word to appear more than once in the sequence, as the word `TCC` does. It appears in positions 3 and 9 of the first sequence. In such a case, we store both positions in a list of the same PSL node. Such a case is the word `AAA` in the hash table. It is located in positions 1, 8 and 9 of the $p_2$ sequence. Moreover, if we have many database sequences, then it is likelihood the same word to belong to more than one database sequences or to all of them, as the word `CCT` does. It is a member of two word lists. In such a case, we store all the *database identifiers* and the correspondent positions in separate nodes of PSL list as it is shown in Figure 4.2.

After we have constructed the hash table and whenever the query sequence comes, we are ready to search which words of the word list of the query sequence $Q$ are also words of the database sequences. The word list of the query sequence is presented in Table 4.1. As each word of the query sequence is generated, we access the correspondent entry of the hash table that is indicated by the hash value of that word. For example generating the second word for the query sequence, that is the word `CCT`, it corresponds to hash value 23, we access the twenty-third entry of the hash table and read that this word belongs to the sequences with database identifiers $p_1$ and $p_2$ and it is located to the fourth and tenth position of the first sequence and to the fifth position of the second sequence. So, the position of this word is added to the NSL list and will be examined by the third step of the algorithm. Thus as it is also presented by Figure 4.2, the NSl list corresponding to `CCT` word has two nodes storing 2 and 11 meaning that these are the two positions of the query sequence where the `CCT` is located.

## 4.3 Experimental Evaluation

Having presented our algorithm, we now move to the experiment evaluation and the comparison to the original BLAST algorithm.

### 4.3.1 Settings

The algorithm described in Section 4.2 was implemented in C, and all the experiments were run on a PC, with a Pentium III 1.7GHz processor, with 1GB RAM, running Linux. The time shown in the graphs is elapsed time in milliseconds and no other processes were run on the PC during the experiments. For each of our conducted experiments we followed a standard procedure described below.

We considered the case of DNA sequences and we implemented our algorithm and the other two steps of BLAST algorithm, as they described in Section 3.4.2, to handle DNA sequences. Thus, we made ourselves familiar with the source code of the `blastn` program, the one of the BLAST family's programs that handles DNA sequences exclusively. For the comparison of our method against the second step of the original BLAST algorithm we used the 1.4 Version of `blastn` belonging to the WU-BLAST of Washington University. We downloaded the source code of the 1.4 version and properly modified it being able to calculate the running time of the second step of the algorithm.

The data that we used to conduct our experiments were real genomic DNA sequences. We selected the Fugu cDNA database that contains 47036 sequences to be used as the long standing queries. For the query sequences we made use of the 100 EST sequences of seabream that were produced by Mr. Bargelloni for the purposes of the European project *Bridgemap*[3]. The fugu cDNA sequences are saved in a single text file in FASTA format while the EST sequences of seabream are saved in separate text files following the same FASTA format.

Having decided on these, we populate the data structures of the algorithm, load the query sequence into main memory and run each algorithm. All the experiments are run five times and the results are averaged to eliminate any fluctuations in the time measurements. Note that when we refer to the matching time of a query sequence against a database of sequences we mean only the time

[3]http://www.bridgemap.tuc.gr

needed by the algorithm to discover which words of the query sequence are also words of the database sequences. This time does not include the time needed to upload the sequences from the secondary storage to the main memory, or the time to construct the index for the database sequences. Moreover, when we refer to the memory usage of the algorithms we mean the memory space needed by each algorithm to discover which words of the query sequence are also words of the database sequences. For our algorithm this memory space includes the memory needed for the construction of the hash table and the other data structures. For the BLAST algorithm this memory space equals to the memory consumed by the automaton.

### 4.3.2 Varying The Number Of Long Standing Queries

We evaluated our algorithm under different number of database sequences measuring the first time the memory usage and the second time the matching time for both algorithms. The experiments were carried out using up to 47036 sequences from the cDNA library as the long standing queries and some of the 100 sequences as the query ones.

#### Matching Time

As we can see in Figure 4.5 both algorithms are linear to the number of database sequences, however SeedsIndex seems to be faster. Due to the use of an index structure for all the words of the database sequences, we observe that the time for the matching procedure, the second step of BLAST, is reduced to a considerable quantity and does not exceed 1 millisecond. We also observe that SeedsIndex seems to be independent to the number of the database sequences. This happens because as the number of database sequences increases, the total number of words also increases but the number of the distinct words remains constant since their number is finite.

#### Memory Usage

The main objective of this experiment was to compare the two algorithms, our SeedsIndex algorithm and the original BLAST algorithm, in terms of memory usage and make up our minds which of two is finally worth of using. The experiment was carried out with the same kind and number of data as the previous experiment.

In Figure 4.6, we can see that both algorithms are linear to the number of database sequences with the original BLAST algorithm to make use of considerably less quantity of memory space and to be independent to the number of database sequences. This is because the only memory space it needs is the one

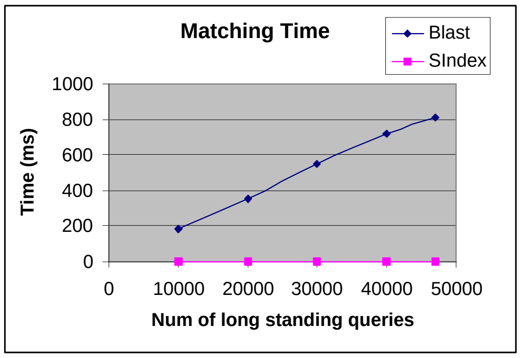


Figure 4.5: Effect of the number of long standing queries in matching time

for the construction of the automaton. We remind, at this point, that the algorithm constructs the automaton that recognizes the words of the query sequence. Because the construction of the automaton happens while reading the query sequence, it does not need extra memory space to save the word list of the query sequence. The needed memory space for the automaton depends on the size of the automaton which, in turn, depends on the size of the word list, which as we described in 3.4.2 depends on the word size $w$. Moreover it is highly depended on the sequence composition. If the same nucleotides are sequentially repeated, we end up with multiple appearances of the same word in different positions, so the size of the automaton is smaller. As for the SeedsIndex algorithm, it needs considerably more memory space than the BLAST algorithm due to the construction of the hash table and all the appropriate data structures, described in subsection 4.2.2. We must observe that as the number of database sequences increases, due the constant size of hash table only the secondary data structures, that save information about the words, are populated. Thus, as the number of database sequences increases the memory space needed decreases since the secondary data structures make use of less memory space.

### SeedsIndex vs original BLAST

Comparing the results of the two experiments, we conclude that if time is in great importance we have to use more memory space and follow the algorithmic approach of SeedsIndex. On the other hand if we are not concerned about time, then the original BLAST algorithm is the best choice since it spends less memory. If we think that nowadays memory space is quite cheap to be bought, our approach of SeedsIndex seems to be useful.

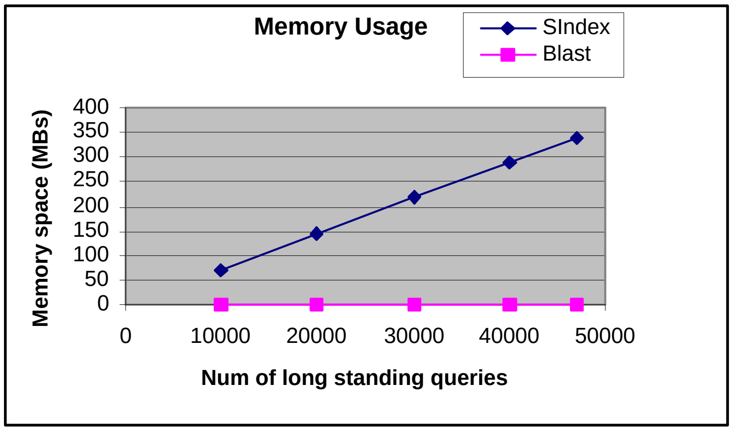


Figure 4.6: Memory usage for the two algorithms during the matching procedure

## 4.4 Using SeedsIndex in an Information Alert Scenario

Now that we have finished describing our proposed algorithm, we describe a whole information alert scenario where our algorithm could also be used.

Let us assume that a geneticist studies the genome of an organism. After he has cloned and sequenced the DNA of that organism, he has tens of sequences representing segments of the whole genome of the organism. His purpose is to identify the functionality of these sequences and locate their physical positions on the genome. That's why before anything else, he compares these sequences to public genomic databases looking for others with similarities and identified functionality. If the search results to great similarity with sequences of already studied organisms, then precious information can be extracted for the under study organism. If it does not, it is likely that the non informative search of today might be informative in the near future, due to the great amount of sequences that become publicly available almost every week. Consequently, it is crucial for the geneticist to be informed of anything new that concerns his study. Under an information alert scenario, the researcher will be alerted as soon as a new sequence becomes known and only if the comparison against his set of sequences results in a satisfying similarity. The similarity that is satisfying can be different for each sequence and is determined by him. Concluding, the user will be alerted if and only if a higher percentage of similarity than the given one comes up, even for just one of the sequences.

An information alert scenario can be implemented through a system, the graphical representation of which is shown in Figure 1.1. We remind that each user will submit a profile to the system setting his preferences and requirements.

In our case, this profile will contain the set of sequences and the minimum similarity percentage that satisfies the user's criteria for each of them. Moreover, because we search for similarity among sequences, we can use the *SeedsIndex* algorithm 4.2 as the search algorithm. In such a case the database sequences will correspond to the set sequences of all of users' profiles while the query sequence will correspond to any new incoming sequence that becomes known to the system.

## 4.5 Summary

In this chapter we have presented in detail the main memory algorithm *SeedsIndex*. We have also evaluated this algorithm experimentally using real genomic data and compared to the original BLAST algorithm. Finally, we described an information alert scenario where *SeedsIndex* could be used.

# Chapter 5

# Implementation of BioAlert, an Information Alert System

After the previous chapter and the discussion of our proposed algorithm for sequence comparison, we continue with this chapter where we present an alert system specialized to be used by biologists. As we discussed in Chapter 1, biologists have to cope with the problem of the great quantity of information available and the problem of being informed of that new information, something that is crucial for their research. An alert system focused on biologists' needs can be a good solution to these problems. Thus, we implemented such a system which was named BioAlert and is part of the European genomic project *Bridgemap* [2].

In the first part of this chapter we present the data model and the query language which was defined for that system. The second part is dedicated to the system itself, presenting a detailed description of its implementation.

We must note that in the following sections we make use of the terms *document* and *profile*[1] as they are used in the system DIAS discussed in Section 2.1.2. These terms correspond to Swiss-Prot *entries* and *user preferences* respectively.

## 5.1 Data Model and Query language

In this section we present the data model and the query language for our system following the formal approach of [12, 23, 25]. This data model is based on free text and on *attributes* or *fields* with finite-length strings as values. Formally, the meaning of free text is captured by the concepts of *text value* and *sequence value* which are defined. The query language is based on the *Boolean* model of Information Retrieval (IR) and it offers the comparison operators "contains" and "similar" on the attribute values.

[1]The term query can also be used because in an information alert setting, a profile is simply a long-standing query. Thus, in this chapter we will use these terms interchangeably.

### 5.1.1 Syntax

We assume that we have a finite *alphabet* $\Sigma$. A *word* is a finite non-empty sequence of letters from $\Sigma$. We also assume the existence of an infinite set of words called the *vocabulary* and denoted by $\mathcal{V}$.

**Definition 12** *A* text value *$s$ of length $n$ over vocabulary $\mathcal{V}$ is a total function $s : \{1, 2, \ldots, n\} \rightarrow \mathcal{V}$.*

In other words, a text value $s$ is a finite sequence of words drawn from the vocabulary $\mathcal{V}$. For each text value $s$, $s(i)$ denotes the $i$-th element of $s$ while its length (i.e., its number of words) will be denoted by $|s|$. Text values can be used to represent finite-length strings consisting of words separated by blanks.

**Example 14** *In all the examples, our vocabulary will be the vocabulary of the English language. The string*

*The human genome almost is complete*

*can be represented by a text value $s$ of length 6 over the English vocabulary with $s(1) = The$, $s(2) = human$ etc. The text value "human genome" is included in $s$.*

In order to define the concept of *sequence value*, we assume the existence of the alphabet $\Sigma_{amino}$, a finite set of twenty letters representing the twenty amino acids found in proteins, as it is shown in Table 3.2.

**Definition 13** *A* sequence value *$s$ of length $n$ over alphabet $\Sigma_{amino}$ is a total function s: $\{1,2,\ldots,n\} \rightarrow \Sigma_{amino}$.*

In other words, a sequence value is an ordered succession of characters drawn from an alphabet $\Sigma_{amino}$. For each sequence value s, $s(i)$ denotes the $i$-th character of $s$ while its length will be denoted by $|s|$. If the length of a sequence value is 0, then we say that we have an *empty sequence value* that will be denoted by $\varepsilon$.

**Example 15** *The succession of amino acids*

*GDVEKGKKIFVQKCAQCHTV*

*is a sequence value of length 20.*

Let us assume that we have a finite set of attributes called the *attribute universe*, which will be denoted by $\mathcal{A}$. In our case the attributes will come from the namespaces of biological applications. Specifically, they will come from the attributes of the entries of Swiss-Prot database. In Section 2.2.1 a representative entry of Swiss-Prot was discussed.

For our queries we use two types of operators. The first one is the *contains* operator. It is denoted by the symbol $\sqsupseteq$ and is applied to text values. A formula that uses the *contains* operator is an expression of the form $A \sqsupseteq s$, where $A$ is an attribute and $s$ is a text value. The second operator is the operator of *similarity*, that will be denoted by the symbol $\sim$. A similarity formula is an expression of the form $A \sim t$, where $A$ is an attribute and $t$ is a sequence. The similarity operator is used to capture the concept of similarity between the sequences as it is defined in Section 3.1 of Chapter 3.

A document $d$ is a set of attribute-value pairs $(A, t)$ where $A \in \mathcal{A}$, $t$ is either a text value over $\mathcal{V}$ or a sequence value over $\Sigma_{amino}$, and all attributes are distinct. In our case a document will be an entry of the Swiss-Prot database.

**Example 16** *An example of a document is presented in Figure 5.1.*

A query is a conjunction of the form

$$A_1 \sqsupseteq s_1 \wedge \ldots \wedge A_n \sqsupseteq s_n \wedge A_{n+1} \sim t$$

where each $A_i \in \mathcal{A}$, each $s_i$ is a text value, each $t$ is a sequence value and $\sqsupseteq$, $\sim$ are the contains and similarity operators respectively.

**Example 17** *The following formula is a query:*

$$ORGANISM\ CLASIFICATION \sqsupseteq \text{“}Bacteria;\text{”} \wedge$$

$$DESCRIPTION \sqsupseteq \text{“}Trypsin - like\ protease\ precursor\text{”} \wedge$$

$$\begin{aligned} SEQUENCE\ DATA \sim \text{“}&MLTVTTLVQL\ MKRTLAVGAV\ ALAAVSLQPG \\ &TATAGPAPVV\ GGTRAAQGEF\ PFMVRLSMGC \\ &GGALYTQQIV\ LTAAHCVSGS\ GNNTSITATA \\ &GVVDLNSSSA\ \ IKVKSTKVLQ\ \ APGYNGKGKD \\ &WALIKLAKPI\ \ NLPTLKIADT\ \ KAYDNGTFTV \\ &AGWGAAREGG\ \ GQQRYLLKAN\ \ VPFVSDASCQ \\ &SSYGSDLVPS\ \ EEICAGLPQG\ \ GVDTCQGDSG \\ &GPMFRRDNNN\ \ AWIQVGIVSW\ \ GEGCARPNYP \\ &GVYTEVSTFA\ \ AAIKSAAAGM\text{”} \end{aligned}$$

*A second one is:*

$$Species \sqsupseteq \text{“}Homo\ sapiens;\text{”} \wedge$$

$$Description \sqsupseteq \text{“}Trypsin - like\ protease\ precursor\text{”}.$$

```
ID   TRYP_STREX     STANDARD;      PRT;   260 AA.
AC   P80420; Q6U1K3;
DT   01-NOV-1995 (Rel. 32, Created)
DT   01-OCT-2004 (Rel. 45, Last sequence update)
DT   01-OCT-2004 (Rel. 45, Last annotation update)
DE   Trypsin-like protease precursor (EC 3.4.21.-).
GN   Name=tlp;
OS   Streptomyces exfoliatus (Streptomyces hydrogenans).
OC   Bacteria; Actinobacteria; Actinobacteridae; Actinomycetales;
OC   Streptomycineae; Streptomycetaceae; Streptomyces.
OX   NCBI_TaxID=1905;
RN   [1]
RP   SEQUENCE FROM N.A.
RC   STRAIN=SMF13;
RA   Lee D.H., Kim Y.-H., Kim I.S., Lee K.J.;
RT   "Role of trypsin-like protease on morphological differentiation of
RT   Streptomyces exfoliatus SMF13.";
RL   Submitted (SEP-2003) to the EMBL/GenBank/DDBJ databases.
RN   [2]
RP   SEQUENCE OF 39-58.
RC   STRAIN=SMF13;
RX   MEDLINE=95291424; PubMed=7773379;
RA   Kim I.S., Lee K.J.;
RT   "Physiological roles of leupeptin and extracellular proteases in
RT   mycelium development of Streptomyces exfoliatus SMF13.";
RL   Microbiology 141:1017-1025(1995).
CC   -!- FUNCTION: Involved in mycelium differentiation.
CC   -!- SIMILARITY: Belongs to peptidase family S1.
CC   --------------------------------------------------------------------------
CC   This SWISS-PROT entry is copyright. It is produced through a collaboration
CC   between  the Swiss Institute of Bioinformatics  and the  EMBL outstation -
CC   the European Bioinformatics Institute.  There are no  restrictions on  its
CC   use  by  non-profit  institutions as long  as its content  is  in  no  way
CC   modified and this statement is not removed.  Usage  by  and for commercial
CC   entities requires a license agreement (See http://www.isb-sib.ch/announce/
CC   or send an email to license@isb-sib.ch).
CC   --------------------------------------------------------------------------
DR   EMBL; AY380806; AAQ88430.1; -.
DR   MEROPS; S01.101; -.
DR   InterPro; IPR001254; Peptidase_S1.
DR   PROSITE; PS50240; TRYPSIN_DOM; 1.
DR   PROSITE; PS00134; TRYPSIN_HIS; 1.
DR   PROSITE; PS00135; TRYPSIN_SER; 1.
KW   Direct protein sequencing; Hydrolase; Serine protease; Signal;
KW   Zymogen.
FT   SIGNAL        1     34       Potential.
FT   PROPEP       35     38       Activation peptide.
FT   CHAIN        39    260       Trypsin-like protease.
FT   ACT_SITE     75     75       Charge relay system (By similarity).
FT   ACT_SITE    120    120       Charge relay system (By similarity).
FT   ACT_SITE    209    209       Charge relay system (By similarity).
FT   DISULFID     60     76       By similarity.
FT   DISULFID    179    194       By similarity.
FT   DISULFID    205    234       By similarity.
FT   CONFLICT     39     39       V -> R (in Ref. 2).
FT   CONFLICT     49     49       E -> N (in Ref. 2).
FT   CONFLICT     53     54       MV -> QQ (in Ref. 2).
SQ   SEQUENCE   260 AA;  26593 MW;  9B56032D44F8490B CRC64;
     MLTVTTLVQL MKRTLAVGAV ALAAVSLQPG TATAGPAPVV GGTRAAQGEF PFMVRLSMGC
     GGALYTQQIV LTAAHCVSGS GNNTSITATA GVVDLNSSSA IKVKSTKVLQ APGYNGKGKD
     WALIKLAKPI NLPTLKIADT KAYDNGTFTV AGWGAAREGG GQQRYLLKAN VPFVSDASCQ
     SSYGSDLVPS EEICAGLPQG GVDTCQGDSG GPMFRRDNNN AWIQVGIVSW GEGCARPNYP
     GVYTEVSTFA AIKSAAAGM
//
```

Figure 5.1: A document in the Swiss-Prot style

### 5.1.2 Semantics

Let us now define the semantics of the above query language in our setting. We start by defining when a document satisfies a query.

**Definition 14** *Let $\mathcal{D}$ be a document schema, $d$ a document over $\mathcal{D}$ and $\phi$ a query over $\mathcal{D}$. The concept of document $d$* satisfying *query $\phi$ (denoted by $d \models \phi$) is defined as follows:*

1. *If $\phi$ is of the form $A \sqsupseteq s$ then $d \models \phi$ iff there exists a pair $(A, \mathcal{V}) \in \lceil$ and $s$ is contained in $\mathcal{V}$.*
2. *If $\phi$ is of the form $A \sim t'$ then $d \models \phi$ iff there exists a pair $(A, t) \in d$ and the the global of $t$ and $t'$ as computed by the BLAST algorithm is greater than $0$.*
3. *If $\phi$ is of the form $\neg\phi_1$ then $d \models \phi$ iff $d \not\models \phi_1$.*
4. *If $\phi$ is of the form $\phi_1 \wedge \phi_2$ then $d \models \phi$ iff $d \models \phi_1$ and $d \models \phi_2$.*
5. *If $\phi$ is of the form $\phi_1 \vee \phi_2$ then $d \models \phi$ iff $d \models \phi_1$ or $d \models \phi_2$.*

**Example 18** *The first query of Example 17 is satisfied by the document of Example 16 while the second one is not satisfied.*

## 5.2 System description

Now that we have presented the data model and its corresponding query language and before we give details about system's implementation we give a description of the system and discuss the algorithm that is used.

Our goal is to build an alert system where users express their desire for information by posting profiles or long-standing queries to the system and the system informs them whenever the incoming information satisfies their profile. Such systems for information dissemination have been studied and implemented (e.g., SIFT [38], Hermes [14] and DIAS [25]).

In our case we cope with textual information however part of this information is genomic sequences, as they are presented in Section 3.1. *BioAlert* has been implemented to combine the algorithms for the text filtering and the algorithms for sequence comparing, explained in Chapter 3. A definition for the filtering problem as it arises in textual information dissemination systems is as follows: Given a database of profiles *db* and an incoming document *d*, find all profiles $q \in db$ that match *d*.

Before we continue we must remind the reader that our current implementation of *BioAlert* is based on a related service, which is called Swiss-Shop [8] and was presented in Chapter 2.

### 5.2.1 Functionalities

Let us now give a brief description of the functionalities that our system supports. Each user may pose his desire for information to the system. This user's requirement is called a *profile*. A user may have more than one profiles posed to the system and there is not a maximal number of profiles that a user can pose to the system. Apart from the addition of new profiles, a user has the capability to remove any obsolete profiles from the system as well as to list his own profiles.

Let us see now what pieces of information compose a *profile*. A profile is composed by the text part and/or the sequence data part. A keyword based search is performed on the text part while a sequence based search is performed on the sequence data part. The text part is separated in attributes and for each of them one or more keywords are given by the user. The attributes are conjunctive to each other. This means that a profile will match to the new incoming information if and only if the words of all of the attributes are contained to the new information. Due to the fact that our implementation is based on Swiss-Prot database [7], the sequence data are proteins and the attributes of the text part correspond to the fields of a Swiss-Prot entry. The format which follows a Swiss-Prot entry was presented in Section 2.2.1 of Chapter 2.

On the contrary to the profiles that a user poses to the system, the new incoming information is retrieved by the system itself. In our implementation the new incoming information is the entries of the latest update of the Swiss-Prot database. As soon as we have new information a search is performed by the system. The users are alerted as soon as a search has been performed and if and only if the search resulted to matches between his/her *profiles* and the new incoming information.

### 5.2.2 The Algorithm

Now that we have given a description of our system and described its functionalities, we discuss the algorithm that we use.

Because our system handles both textual and biological information, we first present the algorithm used for each kind of information involved and then we present the algorithm, that we designed and implemented, and which combines the other two algorithms.

#### Algorithm for the Text Part

For the text part we chose to use one of the four algorithms that was studied and implemented in [36]. It is named Tree and it is the main memory version of one of the algorithms that were originally developed in SIFT [38]. Tree algorithm introduces a *trie-like* data structure for profile storage in order to exploit the similarities between the profiles during filtering. On the following

| **Profile** | **Identifying prefix** |
|---|---|
| $P_1$: (a,b) | (a,b) |
| $P_2$: (a,d) | (a,d) |
| $P_3$: (a,d,e) | (a,d,e) |
| $P_4$: (b,f) | (b) |
| $P_5$: (c,d,e,f) | (c) |
| $P_6$: (d,f,g) | (d) |
| $P_7$: (d,f,g) | (d) |

Table 5.1: A set of profiles and their identifying prefixes

lines we present how the algorithm works verbatim from [36] and we start by presenting some definitions first.

**Definition 15** *Let* p *be a profile* ($w_1$,$w_2$,...,$w_k$) *of* k *words, and* $0 \leq i < k$. *The* ($w_1$,$w_2$,...,$w_i$) *is called a* prefix *of p with* ($w_{i+1}$,...,$w_k$) *its corresponding* postfix.

**Definition 16** *Let P be a set of profiles. A prefix* $(w_1, w_2, \ldots, w_i)$ *identifies a profile p in P if* $i = k$ *or if there is no other profile in P, except those identical to p, that have* $(w_1, w_2, \ldots, w_i)$ *as a prefix. The shortest prefix that identifies a profile is called the* identifying prefix *of that profile.*

As profiles arrive from subscribing users, TREE organises their identifying prefixes into a profile tree. The root of the profile tree (level 0) corresponds to the empty prefix. A node $n$ at level $i$ corresponds to a prefix $\sigma = (w_1, w_2, \ldots, w_i)$ of some identifying prefixes. All prefixes identical to $\sigma$ are represented by this node $n$. Its children are nodes corresponding to prefixes $(w_1, w_2, \ldots, u)$ of some identifying prefixes (where $u$ is a word). In Figure 5.2 we present an example of a profile tree which corresponds to the profiles of Table 5.1. As the Figure 5.2 presents, each node has attached a linked list of profile identifiers of which $\sigma$ is the identifying prefix and a second linked list of words that form the postfix of the identifying profiles. To speed up the search, a hash table is used to index the children of the root of the profile tree. Thus, a profile tree T is actually implemented as a forest where pointers to the root nodes of all trees rooted at level 1 are stored in a hash table called the root hash table of T.

As a document comes along, the profile tree is searched in a breadth-first way to discover all matching profiles. More details can be found in [36].

Tree algorithm follows one of the three existing and classical Information Retrieval models, the *Boolean* model. The other two are the *vector* and the *probabilistic* model, while there are other numerous alternatives for each type. We must note that the above models refer to the text context of a document. However, there are other models that refer to the structure of a written text. We mention some words for the boolean model, verbatim from [36]. The boolean

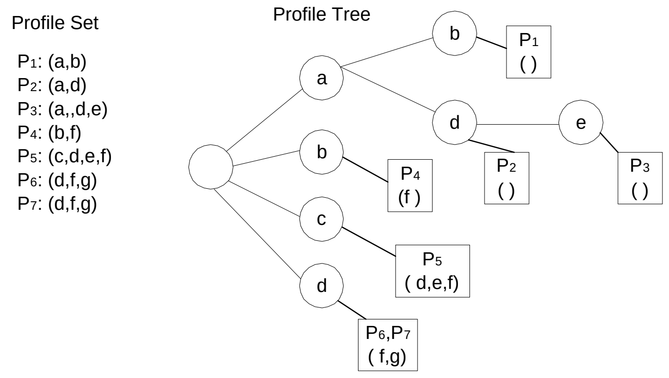


Figure 5.2: An example of the profile tree constructed by the TREE algorithm.

model considers that index terms are present or absent in a document. Index term is a word whose semantics are considered to capture the main theme of a document. Thus, a document is described by a set of representative keywords called index terms. Because some terms capture a documents's content better than others, we have introduced the meaning of weight. Weight is a number which quantifies the importance of an index term for describing a document's content. As a result, the index term weights are all assumed to be binary. A query is composed of index terms linked by three connectives: *not*, *and*, *or*. Thus a query is a conventional boolean expression which can be representative as a disjunction of conjunctive vectors. A similarity of a document to a query is defined as the match of the query vector keyword values with the corresponding document vector values. The above mentioned similarity function outputs one, if there is relevance between the document and the query relevance or zero otherwise. There is no notion of a partial match to the query conditions, so the boolean model is more of a data model than an information retrieval model. In [23],[25] sophisticated models and query languages for textual information dissemination are proposed. The more sophisticated data model *AWPS* and its corresponding language, is an attribute-based model which uses linguistically motivated concepts such as word and not arbitrary strings, and allows boolean and proximity queries along with a "similarity" operator based on the vector space model. The latest query language proposed by the same group can be found published in [12].

As we remarked the text part of profiles supports attributes. However, TREE algorithm does not follow an attribute-based model, but instead the *Boolean* model as described above. Consequently, we adjusted our implementation to take care of this incompatibility. Because each attribute is conjunctive to others, we apply the TREE algorithm to each attribute independently, as we explain in Section 5.2.2. Finally, we combine the results conjunctively in order to have the final result. Thus, a profile matches a text if and only if all the attributes matches the text.

```
function AddProfile (list AttributeList, UserInfo ui) {
          Profile p;
          int i, attrNum;
          p = setProfileCharacteristics();
          attrNum = AttributeList.length;
          for each  attribute $attr \in$ AttributeList except attr seqdo
                    TREEInsertProfile(p, ui, i, attrNum);
                    i++;
          end for
          if attribute seq != NULL then
             insertSeq();
          end if
}
```

Figure 5.3: Pseudo-code of the algorithm for adding a profile in BioAlert

## Algorithm for the Sequence Part

For the sequence part, we use the BLAST algorithm as the sequence comparison algorithm. A detailed presentation for this algorithm can be found in Section 3.4.2. We must note that for our implementation we downloaded and installed the latest version of the stand-alone version of the algorithm which is distributed by NCBI[2].

As we discussed we have protein sequences so we need and call the blastp program. We also use the *formatdb* program to get the database ready so as to be used by blastp program. Moreover, all the sequences that we handle are stored in FASTA format. This format is supported by BLAST. We compare each sequence against a database of protein sequences. This database of protein sequences is constituted by the protein sequences that each user pose to the system with his profile. Just one protein sequence is submitted with each profile and it is called *profile sequence*. So, we accumulate all the profile sequences and these constitute the database against which a BLAST search will be run. Thus, each time a new profile is posed to the system, the protein sequence is added to the database and this is formatted with *formatdb*.

## The general Algorithm

Now that we presented the two algorithms, we present the algorithms for adding a profile and the matching procedure for *BioAlert* service. In Figure 5.3 the pseudo-code of adding a profile to *BioAlert* is shown. As one can notice, the function takes as input a list with the attribute values (*AttributeList*) of the profile, that will be added, and some user information (*ui*) of the user that owns that profile.

[2]National Center for Biotechnology Information

For each text attribute the TREEInsertProfile function is called which adds the specified attribute value to a tree-like structure according to TREE algorithm. Consequently, we have as many tree-like structures as the number of attributes supported by our service. In each tree-like structure the TREE algorithm is applied while they are not organized or stored in a distinct data structure. In this way, we implement an attribute-based model using an algorithm that follows the boolean model.

If the user's profile contain a protein sequence, and consequently the attribute *seq* has been set, the function insertSeq is applied to that sequence attribute *seq*. This function adds the protein sequence, that the user submits, to a file in FASTA format and suitably formats it so as to be ready to be used by BLAST.

In Figure 5.4 the pseudo-code of the matching algorithm of *BioAlert* is presented. We notice that we have three main categories concerning the profiles. We have the profiles that contain only textual information (*onlyText*), these that contain only biological information (*onlySeq*), that is protein sequences, and those that contain both (*bothTextSeq*). In case that we have profiles that contain textual information, we apply the matching procedure which follows the TREE algorithm and is described in subsection 5.2.2. This is coded by the function matchText and its pseudo-code is presented in the Figure 5.5. In case that we have profiles that contain only protein sequences, we just run a BLAST search against them and the new incoming sequences. In the third case that we have profiles that contain both kind of information, we run both blast search and the text matching. The final result arises from the conjunction of the two results. Finally, in each case the users that have to be notified are notified by sending them an email.

The function matchText does the actual matching. It accepts as input the document *Doc* for which the matching will be done. Its output is a list of the profiles that match that document. For each attribute the TREEMatch function that applies the TREE matching algorithm is called. Because each attribute is conjunctive to others, the final list contains only those profiles for which matches have been found for all of their attributes. This is what the last two *for* loops implement.

## 5.3 System organisation

Now that we have presented the data model and the query language and we have given a detailed description of the system itself and the used algorithm, we continue with a presentation of its high level organisation by presenting some basic modules of it. Finally, we discuss some technical issues.

A high level view of the internal organisation of *BioAlert* is shown in Figure 5.6. As we can see, a GUI mediates between the user and the system forwarding the demands of the former to the latter and messages from the system to the user. There is also a number of modules which represent the supported functionalities.

```
function match (Document doc) {
        list onlySeq, onlyText, bothTextSeq;
        list matchedProfiles, finalList;
        if  onlySeq != NULL then
            for each  profile sequence $q \in$ onlySeq do
                          res = Blast($q$, $newSeqs$, $p$);
                          notifyUser($q.emailAddress$, $res$);
            end for
        end if
        if onlyText != NULL || bothTextSeq != NULL then
          matchedProfiles = matchText($doc$);
          if onlyText != NULL then
             finalList = matchedProfiles - (profiles $\in$ bothTextSeq);
             for each  profile $p \in$ finalList do
                          notifyUser($p.emailAddress$);
             end for
          end if
          if bothTextSeq != NULL then
             finalList = matchedProfiles - (profiles $\in$ onlyText);
             res = Blast(doc.seq, finalList.seqs);
             finalList = finalList $\wedge$ res;
             for each  profile $p \in$ finalList do
                          notifyUser($p.emailAddress$);
             end for
          end if
        end if
}
```

Figure 5.4: Pseudo-code of the algorithm for the matching procedure of BioAlert

```
function matchText(Document Doc) {
    list res, MatchedProfiles;
    int matchedAttr;
    for each attribute attr do
        matches = TREEMatch(Doc);
        addTo(res, matches);
    end for
    for each entry of profile sequence $q \in$ res do
        updateProfile($q$, matchedAttr++);
    end for
    for each profile sequence $q$ do
        if q.matchedAttr == q.AttrNum then
            addTo(MatchedProfiles, $q$);
        end if
    end for
    return MatchedProfiles;
}
```

Figure 5.5: Pseudo-code of the algorithm for the text matching

**Connect/disconnect** This module refers to the user's opportunity to connect or to disconnect to or from the system. While a user is connected to the system, can use the rest the functionalities. When he finishes and desires to terminate the connection, he selects the functionality of disconnection.

**Add a new profile** The users have the capability to pose profiles to the system. Because of the two parts of a profile, a profile parser is responsible to separate the text data from the sequence data and save them in different data structures. We referred these structures in Section 5.2.2 when we presented the algorithms we used.

**Create new user account** When a user wants to connect and use the system for the first time, he must create an account. He defines the information that the system requires as well as a password and a user name that are unique. All these information is stored in a data structure and are available any time the user tries to connect to the system.

**List all the existing profiles for a user** This functionality provides the users with a list of all of their profiles posed to the system. Having that list, they are capable to edit it by adding or removing profiles from it.

**Remove an existing profile** Each user has the capability to remove any obsolete profiles. These profiles are actually filtered out from the final answer

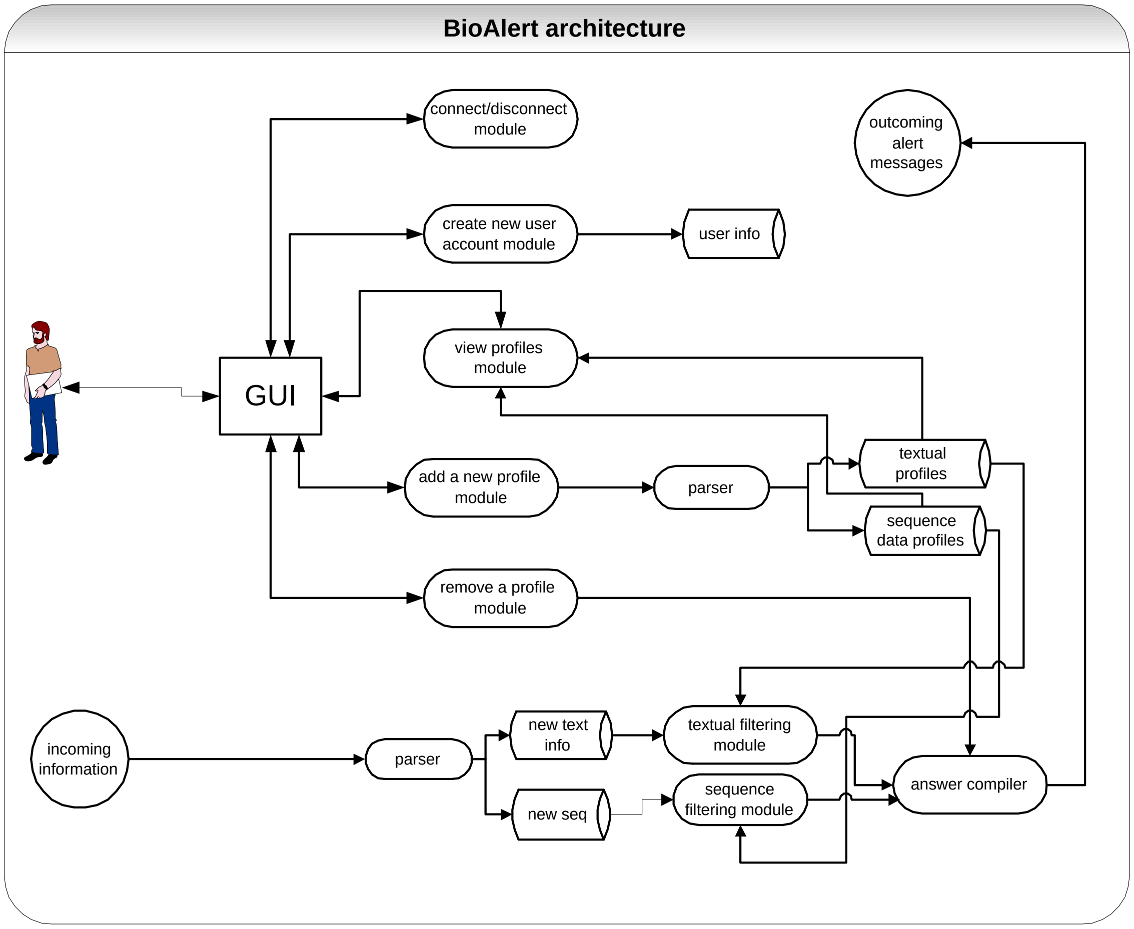
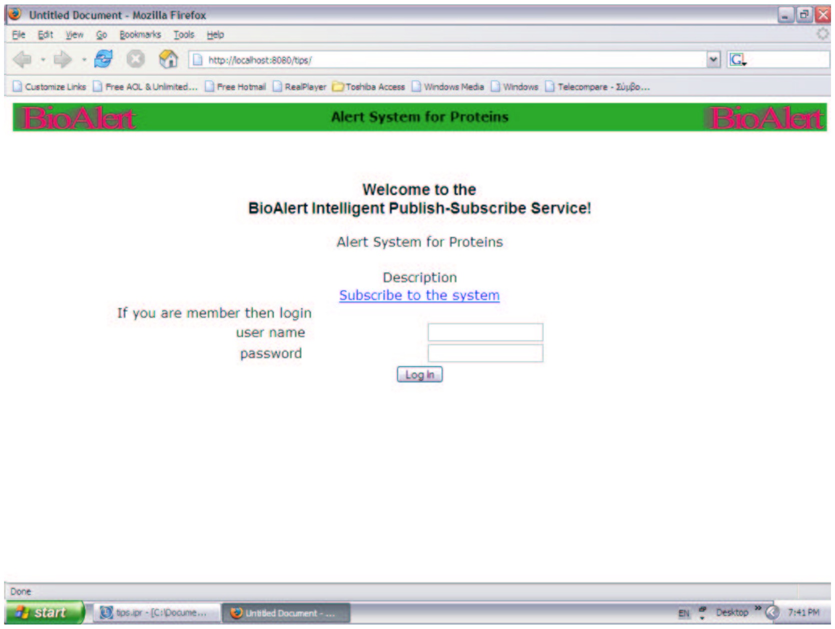


Figure 5.6: BioAlert organisation


Untitled Document - Mozilla Firefox
http://localhost:8080/tips/
BioAlert
Alert System for Proteins
BioAlert
Done

Welcome to the
BioAlert Intelligent Publish-Subscribe Service!

Alert System for Proteins

Description
Subscribe to the system

If you are member then login
user name
password
Log In

Figure 5.7: BioAlert's main page.

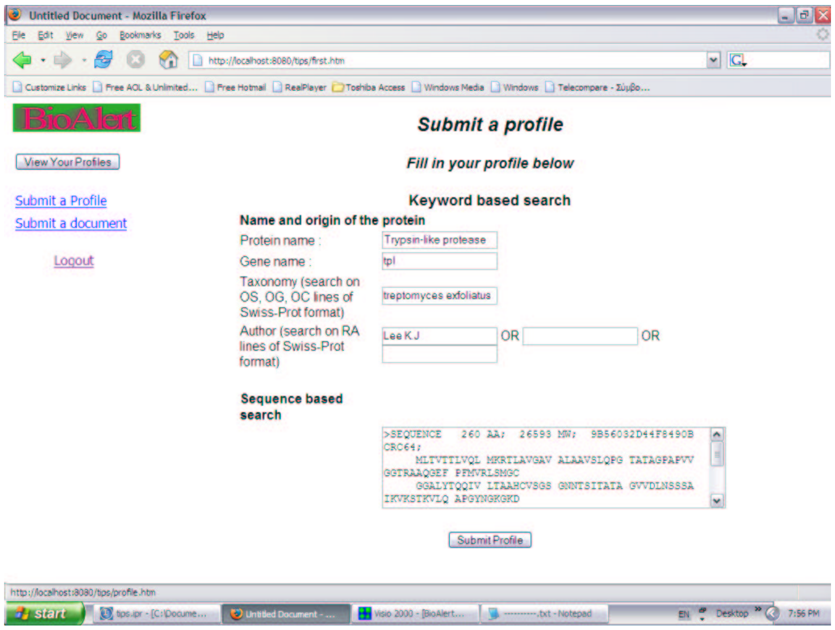
list.

Although all the above capabilities are user-initiative, the functionality to alert the users is not. As we can see in Figure 5.6, the incoming information after a parsing is separated in textual and sequence data and queued for the filtering procedure. Only when both processes have been terminated, the final answer list is compiled. Having the final answer list, the users are alerted by email.

Before we continue with a comparison of the system and the existing service of Swiss-Shop, we present some screen-shots of *BioAlert*. In Figure 5.7 the main page of BioAlert is depicted. In Figures 5.8, 5.9 the form for specifying a profile and a list showing the user's profiles are presented respectively.

## 5.4 BioAlert vs. Swiss-Shop

In this section we present how our system *BioAlert* differs from *Swiss-Shop* service. Both of them are services that allow the specification of long-standing queries on textual as well as sequence data of the same protein database. The protein database of Swiss-Prot is used by the systems because its entries are annotated very well. Apart from the sequence data of proteins, descriptions, comments, cross-references to other databases and many other information are given. So, it is meaningful to search on textual as well as sequence data. This functionality is supported by both systems. In both systems, a user can specify


Untitled Document - Mozilla Firefox
http://localhost:8080/tips/first.htm
BioAlert
View Your Profiles
Submit a Profile
Submit a document
Logout
Submit Profile
http://localhost:8080/tips/profile.htm

**Submit a profile**

*Fill in your profile below*

**Keyword based search**

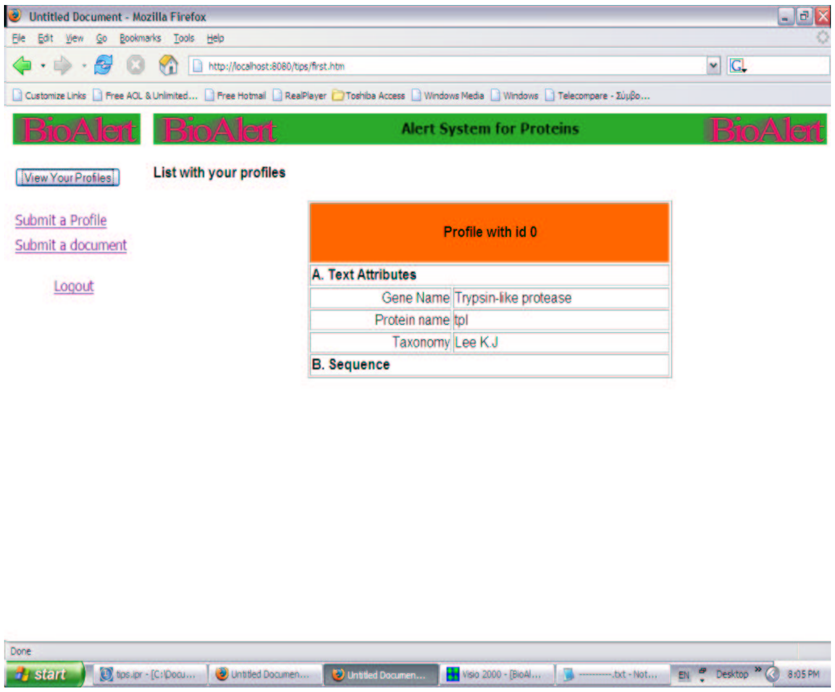
**Name and origin of the protein**

| | |
|---|---|
| Protein name : | Trypsin-like protease |
| Gene name : | tpl |
| Taxonomy (search on OS, OG, OC lines of Swiss-Prot format) | treptomyces exfoliatus |
| Author (search on RA lines of Swiss-Prot format) | Lee K.J OR OR |

**Sequence based search**

```
>SEQUENCE   260 AA;  26593 MW;  9B56032D44F8490B
CRC64;
     MLTVTTLVQL MKRTLAVGAV ALAAVSLQPG TATAGPAPVV
GGTRAAQGEF PFMVRLSMGC
     GGALYTQQIV LTAAHCVSGS GNNTSITATA GVVDLNSSSA
IKVKSTKVLQ APGYNGKGKD
```

Figure 5.8: Specifing a long-standing query in BioAlert.


Untitled Document - Mozilla Firefox
http://localhost:8080/tips/first.htm
BioAlert
Alert System for Proteins
View Your Profiles
Submit a Profile
Submit a document
Logout
Done

**List with your profiles**

| **Profile with id 0** | |
|---|---|
| **A. Text Attributes** | |
| Gene Name | Trypsin-like protease |
| Protein name | tpl |
| Taxonomy | Lee K.J |
| **B. Sequence** | |

Figure 5.9: Viewing profiles in BioAlert.

a long-standing query where he will specify a protein given its sequence data or he will specify keywords. In the first case only the entries with similar proteins to the specified one are searched while in the second case only the entries that contain that keywords are searched. For the textual searches we use a simple query language that supports the functionality we want. Other more expressive languages such as these ones that are presented in [12, 37] can replace it.

However, the main difference of our system to *Swiss-Shop* is that our system permits both textual and sequence similarity searches to be involved to the specification of the same query. Users set queries where they search database entries which they match or contain the specified textual data and at the same time the protein sequence is similar to the specified one. In this way, users are able to specify more complex queries and receive meaningful and more informative answers.

We continue and present some technical details about the system.

## 5.5 Technical details

*BioAlert* system described in Section 5.2 was implemented in Java and it runs on a PC, with a Pentium II 1.3GHz processor, with 256 MB RAM running Windows XP. The interface is implemented using JavaServerPages technology. For that reason the latest version of Tomcat sever has been installed. Because of the mails that are sent to the users, a mail server, called Free SMTP Server, has also been installed. The stand-alone version of the latest BLAST algorithm has also been installed. Because the TREE algorithm that we used has been implemented in C, we used JNI technique in order to integrate and call it from Java. Finally, our implementation makes use of JavaBeans and Servlet technology so as to make the whole system to work.

## 5.6 Summary

In this chapter we discussed in detail *BioAlert*, the alert system that we implemented. We presented the data model and the query language we used as well as the algorithm we implemented. A high level organisation of the system and the functionality supported by it was also presented. Finally, we discussed implementation issues as the technologies that we used for the system's development.

# Chapter 6

# Concluding Remarks

Now that this dissertation has come to an end, let us summarise our main achievements. We distinguish two parts of our work. In the first part, we focused our attention on sequence comparison, the fundamental operation in computational biology, while in the second part we concentrated on the problem of information dissemination focusing mainly on the view of molecular biology.

We started this work with a presentation of related systems which study the problem of information dissemination either on textual information or on the view of molecular biology.

We continued with a detailed and complete introduction to some molecular biology's meanings and terms. We presented definitions and examples as a helpful guide for a computer scientist. Having defined the basic terms that we use in this work, we presented basic algorithms for sequence comparison and we concentrated on the algorithm of BLAST which is also the most popular one. We proposed a variation of that algorithm that trades space for time. To prove our case we continued with a experimental evaluation of it and a comparison to the existing original algorithm. We designed a realistic scenario under which our algorithm and the original one were run using real genomic data which come from of the European project Bridgemap [2]. After several measurements that we made on the running time and the memory space for both algorithms, we made useful conclusions. The major one states that you have to spend a significant amount of memory space if you want to decrease the running time of the algorithm significantly.

In the second part, we put our focus on the problem of information dissemination. We designed and implemented an alert service for molecular biologists that deals with both textual and sequence data. Our system allows the specification of long-standing queries on textual as well as sequence data of protein databases. A very similar service was presented in Section 2.2.1 of Chapter 2. However, our system offers more functionality to users as it was stated in Section 5.4. Users specify long-standing queries either on textual or on sequence data, as they did using Swiss-Shop, but now they can also specify queries on both types of data.

Moreover, we presented in detail the data model, the query language and the algorithm that we used for the implementation of our system.

## 6.1 Future Work

As for the alternative sequence comparison, we can address its implementation for the case of protein sequences, too. Since the original algorithm is implemented to work using more than one processors, it can be designed carefully to work as a multiprocessor algorithm.

Concerning the implementation of the alert system, it can be extended to use other databases apart from Swiss-Prot. There are also several improvements that can be implemented such as editing and deleting the profiles from the data structures of the algorithm. Moreover, the use of other models than the boolean one can be used to implement such a service which will support a more expressive language. Such models and languages have been proposed by our laboratory as you can read in [23, 25]. Finally, such a service would be interesting to be implemented in terms of a P2P system [19, 18, 20].

# Bibliography

[1] BLAST Home page. http://ncbi.nlm.nih.gov/BLAST/.

[2] Bridgemap website. http://www.bridgemap.tuc.gr.

[3] http://au.expasy.org/proteomics_def.html.

[4] http://cbr-rbc.nrc-cnrc.gc.ca/srs6bin/cgi-bin/wgetz?-page+top+-id+4looq1l3ccg.

[5] http://www.isb-sib.ch/.

[6] Prosite website. http://au.expasy.org/prosite/.

[7] Swiss-prot website. http://au.expasy.org/sprot/.

[8] Swiss-shop home page. http://au.expasy.org/swiss-shop/.

[9] Uniprot website. http://www.expasy.uniprot.org/index.shtml.

[10] Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. Basic Local Alignment Search Tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

[11] D. L. Brutlag, J. P. Dautricourt, S. Maulik, and J.Relph. Improved sensitivity of biological sequence database searches. *Computer Applications in Biosciences*, 6:237–45, 1990.

[12] M. Koubarakis C. Tryfonopoulos and Y. Drougas. Filtering Algorithms for Information Retrieval Models with Named Attributes and Proximity Operators. In *Proceedings of the 27th Annual ACM SIGIR Conference*, July 25-July 29, Sheffield, United Kingdom 2004.

[13] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Achieving scalability and expressiveness in an internet-scale event notification service. In *Proceedings of the 19th ACM Symposium on Principles of Distributed Computing (PODC'2000)*, pages 219–227, 2000.

[14] D. Faensen and L. Faulstish and H. Schweppe and A. Hinze and A. Steidinger. Hermes- A Notification Service for Digital Libraries. In *Proceedings of theJoint ACM/IEEE Conference on Digital Libraries (JCDL '01).* Roanoke, Virginia, USA, 2001.

[15] Dan Gusfield. *Algorithms on strings, trees, and sequences: computer science and computational biology.* Cambridge University Press, 1999.

[16] Hedvig Hegyi, Jen-Mai Lai, and Peer Bork. The Sequence Alerting Server - a new WEB server. *CABIOS APPLICATIONS NOTE*, 13(6):619–620, 1997.

[17] S. Henikoff and J. G. Henikoff. Amino acid substitution matrices from protein blocks. *Proc. Natl. Academy Science*, 89(10):915–19, 1992.

[18] S. Idreos and M. Koubarakis. P2P-DIET: Ad-hoc and Continuous Queries in Peer-to-Peer Networks using Mobile Agents. In *3rd Hellenic Conference in Artificial Intelligence*, volume 3025, pages 23–32, Samos, Greece, May 5-8 2004.

[19] S. Idreos, M. Koubarakis, and C. Tryfonopoulos. P2P-DIET: An Extensible P2P Service that Unifies Ad-hoc and Continuous Querying in Super-peer Networks. In *Proceedings of the ACM SIGMOD/PODS 2004 Conference.*, Maison de la Chimie, Paris, France, June 13-18, 2004.

[20] Stratos Idreos, Manolis Koubarakis, and Christos Tryfonopoulos. P2P-DIET: One-Time and Continuous Queries in Super-peer Networks. In *Proceedings of the IX International Conference on Extending Database Technology (EDBT04). In LNCS* , volume 2992, pages 851–853, Heraklion, Crete, Greece, March 14-18, 2004.

[21] S. Karlin and S. F. Altschul. Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. *Proc. Natl. Academy Science*, 87:2264–68, 1990.

[22] S. Karlin and S. F. Altschul. Applications and statistics for multiple high-scoring segments in molecular sequences. *Proc. Natl. Academy Science*, 90:5873–77, 1993.

[23] M. Koubarakis, C. Tryfonopoulos, P. Raftopoulou, and T. Koutris. Data models and languages for agent-based textual information dissemination. In *Proceedings of the 6th International Workshop on Cooperative Information Agents (CIA2002)*, volume Lecture Notes in Computer Science, 2002.

[24] D. J. Lipman and W. R. Pearson. Rapid and sensitive protein similarity searches. *Science*, 227:1435–41, 1985.

[25] M. Koubarakis and T. Koutris and C. Tryfonopoulos and P. Raftopoulou. Information Alert in Distributed Digital Libraries: The Models, Languages and Architecture of DIAS. In *Proceedings of the 6th European Conference on Digital Libraries (ECDL2002)*, volume 2458 of Lecture Notes in Computer Science, page 527, 2002.

[26] M. McClure and T. Vasi and W. Fitch. Comparative analysis of multiple protein-sequence alignment methods. *Mol. Biol. Evolution*, 11:571–92, 1994.

[27] E. W. Mayers and W. Miller. Optimal alignments in linear space. *Computer Applications in Biosciences*, 4(1):11–17, 1988.

[28] E. W. Myers. A sublinear algorithm for approximate keyword searching. *Algorithmica*, 12:345–74, 1994.

[29] W. R. Pearson and D. J. Lipman. Improved tools for biological sequence comparison. *Proc. Natl. Academy Science*, 85:2444–2448, 1988.

[30] Pierre Baldi and Soren Brunal. *BIOINFORMATICS: The machine learning approach*. The MIT Press, second edition edition.

[31] R. D. Fleischmann and M. D. Adams and O. White and R. A. Clayton and E. F. Kirkness and A. R. Kerlavage and C. J. Bult and J. F. Tomb and B. A. Dougherty and J. M. Merrick, et al. Whole-genome random sequencing and assembly of Haemophilus influenzae Rd. *Science*, 269(496):496–512, 1995.

[32] R. F. Doolittle. Searching through sequence databases. In R. F. Doolittle, editor, *Methods in Enzymology Vol. 183. Molecular Evolution: Computer Analysis of Protein and Nucleic Acid Sequences*, pages 99–110. Academic Press, New York, 1990.

[33] R. Schwarz and M. Dayhoff. Matrices for detecting distant relationships. In M. Dayhoff, editor, *Atlas of Protein Sequences*, pages 353–58. Natl. Biomed. Res. Found., 1979.

[34] J. C. Setubal and J. Meidanis. *Introduction to Computational Molecular Biology*. PWS Publishing Company, 1997.

[35] SunOS5.5. *BLAST Manual Pages*, 1995.

[36] Theodoros Koutris. Textual Information Dissemination in Distributed Agent Systems: Architectures and Efficient Filtering Algorithms. Master's thesis, Technical University of Crete, Greece., 2003.

[37] C. Tryfonopoulos. Agent-Based Textual Information Dissemination: Data Models, Query Languages, Algorithms and Computational Complexity. Master's thesis, Technical University of Crete, Greece., 2002.

[38] T.W. Yan and H. Garcia-Molina. The SIFT information dissemination system. *ACM Transactions on Database Systems*, 24(4):529–565, 1999.

[39] W. R. Pearson. Effective protein sequence comparison. In R. F. Doolittle, editor, *Methods in Enzymology Vol. 266. Computer Methods for Macromolecular Sequence Analysis*, pages 227–58. Academic Press, New York, 1996.

[40] W. R. Pearson. Protein sequence comparison and Protein evolution. Tutorial - ISMB2000, 2001.

[41] Warren J. Ewens and Gregory R. Grant. *Statistical Methods in Bioinformatics: An Introduction*. Springer, 2001.