VOT 74289

THE DEVELOPMENT OF SEMANTIC META-DATABASE: AN ONTOLOGY BASED SEMANTIC INTEGRATION OF BIOLOGICAL DATABASES

(PEMBANGUNAN PENGKALAN DATA-META SEMANTIK : ONTOLOGI BERASASKAN INTEGRASI SEMANTIK KEPADA PENGKALAN DATA BIOLOGI)

RUHAIDAH BINTI SAMSUDIN
SAFAAI BIN DERIS
MUHAMAD RAZIB BIN OTHMAN
ROSLI BIN MD ILLIAS
SAFIE BIN MAT YATIM

RESEARCH VOT NO:
74289

Jabatan Kejuruteraan Perisian
Fakulti Sains Komputer Dan Sistem Maklumat
Universiti Teknologi Malaysia

2007

ABSTRACT

Protein sequence annotation is important for the preservation and reuse of knowledge, for content-based queries, and for the understanding of its function. Traditional wet-lab methods are labor intensive and prone to human error. Alternatively, existing tools are time intensive and require high investment in computing facilities for offline usage. On the other hand, these tools are highly dependent on internet stability and speed for online usage. Therefore, a simple and practical computational method that is more accurate, faster, easy to configure and use, and bears low computing cost is needed particularly for offline usage. In this study, a Gene Ontology (GO) based protein sequence annotation tool named *extended* UTMGO is developed to meet these features. The GO is selected because of its ability to provide dynamic, precisely defined, structured, and controlled terms that describe genes and their functions and products in any organism. Furthermore, the GO terms are linked with gene products and their protein sequences from various species provided by Gene Ontology Annotation (GOA). Thus, assigning highly correlated GO terms of annotated protein sequences to partially annotated or newly discovered protein sequences can be made. The tool comprises two intelligent algorithms. The first algorithm combines parallel genetic algorithm with the split-and-merge algorithm. The idea is to cluster the GO terms into number *k* of clusters in order to split the monolithic GO RDF/XML file into smaller files. Thus, it enables protein sequences and Inferred from Electronic Annotation (IEA) evidence associations to be included in those files. The second algorithm incorporates parallel genetic algorithm with the semantic similarity measure algorithm. The motive is to search for a set of semantically similar GO terms from the fragmented GO RDF/XML files to a given query. In addition, its basic version which is a GO browser based on semantic similarity search is also introduced to overcome the weaknesses of conventional approach: the keyword matching.

ABSTRAK

Penganotasian jujukan protein adalah penting untuk pemeliharaan dan penggunaan semula pengetahuan, pertanyaan berasaskan-kandungan dan pemahaman terhadap fungsinya. Kaedah makmal-basah tradisional adalah intensif buruh dan terdedah kepada ralat manusia. Sebagai alternatif, alatan sedia ada adalah intensif masa dan memerlukan pelaburan kemudahan pengkomputeran yang tinggi untuk penggunaan luar talian. Selain itu, ia sangat bergantung kepada kestabilan dan kelajuan internet untuk penggunaan dalam talian. Maka, kaedah komputasi yang mudah dan praktikal yang lebih tepat, pantas, mudah dikonfigurasi dan diguna serta dengan kos pengkomputeran yang murah diperlukan terutamanya untuk penggunaan luar talian. Dalam pengajian ini, alatan penganotasian jujukan protein berasaskan Ontologi Gen (GO) iaitu UTMGO *lanjutan* dibangunkan untuk memenuhi ciri-ciri tersebut. GO dipilih kerana keupayaannya menyediakan istilah yang dinamik, takrifan tepat, berstruktur dan terkawal yang menerangkan gen dan fungsi serta produknya dalam sebarang organisma. Tambahan pula, istilah GO dihubungkan dengan produk gen dan jujukan proteinnya daripada pelbagai spesies yang disediakan oleh Anotasi Ontologi Gen (GOA). Dengan itu, penentuan istilah GO bagi jujukan protein yang amat tinggi hubung kaitnya kepada jujukan protein yang telah separa dianotasi atau baru ditemui boleh dibuat. Alatan ini mengandungi dua algoritma pintar. Algoritma pertama menggabungkan algoritma genetik selari dengan algoritma pisah-dan-cantum. Tujuannya ialah untuk mengelompokkan istilah GO kepada sejumlah *k* kelompok bagi memisahkan fail GO RDF/XML yang besar kepada fail-fail yang kecil. Dengan itu, jujukan protein dan perhubungan bukti Disimpul daripada Anotasi Elektronik (IEA) boleh ditambah ke dalam fail-fail tersebut. Algoritma kedua menggabungkan algoritma genetik selari dengan algoritma sukatan keserupaan semantik. Tujuannya ialah untuk mencari satu set istilah GO yang semantiknya serupa dengan pertanyaan yang ditentukan daripada fail-fail GO RDF/XML yang kecil. Selain itu, versi asasnya iaitu pelayar GO yang berasaskan kepada carian keserupaan semantik juga diperkenalkan untuk mengatasi masalah pendekatan konvensional: padanan kata kunci.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

BIC	-	Bayesian Information Criterion
BLAST	-	Basic Local Alignment Search Tool
BLOSUM	-	Blocks Substitution Matrix
CDS	-	Coding Sequence
CFG	-	Comparing Factor Group
DAG	-	Directed Acyclic Graph
DBI	-	Davies-Bouldin index
DDBJ	-	Deoxyribonucleic acid Data Bank of Japan
DNA	-	Deoxyribonucleic acid
EBI	-	European Bioinformatics Institute
EMBL	-	European Molecular Biology Laboratory
EST	-	Expressed Sequence Tag
ExPASy	-	Expert Protein Analysis System
EXProt	-	Experimentally Verified Protein Functions
FASTA	-	Fast Alignment
GO	-	Gene Ontology
GOA	-	Gene Ontology Annotation
GPP	-	Graph Partitioning Problem
IC	-	Inferred from Curator
IEA	-	Inferred from Electronic Annotation
IMP	-	Inferred from Mutant Phenotype
JSP	-	Java Server Pages
MGI	-	Mouse Genome Informatics
NCBI	-	National Center for Biotechnology Information
NHGRI	-	National Human Genome Research Institute
OBO	-	Open Biomedical Ontologies

PAM	-	Point Accepted Mutations
PANDIT	-	Protein and Associated Nucleotide Domains with Inferred Trees
PC	-	Personal Computer
PDB	-	Protein Data Bank
PIR	-	Protein Information Resource
PRF	-	Protein Research Foundation
QOC	-	Quality of Clustering
RCA	-	Inferred from Reviewed Computational Analysis
RDBMS	-	Relational Database Management Systems
RefSeq	-	Reference Sequence
SFG	-	Selecting Factor Group
SGD	-	Saccharomyces Genome Database
SIB	-	Swiss Institute of Bioinformatics
Swiss-Prot	-	Swiss Protein
TAIR	-	The Arabidopsis Information Resource
TCDB	-	Transporter Classification Database
TrEMBL	-	Translated European Molecular Biology Laboratory
UniParc	-	Universal Protein Resource Archive
UniProt	-	Universal Protein Resource
UniProtKB	-	Universal Protein Resource Knowledgebase
UniRef	-	Universal Protein Resource Reference Clusters
US NLM	-	United States National Library of Medicine
USDA	-	United States Department of Agriculture
VLSI	-	Very Large Scale of Integration
WWW	-	World Wide Web

CHAPTER 1

INTRODUCTION

1.1 Overview

Bioinformatics is the application of computer technology to store, organize, and analyze the vast amount of biological data which is available in the form of sequences and structures of proteins (the building blocks of organisms) and nucleic acids (the information carrier). The biological information of nucleic acids is available as sequences while the data of proteins is available as sequences and structures. The protein sequence is a chain of amino acids that represents the primary structure of a protein. It plays a central role to determine the structure, homology, and function of a protein. Annotation of a protein sequence is important for the preservation and reuse of knowledge and for content-based queries. Annotation is a process of associating additional information with a particular point in a piece of information. The protein sequence annotation is done either manually by several expert biologists, automatically using bioinformatics tools like Basic Local Alignment Search Tool (BLAST), or both combinations. By supplementing additional information to a protein sequence, it increases the value of the resource for users and can be regarded to be highly reliable. Recently, the Gene Ontology (GO; http://www.geneontology.org/) has been widely used in protein sequence annotation. This is due to characteristics of the GO that the data is continuously evolved and refined, the structure is simple and relatively easy to understand and use, direct input from the biological community, and active curation to sustain the quality and integrity of data. The GO is a collection of nearly 23 thousand terms to describe gene and gene product attributes in any organism. The terms are structured, controlled vocabularies and organized as a Directed Acyclic Graph (DAG) in three aspects: cellular component, biological process, and molecular function.

1.2 Current Methods for Protein Sequence Annotation

Instead of traditional wet-lab methods that are manually done by the biologists, the computational methods for automated protein sequence annotation can be divided into four main categories as follows:

(i) Sequence-similarity-based method depends on the determination of a local or global similarity between the not-yet annotated protein sequence and protein sequences with known annotation. This method uses sequence similarity search algorithms such as Smith-Waterman and Needleman-Wunsch algorithms.

(ii) Controlled-vocabulary-based method employs the most widely used biological ontology, the GO along with its annotation databases to annotate protein sequence.

(iii) Literature-based method relies on natural language processing and text mining techniques to extract information from the biomedical literature as evidence to annotate protein sequence.

(iv) Rule-based method annotates protein sequence based on condition and existence of certain rules. The rules are created according to information extracted from the secondary databases such as protein families, domains, and functional sites databases.

Recently, the GO is an emerging ontology that is gaining momentum for the purpose of genome, expressed sequence tag (EST), and protein annotations. The advantages of using the GO for protein sequence annotation are:

(i) The GO data is dynamic and constantly evolves according to the current state of biological knowledge advances.

(ii) The GO data is publicly available and can be downloaded at any time on the World Wide Web (WWW) in various formats that can be understandable and processable by human and machine alike.

(iii) The common GO terms shared by gene and protein sequences in multiple organisms in different databases can facilitate uniform queries across them.

(iv) The association of GO terms with nearly 2.5 million gene products that are supported by citation and evidence can affirm its reliability for future evaluation and use.

1.3 Challenges of Protein Sequence Annotation

Application of the GO terms to annotate protein sequences is not easy, especially for species not yet inserted in public biological databases. Furthermore, for bioscientists with little computational knowledge or limited facilities it is a hard task to annotate those protein sequences. This is due to the fact that generally the existing GO-based protein sequence annotation tools are:

(i) Dependent on BLAST which is computationally intensive and requires high-cost and high-specification hardware since sequence alignment is performed to all protein sequences but not to protein sequences only that indicate higher similarity.

(ii) Dependent on Relational Database Management Systems (RDBMS) which require the user to setup the RDBMS software and to import the data or sources into the RDBMS format.

(iii) Partially based on the GO data which requires the user to download the Gene Ontology Annotation (GOA) data or protein sequence data sets from several sources.

Furthermore, the traditional wet-lab methods are labor intensive and prone to human error. On the other hand, sequence-similarity-based tools like BLAST that are used by most of the computational methods as described in Section 1.2 are time intensive and require high investment in computing facilities such as cluster server or grid computing if being used locally. Moreover, for remote users, these tools are subject to internet stability and speed to access the tools and to get the results online.

1.4 Statement of the Problem

The macro (application) problem that is tried to be solved in this study can be described as follows:

> "Given a protein sequence, it is a challenging task to develop a new GO-based method to annotate protein sequences that does not depend on BLAST and RDBMS and is fully based on the GO data. At the same time it is capable of producing better results and requires a reasonable amount of running time with low computing cost specifically for offline usage".

In order to develop the new GO-based method to annotate protein sequences, the following factors need to be considered:

(i) The first factor relates to the process of splitting the monolithic GO RDF/XML file into smaller files. The aims are to avoid dependency on RDBMS format, to fully use the GO data by adding the GOA data and the protein sequence data sets into the files since they are excluded in the original GO RDF/XML file, and to make it easier to be accessed and processed.

(ii) The second factor relates to the process of searching the smaller and fragmented GO RDF/XML files. The aim is to find a group of GO terms with higher term similarity score to a GO term which is foreseen to have higher relationship with the query protein sequence.

(iii) The third factor relates to the process of verifying the results obtained from the second factor by computing sequence alignment score between the query protein sequence and all sequences attached to the predicted GO terms. The aim is to ensure that sequence alignment is not carried out to all protein sequences but only to protein sequences with higher outguessed similarity. Hence, it will require low cost and minimum hardware specification and less amount of processing time.

The factors as described above lead to more technical and theoretical problems. These micro (research) problems are related to automatic clustering and semantic similarity searching. Automatic clustering is an unsupervised learning problem that tries to divide a set of elements into a number *k* of clusters. Thus, elements in the same cluster are as similar as possible and elements in different clusters are as dissimilar as possible. Determining the number *k* of clusters is done by the algorithm and it can be regarded as a hard algorithmic problem. To cluster the GO terms into the number *k* of clusters in order to split the monolithic GO RDF/XML file, the following questions need to be answered:

(i) What is the most suitable clustering algorithm that provides optimal solution and offers reasonable amount of processing time?

(ii) What is the precise criterion for identifying the number *k* of clusters and for measuring the goodness of those clusters?

On the other hand, semantic similarity searching relates to the problem of determining semantic relatedness between terms either by virtue of their likeness (*bank-trust company*), synonymy (*car-automobile*), meronymy (*computer-keyboard*), antonymy (*rich-poor*), functional relationship (*marker pen-white board*), or frequent association (*orang utan-Borneo*). For semantically similar GO terms, the terms are related according to “association”: a table storing information that is shared among the GO terms. Particularly, this table provides an annotation record that is basically a link between a gene product and a GO term provided by the GOA. To search the GO terms, the following questions need to be answered:

(i) What is the most suitable search algorithm that provides optimal solution and offers reasonable amount of processing time?

(ii) What is the precise criterion for this biology-related search for measuring the semantic similarity between the GO terms?

1.5 Objective of the Study

The goal of this study is to develop a computational method to annotate protein sequences using information in the GO. Therefore, this study has several objectives to achieve as follows:

(i) To study and design a GO-based method that uses intelligent techniques and the GO in order to annotate protein sequences.

(ii) To develop an automatic clustering algorithm using the genetic split-merge algorithm in order to split the monolithic GO RDF/XML file.

(iii) To develop a similarity search algorithm using the genetic similarity algorithm in order to find a group of semantically similar GO terms.

(iv) To develop a tool as a proof-of-concept study that applied both algorithms mentioned above in order to highlight the capabilities of the proposed GO-based method.

1.6 Scope and Significance of the Study

Protein sequence annotation is important for the preservation and reuse of knowledge and for content-based queries. Traditional wet-lab methods are labor intensive and prone to human error. Alternatively, sequence-similarity-based tools are time intensive and require high investment in computing facilities for offline usage. On the other hand, these tools are highly dependent on internet stability and speed for online usage. Therefore, a simple and practical computational method that is more accurate, faster, easy to configure and use, and bears low computing cost is needed particularly for offline usage. In this study, a GO-based protein sequence annotation tool named *extended* UTMGO is developed to meet these features. The tool employs two primary intelligent algorithms. The first algorithm named genetic split-merge algorithm is used to split the monolithic GO RDF/XML file. The genetic split-merge algorithm applies the parallel genetic algorithm and the split-and-merge algorithm. The split-and-merge algorithm is implemented to improve infeasible clusters in order to efficiently estimate the number *k* of clusters. The second algorithm named genetic similarity algorithm is used to search for semantically similar GO terms from the fragmented GO RDF/XML files. The genetic similarity algorithm applies the parallel genetic algorithm and the semantic similarity measure algorithm. The semantic similarity measure algorithm is implemented due to its ability to improve the precision and recall of information retrieval by identifying the relation between GO terms. This is acquired by computing the distance or the amount of information those GO terms share in common. Both algorithms use the parallel genetic algorithm because of its capability of being adaptive, efficient, robust, and a global search method that is suitable to address a situation where the search space is large. Moreover, the parallel genetic algorithm optimizes its fitness function by utilizing the genetic operators to find an optimal solution. It can also be executed on a low-cost Personal Computer (PC) cluster using message passing interface libraries that are open source and easy to install.

1.7 Organization of the Thesis

This thesis is organized into 7 chapters. A brief description of the contents of each chapter is given as follows:

(i) Chapter 1 describes the problems, objective, scope, and significance of the study.

(ii) Chapter 2 reviews main subjects used in the thesis that include protein sequence annotation, the GO, algorithms for automatic clustering of

GO RDF/XML file and semantic similarity searching of GO terms, and related tools for protein sequence annotation.

(iii) Chapter 3 describes the operational framework adopted to achieve the objective of the study including the results analysis, instrumentations, and data sources used in the thesis.

(iv) Chapter 4 describes a solution of splitting the monolithic GO RDF/XML file using the genetic split-merge algorithm. The genetic split-merge algorithm combines the parallel genetic algorithm and the split-merge algorithm. The parallel genetic algorithm finds the best combination of node-cluster and the split-merge algorithm identifies the best number k of clusters k_{best}.

(v) Chapter 5 describes a solution of finding a group of semantically similar GO terms using the genetic similarity algorithm. The genetic similarity algorithm combines the parallel genetic algorithm and the semantic similarity measure algorithm. The semantic similarity measure algorithm computes the degree of relationship between the GO terms and the parallel genetic algorithm generates a solution comprising a group of semantically similar GO terms. A GO browser named *basic* UTMGO is introduced to show the applicability of the genetic similarity algorithm.

(vi) Chapter 6 describes a solution of annotating anonymous protein sequence using a GO-based protein sequence annotation tool named *extended* UTMGO. The *extended* UTMGO comprises two intelligent algorithms: the genetic split-merge algorithm and the genetic similarity algorithm.

(vii) Chapter 7 draws general conclusions about achieved results and presents the contributions and future works of the study.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

Through bioinformatics, a scientist can use a genomic DNA (Deoxyribonucleic acid) sequence to: predict which part of the DNA sequence is a gene; compare the gene sequence to other known genes to predict a function; convert the DNA sequence into the protein sequence to predict a function; compare the sequences to study evolutionary relationships; analyze the protein sequence to predict when, how, and where it might function; and generate a 3-D model of the predicted protein. Bioinformatics has shown that protein sequence information from simpler organisms such as microbes can be used to understand protein sequences in complex organisms such as animals and plants. Using the relationships and predictions generated by bioinformatics, scientists can better understand how an organism functions, from simple to complex. However, to annotate a protein sequence, meaning that interpreting the features of the protein sequence and adding additional information to the protein sequence using computational tools and combined with biological knowledge, is not an easy task. Even though, controlled vocabulary such as GO has imposed itself as a standard for proteome annotation and function prediction of proteins. This chapter begins with explanation about protein sequence annotation (Section 2.2). Following that, this chapter describes the GO (Section 2.3) followed by a review of the automatic clustering algorithms (Section

2.4) and the semantic similarity searching algorithms (Section 2.5) that relates to the objectives of the study. A review of the protein sequence annotation tools is given in Section 2.6. Finally, the chapter concludes with findings of the literature review.

2.2 Protein Sequence Annotation

A protein sequence is a chain of amino acids that represents the primary structure of a protein as shown in Figure 2.1. The protein sequence plays a central role to determine the structure, homology, and function of a protein as depicted in Figure 2.2.

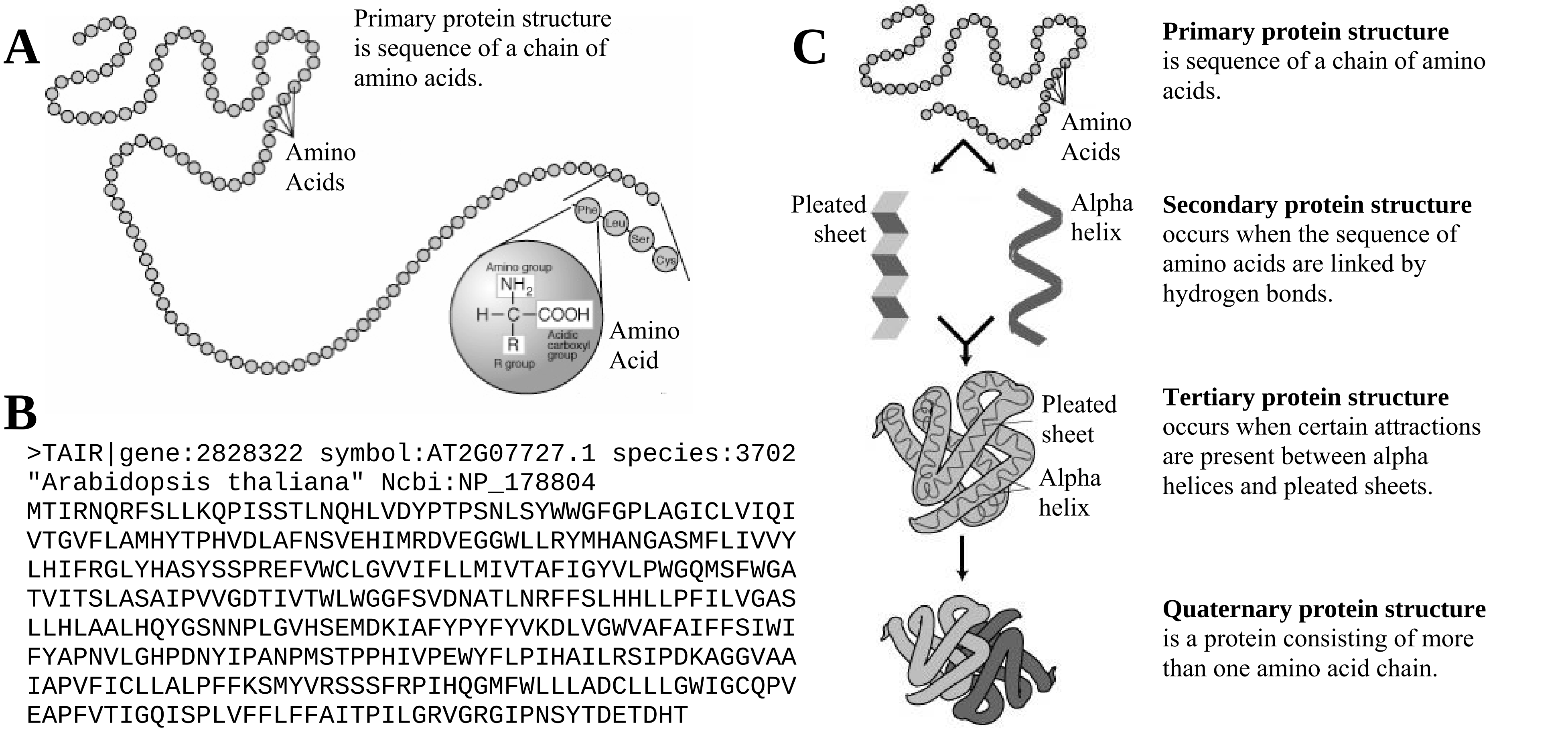
The database of protein sequences can be considered as primary database. It serves as a source for the construction of secondary databases that contain the results of analysis of the protein sequences in the primary databases. The secondary databases are related to protein families, domains, and functional sites. Examples of secondary databases are:

(i) PROSITE (http://www.expasy.ch/prosite/) is a database of protein families, domains, and functional sites. The PROSITE is provided by the Expert Protein Analysis System (ExPASy) proteomics server of the Swiss Institute of Bioinformatics (SIB).

(ii) Pfam (http://www.sanger.ac.uk/Pfam/) comprises many common protein families and domains. It is a database managed by the Wellcome Trust Sanger Institute.

(iii) Protein and Associated Nucleotide Domains with Inferred Trees (PANDIT; http://www.ebi.ac.uk/goldman-srv/pandit/) is a protein families database developed and maintained by the European Bioinformatics Institute (EBI).

Recently, many works have used the protein sequence databases as main resource to predict protein-protein interactions [1], metabolic pathway [2], and protein subcellular localization [3].



```
>TAIR|gene:2828322 symbol:AT2G07727.1 species:3702
"Arabidopsis thaliana" Ncbi:NP_178804
MTIRNQRFSLLKQPISSTLNQHLVDYPTPSNLSYWWGFGPLAGICLVIQI
VTGVFLAMHYTPHVDLAFNSVEHIMRDVEGGWLLRYMHANGASMFLIVVY
LHIFRGLYHASYSSPREFVWCLGVVIFLLMIVTAFIGYVLPWGQMSFWGA
TVITSLASAIPVVGDTIVTWLWGGFSVDNATLNRFFSLHHLLPFILVGAS
LLHLAALHQYGSNNPLGVHSEMDKIAFYPYFYVKDLVGWVAFAIFFSIWI
FYAPNVLGHPDNYIPANPMSTPPHIVPEWYFLPIHAILRSIPDKAGGVAA
IAPVFICLLALPFFKSMYVRSSSFRPIHQGMFWLLLADCLLLGWIGCQPV
EAPFVTIGQISPLVFFLFFAITPILGRVGRGIPNSYTDETDHT
```

Primary protein structure is sequence of a chain of amino acids.

Secondary protein structure occurs when the sequence of amino acids are linked by hydrogen bonds.

Tertiary protein structure occurs when certain attractions are present between alpha helices and pleated sheets.

Quaternary protein structure is a protein consisting of more than one amino acid chain.

Figure 2.1: The protein sequence illustration: (A) The protein primary structure (source: the National Human Genome Research Institute (NHGRI)); (B) The protein sequence of *AT2G07727.1* (Gene:2828322) in FASTA format (source: TAIR); (C) The four levels of protein structure (source: NHGRI).

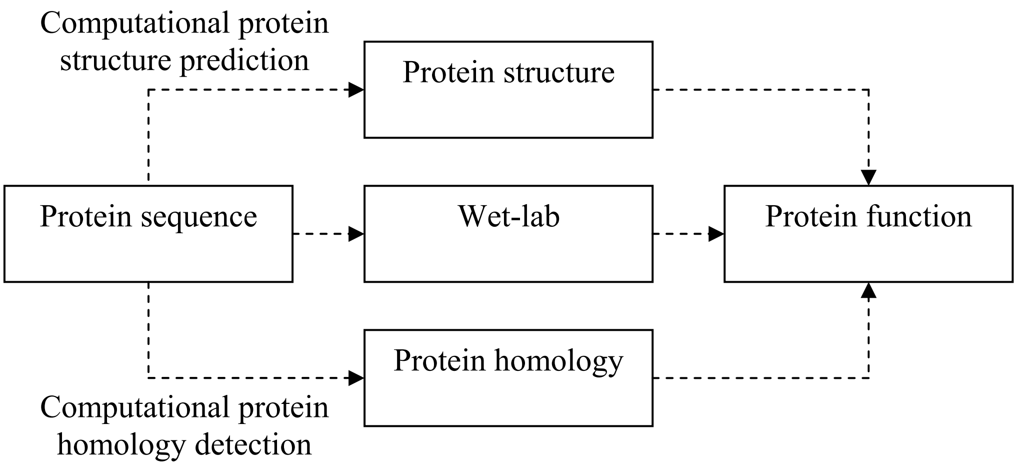

Computational protein structure prediction
Protein structure
Protein sequence
Wet-lab
Protein function
Protein homology
Computational protein homology detection

Figure 2.2: Three different ways of inferring protein function from the protein sequence.

The protein sequence databases are divided into two categories: the protein sequence repositories and the annotated protein sequence databases. The discussions of protein sequence databases have been presented by Whitfield *et al.* [4], Brooksbank *et al.* [5], and Apweiler *et al.* [6]. The protein sequence repositories are highly redundant and with little or no additional information to aid further analysis of the records. Among protein sequence repositories are National Center for Biotechnology Information (NCBI) Entrez Protein (http://www.ncbi.nlm.nih.gov/ entrez/query.fcgi?db=Protein) and Reference Sequence (RefSeq; http://www.ncbi. nlm.nih.gov/RefSeq/). On the other hand, the annotated protein sequence databases provide non-redundant set of protein sequences by consolidating all reports for a given protein sequence into one unique record. The annotation is done either manually by several expert biologists, automatically using bioinformatics tools like BLAST, or both combinations. By supplementing additional information to a protein sequence, it increases the value of the resource for users and can be regarded to be highly reliable. The most comprehensive annotated protein sequence database is Universal Protein Resource (UniProt; http://www.ebi.uniprot.org/). The UniProt merges the information contained in UniProtKB/Swiss-Prot (Swiss Protein; http://www.ebi.ac.uk/swissprot/), UniProtKB/TrEMBL (Translated European Molecular Biology Laboratory; http://www.ebi.ac.uk/trembl/), and Protein Information Resource (PIR; http://pir.georgetown.edu/). The aim is to provide a central resource on protein sequences and functional annotation. The UniProt consists of three main components:

(i) UniProt Knowledgebase (UniProtKB) provides extensive cross-references, functional and feature annotations, and literature-based evidence attribution for easy analysis and cross-database search. It comprises the manually annotated UniProtKB/Swiss-Prot section and the automatically annotated UniProtKB/TrEMBL section.

(ii) UniProt Reference Clusters (UniRef) offers speed similarity searches through sequence space compression by combining closely correlated sequences into a single record.

(iii) UniProt Archive (UniParc) stores all publicly available protein sequences, including their history and links to the source databases.

The UniProt is maintained collaboratively by the SIB and the EBI. Other annotated protein sequence databases are Experimentally Verified Protein Functions (EXProt; http://www.cmbi.kun.nl/EXProt/), Protein Research Foundation (PRF; http://www.prf.or.jp/en/), and Transporter Classification Database (TCDB; http://www.tcdb.org/).

The most systematic protein sequence annotation is carried out by the UniProt. The protein sequences in the UniProt undergo three major phases of annotation as shown in Figure 2.3. The process starts when the wet-lab researchers submit their nucleotide sequence to the European Molecular Biology Laboratory (EMBL). A similarity analysis including search for protein domains and the coding sequence (CDS) expected should be determined by the wet-lab researcher. Secondly, the CDS is translated into protein sequence. The protein sequence is then annotated automatically and stored in the UniProtKB/TrEMBL. The automated annotation is performed using automatically generated rules as in Spearmint [7] or manually curated rules based on protein families, including PIRSF classification-based name rules and site rules [8], HAMAP family rules [9], and RuleBase rules [10]. The UniProtKB/TrEMBL also received nucleotide sequences from GenBank (http://www.ncbi.nlm.nih.gov/Genbank/) and DNA Data Bank of Japan (DDBJ; http://www.ddbj.nig.ac.jp/) and protein sequences extracted from the literature or directly sent to the UniProtKB/Swiss-Prot. Thirdly, protein sequences in the UniProtKB/TrEMBL are selected for full manual annotation and consolidation into the UniProtKB/Swiss-Prot. The manual annotation is done by biologists and is based on literature curation and sequence analysis. The manual annotation procedures were described in detail by Apweiler *et al*. [11]. Further explanation of the annotation processes in the UniProt can be found in [12], [13].

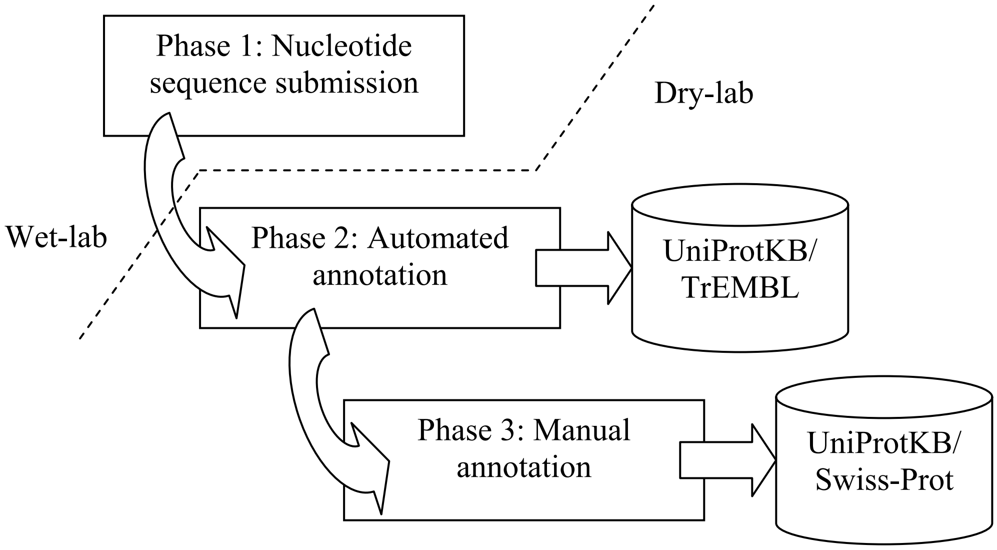

Phase 1: Nucleotide sequence submission
Dry-lab
Wet-lab
Phase 2: Automated annotation
UniProtKB/ TrEMBL
Phase 3: Manual annotation
UniProtKB/ Swiss-Prot

Figure 2.3: Phases of protein sequence annotation in the UniProt.

Lately numerous methods have been proposed for automated protein sequence annotation. These methods can essentially be divided into four main categories as follows:

(i) Sequence-similarity-based method depends on the determination of a local or global similarity between the not-yet annotated protein sequence and protein sequences with known annotation. This method uses sequence similarity search algorithms such as Smith-Waterman and Needleman-Wunsch algorithms. Examples of works have been carried out by Snyder *et al*. [14] and Koski *et al*. [15].

(ii) Controlled-vocabulary-based method employs the most widely used biological ontology, the GO along with its annotation databases to annotate protein sequence such as studies done by Jones *et al*. [16] and Prlic *et al*. [17].

(iii) Literature-based method relies on natural language processing and text mining techniques to extract information from the biomedical literature as evidence to annotate protein sequence. Some recent studies have been conducted by Yuan *et al*. [18] and Chiang and Yu [19].

(iv) Rule-based method annotates protein sequence based on condition and existence of certain rules. The rules are created according to information extracted from the secondary databases. This method has been applied by Sigrist *et al*. [20] and Yu [21].

2.3 The Gene Ontology

The GO project started in 1998 by collaboration between three model organism databases: FlyBase (http://flybase.bio.indiana.edu/), Saccharomyces Genome Database (SGD; http://www.yeastgenome.org/), and Mouse Genome Informatics (MGI; http://www.informatics.jax.org/). Currently, databases participated in the GO project covers model organisms like *Arabidopsis thaliana*, *Caenorhabditis elegans*, *Danio rerio*, *Dictyostelium discoideum*, *Oryza, Rattus norvegicus*, and several protozoan parasites including *Leishmania major*, *Plasmodium falciparum*, and *Trypanosoma brucei*. The GO project is developed and maintained by the GO Consortium. The GO Consortium is currently formed by 16 entities such as EBI, University of Cambridge, University of California Berkeley, The Jackson Laboratory, Stanford University, and Princeton University. The GO is one of the ontologies that take part in the Open Biomedical Ontologies (OBO; http://obo.sourceforge.net/). The OBO is an umbrella project providing well-structured controlled vocabularies that are freely available and can be used across different biological and medical domains.

The goal of the GO project is to construct a well defined and standardized vocabulary for describing the roles of genes and gene products in any organism, even if the cell is evolving and their roles in the cells are changing. The purposes of producing the controlled vocabularies are to manage different names for the same concepts existing in various species, to support cross-species comparison and cross-databases search, and to assist annotation of vast amounts of biological data held in genome and protein databases. The main concept used in the development of the GO is ontology. The ontology is an explicit description of a domain. The ontology is created to define common vocabulary and to share common understanding of the meaning of any vocabulary used. The ontology has been developed in many fields such as chemical process engineering [22], ecoinformatics [23], and multimedia [24]. The ontology has also been implemented to solve various problems related to semantic web search [25], verification of conceptual models [26], and database integration [27].

The GO comprises three sub-ontologies as shown in Figure 2.4. The cellular component describes locations that refer to the place in the cell where a gene product is active like “cytoplasm” (GO:0005737). The biological process describes biological goals contributed by the gene or gene product such as “cell cycle” (GO:0007049). Finally, the molecular function describes activity of a gene product at the molecular level, an example includes “protein kinase activator activity” (GO:0030295). The vocabulary of the GO is called term. Each GO term is related to its parent either via: an “is-a” relationship like “intracellular part” (GO:0044424) is a “cell part” (GO:0044464); or a “part-of” relationship such as “intracellular part” (GO:0044424) is part of “intracellular” (GO:0005622). The properties of the GO term are depicted in Figure 2.5. Each gene product associated to the GO term is supported by an evidence code and a specific reference. For example, an association between gene product “easily shock” (*eas*; FBgn0000536) and GO term “mechanosensory behavior” (GO:0007638) is supported by an evidence code of Inferred from Mutant Phenotype (IMP) and a literature reference PMID:7932299 from PubMed (http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?DB=pubmed). The evidence codes and its description can be found at http://www.geneontology.org/GO.evidence.shtml. The association of gene products to the GO terms is provided by GOA (http://www.ebi.ac.uk/GOA/). The GOA had successfully annotated proteins in the UniProtKB from a variety of species to the GO terms [28].

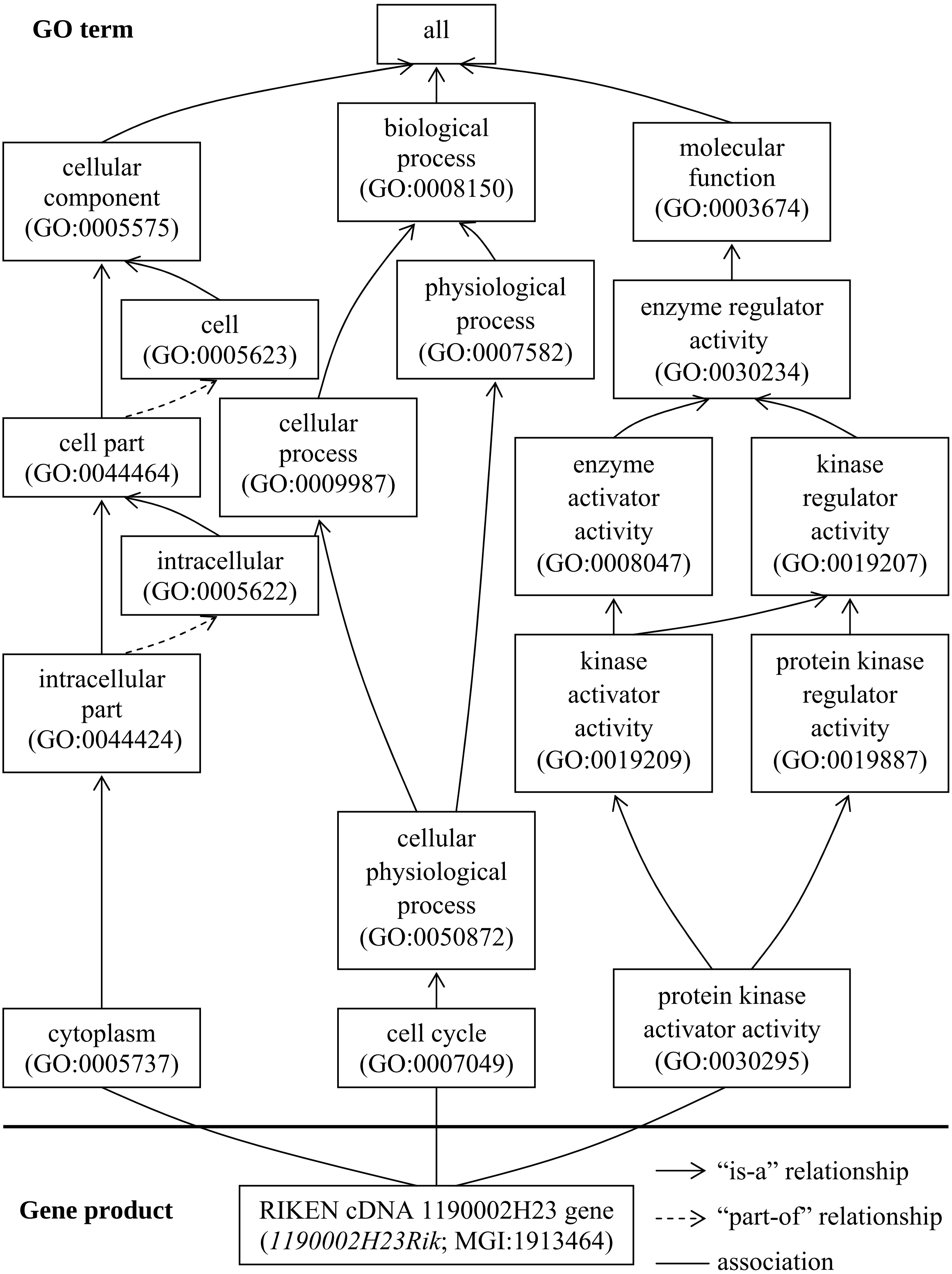
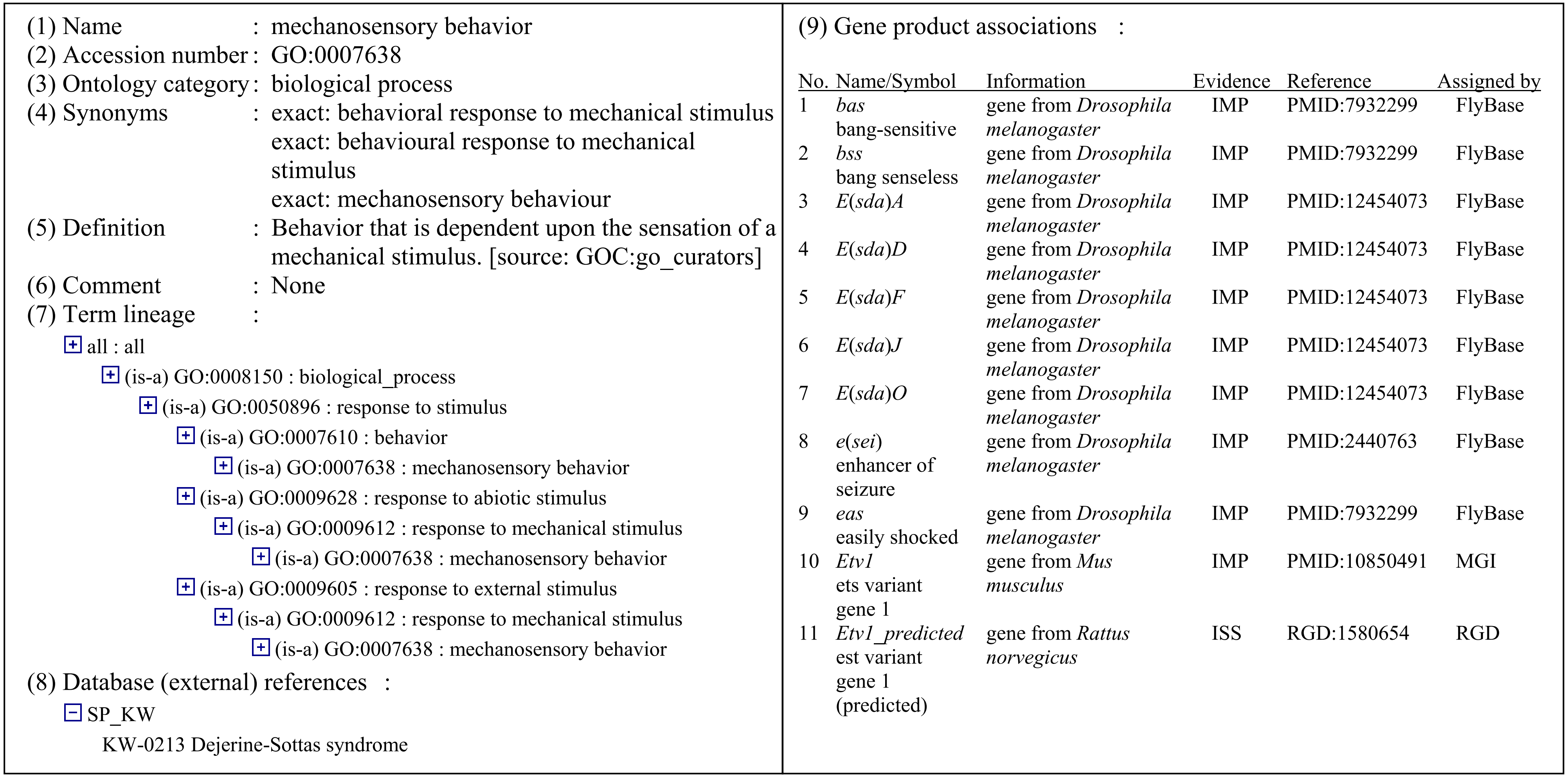

GO term
all
cellular component (GO:0005575)
biological process (GO:0008150)
molecular function (GO:0003674)
cell (GO:0005623)
physiological process (GO:0007582)
enzyme regulator activity (GO:0030234)
cell part (GO:0044464)
cellular process (GO:0009987)
enzyme activator activity (GO:0008047)
kinase regulator activity (GO:0019207)
intracellular (GO:0005622)
intracellular part (GO:0044424)
kinase activator activity (GO:0019209)
protein kinase regulator activity (GO:0019887)
cellular physiological process (GO:0050872)
protein kinase activator activity (GO:0030295)
cytoplasm (GO:0005737)
cell cycle (GO:0007049)
Gene product
RIKEN cDNA 1190002H23 gene (1190002H23Rik; MGI:1913464)
"is-a" relationship
"part-of" relationship
association

Figure 2.4: The GO sub-ontologies



(1) Name : mechanosensory behavior
(2) Accession number : GO:0007638
(3) Ontology category : biological process
(4) Synonyms : exact: behavioral response to mechanical stimulus
exact: behavioural response to mechanical stimulus
exact: mechanosensory behaviour
(5) Definition : Behavior that is dependent upon the sensation of a mechanical stimulus. [source: GOC:go_curators]
(6) Comment : None
(7) Term lineage :

- ⊞ all : all
  - ⊞ (is-a) GO:0008150 : biological_process
    - ⊞ (is-a) GO:0050896 : response to stimulus
      - ⊞ (is-a) GO:0007610 : behavior
        - ⊞ (is-a) GO:0007638 : mechanosensory behavior
      - ⊞ (is-a) GO:0009628 : response to abiotic stimulus
        - ⊞ (is-a) GO:0009612 : response to mechanical stimulus
          - ⊞ (is-a) GO:0007638 : mechanosensory behavior
      - ⊞ (is-a) GO:0009605 : response to external stimulus
        - ⊞ (is-a) GO:0009612 : response to mechanical stimulus
          - ⊞ (is-a) GO:0007638 : mechanosensory behavior

(8) Database (external) references :

- ⊟ SP_KW
  - KW-0213 Dejerine-Sottas syndrome

(9) Gene product associations :

No.	Name/Symbol	Information	Evidence	Reference	Assigned by
1	*bas* bang-sensitive	gene from *Drosophila melanogaster*	IMP	PMID:7932299	FlyBase
2	*bss* bang senseless	gene from *Drosophila melanogaster*	IMP	PMID:7932299	FlyBase
3	*E(sda)A*	gene from *Drosophila melanogaster*	IMP	PMID:12454073	FlyBase
4	*E(sda)D*	gene from *Drosophila melanogaster*	IMP	PMID:12454073	FlyBase
5	*E(sda)F*	gene from *Drosophila melanogaster*	IMP	PMID:12454073	FlyBase
6	*E(sda)J*	gene from *Drosophila melanogaster*	IMP	PMID:12454073	FlyBase
7	*E(sda)O*	gene from *Drosophila melanogaster*	IMP	PMID:12454073	FlyBase
8	*e(sei)* enhancer of seizure	gene from *Drosophila melanogaster*	IMP	PMID:2440763	FlyBase
9	*eas* easily shocked	gene from *Drosophila melanogaster*	IMP	PMID:7932299	FlyBase
10	*Etv1* ets variant gene 1	gene from *Mus musculus*	IMP	PMID:10850491	MGI
11	*Etv1_predicted* est variant gene 1 (predicted)	gene from *Rattus norvegicus*	ISS	RGD:1580654	RGD

Figure 2.5: The properties of the GO term. For example, the "mechanosensory behavior" (GO:0007638) and part of its gene product association.

The size of the GO data (as of January 2007) is shown in Table 2.1. The GO data is stored in the following database categories:

(i) *termdb* is a database that contains information on the GO terms and relationships only.

(ii) *assocdb* is a database which subsumes data in the *termdb* and addition with associations between the GO terms and gene products.

(iii) *seqdb* is a database containing protein sequences that associate with gene products and all data in the *assocdb*.

(iv) *seqdblite* is a database which is same as *seqdb*, except all Inferred from Electronic Annotation (IEA) evidence associations have been taken out.

The GO data is in OBO, OWL, RDF/XML, and MySQL formats. The OBO and OWL formats are available just on the *termdb*. The MySQL format can be downloaded on all database categories. Meanwhile, the RDF/XML format comes without protein sequences and IEA evidence associations.

Table 2.1: Size of the GO data.

Item	No. of records
GO terms	22,954
Definitions of GO terms	22,086
Synonyms for GO terms	20,797
Relationships between GO terms	35,006
All paths in GO graph	1,970,267
External database identifier entities	5,833,963
Links from GO terms to other databases	92,670
Gene products	2,498,910
Synonyms for gene products	330,752
Link between gene product and GO term	10,380,867
Gene product counts per GO term	550,392
Evidence type and reference for an association between gene product and GO term	11,866,795
External database links for an association between gene product and GO term	11,436,198
Protein sequences	2,310,180
Link between gene product and protein sequence	2,315,391
External database links for a protein sequence	21,761,312
Species	268,435

The GO has been used in many applications including gene expression studies [29], proteomics studies [30], comparative genomics [31], and data and text mining [32]. This is due to characteristics of the GO that the data is continuously evolved and refined, the structure is simple and relatively easy to understand and use, direct input from the biological community, and active curation to sustain the quality and integrity of data. Detail discussion about GO can be found in [33]–[39].

2.4 Automatic Clustering Algorithms

Automatic clustering is a process of dividing a set of elements into unknown clusters, where the best number k of clusters is determined by the clustering algorithm. That is, elements within each cluster should be highly similar to each other than to elements in any other cluster. Finding the k automatically is a hard algorithmic problem. The automatic clustering problem can be defined as follows:

> "Let $X = \{X_1, X_2, \ldots, X_n\}$ be a set of n element. These elements are clustered into non-overlapping clusters $C = \{C_1, C_2, \ldots, C_k\}$, where C is called a cluster, k is the unknown number of clusters, $C_i \cap C_j = \emptyset$ for $i \neq j$, $C_1 \cup C_2 \cup \ldots \cup C_k = X$, $C_i \subseteq X$, and $C_i \neq \emptyset$."

The clustering problem is omnipresent in many fields of science and engineering. It has been solved by various techniques such as k-means [40], genetic algorithm [41], self-organizing map [42], fuzzy c-means [43], and particle swarm optimization [44]. Survey of clustering techniques can be found in [45]–[47]. Recently, the increasing amount of data has made the number k of clusters difficult to guess, and the value supplied by the user based on prior knowledge, presumptions, and practical experiences is often inaccurate. Therefore, reasonable ways of identifying the number k of clusters automatically is required to avoid trial-and-error work. Lately, several techniques have been proposed to determine the number k of clusters. Most of the techniques are wrapped around k-means or genetic algorithm. Split and/or merge rules are the most famous wrapper methods to increase or decrease the number *k* of clusters while the algorithm continues. Among these techniques are:

(i) X-means [48]; in this the splitting decision is performed by computing the Bayesian Information Criterion (BIC) until the upper bound of *k* is attained.

(ii) G-means [49]; it starts with small number of k-means centers and raises the number of centers using Gaussian distribution.

(iii) CLUSTERING [50]; it is an automatic clustering based on heuristic strategy that uses the nearest neighbor to group those data that are situated close to one and another. Then, genetic algorithm is used to group the smaller clusters into larger ones.

(iv) S+G [51]; it is also a two stage method, which in the beginning uses a self-organizing feature map to determine the number *k* of clusters and then employs a genetic algorithm based clustering to find the final solution.

In the GO context, the GO terms are structured as DAG. Let GO graph $G = \{V, E\}$, where V is a set of nodes that represent the GO terms and E is a set of directed edges that represent relationships between the GO terms. Partitioning the GO graph in order to cluster the GO terms can be considered as a Graph Partitioning Problem (GPP). The aim of GPP is to cut a vertex set V into k disjoint and non-empty subsets such that the number of edges connecting nodes in different subsets is minimized and the number of edges connecting the nodes in the same subsets is maximized. GPP is a fundamental combinatorial optimization problem that has numerous practical applications in many areas including design of Very Large Scale of Integration (VLSI) circuits [52], mesh partitioning in parallel processing [53], image segmentation in computer vision [54], and gene expression analysis in bioinformatics [55]. An extensive study of Kerninghan-Lin algorithm, simulated annealing, tabu search, watermarking, and normalized cut have been carried out by [56]–[59], [54] respectively to solve the GPP. Review of the GPP techniques can be found in [60], [61]. Several studies using genetic algorithm for the GPP have also been done by:

(i) Bui and Moon [62] introduced a schema of preprocessing phase before the initialization of population to ameliorate the quality of the chromosome. The different classes of graphs: random graph, random geometric graph, random regular graph, and caterpillar graph consisting of 134 to 5,252 nodes, were tested with the algorithm.

(ii) Kaveh and Bondarabady [63] implemented genetic algorithm for finite element decomposition of 1,640 to 6,720 elements. Sequences of coarsening and uncoarsening process are performed to transform the large scale graph G_0 into a smaller size graph G_n and vice versa such that a suitable size of graph can be partitioned by genetic algorithm.

(iii) Kohmoto *et al.* [64] has incorporated simulated annealing into genetic algorithm to generate feasible solutions. The algorithm is then applied to undirected graph with 124 to 250 nodes.

For the ontology clustering, very little effort has been done in this area. Stuckenschmidt and Klein [65] have proposed a method for automatic clustering of large ontologies based on the structure of the class hierarchy. The method consists of three steps:

(i) In the first step, a dependency graph is created from ontology source file using PROLOG-based tool that reads OWL and RDF schema files. It then displays the dependency graph using networks analysis tool Pajek.

(ii) In the second step, the strength of the dependencies between the concepts in the dependency graph is determined by computing the propositional strength network.

(iii) In the third step, an island algorithm is used to determine the modules existing in the dependency graph.

2.5 Semantic Similarity Searching Algorithms

Ontology is a description of concepts in a domain and the relationships between the concepts. Ontology can be represented as a directed graph. The ontology graph comprises the concepts including the descriptions as nodes and semantic relationships as edges. Recently, there has been growing development of ontology in the bioinformatics field such as Sequence Ontology [66], Cell Ontology [67], Chemical Ontology [68], Multiple Alignment Ontology [69], Biodynamic Ontology [70], and Protein-Interactions Ontology [71]. However, the “ontology searching”, which refers to the activity of retrieving concepts in the ontology graph, is not accurately performed by the traditional search engines that are based on keywords. These search engines neglect the semantic relationships between the search concepts and only consider those concepts as character strings. Thence, a mechanism to measure the similarity between concepts in the ontology graph is required to reduce dependency of specialists of a certain domain to input relevant concepts as search words.

There are numerous search techniques that are frequently and extensively used in computer science, engineering, mathematics, and other fields such as:

(i) Tabu search is a local search technique. It uses a local or neighborhood search procedure to repetitively move from a solution *x* to a solution *x*' in the neighborhood of *x*, until termination criterion is satisfied. Examples of application include flow shop problem [72] and facility location problem [73].

(ii) Simulated annealing is a global optimization technique that is based on probabilistic methods. It traverses the search space by producing neighboring solutions of the current solution. The simulated annealing has been applied in flexible manufacturing system [74] and heterogeneous distributed system [75].

(iii) Genetic algorithms are a global search heuristics. These algorithms work by seeking potential solutions and evaluating them. The best solutions are modified to form a new population. This operation is repeated until no better solutions are generated. The genetic algorithms have solved various problems such as nurse rerostering problem [76] and personnel assignment problem [77].

(iv) Ant colony optimization is a population-based technique that tries numerous solution options at each step of the algorithm. The ant colony optimization is inspired by the behavior of ants in discovering routes from the colony to food. It has been applied in water distribution system [78] and solved the nonlinear resource allocation problem [79].

Other techniques include particle swarm optimization [80], hill climbing [81], and cross-entropy method [82]. A detailed comparison among these techniques can be found in [83]–[85].

In the case of semantic similarity search, researchers have used different measures to identify similarity between two concepts being compared. Lately, several new semantic similarity measures have been introduced such as:

(i) Edge-similarity measure [86] is applied to varying image illumination and contrast.

(ii) Quantitative tract similarity measure [87] is based on the shape and length of the two tracts being analyzed to improve image segmentation reproducibility.

(iii) Trainable similarity measure [88] applied the matching-pursuit approach for road-sign classification.

(iv) Clip-based similarity measure [89] is based on two bipartite graph matching algorithms (maximum matching and optimal matching) for video retrieval and video summarization.

(v) Spectral similarity measures [90] consist of four spectral measures (spectral angel measure, Euclidean distance measure, spectral correlation measure, and spectral information divergence) for the analysis of hyperspectral imagery.

Other semantic similarity measures are: Chen *et al*. [91] has proposed fuzzy similarity measure for distorted fingerprints matching; and Lee and Crawford [92] and Moghaddam *et al*. [93] have created Bayesian similarity measure for image segmentation and image matching respectively. Evaluation of different semantic similarity measures have been done by Skerl *et al*. [94] for rigid registration of medical images and Núñez *et al*. [95] on improving case-based reasoning for environmental decision support systems.

On the other hand, for the GO, semantic similarity search is required in order to search for semantically similar GO terms and to reduce dependency on the specialists. Thence, it avoids the users from investing lots of time browsing the GO terms. However, this approach involves computing the amount of information the GO terms share in common and/or calculating the depth and the local network density of the GO term. This scenario becomes complicated since the GO terms are structured as a DAG and searching the GO graph is an NP-complete problem. By contrast, the existing GO browsers to support basic needs for scientists to search the GO terms are still using conventional approach which is based on keyword matching. Thus, for a scientist to find a group of GO terms that have semantically similar properties is time consuming and a hard task. A list of tools for searching and browsing the GO terms can be found at http://www.geneontology.org/GO.tools.browsers.shtml. All these tools are free to academics, among them are:

(i) CGAP GO Browser is developed by The Cancer Genome Anatomy Project. It allows the user to browse the GO terms using the hierarchy view and find the known human and mouse genes assigned to each term. This tool can be used at http://cgap.nci.nih.gov/Genes/GOBrowser/.

(ii) GOFish is created using Java applet by the Roth Laboratory at the Harvard University. It uses term name or accession number as an input and then performs keyword matching. This tool allows the user to construct arbitrary Boolean queries using GO terms, and ranks gene products that satisfy the queries. The GOFish can be found at http://llama.med.harvard.edu/software.html.

(iii) Ontology Lookup Service is provided by the EBI. It is based on partial keyword search. As the users types into the search box, they will see recommended terms that match what are being entered in the list box. This tool was developed to merge all publicly available biomedical ontologies into a single database. It can be viewed at http://www.ebi.ac.uk/ontology-lookup/.

Other browsers are AmiGO (http://godatabase.org/), EP GO Browser (http://ep.ebi.ac.uk/EP/GO/), QuickGO Browser (http://www.ebi.ac.uk/ego/), GenNav Browser (http://mor.nlm.nih.gov/perl/gennav.pl), and MGI GO Browser (http://www.informatics.jax.org/searches/GO_form.shtml).

2.6 Protein Sequence Annotation Tools

Bioinformatics is the application of computer technology to store, retrieve, analyze, simulate, or predict the composition or the structure of biomolecules. It involves the development of algorithms and statistical techniques, databases, and tools. The bioinformatics tools should be developed using open source and web technologies. Therefore, these tools can be distributed freely and used extensively by the bioscientists. However, an excellent tool should be easy to be setup and used, can be run on low-cost hardware, and requires a short execution time.

Recently, a number of bioinformatics tools have been developed for protein sequence annotation based on the GO. These tools are:

(i) Blast2GO employs BLAST to find homologous sequences to Fast Alignment (FASTA) formatted input protein sequences. The Blast2GO extracts the GO terms for each found hit by mapping to existing annotation associations. An annotation rule finally assigns GO terms to the query protein sequence. This tool can be accessed at http://bioinfo.ivia.es/blast2go/. It is maintained by the Centro de Genómica at the Instituto Valenciano de Investigaciones Agrarias.

(ii) GoAnna can be applied for protein sequence annotation using a sequence similarity search. This tool accepts a list of protein sequences in FASTA format. The GoAnna conducts BLAST search against AgBase databases or GO annotated databases like UniProtKB/Swiss-Prot and UniProtKB/TrEMBL. This tool is developed by the Mississippi State University and can be used at http://agbase.msstate.edu/GOAnna.html.

(iii) HT-GO-FAT provides the bioscientists with a high-throughput mapping of unknown protein sequence to GO annotation. It uses BLAST for sequence similarity search. The HT-GO-FAT can be downloaded from http://liru.ars.usda.gov/mainbioinformatics.html. This tool is developed by the Livestock Issues Research Unit at the United States Department of Agriculture (USDA) Agricultural Research Service.

(iv) InGOt is capable to assign up-to-date GO terms to a given protein sequence. The InGOt claims to have more sequences than any public resource and assignments harvested from the broadest possible GO-linked resources. It is proprietary software by Inpharmatica Ltd. A free two week trial of this tool can be downloaded at http://www.inpharmatica.co.uk/ingot/.

Other GO-based protein sequence annotation tools are: GOPET is addressable via http://genius.embnet.dkfz-heidelberg.de/menu/biounit/open-husar/, and it has been developed by the German Cancer Research Center; GOtcha (http://www.compbio.dundee.ac.uk/gotcha/gotcha.php) by the Barton Group at the University of Dundee; GoFigure (http://udgenome.ags.udel.edu/gofigure/) is under the UDGenome project by the University of Delaware; GOblet (http://goblet.molgen.mpg.de/) is introduced by the Max Planck Institute for Molecular Genetics; and lastly JAFA (http://jafa.burnham.org/) is maintained by the Burnham Institute for Medical Research.

In parallel, several works using computational intelligence techniques for protein sequence annotation have also been done by:

(i) Kirac *et al*. [96] introduced a data mining technique that calculates the probabilistic relationships between the GO annotations of proteins on protein-protein interaction data. Then, it assigns highly associated GO terms of annotated proteins to the target protein sequence.

(ii) Ray and Craven [97] built a system to annotate a given protein sequence with codes from the GO using the text of an article from the biomedical literature as evidence. This system relies on statistical techniques namely the *n*-gram models and the Naïve Bayes models.

(iii) Ponomarenko *et al*. [98] shows how protein sequence annotation can be improved and corrected if protein structures are available. They used the combinatorial extension algorithm to compare the structure. Then, it widens the protein annotation provided by the GOA to further annotate the protein sequences in the Protein Data Bank (PDB; http://www.rcsb.org/pdb/).

There are also varieties of protein sequence annotation tools that have been developed without depending on the GO data such as FeatureMap3D (http://www.cbs.dtu.dk/services/FeatureMap3D/), KOBAS (http://kobas.cbi.pku.edu.cn/), MineBlast (http://leger2.gbf.de/cgi-bin/MineBlast.pl), ProtoBee (http://www.protobee.cs.huji.ac.il/), and ProFAT (http://cluster-1.mpi-cbg.de/profat/).

2.7 Trends and Tendencies

Protein sequences are stored in a database called primary database. The primary database provides a source for the prediction of structure, homology, and function of a protein. The primary databases are divided into protein sequence repositories such as NCBI Entrez Protein and RefSeq and annotated protein sequence databases such as UniProt and EXProt. The annotated protein sequence databases provide non-redundant set of protein sequences with additional information compared to the protein sequence repositories. The most systematic protein sequence annotation is done by the UniProt which involves three major phases: similarity analysis of the submitted nucleotide sequence, translation into protein sequence and automated annotation, and manual annotation for verification. Currently for automated annotation, four methods have been identified: sequence-similarity [14], controlled-vocabulary [16], literature [18], and rule [20] -based methods. Lately, the controlled-vocabulary-based method using GO has been widely applied to annotate protein sequences. This is because the GO data constantly evolves and it is publicly available, well defined and a consistent biological terminology, and associated with a large number of gene products that are supported by citation and evidence.

In the case of splitting the monolithic GO RDF/XML file, the process can be regarded as GPP. Several works done by [62]–[64] have shown that the GPP can be efficiently solved by genetic algorithm. Furthermore, algorithms such as CLUSTERING [50] and S+G [51] have shown that genetic algorithm can be combined with other algorithms to find the number *k* of clusters automatically. However, applications of genetic algorithm to split the monolithic GO RDF/XML file is not easy since very little work has been done in ontology clustering as references. Another focus of this study is to perform semantic similarity searching on the GO terms. Currently, most of the GO browsers such as AmiGO and GOFish are based on keyword matching. On the other hand, existing searching algorithms such as genetic algorithm are not capable of executing the task alone. Therefore, a suitable semantic similarity measure for ontology searching is required to combine with the genetic algorithm. However, most of the existing semantic similarity measures [86]–[95] are specifically designed for image segmentation and image matching. Lastly, although most of the protein sequence annotation tools such as GoAnna and HT-GO-FAT are publicly available via the internet, yet they depend on BLAST to perform sequence similarity that requires high computing power and high implementation cost especially for offline usage. Therefore, a simple and practical tool that is easy to be configured with low computing cost needs to be developed.

2.8 Summary

This chapter gives broad review of basic concepts of the protein sequence, protein sequence databases, and processes involved in the protein sequence annotation for better understanding of the nature of the problems, together with explanation about GO including its properties, characteristics, and applications. This chapter also presents related algorithms for clustering, automatic clustering, GPP, and ontology clustering including algorithms for searching, semantic similarity searching, and protein sequence annotation. Reviews of GO browsers and protein sequence annotation tools are also presented in this chapter.

CHAPTER 3

RESEARCH METHODOLOGY

3.1 Introduction

One of the advantages of the GO terms is that it can cope with synonyms and can describe biological function. Furthermore, the GO terms are linked with approximately 10.38 million associations, 2.50 million different gene products, and with the largest set covering around 2.31 million protein sequences from 0.27 million species. Thence, specific protein sets can easily be compared with respect to common functional features [30], [99], protein databases such as MiGenes [100] and PA-GOSUB [101] can be explored through complicated queries, and large-scale protein database can simply be annotated [28], [102] based on the GO terms. However, direct use of the GO terms to annotate protein sequences is not easy, especially from small sequencing projects or for species not commonly represented in biological databases. Furthermore, for small group of scientists with little computational background or without appropriate facilities it is a tedious task to annotate those protein sequences. Therefore, in Section 3.2, we present the framework of the study that discusses the development of the *extended* UTMGO including its basic version for browsing the GO terms. The framework also discusses the intelligent algorithms of the *extended* UTMGO: the genetic split-merge algorithm and the genetic similarity algorithm that are used to split the monolithic GO RDF/XML file and to search a group of semantically similar GO terms respectively.

The data sets used as well as the instrumentation and analysis of the results of the algorithms and tools are also discussed in Sections 3.3 and 3.4 respectively.

3.2 Framework of the Study

The framework of the study involved three main phases namely the ontology clustering phase, the ontology searching phase, and the bioinformatics tool development phase as depicted in Figure 3.1.

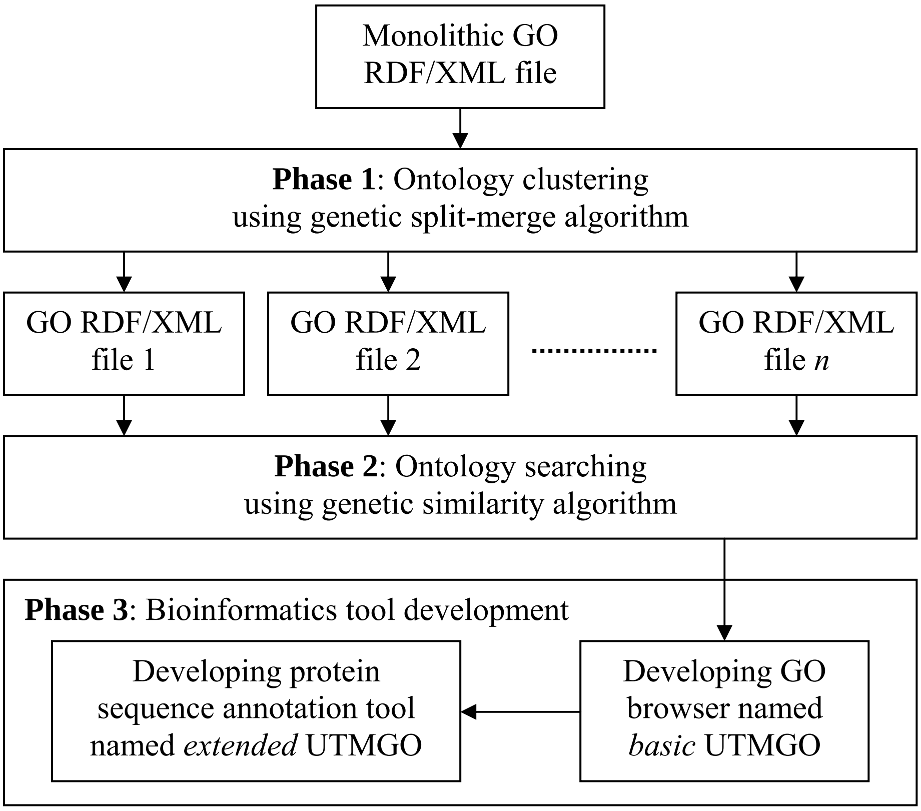

Monolithic GO RDF/XML file
Phase 1: Ontology clustering using genetic split-merge algorithm
GO RDF/XML file 1
GO RDF/XML file 2
....
GO RDF/XML file *n*
Phase 2: Ontology searching using genetic similarity algorithm
Phase 3: Bioinformatics tool development
Developing protein sequence annotation tool named *extended* UTMGO
Developing GO browser named *basic* UTMGO

Figure 3.1: The framework of the study.

In the ontology clustering phase, the genetic split-merge algorithm is formed to cluster the GO terms. The aim is to split the monolithic GO RDF/XML file into a number of smaller files. The genetic split-merge algorithm is a combination of parallel genetic algorithm and split-and-merge algorithm. The detail about genetic split-merge algorithm is discussed in Chapter 4. The genetic split-merge algorithm can be summarized as follows:

(i) Initialization of a population of chromosomes where alleles for each chromosome show the cluster number and loci represent the GO terms accession number.

(ii) Evaluate the fitness of each chromosome.

(iii) Select chromosomes for reproduction using the roulette wheel selection scheme.

(iv) Apply partial match crossover and swap mutation operators.

(v) Replace the least fit chromosomes in the existing population by the newly generated offspring.

(vi) Repeat steps (ii)–(v) until the stopping criteria are met.

The inputs for the genetic split-merge algorithm are the GO graph and the minimum number *k* of clusters specified by the user. This algorithm returns the best chromosome representing a *k* number of good clusters. The genetic split-merge algorithm is capable of automatically identifying the number *k* of clusters, producing balanced clusters in terms of number of elements in each cluster, requires reasonable amount of processing time, and generates good clusters.

In the ontology searching phase, the genetic similarity algorithm is developed to perform semantic similarity search. The idea is to find a group of semantically similar GO terms for a given query GO term. The genetic similarity algorithm incorporates semantic similarity measure algorithm in the parallel genetic algorithm. A comprehensive discussion of the genetic similarity algorithm is done in Chapter 5. The genetic similarity algorithm can be summarized as the following steps:

(i) Perform preprocessing using the semantic similarity measure algorithm.

(ii) Initialization of a population of chromosomes where alleles for each chromosome show either the GO terms are retrieved or not retrieved and loci represent the GO terms accession number.

(iii) Evaluate the fitness of each chromosome.

(iv) Select chromosomes for reproduction using the roulette wheel selection scheme.

(v) Apply two-point crossover and swap mutation operators.

(vi) Replace the least fit chromosomes in the existing population by the newly generated offspring.

(vii) Repeat steps (iii)–(vi) until the stopping criteria are met.

The inputs for the genetic similarity algorithm are the GO graph and the query GO term. This algorithm returns the best chromosome representing a group of GO terms that are semantically similar to the query GO term. The genetic similarity algorithm is susceptible of returning the GO terms whose names do not have keywords similar to the name of the query GO term has. Furthermore, it is able to avoid producing many GO terms with low term similarity score and can be executed in a short time.

In the bioinformatics tool development phase, the *basic* UTMGO is developed using web technology. The main goal of this tool is to act as a new way to search the GO terms. The *basic* UTMGO has shown its capability to determine the semantically similar GO terms as compared to other keyword-based GO browsers. This is due to the effectiveness of the genetic split-merge algorithm and the genetic similarity algorithm. The potential of this tool has been broadened to annotate protein sequences. The tool named *extended* UTMGO is able to return a set of GO terms together with their associated protein sequences that have higher sequence alignment score to the query protein sequence. This feature allows bioscientists to annotate protein sequences by only using the GO terms and its properties. Thus, it prevents dependency on BLAST, RDBMS, various sources of data, and high-cost and high-specification hardware unlike other protein sequence annotation tools. The *basic* and *extended* UTMGO are described in Chapter 5 and 6 respectively.

3.3 Data Sources

The GO data used in this study is in RDF/XML format which can be downloaded from http://archive.godatabase.org/. The data is compressed in a GZIP file named go_YYYYMM-assocdb.rdf-xml.gz. In Chapter 4, all GO data in the RDF/XML format is used to test the genetic split-merge algorithm. However, to include protein sequences and IEA evidence associations into the smaller GO RDF/XML files, these data are taken from the MySQL format. The GO data in the MySQL format is stored in a file named go_YYYYMM-seqdb-tables.tar.gz. In Chapter 5, the *basic* UTMGO and the genetic similarity algorithm use 250 GO terms as the query GO terms. These terms are selected randomly which comprise 8% from cellular components, 56% from biological processes, and 36% from molecular functions. In the meantime, to assess the performance of the *extended* UTMGO for annotating protein sequences, 50 protein sequences are selected randomly as the query protein sequence from each species as follows:

(i) *Oryza sativa ssp japonica* from the Gramene database (http://www.gramene.org/Oryza_sativa/index.html).

(ii) *Homo sapiens* is obtained from the Ensembl database (http://www.ensembl.org/Homo_sapiens/index.html).

(iii) *Saccharomyces cerevisiae* from the SGD database.

(iv) *Arabidopsis thaliana* is downloaded from the TAIR database (The Arabidopsis Information Resource; http://www.arabidopsis.org/).

3.4 Instrumentation and Results Analysis

All experiments are run on a 25-node low-cost PC cluster with 2.8GHz Pentium IV of processor, 512MB of memory, and 100Mbps of network speed. The low-cost PC cluster is based on island (coarse-grained) model and it is implemented using MPICH2 libraries (http://www-unix.mcs.anl.gov/mpi/mpich/). The operating system used is Fedora Core 5. The genetic algorithm adopted in this study is an enhancement of the GAlib C++ libraries (http://lancet.mit.edu/ga/). The interface for the *basic* and *extended* UTMGO are developed using Java Server Pages (JSP) scripts.

In Chapter 4, four comparisons are presented to evaluate the performance of the genetic split-merge algorithm. The comparison includes results analysis of different number of processors of the low-cost PC cluster used to run the genetic split-merge algorithm, different fitness functions of the genetic split-merge algorithm, and comparison between genetic split-merge algorithm and other clustering and automatic clustering algorithms. In Chapter 5, the results of the genetic similarity algorithm are analyzed with different semantic similarity measure and different combinations of parameters α and β for depth and local network density factors respectively. Whereas, different semantic similarity and keyword-based GO browsers are used to analyze the results of the *basic* UTMGO. The results analysis of different number of processors of the low-cost PC cluster used to run the *basic* UTMGO and the genetic similarity algorithm as its intelligent engine are also presented in Chapter 5. Lastly, in Chapter 6, the results of the *extended* UTMGO have been analyzed with other GO-based protein sequence annotation tools. The expert and the system that are related to Equation 5.15 and 5.16, discussed in Section 5.6 and 6.4, refer to a biologist who has knowledge of the GO and protein sequence annotations and the *basic* and the *extended* UTMGO, respectively. The results in this study have been validated by the GO Consortium. Some publications of the study have also been included in the GO bibliography (http://www.geneontology.org/cgi-bin/biblio.cgi).

3.5 Summary

The framework of this study has been presented in this chapter to solve the macro (application) and micro (research) problems. However, in the following chapters, we are going to present more details of the techniques for splitting the monolithic GO RDF/XML file, followed by techniques for finding a group of semantically similar GO terms, and then developing the *basic* and *extended* UTMGO.

CHAPTER 4

THE GENETIC SPLIT-MERGE ALGORITHM
FOR SPLITTING THE MONOLITHIC GENE ONTOLOGY RDF/XML FILE

4.1 Introduction

The GO is a collection of dynamic and standardized biological terms used to annotate gene products in any organism. These biological terms are rich with information such as definition, synonyms, external database references, association with annotated gene products and their protein sequences that are provided by the GOA, and relationships with other terms. The GO data is available in RDF/XML, OBO/XML, OWL, and MySQL formats. The GO RDF/XML is created to allow the GO data to be shared and reused across the WWW in a way that it can be interpreted and processed by human and machine alike. The advantage is that, especially for bioscientists, it obviates the need for manually importing the GO data into relational database format every time it is updated. Thus, it prevents them from setting up the database software. The GO RDF/XML has been used by numerous bioinformatics tools such as WEGO [103], a tool for plotting GO annotation results; ErmineJ [104], a tool for the functional analysis of gene sets in microarray gene expression data; DynGO [105], a tool to search for a GO term and its association using batch and semantic retrieval; and COBrA [106], a browser and editor for GO and OBO ontologies that allows the user to make links between terms in those ontologies.

Due to large amount of the GO data as shown in Table 2.1, protein sequences and IEA evidence associations are not included in the GO RDF/XML file by the GO Consortium. But still the astronomical size and massive nature of this single flat file (current size is 472 MB) has caused the GO RDF/XML difficult to be maintained, published, validated, and processed. An alternative way to make the GO RDF/XML more complete, coherent, and easy to browse is to split it into multiple files. Thus, it enables protein sequences and IEA evidence associations to be included in the smaller GO RDF/XML files.

Splitting the GO RDF/XML file requires the GO terms to be grouped into a number *k* of clusters. Since the GO terms are structured as DAG, let GO graph be $G = \{V, E\}$ that consists of two main elements: *V* is a set of nodes that represent the GO terms and *E* is a set of edges that represent relationships between the GO terms. Partitioning the GO graph is a combinatorial problem and can be regarded as a GPP. The intention of GPP is to divide a vertex set *V* into *k* disjoint and non-empty subsets in order to produce partitions that have higher degree of interaction between nodes in the same partition and have lower degree of interaction between nodes in different partitions. The task of partitioning the large GO graph that contains more than 22 thousand nodes and almost 2.0 million paths is characterized as bearing very high computational complexity. Moreover, identifying the number *k* of clusters is a hard algorithmic problem since it is difficult to guess, and it requires a trial-and-error work.

This chapter is organized as follows. Section 4.2 gives related work on clustering, automatic clustering, and GPP. Section 4.3 explains the proposed algorithm to split the monolithic GO RDF/XML file. Section 4.4 describes the testing environment and evaluation measures used in this chapter. Section 4.5 presents experimental results and discussion. Finally, the chapter summary is provided in Section 4.6.

4.2 Related Work

A large number of clustering algorithms have been proposed in the past decade. Among the successfully implemented clustering algorithms are fuzzy logic, e.g. fuzzy clustering by local approximation of membership [107] and fuzzy c-means [108] for clustering DNA microarray data; support vector machines, e.g. clustering support vector machines [109] for protein local structure prediction and support vector clustering [110] for marketing segmentation; k-means, e.g. k-means range algorithm [111] for personalized data clustering in e-commerce and greedy k-means algorithm [112] for global gene trajectory clustering; and evolutionary algorithms, e.g. hybrid-evolutionary-programming algorithms [113] for microbial growth studies and genetic clustering [114] for clustering gene expression data. Other clustering algorithms include hierarchical clustering [115], Bayesian clustering [116], profile hidden Markov model [117], and self-organizing map [118]. There are also hybrid clustering algorithms such as rough fuzzy c-means [119], rough k-means [120], and evolutionary fuzzy c-means [121]. Comparison of clustering algorithms can be found in [122]–[125].

For automatic clustering, several new algorithms have been developed recently. Evolutionary clustering [126] employs merge and split mutation operators to dynamically change the number *k* of clusters that is represented by the length of the chromosome during the evolutionary process. This algorithm is specifically developed to cluster gene expression microarray data. Laszlo and Mukherjee [127] introduces genetic algorithm for evolving centers in the k-means. They exploit the emersion of chromosomes with varying number of genes to simultaneously search for a range of good clusters around the specified *k*. The algorithm has been tested using benchmark data sets of traveling salesman problem. Hybrid niching genetic algorithm [128] applies Selecting Factor Group (SFG) and Comparing Factor Group (CFG). The SFG is used to encourage mating between chromosomes. Meanwhile, the purpose of the CFG is to balance competition during substitution between chromosomes with the same number of clusters and chromosomes with different number of clusters. Three real data sets of iris, breast cancer, and subcellcycle are used in the experiments.

In another part, GPP has been studied by several researchers for different sizes of graph. Aykanat *et al.* [129] has formulated adaptive object space decomposition problem as a GPP. A tool named RM-MeTiS is developed to partition the graph. This tool consists of three phases: multilevel coarsening, initial remapping, and multilevel refinement. The largest graph consists of 109,744 nodes and the experiments are conducted on a 28-node PC cluster. Duarte *et al.* [130] has modeled image segmentation as a GPP. The GPP is resolved by a variant of normalized cut using hierarchical social metaheuristic. The experiments involve the largest graph with 11,155 nodes and 1,817,351 edges. Mitchell and Mancoridis [131] has invented Bunch as a tool for modularization of software systems. This tool uses search techniques and treats the clustering process as a GPP. It has been tested to the largest graph with almost 10,000 nodes and 100,000 edges.

In genetic algorithm based clustering, a population with *ps* number of chromosomes is randomly generated with every chromosome representing a solution. The goodness of each chromosome is evaluated by a fitness function. Salim and Mohemad [132] has introduced mean inter-cluster molecular dissimilarity measure to calculate the fitness function as follows:

$$f_{\mathrm{SM}}(x) = 1 - \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} T_{ij}}{n^2}, \tag{4.1}$$

where T_{ij} represents the Tanimoto coefficient between cluster centroids and n is the number of centroids. In the meantime, Garai and Chaudhuri [133] defines the fitness function of a chromosome as follows:

$$f_{\mathrm{GC}}(x) = \sum_{i=1}^{k} D_{\mathrm{inter}}(C_i) - \sum_{i=1}^{k} D_{\mathrm{intra}}(C_i) \times |B|, \tag{4.2}$$

where $D_{\mathrm{intra}}(C_i)$ represents intra-distance in cluster C_i, $D_{\mathrm{inter}}(C_i)$ represents inter-distance of cluster C_i, and $|B|$ represents the size of dataset.

4.3 The Genetic Split-Merge Algorithm

A genetic split-merge algorithm that combines parallel genetic algorithm with split-and-merge algorithm is proposed to cluster the GO terms. The aim is to split the monolithic GO RDF/XML file into a number of smaller files. The parallel genetic algorithm is used because of its capability of being adaptive, efficient, robust, and a global search method that is suitable to address a situation where the search space is large. Moreover, parallel genetic algorithm optimizes its fitness function by utilizing the genetic operators to find an optimal solution. It can also be executed on a low-cost PC cluster using message passing interface libraries that are open source and easy to install. The split-and-merge algorithm is implemented to improve infeasible clusters in order to efficiently estimate the number *k* of clusters. Generally, the genetic split-merge algorithm works by decomposing the GO terms into a number of clusters and then automatically combines these clusters in several iterations until the best number *k* of clusters is found. The genetic split-merge algorithm uses cohesion-and-coupling metric to measure the goodness of the generated clusters. The genetic split-merge algorithm is expected to be capable of automatically identifying the number *k* of clusters, producing balanced clusters in terms of number of elements in each cluster, requires reasonable amount of processing time, and generates good clusters. The overview of the genetic split-merge algorithm is shown in Figure 4.1.

4.3.1 Chromosome Representation

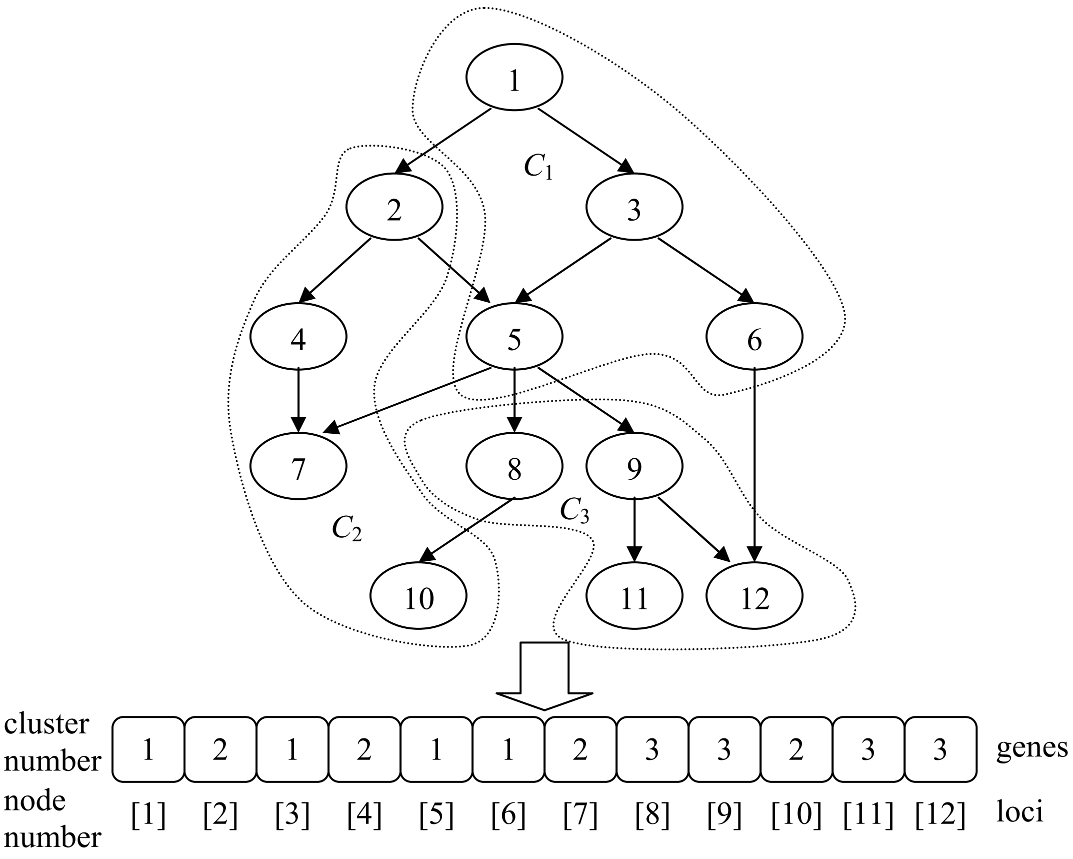
The GO graph is represented by a chromosome using 1D array of integers. The chromosome is built in a way where gene represents the cluster number, loci represents the node number, and the chromosome length represents the number of nodes in the GO graph. This encoding scheme allows any size of graph to be easily represented by the chromosome, increases the convergence velocity of the genetic split-merge algorithm, and makes the gene values to be simply assigned and interpreted. An example of chromosome representation of GO graph with 12 nodes and 3 clusters is shown in Figure 4.2.

```
Genetic-Split-Merge-Algorithm ($G$, $k_{\min}$);
Input: $G = \{V, E\}$ (a Gene Ontology graph) and $k_{\min}$ (a minimum number $k$
of clusters)
Output: $C = (C_1, C_2, \ldots, C_k)$ (a clustering)
begin
    $t := 0$;
    initialize $Pop(t)$; // note that $Pop(t) = \{x_1^t, \ldots, x_{ps}^t\}$ where $Pop(t)$ and
                             $x^t$ are the population and chromosome for generation
                             $t$ respectively and $ps$ is the size of population
    evaluate $Pop(t)$;
    while not termination-condition do
        $t := t + 1$;
        select $Pop(t)$ from $Pop(t-1)$;
        alter $Pop(t)$ by crossover and mutation operators;
        alter $Pop(t)$ by split and merge functions;
        evaluate $Pop(t)$;
    end-while
end
```

Figure 4.1: The genetic split-merge algorithm.

4.3.2 Crossover and Mutation Operators

Two classical and most often-used genetic operators, the crossover and the mutation operators, are employed during the reproduction phase. These operators are chosen since they work effectively with a chromosome that uses 1D array of integers and a fitness function that is based on the cohesion-and-coupling metric. The crossover operator performs a probabilistic process to create new offsprings by combining features of their parents. The mutation operator also performs a probabilistic process to modify one or more genes of each new offspring produced from the crossover process. The reason for using these operators in the genetic split-merge algorithm is to generate new population with higher total fitness in each generation.



cluster number	1	2	1	2	1	1	2	3	3	2	3	3	genes
node number	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	loci

Figure 4.2: An example of chromosome representation.

4.3.3 Split and Merge Functions

By adopting the split-and-merge algorithm, the *k* value in the chromosomes is refined and fixed. After every reproduction by the genetic operators, each new offspring goes through alteration process by the split function $S(x)$ and then by the merge function $M(x)$. The transformation is based on a cluster-by-cluster basis by making modification in a single chromosome ($S(x)$, $M(x)$: $x \rightarrow x'$), which is then evaluated by the fitness function $f_O(x')$ (refer to Equation 4.8). Through these functions, chromosomes with best number k of clusters and high fitness are recreated in each generation. Hence, it indirectly eliminates the process of producing solution with unsuitable number k of clusters and accelerates the pace for convergence.

The main objective of the split function $S(x)$ is to decompose each cluster in chromosome x into reasonable fragmented clusters. This function works by creating clone chromosomes $x_1^c \ldots x_n^c$ from the chromosome $x \in Pop(t)$. For each cluster $C_1 \ldots C_p$ in the clone chromosome x^c, the cluster C_p is divided into two clusters C_{p_1} and C_{p_2}. The chromosome x is replaced by the best clone chromosome x^c that satisfies the following criteria:

(i) The Quality of Clustering (QOC) of the clusters C_{p_1} and C_{p_2} in the clone chromosome x^c is higher than the QOC of the cluster C_p in the chromosome x.

(ii) The dependency index γ (refer to Equation 4.7) of the clusters C_{p_1} and C_{p_2} in the clone chromosome x^c must be greater than the dependency index threshold for small cluster $I_{\min}$.

The QOC of the clusters C_{p_1} and C_{p_2} in the clone chromosome x^c is computed as follows:

$$x^c.QOC(C_{p_1}, C_{p_2}) = \frac{\sum_{i=p_1}^{p_2} \alpha_i}{2} - \frac{\sum_{i=p_1, j=1}^{p_2, k} \beta_{i,j}}{2k-3}, \tag{4.3}$$

where α_i is the cohesion of the cluster i (refer to Equation 4.9) and $\beta_{i,j}$ is the coupling between clusters i and j (refer to Equation 4.10). The QOC of the cluster C_p in the chromosome x is calculated with the following equation:

$$x.QOC(C_p) = \alpha_p - \frac{\sum_{i=p, j=1}^{p,k} \beta_{i,j}}{k-1}. \tag{4.4}$$

The merge function $M(x)$ is carried out for combining the isolated clusters by repairing genes in the chromosome x when necessary. The objective is to guarantee that all the chromosomes repaired by the split function $S(x)$ are genuinely fit to be feasible and near optimal solution. The merge function $M(x)$ is invoked to combine clusters C_p and C_q in the chromosome $x \in Pop(t)$. If the trial consolidation fulfills the following conditions, then the clusters C_p and C_q are permanently merged:

(i) The QOC of the merged clusters C_p and C_q is higher than the QOC of the cluster C_p alone.

(ii) The dependency index γ of the merged clusters C_p and C_q must be less than the dependency index threshold for large cluster $I_{\max}$.

The QOC of the cluster C_p in the chromosome x is computed by Equation 4.5 as shown below:

$$x.QOC(C_p) = \alpha_p - \frac{\sum_{i=p,j=1}^{p,k} \beta_{i,j}}{k-2}. \tag{4.5}$$

The QOC of the merged clusters C_p and C_q in the chromosome x is calculated as follows:

$$x.QOC(C_p, C_q) = \frac{\alpha_p + \alpha_q}{2} - \frac{\sum_{i=p,j=1}^{p,k} \beta_{i,j} + \sum_{i=q,j=1}^{q,k} \beta_{i,j}}{2k-3}. \tag{4.6}$$

After undergoing the split and merge processes, any illegal chromosome is adjusted and then evaluated by the fitness function $f_O(x)$. The illegal chromosome contains one or more clusters which are empty. For example, given k = 4, the chromosome x = (4 1 1 4 1 4 4 4 1) is illegal because cluster number two and three are empty. In some cases the split and merge processes can cause clusters to further split or merge due to strong internal dependencies. This phenomenon creates unbalanced clusters and reflects the aim of creating modular GO RDF/XML files that are easy to be maintained, published, validated, and processed. Therefore, dependency index γ is introduced to stabilize the split-and-merge algorithm and to forbid it from producing micro or giant clusters during splitting or merging process. The dependency index γ_i of the cluster i is given by:

$$\gamma_i = \frac{N_i - k}{\sum_{j=1}^{k} N_j - 1}, \tag{4.7}$$

where N_i is the number of nodes in the cluster i and N_j is the total number of nodes in the GO graph. The target value for dependency index γ_i of the cluster i is 0. The maximum value is 1 which represents the worst case where most of the nodes form a large cluster. Meanwhile, negative value indicates pathological clusters with undersized number of nodes.

4.3.4 Fitness Function

The fitness function $f_{\mathrm{O}}(x)$ to partition the GO graph is based on the cohesion-and-coupling metric and is defined as follows:

$$f_{\mathrm{O}}(x)=\begin{cases}\dfrac{\sum_{i=1}^{k}\alpha_i}{k}-\dfrac{\sum_{i,j=1}^{k}\beta_{i,j}}{\frac{k(k-1)}{2}}-\sqrt{\dfrac{\sum_{i=1}^{k}(\gamma_i-\overline{\gamma})^2}{k}} & \forall k>1 \\ \alpha_i-\sqrt{\dfrac{\sum_{i=1}^{k}(\gamma_i-\overline{\gamma})^2}{k}} & k=1\end{cases} \quad . \tag{4.8}$$

The value of the fitness function $f_{\mathrm{O}}(x)$ vary between [-1…1]. A good quality chromosome has a high value of fitness function $f_{\mathrm{O}}(x)$. The cohesion α_i of the cluster i is calculated by:

$$\alpha_i=\frac{\mu_i}{\frac{N_i\left(N_i-1\right)}{2}}, \tag{4.9}$$

where N_i is the number of nodes in the cluster i and μ_i is the number of its internal edges. The coupling $\beta_{i,j}$ between clusters i and j is given by:

$$\beta_{i,j}=\begin{cases}0 & i=j \\ \dfrac{\varepsilon_{ij}}{N_iN_j} & i\neq j\end{cases}, \tag{4.10}$$

where N_i and N_j are number of nodes in the clusters i and j respectively and ε_{ij} is the number of edges from cluster i to cluster j.

4.3.5 Parallelization Process

Partitioning the GO graph is computationally intensive. This is due to the fact that the GO graph has a large number of nodes and paths. Furthermore, to obtain a good solution, it requires a multitude of chromosomes and many generations of population. This scenario becomes deteriorated when population for each generation is required to go through the reproduction process and the split and merge functions. To solve this problem, the genetic split-merge algorithm is paralleled by exploiting the advantages of island (coarse-grained) model [134]–[136]. It is implemented on a low-cost PC cluster using message passing interface libraries. The parallelization process of the genetic split-merge algorithm is shown in Figure 4.3.

4.4 Testing Preparation and Evaluation Measures

The GO data used in this chapter is in RDF/XML format as released in January 2007 (refer to Table 2.1). The data is compressed in a GZIP file named go_200701-assocdb.rdf-xml.gz. The data is updated monthly and can be downloaded from http://archive.godatabase.org/. The data comes without protein sequences and IEA evidence associations. Therefore, to include both of them into the fragmented GO RDF/XML files these data are taken from the MySQL format. The GO data in MySQL format is stored in a file named go_200701-seqdb-tables.tar.gz. The genetic split-merge algorithm and other algorithms that have been used for comparison are run on a 25-node low-cost PC cluster with 2.8GHz Pentium IV of processor, 512MB of memory, and 100Mbps of network speed. The operating system used is Fedora Core 5. The low-cost PC cluster is implemented using MPICH2 libraries [137]. The genetic algorithm used in this chapter is an enhancement of the existing GAlib C++ libraries created by Wall [138]. The parameters used to run the genetic split-merge algorithm are shown in Table 4.1.

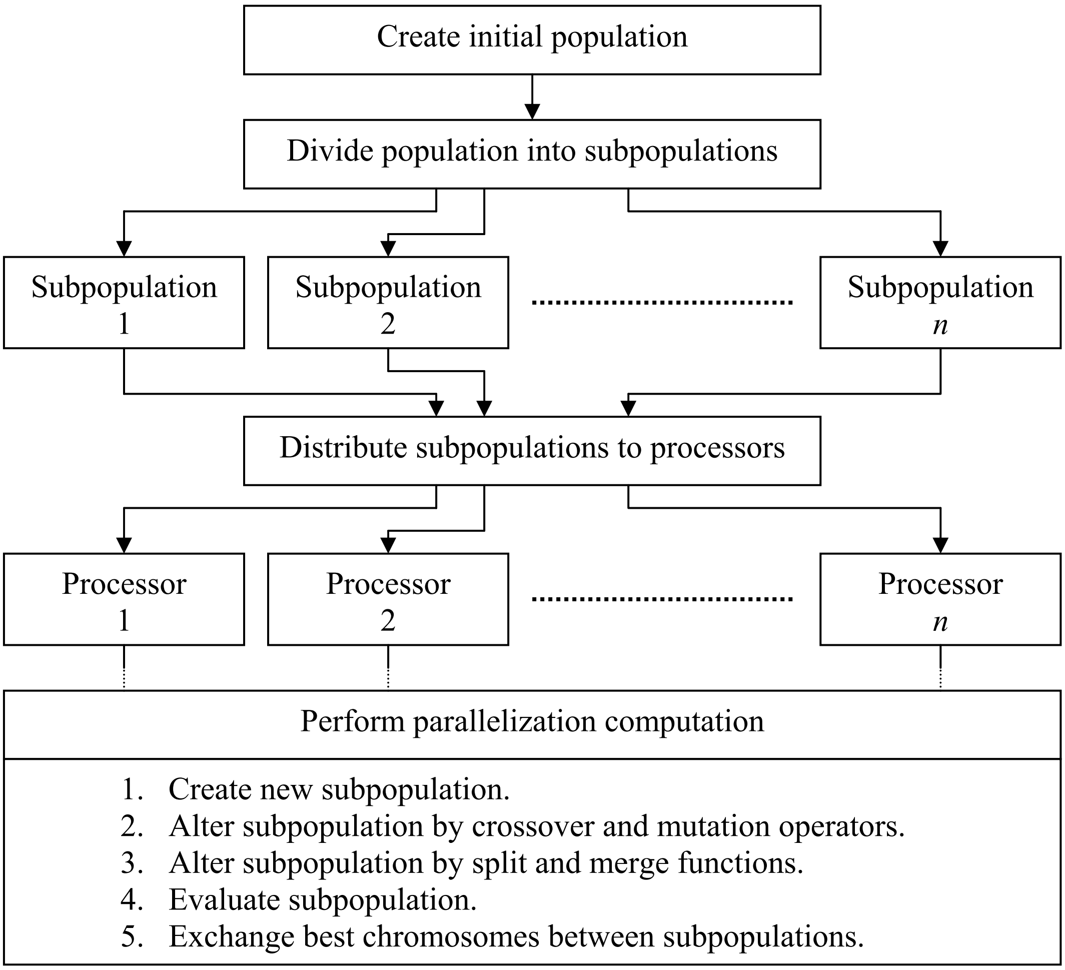

Create initial population
Divide population into subpopulations
Subpopulation 1
Subpopulation 2
Subpopulation n
Distribute subpopulations to processors
Processor 1
Processor 2
Processor n
Perform parallelization computation
1. Create new subpopulation.
2. Alter subpopulation by crossover and mutation operators.
3. Alter subpopulation by split and merge functions.
4. Evaluate subpopulation.
5. Exchange best chromosomes between subpopulations.

Figure 4.3: The parallelization flow of the genetic split-merge algorithm.

To evaluate the goodness of the clustering produced by the genetic split-merge algorithm, two validity measures are used: the Davies-Bouldin index (DBI) measure and the F-measure. The DBI measure is defined as follows:

$$\mathrm{DBI} = \frac{\sum_{i=1}^{k} \max_{i \neq j} \left\{ \frac{d_{\text{intra}}(i) + d_{\text{intra}}(j)}{d_{\text{inter}}(i,j)} \right\}}{k}, \tag{4.11}$$

where d_{intra} is the average distance of all patterns in cluster i to their cluster center c_i and d_{inter} is the distance of cluster centers c_i and c_j. Since the clustering objective is to maximize inter-cluster distance and to minimize intra-cluster distance, a good clustering therefore should have a small value of DBI. The DBI measure has been studied by [139]–[141]. On the other hand, the F-measure combines the precision and recall measures adopted from information retrieval. The F-measure is calculated as follows:

$$F = \frac{2PR}{P + R}, \tag{4.12}$$

where $P = \frac{N_{ij}}{N_i}$, $R = \frac{N_{ij}}{N_j}$, N_{ij} is the number of elements with cluster label i within cluster j, N_i is the number of elements with cluster label i, and N_j is the number of elements of cluster j. The F-measure values are in the interval [0, 1], and the larger value indicates better clustering quality. The F-measure has been used by Ma *et al.* [126], Cui *et al.* [142], and Watts and Porter [143] to validate the clustering results.

Table 4.1: Parameters of the genetic split-merge algorithm.

Item	Parameter
Size of population	100
Number of generations	400
Crossover probability	0.8
Mutation probability	0.01
Length of chromosome	22,954
Replacement percentage	0.5
Type of crossover	Partial match crossover
Type of mutation	Swap mutation
Type of genetic algorithm	Steady-state genetic algorithm
Scaling	Sigma truncation scaling
Fitness function	Maximizing preferences
Number of clone chromosomes	5
Dependency index threshold for small cluster	0.1
Dependency index threshold for large cluster	0.3
Number of subpopulations	25
Isolation time	10 generations
Number of emigrants	1
Type of replacement	Bad by best
Type of migration	Stepping stone

4.5 Results and Discussion

In order to justify the need for executing the genetic split-merge algorithm on a low-cost PC cluster, the effect of using different number of processors in the low-cost PC cluster is analyzed. The effect on the following factors is studied: number of generations to converge g_{best}, number of clusters found k_{found}, CPU time (in seconds), maximum value of fitness function $max\{f_{\text{O}}(x)\}$, DBI, and F-measure. Results in Table 4.2 show that a cluster of 25 processors is the ideal solution to handle the computational problem. Six factors, particularly the CPU time, were highly affected if more number of processors were removed. Otherwise, additional processors only slightly affected those factors.

Table 4.2: The effects of different number of processors used on the performance of the genetic split-merge algorithm.

Item	**Number of processors**					
	5	**10**	**15**	**20**	**25**	**30**
g_{best}	580	470	430	320	250	250
k_{found}	5	5	5	5	5	5
CPU time	12,652.7	2,472.8	996.0	353.5	32.4	31.9
$max\{f_{\text{O}}(x)\}$	0.1051	0.1168	0.1237	0.1305	0.1353	0.1356
DBI	1.75	1.68	1.63	1.58	1.52	1.50
F-measure	0.74	0.81	0.85	0.88	0.92	0.93

To assess the performance of our fitness function $f_{\text{O}}(x)$, its results are compared with fitness functions introduced by Salim and Mohemad $f_{\text{SM}}(x)$ [132] and Garai and Chaudhuri $f_{\text{GC}}(x)$ [133]. The dependency index γ is added to both fitness functions as well as to our fitness function $f_{\text{O}}(x)$ and different minimum number k of clusters k_{min} are used. The results in Table 4.3 show that the earliest number of generations to converge g_{best} is obtained by our fitness function $f_{\text{O}}(x)$ which appeared as early as after 250 generations. The results also show that if the minimum number k of clusters k_{min} is greater than the best number k of clusters k_{best}, then the number k of clusters found k_{found} is bound to it. Furthermore, the results show that our fitness function $f_{\text{O}}(x)$ provides the best value of CPU time (in seconds), DBI, and F-measure which are 32.4 seconds, 1.52, and 0.92 respectively.

Table 4.3: Comparison of different fitness functions.

$k_{\min}$	$f_{\mathrm{O}}(x)$					$f_{\mathrm{SM}}(x)$					$f_{\mathrm{GC}}(x)$				
	g_{best}	k_{found}	CPU time	DBI	F-measure	g_{best}	k_{found}	CPU time	DBI	F-measure	g_{best}	k_{found}	CPU time	DBI	F-measure
1	310	5	38.4	1.57	0.79	480	5	287.0	1.64	0.83	640	5	71.3	1.67	0.66
2	300	5	36.8	1.55	0.81	430	5	266.8	1.61	0.78	610	5	64.4	1.64	0.70
3	290	5	33.5	1.58	0.85	390	5	242.1	1.60	0.71	590	5	62.5	1.65	0.67
4	260	5	32.9	1.54	0.80	370	5	203.9	1.62	0.67	530	5	59.9	1.63	0.71
5	250	5	32.4	1.52	0.92	330	5	194.9	1.60	0.74	430	5	58.9	1.59	0.69
6	270	6	33.0	1.65	0.67	340	6	229.7	1.71	0.61	470	6	62.7	1.79	0.41
7	280	7	33.6	1.64	0.66	380	7	280.2	1.69	0.58	510	7	63.5	1.78	0.38
8	310	8	37.6	1.63	0.64	450	8	308.1	1.68	0.56	560	8	81.7	1.75	0.37
9	320	9	38.3	1.60	0.62	480	9	340.2	1.66	0.51	620	9	87.9	1.71	0.34
10	330	10	41.8	1.59	0.58	530	10	357.0	1.64	0.44	670	10	109.8	1.70	0.28

Table 4.4: Comparison of different clustering algorithms.

k or $k_{\min}$	Genetic split-merge algorithm			k-means			Fuzzy c-means			Support vector clustering		
	CPU time	DBI	F-measure	CPU time	DBI	F-measure	CPU time	DBI	F-measure	CPU time	DBI	F-measure
5	32.4	1.52	0.92	74.9	1.61	0.72	72.1	1.53	0.76	40.9	1.55	0.86
6	33.0	1.65	0.67	79.0	1.78	0.66	74.7	1.75	0.67	41.3	1.74	0.74
7	33.6	1.64	0.66	83.5	1.74	0.62	76.2	1.70	0.64	42.0	1.68	0.67
8	37.6	1.63	0.64	86.7	1.73	0.57	80.1	1.69	0.60	44.7	1.65	0.62
9	38.3	1.60	0.62	88.5	1.70	0.50	86.2	1.62	0.53	53.1	1.64	0.59
10	41.8	1.59	0.58	94.9	1.65	0.44	87.6	1.60	0.51	53.9	1.61	0.56

In this chapter, three most popular clustering algorithms are examined and compared with the genetic split-merge algorithm as shown in Table 4.4. As it is clear from the table, k=5 returns the best DBI and F-measure values for k-means, fuzzy c-means, and support vector clustering which are (1.61, 0.72), (1.53, 0.76), and (1.55, 0.86) respectively. The results indirectly prove that the k_{found}=5 returned by the genetic split-merge algorithm is the best number k of clusters k_{best}. On the other hand, the best CPU time (in seconds), DBI, and F-measure are obtained by the genetic split-merge algorithm when k=5 is examined. The clustering utilization as depicted in Figure 4.4 shows that the dependency index γ plays an important role in creating balanced clusters.

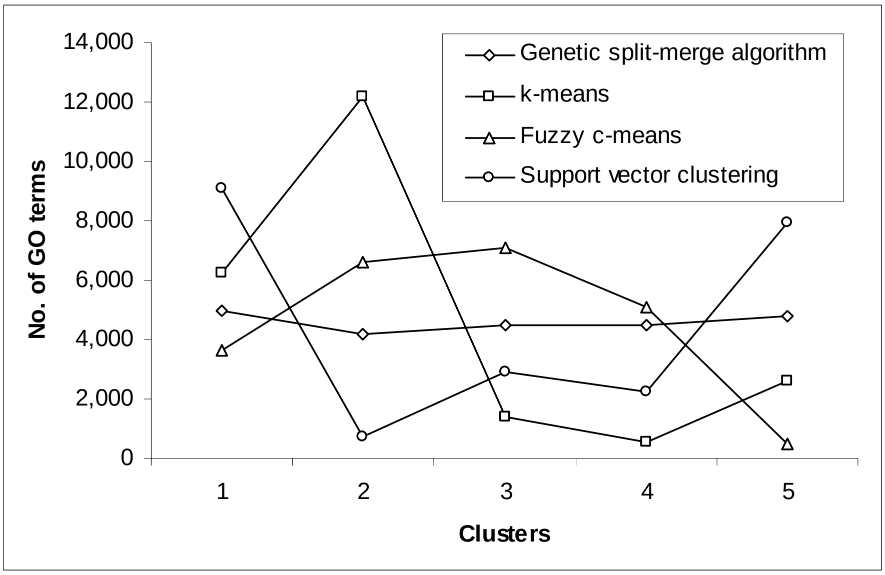


Figure 4.4: Cluster utilization of different clustering algorithms.

The comparison between the genetic split-merge algorithm and other automatic clustering algorithms such as evolutionary clustering [126], Laszlo and Mukherjee's algorithm [127], and hybrid niching genetic algorithm [128] is shown in Table 4.5. The results show that the genetic split-merge algorithm provides the best F-measure (0.92) and obtains the earliest number of generations to converge g_{best} (250 generations), the hybrid niching genetic algorithm offered the best DBI (1.49), and the best CPU time (in seconds) is 31.2 seconds that is taken by the evolutionary clustering. Further, k_{found}=5 is returned as the best number k of clusters k_{best} by all the four algorithms. Figure 4.5 illustrates how the other automatic clustering algorithms are unable to produce balanced clusters compared to the genetic split-merge algorithm.

Table 4.5: Comparison of different automatic clustering algorithms.

	g_{best}	k_{found}	**CPU time**	**DBI**	**F-measure**
Genetic split-merge algorithm	250	5	32.4	1.52	0.92
Evolutionary clustering	300	5	31.2	1.85	0.89
Laszlo and Mukherjee's algorithm	430	5	35.9	2.22	0.74
Hybrid niching genetic algorithm	320	5	40.9	1.49	0.78

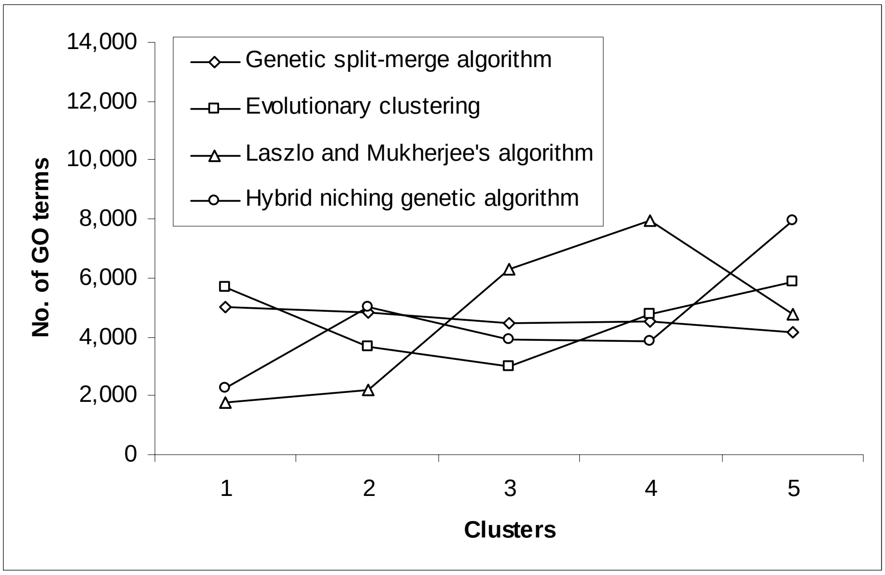


Figure 4.5: Cluster utilization of different automatic clustering algorithms.

An example of part of a smaller Gene Ontology RDF/XML file that has been split by the genetic split-merge algorithm is shown in Figure 4.6. The example shows how a Gene Ontology term “tRNA processing” (GO:0008033) includes “RNA processing” (GO:0006396) from the cluster C_2 (line 6) and “tRNA metabolic process” (GO:0006399) from the cluster C_3 (line 7). The figure also shows the inclusion of an IEA evidence association with the gene product “BC4V2_0_00030” (DDB0218427) from the dictyBase database (line 8–31) and its protein sequence (line 20–28). The example shows that by splitting the monolithic GO RDF/XML file, the smaller GO RDF/XML files can be easily maintained and become exhaustive.

```
1  <go:term rdf:about="http://localhost/go/cluster1#GO:0008033">
2     <go:accession>GO:0008033</go:accession>
3     <go:name>tRNA processing</go:name>
4     <go:definition>The process by which a pre-tRNA molecule is converted to a
5       mature tRNA, ready for addition of an aminoacyl group.</go:definition>
6     <go:is_a rdf:resource="&cluster2; http://localhost/go/cluster2#GO:0006396">
7     <go:is_a rdf:resource="&cluster3; http://localhost/go/cluster3#GO:0006399">
8     <go:association rdf:parseType="Resource">
9       <go:evidence evidence_code="IEA">
10        <go:dbxref rdf:parseType="Resource">
11          <go:database_symbol>DDB_REF</go:database_symbol>
12          <go:reference>10157</go:reference>
13        </go:dbxref>
14      </go:evidence>
15      <go:gene_product rdf:parseType="Resource">
16        <go:name>BC4V2_0_00030</go:name>
17        <go:dbxref rdf:parseType="Resource">
18          <go:database_symbol>DDB</go:database_symbol>
19          <go:reference>DDB0218427</go:reference>
20          <go:sequence>MSPRYKIIYEYIGKSFTGFQRLKYPVVKLPVQQVL
21            EDSLEKIHGYKIPIVGSSRTDHGVSAVGQVSHFDVKTRTSKSGI
22            EMPLLSPEELTMAINYNVGKEYLKSIRIIKTEIVDDKFHCRFNA
23            TSRTYLYRVMANCGRKQIPLELLDRVYLVGPILNLDEMRKAS
24            EMFIGTHDFSSFRSAKCSSTRPIRSISHIKIYDLPLPDIFQYNPSFQ
25            NISRSNTNYPIGDGEKNLDKKNTGLQYFGIEIKARAFLHNQVRI
26            MVASLIKVGEGEISIQQLEEIKDKKDRGAAPPTASPEPLTLLTV
27            SYDDPKVNPSTFQQQQQQQQQQQQQQQQQQQQQQQQQQQQ
28            QQS</go:sequence>
29        </go:dbxref>
30      </go:gene_product>
31    </go:association>
32    <!-- more associations -->
33 </go:term>
34 <!-- more Gene Ontology terms -->
```

Figure 4.6: An example of part of a smaller GO RDF/XML file.

The experimental results have shown that, unlike any other clustering algorithm such as k-means, fuzzy c-means, and support vector clustering, the proposed algorithm with the split-and-merge strategy can automatically find the best number k of clusters k_{best}. Compared to other automatic clustering algorithms such as evolutionary clustering [126], Laszlo and Mukherjee's algorithm [127], and hybrid niching genetic algorithm [128], the genetic split-merge algorithm is capable of producing balanced clusters. The experimental results have shown that the genetic split-merge algorithm requires reasonable amount of execution time and the generated clusters have better DBI and F-measure values compared to the existing clustering and automatic clustering algorithms. Furthermore, the users are allowed to set the minimum number *k* of clusters $k_{\min}$ they wish to maintain.

4.6 Summary

The GO RDF/XML is created to allow the GO data to be shared and reused across the WWW in a way that it can be interpreted and processed by human and machine alike. The GO RDF/XML has been used by numerous bioinformatics tools for analyzing the GO annotation results, analysis of microarray gene expression data, and searching and browsing the GO. However, the increase in size of the GO data has caused the GO RDF/XML difficult to be maintained, published, validated, and processed. One of the solutions is splitting the GO RDF/XML into smaller files. Splitting the monolithic GO RDF/XML file requires the GO terms to be grouped into a number *k* of clusters. Clustering the GO terms is a difficult combinatorial problem and can be modeled as a GPP since they are structured as a DAG. Additionally, deciding the number *k* of clusters to use is not easily perceived and is a hard algorithmic problem. In this chapter, a genetic split-merge algorithm that combines parallel genetic algorithm with split-and-merge algorithm is proposed to handle these problems. The genetic split-merge algorithm uses cohesion-and-coupling metric to measure the goodness of the generated clusters. The performance of the genetic split-merge algorithm has been compared and an example of a smaller GO RDF/XML file is given to show the effectiveness of the genetic split-merge algorithm.

CHAPTER 5

THE GENETIC SIMILARITY ALGORITHM FOR SEARCHING THE GENE ONTOLOGY TERMS

5.1 Introduction

The GO is a collection of nearly 23 thousand terms for providing consistent terms to describe gene and gene product attributes in any organism found in heterogeneous databases. The GO terms are structured, controlled vocabularies organized as a DAG in three aspects: cellular component, biological process, and molecular function. Let GO be a graph $G = \{V, E\}$, where V is a set of nodes representing the GO terms and E is a set of pairs of nodes representing relationships between the GO terms. The GO terms can have more than one parent, as well as multiple children. The GO terms are connected by two relationships: the “is-a” relationship, e.g. “chromatin binding” (GO:0003682) and “structure-specific DNA binding” (GO:0043566) are parents of “chromatin DNA binding” (GO:0031490); and the “part-of” relationship, e.g. “cytoplasmic part” (GO:0044444) is part of “cytoplasm” (GO:0005737). The GO have been utilized in bioinformatics research and has numerous practical applications including prediction of protein-protein interaction networks [30], protein classification [144], prediction of protease types [145], and functional interpretation of microarray data [146].

However, the existing GO browsers that support basic needs of bioscientists for searching the GO terms still use a conventional approach based on keyword matching. Thus, for bioscientists, finding a group of semantically similar GO terms is time consuming and a tedious task. For example, the keyword matching is not capable of computing the relationship between “intracellular organelle” (GO:0043229) and “cytoplasm” (GO:0005737) even though they share the same parent “intracellular part” (GO:0044424) because their names do not exactly or approximately match. Therefore, a GO browser named *basic* UTMGO is introduced in this chapter to overcome the weaknesses of the existing GO browsers. The *basic* UTMGO uses a genetic similarity algorithm that incorporates the parallel genetic algorithm and the semantic similarity measure algorithm. The parallel genetic algorithm is used to generate a solution consisting of a group of semantically similar GO terms that best match to the query GO term, and to accelerate the search in the large GO graph. The search space of the GO graph, *g*(*k*), is astronomical and varies between:

$$2^{\frac{k(k-1)}{2}} \leq g(k) \leq 3^{\frac{k(k-1)}{2}}, \qquad (5.1)$$

where *k* is the number of nodes in the GO graph. Currently the GO graph consists of 22,954 nodes, so the search space of the GO graph is between $2^{263,431,581}$ and $3^{263,431,581}$. A parallel genetic algorithm optimizes its fitness function by utilizing the genetic operators to find an optimal solution. It can also be executed on a low-cost PC cluster using message passing interface libraries that are open source and easy to install. The semantic similarity measure algorithm is added into the parallel genetic algorithm to measure the similitude strength between the GO terms during the creation of initial population and calculation of fitness value. The semantic similarity measure algorithm used is a combination of information content (node-based) and conceptual distance (edge-based). The information content is used to get the amount of information the GO terms share in common, whereas the conceptual distance is applied to know the depth and the local network density of the GO terms.

The remainder of the chapter consists of related work in semantic similarity measure and genetic algorithm and existing tools for searching the GO terms (Section 5.2), technical description of the semantic similarity measure algorithm (Section 5.3), detailed explanation of the proposed genetic similarity algorithm

(Section 5.4), step-by-step explanation of the *basic* UTMGO (Section 5.5), description of the testing environment and evaluation measures used in this chapter (Section 5.6), the results and discussion of experiments (Section 5.7), and followed by the summary (Section 5.8).

5.2 Related Work

Semantic similarity measures play an important role in information retrieval and natural language processing. Example applications include characterization of human regulatory pathways [147], linguistic modeling [148], computer-assisted inter-observer consensus [149], and semantic feature ratings [150]. The choice of semantic similarity measure has the ability to improve the recall and precision of information retrieval by identifying the relation between concepts. This is done by calculating the distance or the amount of information in common between the two concepts being analyzed. Most of the popular measures are based on taxonomic or ontological structure [151–154]. These measures have been analyzed by Budanitsky and Hirst [155], and the evaluation of WordNet (http://wordnet.princeton.edu/) based semantic similarity measures in their study shows that the Jiang and Conrath semantic similarity measure [153] provides the best results. The Jiang and Conrath semantic similarity measure is a combined approach that inherits the conceptual distance approach enhanced with the information content approach. The basic calculation of the Jiang and Conrath semantic similarity measure is expressed as:

$$dist(c_1, c_2) = IC(c_1) + IC(c_2) - 2 \times sim(c_1, c_2), \tag{5.2}$$

where $IC(c) = -\log P(c)$, $sim(c_1, c_2) = \max_{c \in S(c_1, c_2)} \{IC(c)\}$, c is some concept being studied, $P(c)$ is the probability of encountering an instance of concept c, and $S(c_1, c_2)$ is the set of concepts that subsume both c_1 and c_2.

Lord *et al.* [156] has studied the Resnik [154] semantic similarity measure on the GO. They have only considered the GO annotations in Swiss-PROT and the “is-a" relationship. Their work has been extended by Popescu *et al.* [157]. In the meantime, Sevilla *et al.* [158] has compared different semantic similarity measures proposed by Lin [152], Jiang and Conrath, and Resnik. They conclude that the Resnik semantic similarity measure outperforms the other semantic similarity measures. However, their comparisons are based on the gene products rather than the GO terms, and they used the subsets of the GO terms and annotations. Therefore, in this study we use the Jiang and Conrath semantic similarity measure to compute the semantic similarity between pairs of GO terms rather than between pairs of gene products, and we use all the GO terms and annotations provided by the GO Consortium including the "part-of" relationships. The Jiang and Conrath semantic similarity measure is selected since both notions of the shared information content and the conceptual distance of the GO terms in the GO graph are considered as discussed in Section 5.1.

A genetic algorithm is selected because its capabilities as a machine learning technique have been recognized in the information retrieval field. This is due to its capability of being adaptive, efficient, robust, and a global search method that is suitable to address a situation where the search space is large. The properties of the genetic algorithm are as follows: a chromosome (a string of symbols called genes) to represent a solution, an allele to represent the value of the gene (it is usually a binary bit {0, 1}, an integer, or a real number), loci to represent the positions of the genes in the chromosome, a population to represent a set of chromosomes, a fitness function to evaluate each chromosome, a set of genetic operators to generate a new population, and a selection method to select fitter chromosomes for the next generation. The genetic algorithm starts with an initialization step in which an initial population is generated at random. Then it evolves with the following steps in each generation: evaluation of fitness function (the value of each chromosome in the population is calculated according to the fitness function), selection (multiple chromosomes are stochastically selected from the current population based on their fitness to form a new population), and a genetic operation (modification is performed to a newly generated population). These steps are repeated until either a maximum number of generations have been produced or a satisfactory fitness level has been reached for the population. Some reviews of genetic algorithms can be found in [159–161]. Implementations of the genetic algorithm in information retrieval are normally related to web search [162], gene selection [163], spatial information retrieval [164], and document retrieval [165].

For searching the GO terms, most of the present GO browsers respond to user queries by retrieving relevant GO terms based on keyword matching. A list of tools for searching and browsing the GO terms can be found at http://www.geneontology.org/GO.tools.browsers.shtml. Among the popular GO browsers are:

(i) AmiGO is a GO browser developed by the GO Consortium. The keyword-based search is executed either by "exact" or "contains" match over the GO term accession number, name, or synonyms. This tool also allows a user to use a gene product or a protein sequence as a search input.

(ii) GenNav is a GO browser that uses string matching method namely "exact" or "approximate" match that responds to a given GO term or gene product. GenNav is maintained by the United States National Library of Medicine (US NLM).

(iii) QuickGO is a GO browser that allows a user to retrieve the GO terms by "exact" or "wildcard" search for the GO term accession number, name, synonyms, definitions, or comments. This web-based GO browser can be found at the website of the EBI.

(iv) TAIR Keyword Browser is a GO browser that uses the GO term accession number or name as an input and then performs either "contains", "start with", "end with", or "exact" match. This tool is developed by TAIR.

Moreover, DynGO [105] and FuSSiMeG [166] are recently developed GO browsers that perform the semantic similarity search over the GO terms. However, the DynGO has only focused on the information content and has overlooked the role of conceptual distance in finding the significant GO terms. Whereas, the FuSSiMeG is not capable of returning more than one GO term for each query.

5.3 The Semantic Similarity Measure Algorithm

The semantic similarity measure algorithm, as shown in Figure 5.1, takes as input a set of subgraphs of the GO graph and the query GO term. It returns a set of subgraphs of the GO graph with assigned term similarity score for each node in the subgraphs. The term similarity score is used for generation of the initial population and evaluation of the fitness function. The semantic similarity measure algorithm described in this section is adopted from the Jiang and Conrath. It is simplified, and a direct explanation of how the GO is applied to their semantic similarity measure is given.

```
Semantic-Similarity-Measure-Algorithm (G, q);
Input: $G = \{G_1, G_2, \ldots, G_m\}$ (a set of subgraphs of the GO graph) and q
(a query GO term)
Output: $G' = \{G'_1, G'_2, \ldots, G'_m\}$ (a set of subgraphs of the GO graph with
assigned term similarity score)
begin
   for i := 1 to m do   // where m is the number of subgraphs
      for j := 1 to n do   // where n is the number of nodes in the
                              subgraph $G_i$
            calculate the information content $IC(c^i_j)$;  // where $c^i \in G_i$
            calculate the depth $D(c^i_j)$;
            calculate the local network density $E(c^i_j)$;
            calculate the semantic distance $dist(q, c^i_j)$;
            calculate the term similarity score $sim(q, c^i_j)$;
      end-for
   end-for
end
```

Figure 5.1: The semantic similarity measure algorithm.

5.3.1 Information Content Approach

The information content is computed according to the association: a source that presents information shared among the GO terms. The association is a table that stores annotations which provide links between GO terms and gene products that are supported by evidence codes and literature references. For example, gene product "rpl23-A" (*Chloroplast 50S ribosomal protein L23*, GR:P12097), an *Oryza sativa* species from Gramene (http://www.gramene.org) database, is shared among GO terms like "plastid" (GO:0009536), a cellular component that is supported by an evidence code of Inferred from Curator (IC) and a literature reference PMID:12520024; "RNA binding" (GO:0003723), a molecular function, is supported by an evidence code of inferred from Reviewed Computational Analysis (RCA) and literature reference GR.REF:8030; and "translation" (GO:0006412), a biological process, is supported by an evidence code of RCA and literature reference GR.REF:8030. These links are used to calculate the term similarity score between these three GO terms even though they are not directly connected by the "is-a" or "part-of" relationships, are from different categories, and do not have similar keywords. The information content of the GO term $IC(c)$ is represented as follows:

$$IC(c) = -\log(P(c)), \tag{5.3}$$

where $P(c)$ is the probability of occurrence of a GO term c in the association. The probability is measured using maximum likelihood estimation as given below:

$$P(c) = \frac{freq(c)}{N}, \tag{5.4}$$

where N is the total number of occurrences in the association and $freq(c)$ is the number of times that the GO term c and all its descendants occur in the association. The frequency of the GO term c is defined as follows:

$$freq(c) = \sum_{c \in descendants(c_i)} occur(c_i), \tag{5.5}$$

where $descendants(c)$ is a function that returns a set of GO terms that are the descendants of the GO term c. Note that if a GO term c_1 is an ancestor of a GO term c_2, then $freq(c_1) \geq freq(c_2)$ since the GO term c_1 subsumes the GO term c_2 and all its descendants. Therefore, $P(c)$ is larger when the GO term c is nearer to the root term c_0, and $IC(c_1) \leq IC(c_2)$.

5.3.2 Conceptual Distance Approach

The conceptual distance of a GO term is calculated based on the depth and the local network density factors. The depth is referred to as the distance of the GO term in the hierarchy of the GO graph. The local network density is related to the number of children that span out from the GO term. The depth of the GO term $D(c)$ is given as follows:

$$D(c) = \left(\frac{d(c)+1}{d(c)} \right)^{\alpha}, \tag{5.6}$$

where $d(c)$ is the level of the GO term c in the GO graph. The depth of the root term c_0 is 1, and it increases as the altitude of the GO term decreases in the hierarchy. The parameter α controls the degree of how much the depth factor contributes to Equation 5.6, and $\alpha \geq 0$.

The local network density of the GO term $E(c)$ is given by the following equation:

$$E(c) = \left((1-\beta) \times \frac{\overline{E}}{e(c)} \right) + \beta, \tag{5.7}$$

where $e(c)$ is the number of edges that begin from the GO term c and $\overline{E}$ is the number of edges divided by the number of GO terms in the GO graph. The parameter β controls the degree of how much the local network density factor contributes to Equation 5.7, and $0 \leq \beta \leq 1$. The effect of multiple inheritances is not considered in Equation 5.7 since they have been considered during calculation of the information content as mentioned in Equation 5.5. Furthermore, the term similarity score between GO terms c_m and c_n is calculated according to the shortest path that links both of the GO terms via their nearest shared ancestor as formulated in Equation 5.8 and 5.9.

Note that the parameters α and β become less important when α approaches 0 and β approaches 1, since $D(c)$ and $E(c)$ will reach 1 respectively. Furthermore, Equation 5.6 and 5.7 are equal when $\alpha = 0$ and $\beta = 1$.

5.3.3 The Hybrid Approach

The hybrid approach is derived from the notion of the conceptual distance, and by incorporating the information content as a decision factor. Given a sequence of GO terms c_1, …, c_n representing the path from GO term c_1 to c_n with length n. The hybrid approach computes the semantic distance between GO terms c_1 and c_n by the following formula:

$$dist(c_1, c_n) = \sum_{i=0}^{n-1} D(c_i) \times E(c_i) \times \left(IC(c_{i+1}) - IC(c_i)\right), \tag{5.8}$$

where $dist(c_1, c_n)$ is the summation of edge weights along the shortest path that links c_1 with c_n. Thence, the semantic distance between GO terms c_m and c_n is quantified as given below:

$$dist(c_m, c_n) = dist(c_1, v_m) + dist(c_1, v_n), \tag{5.9}$$

where GO term c_1 is the nearest shared ancestor of GO terms c_m and c_n. As the semantic distance is founded on the difference between the information content, the normalization of the semantic distance is given by:

$$dist_{norm}(c_m, c_n) = \min\{1, \frac{dist(c_m, c_n)}{\max\{IC(c)\}}\}. \tag{5.10}$$

Therefore, the term similarity score between GO terms c_m and c_n is measured by converting the semantic distance as follows:

$$sim(c_m, c_n) = 1 - dist_{norm}(c_m, c_n). \tag{5.11}$$

Note that $0 \leq sim(c_m, c_n) \leq 1$ because $0 \leq dist_{norm}(c_m, c_n) \leq 1$.

5.4 The Genetic Similarity Algorithm

An overview of the genetic similarity algorithm is shown in Figure 5.2. The genetic similarity algorithm takes the GO graph and a query GO term as an input.

The best chromosome representing a set of GO terms that have higher term similarity score to the query GO term is returned by the genetic similarity algorithm. The genetic similarity algorithm uses the semantic similarity measure algorithm to calculate the term similarity score which is the semantic similarity measure between each GO term and the query GO term.

```
***Genetic-Similarity-Algorithm*** ($G$, $q$);
**Input:** $G$ (a GO graph) and $q$ (a query GO term)
**Output:** $x_{\text{best}}$ (the best chromosome representing a set of GO terms that
have higher term similarity score to the query GO term)
***begin***
   preprocessing by semantic similarity measure algorithm;
   $t := 0$;
   initialize $Pop(t)$; // note that $Pop(t) = \{x_1^t, \ldots, x_{ps}^t\}$ where $Pop(t)$
                         and $x^t$ are the population and chromosome for
                         generation $t$ respectively and $ps$ is the size of
                         population
   evaluate $Pop(t)$;
   ***while not*** termination-condition ***do***
       $t := t + 1$;
       select $Pop(t)$ from $Pop(t-1)$;
       alter $Pop(t)$ by crossover and mutation operators;
       evaluate $Pop(t)$;
   ***end-while***
***end***
```

Figure 5.2: The genetic similarity algorithm.

5.4.1 Preprocessing

The first step of the genetic similarity algorithm is the calculation of the term similarity score between each node in the subgraphs of the GO graph and the query GO term. The GO graph is partitioned into several subgraphs in order to make calculation of the term similarity score and generation of the initial population easier and faster. The preprocessing step is done by the semantic similarity measure algorithm to improve the quality of the chromosome. This is done by setting the positions of nodes in the chromosome before the initialization step. Thus, the first chromosome created contains the nodes with the highest term similarity score in each subgraph. The second chromosome contains the second best and so on, as shown in example in Figure 5.3 for a GO graph with 4 subgraphs and 20 nodes in which "4" is the query GO term. Note that the GO term accession number is mapped to the node number according to the identification in the "term" table.

Step 1: Given a GO graph with 4 subgraphs and 20 nodes.

Given a query GO term "4"

Subgraph 1	Subgraph 2	Subgraph 3	Subgraph 4
1	4	2	5
3	6	7	8
9	10	12	15
13	11	16	19
20	14	18	
	17		

Step 2: Calculate the term similarity score between each node in the subgraphs and the query GO term "4".

Subgraph 1	Subgraph 2	Subgraph 3	Subgraph 4
1 (0.29)	4 (1.00)	2 (0.38)	5 (0.41)
3 (0.13)	6 (0.23)	7 (0.76)	8 (0.27)
9 (0.33)	10 (0.16)	12 (0.80)	15 (0.45)
13 (0.66)	11 (0.73)	16 (0.19)	19 (0.20)
20 (0.05)	14 (0.28)	18 (0.42)	
	17 (0.51)		

Step 3: Sort the nodes in each subgraph according to their term similarity score.

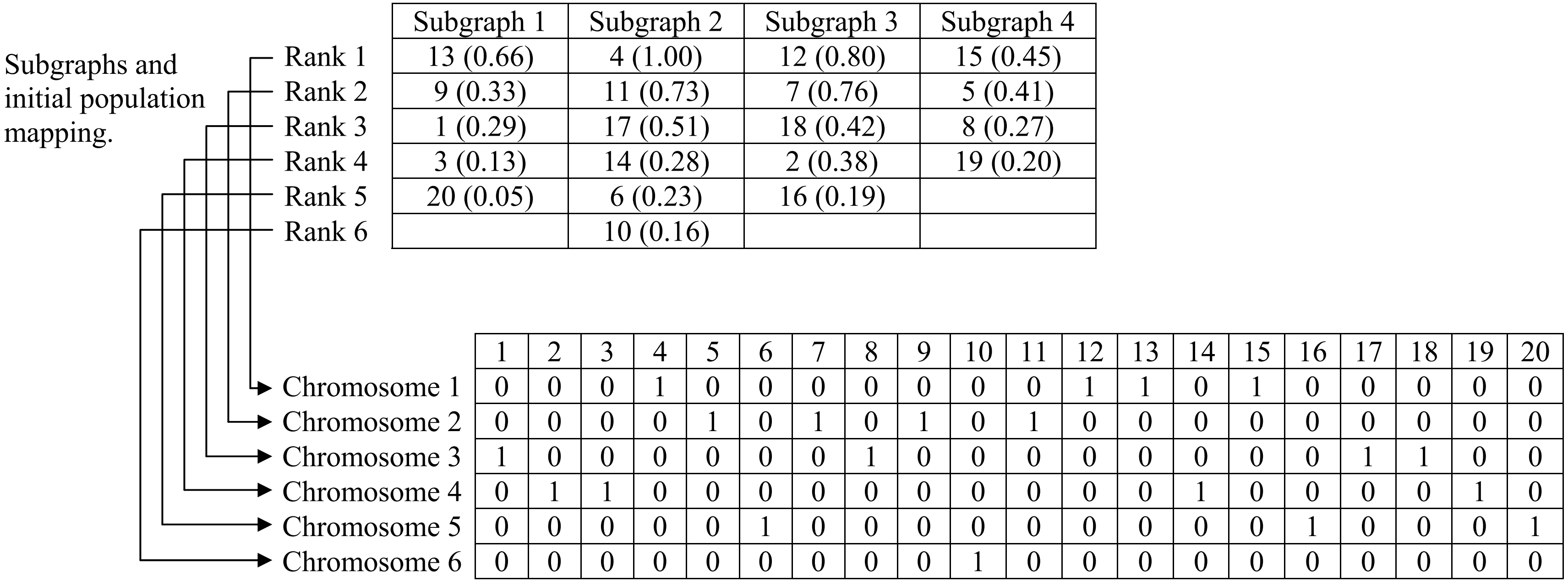
	Subgraph 1	Subgraph 2	Subgraph 3	Subgraph 4
Rank 1	13 (0.66)	4 (1.00)	12 (0.80)	15 (0.45)
Rank 2	9 (0.33)	11 (0.73)	7 (0.76)	5 (0.41)
Rank 3	1 (0.29)	17 (0.51)	18 (0.42)	8 (0.27)
Rank 4	3 (0.13)	14 (0.28)	2 (0.38)	19 (0.20)
Rank 5	20 (0.05)	6 (0.23)	16 (0.19)	
Rank 6		10 (0.16)		

Figure 5.3: An example of preprocessing.

5.4.2 Chromosome Representation

Based on the results returned by the semantic similarity measure algorithm, the initial population is generated according to the following representations: population size is the size of the subgraph with the highest node compared to other subgraphs; chromosome length is the number of nodes in the GO graph; loci represent the node number; a gene specifies whether a node in the pool of nodes is represented by a chromosome or not; and an allele is formed by two binary elements either 0 or 1, where 1 shows presence (retrieved) and 0 shows absence (not retrieved) of a node in a chromosome.

A chromosome is created by taking a node from each subgraph beginning with the ones with higher term similarity score, as shown in example in Figure 5.4. If the cardinality of a subgraph is smaller than the number of chromosomes to be produced, then that subgraph will not be present in each chromosome. An example of mapping of a GO graph into a chromosome is shown in Figure 5.5. This representation is crucial to ensure that the large GO graph can be presented with a simple and straightforward representation; the processing time taken to converge can be shortened since the chromosome is represented using 1D binary string; and the evolution of the genetic similarity algorithm is started with an initial population such that $t_1(x_i) \geq t_1(x_j)$, where $t_1(x)$ is the sum of the term similarity score of the nodes in a chromosome x, $\forall i,j \in \{1, 2, \ldots, ps\}$, ps is the size of population, and $i < j$.



Subgraphs and initial population mapping.

	Subgraph 1	Subgraph 2	Subgraph 3	Subgraph 4
Rank 1	13 (0.66)	4 (1.00)	12 (0.80)	15 (0.45)
Rank 2	9 (0.33)	11 (0.73)	7 (0.76)	5 (0.41)
Rank 3	1 (0.29)	17 (0.51)	18 (0.42)	8 (0.27)
Rank 4	3 (0.13)	14 (0.28)	2 (0.38)	19 (0.20)
Rank 5	20 (0.05)	6 (0.23)	16 (0.19)	
Rank 6		10 (0.16)		

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Chromosome 1	0	0	0	1	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0	0
Chromosome 2	0	0	0	0	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0
Chromosome 3	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0
Chromosome 4	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0
Chromosome 5	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	1
Chromosome 6	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0

Figure 5.4: An example of generating initial population. Note that subgraphs “1” and “3” are not present in chromosome “6” and subgraph “4” is not present in chromosome “5” and “6” since their cardinality is smaller than the size of population.

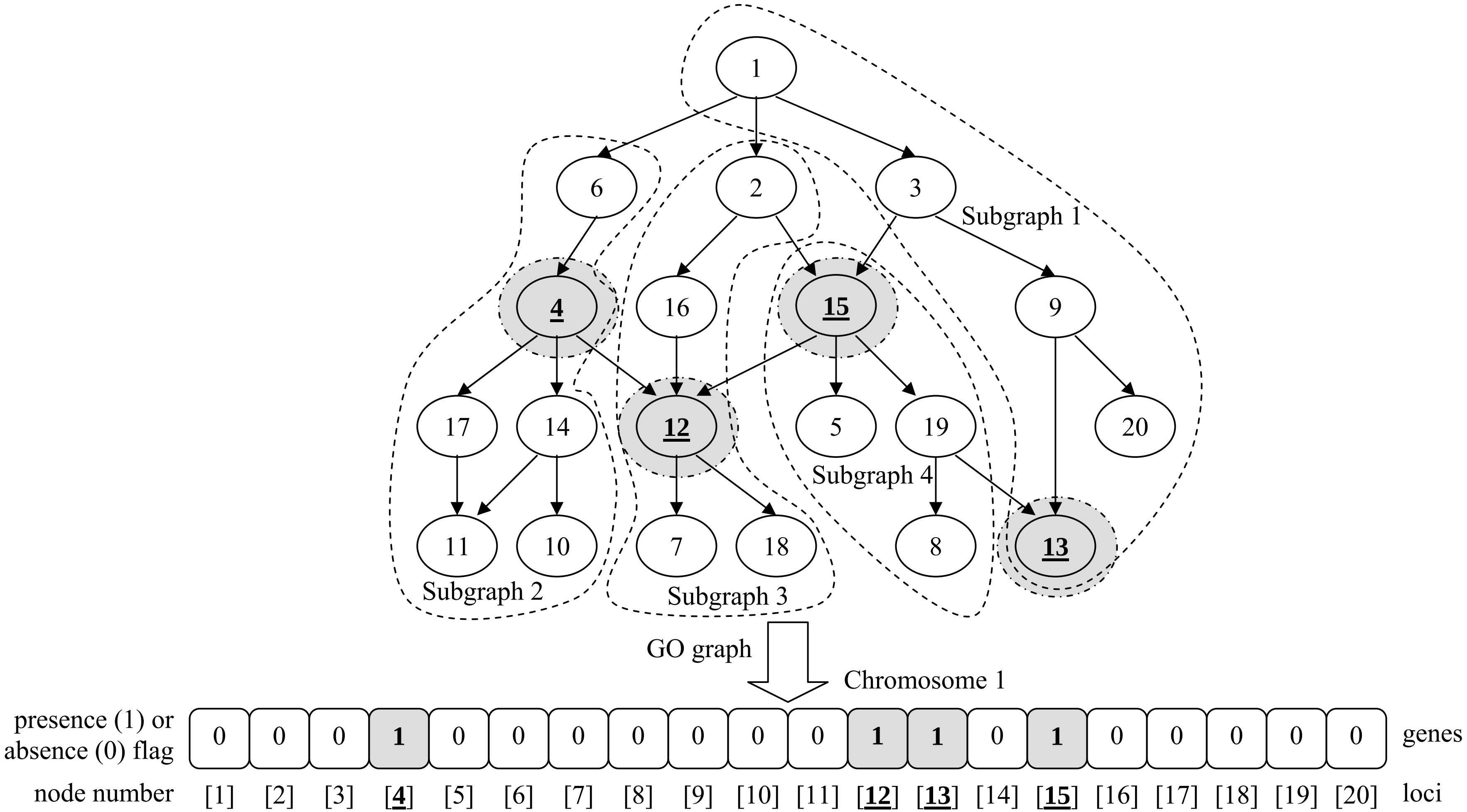


Figure 5.5: An example of mapping of a GO graph into a chromosome. The mapping of nodes "13", "4", "12", and "15" with the highest term similarity score from subgraphs "1", "2", "3", and "4" respectively into chromosome "1".

5.4.3 Crossover and Mutation Operators

In order to keep the genetic similarity algorithm as generic as possible, it uses normal crossover and mutation operators. These operators are chosen since they are formed effectively with a simple 1D binary string representation and with a fitness function that uses the semantic similarity measure. At each generation, the genetic similarity algorithm implements the fitness function as criteria to evaluate the goodness of each chromosome of the current population to create a new set of artificial creatures (a new population). Thence, the fitness value of the best chromosome in each generation can be maximized, as shown in example in Figure 5.6.

The above objective is attained by the crossover and mutation operators that try to improve the total fitness value of the current population by fixing the old ones. Through the crossover operator, the chromosomes reproduced in the new mating pool are matched randomly and afterward each couple of chromosomes, say x_a and x_b, undergoes a cross change. Then, the mutation operator plays a secondary role to forbid an irrecoverable loss of potentially useful information which occasionally crossover can cause. This operator conducts a random alteration of the allelic value of a chromosome.

5.4.4 Fitness Function

The fitness function used focuses on maximizing the preferences for term similarity score. The decision is inspired by the demand of searching for a set of GO terms with higher term similarity score that perfectly match the query GO term. The fitness function $f(x)$ for chromosome x is shown below:

$$f(x) = \chi \times t_1(x) + \delta \times t_2(x), \tag{5.12}$$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	*f*(*x*) value
Generation 0	0	0	0	1	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0	0	145.77
Generation 10	0	0	0	0	1	0	1	0	1	0	1	0	0	1	0	0	0	0	1	0	202.71
Generation 20	0	0	0	0	1	0	1	0	1	0	1	0	0	1	0	0	0	0	1	0	202.71
Generation 30	0	0	0	0	1	0	1	0	1	0	1	0	0	1	0	0	0	0	1	0	202.71
Generation 40	0	0	0	0	1	1	1	1	1	0	1	0	0	1	0	0	0	0	1	0	336.54
Generation 50	0	0	0	0	1	1	1	1	1	0	1	0	0	1	0	0	0	0	1	0	336.54
Generation 60	0	0	0	1	1	1	1	1	1	0	1	0	0	1	0	0	0	0	0	0	337.34
Generation 70	0	0	0	1	1	1	1	1	1	0	1	0	0	1	0	0	0	0	0	0	337.34
Generation 80	0	0	0	1	1	1	1	1	1	0	1	0	0	1	0	0	0	0	0	0	337.34
Generation 90	0	1	0	1	1	1	1	1	0	0	1	0	1	1	0	0	0	0	0	0	504.72
Generation 100	0	1	0	1	1	1	1	1	1	0	1	0	1	0	0	0	0	1	0	0	1,005.19

Figure 5.6: An example of the best chromosome produced by mutation and crossover operators. Note that the evolution stopped after a convergence occurred at 100 generations, the fitness value of the best chromosome is 1,005.19, and the best chromosome returns {“2”,“4”, “5”, “6”, “7”, “8”, “9”, “11”, “13”, “18”} as a set of GO terms that semantically similar to the query GO term “4””.

where χ and δ are control parameters so that the contributions given by factors $t_1(x)$ and $t_2(x)$ are harmonious. The value of the fitness function is stated as a positive value that is higher for the best chromosome.

The fitness function comprises two factors. The first factor is the sum of the term similarity score of the nodes in chromosome x, and is given as follows:

$$t_1(x) = \sum_{u_i \in x} score(u_i)\,, \tag{5.13}$$

where $score(u_i)$ is the term similarity score between the query GO term and nodes that are present in chromosome x. This factor considers the positive effect of having as many nodes with high term similarity score as possibly present in a chromosome. Nonetheless, a chromosome with many nodes with low score could create a fitness value higher than another one with a few good nodes. To avoid this consequence, the dimension index $t_2(x)$ is introduced as follows:

$$t_2(x) = \frac{k}{abs(cnt(x) - ID) + 1}\,, \tag{5.14}$$

where k is the number of nodes in the GO graph, $cnt(x)$ is the number of nodes present in chromosome x, and ideal dimension ID is the number of matched GO terms that are preferred to be returned to the user. Note that $0 < t_2(x) \leq k$ since if the number of nodes present in chromosome x is exactly equal to the ideal dimension, then maximum k is reached. Otherwise, it is rapidly lessened when the number of nodes present in chromosome x is smaller or greater than the ideal dimension.

5.4.5 Parallelization Process

The major computational challenge of searching a group of semantically similar GO terms is the size of the search space of the GO graph because the GO graph has almost 23 thousand nodes and almost 2.0 million paths. Moreover, to obtain a good solution, it requires a multitude of chromosomes, many generations of population, and it undergoes several iterations of the genetic operation by crossover and mutation operators. To overcome these matters, the genetic similarity algorithm is parallelized by exploiting the advantages of the island (coarse-grained) model [134–136] as shown in Figure 5.7. It is implemented on a low-cost PC cluster using message passing interface libraries. The core process of the parallelization is to divide the population into equal size subpopulations. Hereafter, each subpopulation is assigned to a processor where it evolves independently. During the process, a group of best chromosomes called emigrants are transferred to replace a group of worst chromosomes among the subpopulations. This migration process is performed periodically at certain cycles of generations called isolation time. The rationale for implementing the island model is to reduce the execution time by decreasing the communication overhead involved in the exchange of chromosomes between processors, and to improve the quality of the solutions reached by increasing population sizes without increasing the time complexity.

5.5 The *basic* UTMGO

In order to show the practicality of this study, we present the *basic* UTMGO, a tool that uses genetic similarity algorithm to find a group of semantically similar GO terms. A screenshot of the *basic* UTMGO is shown in Figure 5.8 wherein “DNA binding” (GO:0003677) is used as an example of the query GO term. A brief explanation of the processing behind the *basic* UTMGO is as follows:

(i) Public GO data in MySQL and RDF/XML formats are downloaded from the GO website.

(ii) The single humongous GO RDF/XML file is split into smaller files (refer to Chapter 4).

(iii) Corresponding gene products together with protein sequences and evidence associations with the GO terms, either based on IEA or non-IEA evidence code, from the GO MySQL database are inserted into the fragmented GO RDF/XML files.

(iv) The *basic* UTMGO requires the user to enter a GO term and the number of matched GO terms to be returned N_t.

(v) The semantic similarity searching is performed by the genetic similarity algorithm. The results return N_t GO terms with higher term similarity score to the query GO term. The information displayed to the user is the GO terms accession number, followed by a short description of the GO term, its category (either cellular component (C), molecular function (F), or biological process (P)), and the term similarity score.

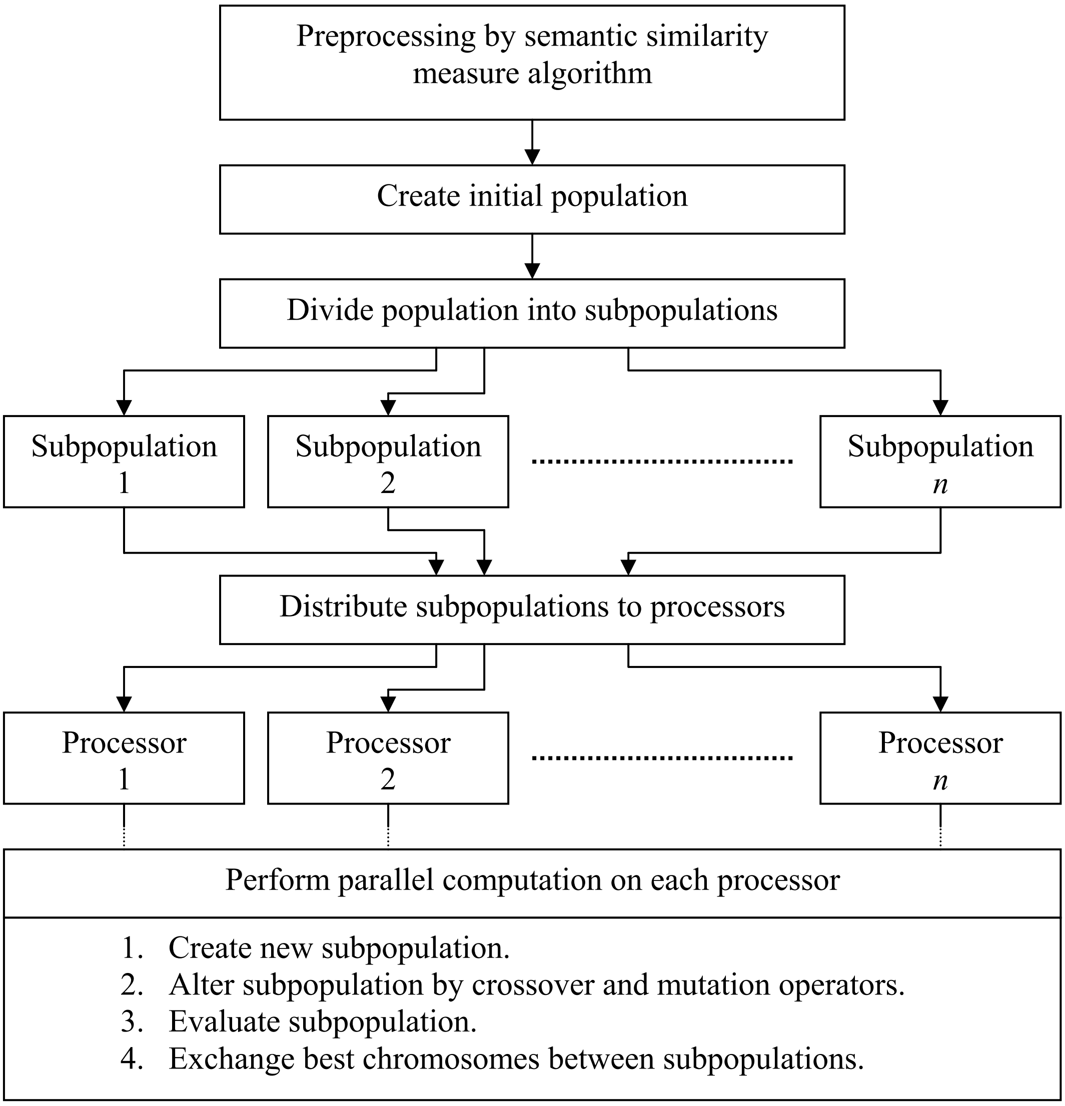
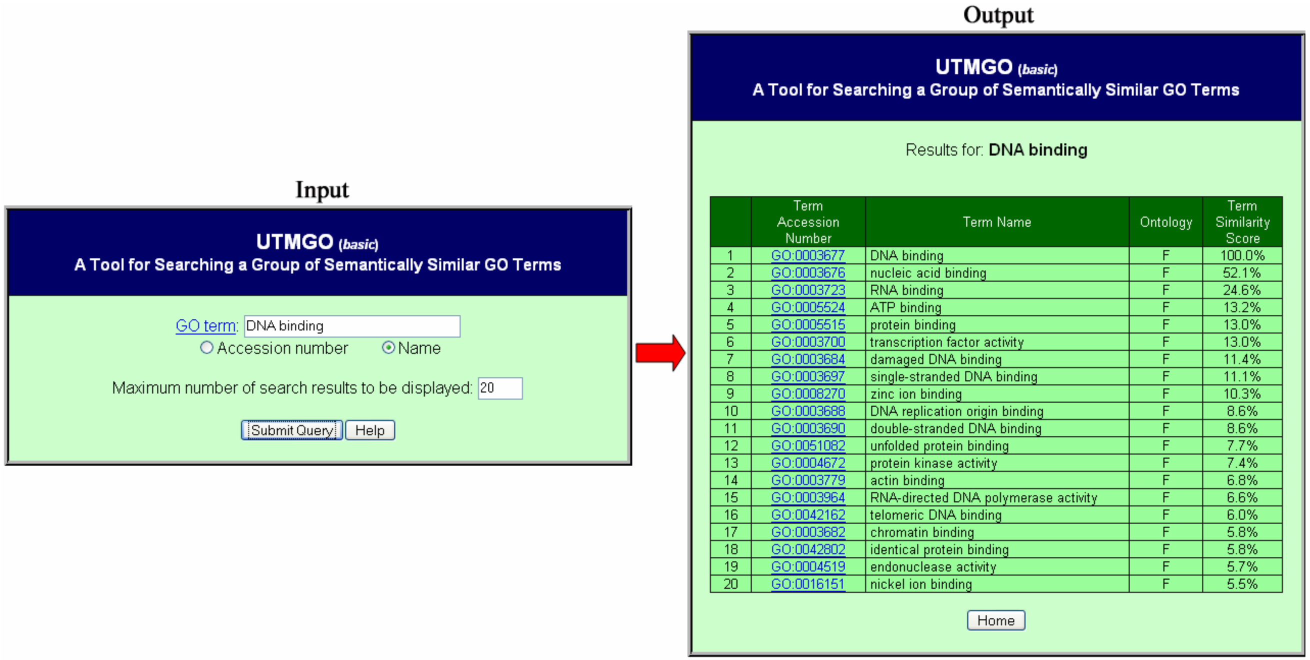

Preprocessing by semantic similarity measure algorithm
Create initial population
Divide population into subpopulations
Subpopulation 1
Subpopulation 2
Subpopulation n
Distribute subpopulations to processors
Processor 1
Processor 2
Processor n
Perform parallel computation on each processor
1. Create new subpopulation.
2. Alter subpopulation by crossover and mutation operators.
3. Evaluate subpopulation.
4. Exchange best chromosomes between subpopulations.

Figure 5.7: The parallelization flow of the genetic similarity algorithm.



UTMGO (*basic*)
A Tool for Searching a Group of Semantically Similar GO Terms

GO term: DNA binding
○ Accession number ● Name

Maximum number of search results to be displayed: 20

UTMGO (*basic*)
A Tool for Searching a Group of Semantically Similar GO Terms

Results for: **DNA binding**

	Term Accession Number	Term Name	Ontology	Term Similarity Score
1	GO:0003677	DNA binding	F	100.0%
2	GO:0003676	nucleic acid binding	F	52.1%
3	GO:0003723	RNA binding	F	24.6%
4	GO:0005524	ATP binding	F	13.2%
5	GO:0005515	protein binding	F	13.0%
6	GO:0003700	transcription factor activity	F	13.0%
7	GO:0003684	damaged DNA binding	F	11.4%
8	GO:0003697	single-stranded DNA binding	F	11.1%
9	GO:0008270	zinc ion binding	F	10.3%
10	GO:0003688	DNA replication origin binding	F	8.6%
11	GO:0003690	double-stranded DNA binding	F	8.6%
12	GO:0051082	unfolded protein binding	F	7.7%
13	GO:0004672	protein kinase activity	F	7.4%
14	GO:0003779	actin binding	F	6.8%
15	GO:0003964	RNA-directed DNA polymerase activity	F	6.6%
16	GO:0042162	telomeric DNA binding	F	6.0%
17	GO:0003682	chromatin binding	F	5.8%
18	GO:0042802	identical protein binding	F	5.8%
19	GO:0004519	endonuclease activity	F	5.7%
20	GO:0016151	nickel ion binding	F	5.5%

Figure 5.8: A screenshot of the *basic* UTMGO.

5.6 Testing Preparation and Evaluation Measures

The testing is executed using a low-cost PC cluster that consists of 25 Pentium IV 2.8GHz processors with 512MB memory and 100Mbps network speed. The genetic similarity algorithm is compiled using GNU GCC compiler under Fedora Core 5 operating system. The low-cost PC cluster is implemented using MPICH2 libraries [137] developed by the Argonne National Laboratory. The genetic similarity algorithm is implemented by enhancing the GAlib C++ libraries [138].

In this chapter, the GO data released in January 2007, as shown in Table 2.1, is explored in the experiments. The full GO graph that consists of 22,954 GO terms (1,977 cellular components, 12,903 biological processes, and 8,074 molecular functions) is input to the genetic similarity algorithm and it becomes the chromosome length. The parameters set for the genetic similarity algorithm are depicted in Table 5.1. A total of 250 GO terms in which 20 GO terms from cellular components, 140 GO terms from biological processes, and 90 GO terms from molecular functions were selected randomly as the query GO terms to evaluate the performance of the *basic* UTMGO and its genetic similarity algorithm.

The effectiveness of the *basic* UTMGO is validated using standard information retrieval measures: recall and precision. Recall is the ratio of the number of relevant GO terms retrieved to the total number of relevant GO terms in the GO database. Precision is the number of relevant GO terms retrieved to the total of irrelevant and relevant GO terms retrieved. These are formulated as:

$$Recall = \frac{a}{(a+b)} \times 100 \text{ and} \tag{5.15}$$

$$Precision = \frac{a}{(a+c)} \times 100\,, \tag{5.16}$$

where *a* is the number of relevant GO terms retrieved (i.e., the system and the expert agree with the matches), *b* is the number of relevant GO terms not retrieved (i.e., the system disagrees with the matches but the expert agrees), *c* is the number of irrelevant GO terms retrieved (i.e., the expert disagrees with the matches but the system agrees), and *d* is the number of irrelevant GO terms not retrieved (i.e., the system and the expert disagrees with the matches).

Table 5.1: Parameters of the genetic similarity algorithm.

Parameter	**Value**
Size of population	500
Number of generations	1,000
Crossover probability	0.6
Mutation probability	0.05
Length of chromosome	22,954
Replacement percentage	0.5
Type of crossover	Two-point crossover
Type of mutation	Swap mutation
Type of genetic algorithm	Steady-state genetic algorithm
Scaling	Sigma truncation scaling
Fitness function	Maximizing preferences
Isolation time	10 generations
Number of subpopulations	25
Number of emigrants	1
Type of replacement	Bad by best
Type of migration	Stepping stone
Parameter α for depth factor	0.5
Parameter β for local network density factor	0.3
Parameter χ for fitness function	1
Parameter δ for fitness function	0.05
Ideal dimension for dimension index	20

5.7 Results and Discussion

Different semantic similarity measures proposed by Lin (sim_{L}) [152], Leacock and Chodorow (sim_{LC}) [151], and Resnik (sim_{R}) [154] are used to assess the performance of our semantic similarity measure (sim_{O}) that has been built according to the Jiang and Conrath semantic similarity measure [153]. The average results of the 250 query GO terms, as shown in Table 5.2, show that sim_{O} provides the best values of recall, precision, and maximum value of fitness function, i.e., 70.35%, 83.80%, and 1,034.02 respectively. However, the earliest number of generations to converge is obtained by sim_{LC} which converged as early as after 470 generations. Again, the best processing time (0.10 seconds) is obtained by sim_{LC}. Table 5.3 shows an example of comparison of different semantic similarity measures in which each GO term is matched with "organelle inner membrane" (GO:0019866): the term similarity score is given in percentage. GO terms such as "infected host cell surface knob" (GO:0020030), "host cell nucleus" (GO:0042025), and "membrane-bound organelle" (GO:0043227) are detected by sim_{O} whereas these are not detected by the other semantic similarity measures. Furthermore, the term similarity score for sim_{O} is higher than the other semantic similarity measures.

Table 5.2: Comparison of genetic similarity algorithm with different semantic similarity measures.

Item	$\boldsymbol{sim_{\mathrm{O}}}$	$\boldsymbol{sim_{\mathrm{L}}}$	$\boldsymbol{sim_{\mathrm{LC}}}$	$\boldsymbol{sim_{\mathrm{R}}}$
g_{best}	540	610	470	490
CPU time	0.13	0.16	0.10	0.11
$max\{f(x)\}$	1,034.02	945.58	889.08	827.10
Recall	70.35	66.43	64.71	62.50
Precision	83.80	78.19	74.92	69.93

To examine the sensitivity of parameters α and β, different combinations of parameters α and β are analyzed. Based on the average results of the 250 query GO terms, the results from Table 5.4 confirm that the combination of $\alpha = 0.5$ and $\beta = 0.3$, used in this study as shown in Table 5.1, outperform other combinations. In the meantime, in order to justify the need for executing the genetic similarity algorithm on a low-cost PC cluster, the effect of using different numbers of processors in the low-cost PC cluster is analyzed. The effects on the following factors are studied: processing time, number of generations to converge, maximum value of fitness function, recall, and precision. The average results of the 250 query GO terms, shown in Table 5.5, show that a cluster of 25 processors is the ideal solution to handle the computational problem. Five factors, particularly the processing time, were highly affected if more processors were removed. Otherwise, additional processors only slightly affected those factors.

Table 5.3: An example of comparison of different semantic similarity measures.

GO term accession number	GO term name	sim_{O}	sim_{L}	sim_{LC}	sim_{R}
GO:0005652	nuclear lamina	5.7	4.0	3.4	2.3
GO:0005787	signal peptidase complex	5.8	4.1	3.6	2.3
GO:0009528	plastid inner membrane	16.1	7.7	6.5	4.3
GO:0009529	plastid intermembrane space	1.6	1.1	0.9	0.5
GO:0009536	plastid	9.1	5.8	2.7	0.5
GO:0016023	cytoplasmic membrane-bound vesicle	6.5	4.3	2.1	0.5
GO:0017090	meprin A complex	5.8	3.0	2.6	2.3
GO:0019815	B cell receptor complex	6.5	3.3	3.0	2.9
GO:0019866	organelle inner membrane	100.0	100.0	100.0	89.0
GO:0019867	outer membrane	7.8	7.5	6.0	4.9
GO:0020006	parasitophorous vacuolar membrane network	4.0	2.3	1.9	1.7
GO:0020007	apical complex	2.2	2.0	1.7	1.1
GO:0020016	flagellar pocket	2.2	1.9	1.0	0.5
GO:0020030	infected host cell surface knob	2.8	0.0	0.0	0.0
GO:0020031	polar ring of apical complex	1.8	1.4	1.3	1.1
GO:0030134	ER to Golgi transport vesicle	3.9	2.8	1.4	0.5
GO:0030386	ferredoxin:thioredoxin reductase complex	1.6	1.1	0.9	0.5
GO:0031090	organelle membrane	12.4	10.1	8.6	3.0
GO:0031300	intrinsic to organelle membrane	8.8	5.5	4.8	3.0
GO:0031471	ethanolamine degradation polyhedral organelle	1.6	1.2	0.9	0.5
GO:0042025	host cell nucleus	5.1	0.0	0.0	0.0
GO:0042601	endospore-forming forespore	1.8	1.6	1.5	1.1
GO:0042995	cell projection	6.4	5.0	4.2	1.1
GO:0043227	membrane-bound organelle	12.7	0.0	0.0	0.0
GO:0043231	intracellular membrane-bound organelle	13.8	8.2	4.0	0.5

Table 5.4: The effects of different combinations of parameters α and β on the values of the recall (r), precision (p), and maximum value of fitness function (f).

Parameter α for depth factor	**Parameter β for local network density factor**				
	β = 1.0	**β = 0.7**	**β = 0.5**	**β = 0.3**	**β = 0.0**
$\alpha = 2.0$	$r = 67.97$ $p = 81.42$ $f = 796.70$	$r = 68.19$ $p = 81.64$ $f = 818.35$	$r = 68.63$ $p = 82.08$ $f = 862.61$	$r = 68.85$ $p = 82.30$ $f = 884.79$	$r = 67.37$ $p = 80.82$ $f = 736.53$
$\alpha = 1.5$	$r = 68.88$ $p = 82.33$ $f = 887.32$	$r = 69.79$ $p = 83.24$ $f = 978.32$	$r = 69.80$ $p = 83.25$ $f = 979.31$	$r = 70.16$ $p = 83.61$ $f = 1{,}015.50$	$r = 68.03$ $p = 81.48$ $f = 802.84$
$\alpha = 1.0$	$r = 69.14$ $p = 82.58$ $f = 912.77$	$r = 69.81$ $p = 83.26$ $f = 980.03$	$r = 69.94$ $p = 83.39$ $f = 993.91$	$r = 70.27$ $p = 83.72$ $f = 1{,}026.49$	$r = 68.40$ $p = 81.85$ $f = 839.83$
$\alpha = 0.5$	$r = 69.35$ $p = 82.80$ $f = 934.13$	$r = 69.93$ $p = 83.38$ $f = 992.78$	$r = 70.04$ $p = 83.49$ $f = 1{,}003.12$	$r = 70.35$ $p = 83.80$ $f = 1{,}034.02$	$r = 68.66$ $p = 82.11$ $f = 865.45$
$\alpha = 0.0$	$r = 67.02$ $p = 80.47$ $f = 701.46$	$r = 67.65$ $p = 81.10$ $f = 764.67$	$r = 67.76$ $p = 81.21$ $f = 775.71$	$r = 68.68$ $p = 82.13$ $f = 867.52$	$r = 66.94$ $p = 80.39$ $f = 693.96$

Table 5.5: The effects of different number of processors used on the performance of the genetic similarity algorithm.

Item	**Number of processors**					
	5	**10**	**15**	**20**	**25**	**30**
g_{best}	690	660	610	580	540	540
CPU time	2,372.37	1,053.85	374.96	68.07	0.13	0.12
$max\{f(x)\}$	717.05	781.18	836.51	898.62	1,034.02	1,034.09
Recall	67.26	67.72	68.40	69.53	70.35	70.39
Precision	80.22	81.08	81.69	82.55	83.80	83.86

To prove the capability of the *basic* UTMGO that uses the genetic similarity algorithm as its intelligent engine, its output is compared with other GO browsers. The comparison is done with keyword-based GO browsers such as AmiGO (developed by the GO Consortium), GenNav (developed by the US NLM), QuickGO (developed by the EBI), and TAIR Keyword Browser (developed by the TAIR), and also with semantic similarity-based GO browsers such as DynGO [105] and FuSSiMeG [166]. The performance is shown in Table 5.6 for the average results of the 250 query GO terms. Hence, the *basic* UTMGO showed better recall and precision, but the AmiGO gives the best processing time (0.11 seconds) which is

0.02 seconds faster than the *basic* UTMGO. Nevertheless, the AmiGO provides the lowest recall (54.96%) and its precision is 21.96% lower than the *basic* UTMGO. The results also show that the semantic similarity-based GO browsers outmatched the keyword-based GO browsers in terms of recall and precision. An example of a query that is based on "DNA binding" (GO:0003677) as the input GO term is shown in Table 5.7 (for the first 10 returned GO terms). Our semantic similarity measure is used to calculate the term similarity score: the value is given in percentage. The results from Table 5.7 show that all GO terms with term similarity score equal or higher than "DNA replication origin binding" (GO:0003688, 8.6%) are returned and descendingly sorted by the *basic* UTMGO. The results generated by the semantic similarity-based GO browsers are attractive because they return GO terms that do not comprise keywords associated with the query GO term. For example, "transcription factor activity" (GO:0003700), "endonuclease activity" (GO:0004519), and "protein kinase activity" (GO:0004672) are returned by the *basic* UTMGO, DynGO, and FuSSiMeG respectively.

Table 5.6: Comparison of performance between *basic* UTMGO and other keyword-based and semantic similarity-based GO browsers.

GO Browser	Recall	Precision	CPU time
basic UTMGO	70.35	83.80	0.13
DynGO	67.88	75.04	0.19
FuSSiMeG	70.26	79.41	0.23
AmiGO	54.96	61.84	0.11
GenNav	56.78	60.92	0.16
QuickGO	57.39	60.43	0.22
TAIR Keyword Browser	56.08	61.12	0.15

Table 5.7: An example of comparison between *basic* UTMGO and other keyword-based and semantic similarity-based GO browsers.

Rank	***basic* UTMGO**		**DynGO**		**FuSSiMeG**		**AmiGO**	
	GO term accession number	**$\mathrm{SSM_O}$**	**GO term accession number**	**$\mathrm{SSM_O}$**	**GO term accession number**	**$\mathrm{SSM_O}$**	**GO term accession number**	**$\mathrm{SSM_O}$**
1	GO:0003677	100.0	GO:0003677	100.0	GO:0003677	100.0	GO:0003680	5.4
2	GO:0003676	52.1	GO:0005524	13.2	GO:0003676	52.1	GO:0050692	1.9
3	GO:0003723	24.6	GO:0005515	13.0	GO:0004672	7.4	GO:0003677	100.0
4	GO:0005524	13.2	GO:0003688	8.6	GO:0003697	11.1	GO:0051880	2.5
5	GO:0005515	13.0	GO:0008534	3.0	GO:0008270	10.3	GO:0003681	4.2
6	GO:0003700	13.0	GO:0003691	3.7	GO:0005515	13.0	GO:0019237	4.6
7	GO:0003684	11.4	GO:0031490	3.6	GO:0019237	4.6	GO:0031490	3.6
8	GO:0003697	11.1	GO:0003681	4.2	GO:0003908	3.3	GO:0003684	11.4
9	GO:0008270	10.3	GO:0050692	1.9	GO:0042162	6.0	GO:0003690	8.6
10	GO:0003688	8.6	GO:0004519	5.7	GO:0003682	5.8	GO:0003691	3.7
Rank	***basic* UTMGO**		**GenNav**		**QuickGO**		**TAIR Keyword Browser**	
	GO term accession number	**$\mathrm{SSM_O}$**	**GO term accession number**	**$\mathrm{SSM_O}$**	**GO term accession number**	**$\mathrm{SSM_O}$**	**GO term accession number**	**$\mathrm{SSM_O}$**
1	GO:0003677	100.0	GO:0003680	5.4	GO:0003677	100.0	GO:0003680	5.4
2	GO:0003676	52.1	GO:0003681	4.2	GO:0006260	3.4	GO:0003677	100.0
3	GO:0003723	24.6	GO:0019237	4.6	GO:0051880	2.5	GO:0003681	4.2
4	GO:0005524	13.2	GO:0031490	3.6	GO:0003899	5.3	GO:0003684	11.4
5	GO:0005515	13.0	GO:0003684	11.4	GO:0003887	5.0	GO:0003690	8.6
6	GO:0003700	13.0	GO:0050692	1.9	GO:0050692	1.9	GO:0003691	3.7
7	GO:0003684	11.4	GO:0003677	100.0	GO:0003908	3.3	GO:0003692	4.6
8	GO:0003697	11.1	GO:0003690	8.6	GO:0003964	6.6	GO:0003695	3.1
9	GO:0008270	10.3	GO:0003691	3.7	GO:0008534	3.0	GO:0000182	4.9
10	GO:0003688	8.6	GO:0051880	2.5	GO:0003886	3.0	GO:0003696	5.2

The search results have indicated that the *basic* UTMGO is able to find a group of semantically similar GO terms with higher recall and precision and reasonable processing time as compared to other semantic similarity-based GO browsers such as DynGO [105] and FuSSiMeG [166]. Furthermore, as compared to other keyword-based GO browsers such as AmiGO (http://godatabase.org/), GenNav (http://mor.nlm.nih.gov/perl/gennav.pl), QuickGO (http://www.ebi.ac.uk/ego/), and TAIR Keyword Browser (http://www.arabidopsis.org/servlets/Search?action= new_search&type=keyword), the *basic* UTMGO is capable of finding GO terms that do not contain the keyword specified by the user and with higher recall and precision.

5.8 Summary

A genetic similarity algorithm is introduced in this chapter to find a group of semantically similar GO terms. The genetic similarity algorithm combines semantic similarity measure algorithm with parallel genetic algorithm. The semantic similarity measure algorithm is used to compute the similitude strength between the GO terms. Then, the parallel genetic algorithm is employed to perform batch retrieval and to accelerate the search in large search space of the GO graph. The genetic similarity algorithm is implemented in the GO browser named *basic* UTMGO to overcome the weaknesses of the existing GO browsers which use a conventional approach based on keyword matching. The computational results and comparison with other related GO browsers are presented to show the effectiveness of the genetic similarity algorithm and the *basic* UTMGO.

CHAPTER 6

extended UTMGO: A GENE ONTOLOGY-BASED PROTEIN SEQUENCE ANNOTATION TOOL

6.1 Introduction

As outlined by the EBI, annotation of an anonymous protein sequence should be inferred from annotations of the nucleotide sequences, analogies with already understood proteins, plus references to patterns and motifs as characteristics of particular protein functions. Annotation of anonymous protein sequences is important for the preservation and reuse of knowledge and for content-based queries. The traditional wet-lab methods are labor intensive and prone to human error. On the other hand, the sequence-similarity-based tools like BLAST are time intensive and require high investment in computing facilities such as cluster server or grid computing if being used locally. Furthermore, for remote users, these tools are subject to internet stability and speed to access the tools and to get the results online. Therefore, a simple and practical method that is capable of producing better results and requires a reasonable amount of running time with low computing cost specifically for offline usage is needed.

In the last few years, the GO terms have been widely used to annotate various protein sets such as in NOPdb [167], a database of nucleolar proteome; SCOPPI [168], a database of protein domain-domain interactions; DRTF [169], a database of rice transcription factor; and MolMovDB [170], a database of macromolecular motions. In addition, GO terms have been successfully implemented in large-scale protein annotation projects involving SWISS-PROT, TrEMBL, and InterPro databases [102]. The GO is a project to provide a rich and comprehensive unified vocabulary to describe genes and their functions and products. Currently the GO comprises more than 22 thousand terms and is updated every 30 minutes, which tally with the growth activities in the bioinformatics field. The advantages of using the GO are as follows: the GO data is dynamic and constantly evolves according to the advances in current state of biological knowledge; the GO data is publicly available and can be downloaded at any time from the WWW in MySQL, RDF/XML, OBO/XML, and OWL formats that can be understandable and processable by human and machine alike; the common GO terms shared by gene and protein sequences in multiple organisms in different databases can facilitate uniform queries across them; and the association of GO terms with nearly 2.5 million gene products supported by the evidence and citation can affirm its reliability for future evaluation and use. The link between the GO terms and gene products is provided by the GOA. In the GOA project, electronic mappings and manual curation are used to assign the GO terms to all proteomes existing in the UniProt, Ensembl, and other organism databases. It covers 2.3 million protein sequences from 0.26 million species.

However, application of the GO terms to annotate anonymous protein sequences is not easy, especially for species not yet inserted in public biological databases. Furthermore, for bioscientists with little computational knowledge or limited facilities it is a hard task to annotate those anonymous protein sequences. The difficulties arise because generally the existing GO-based tools are (1) dependent on BLAST which is computationally intensive and requires high-cost and high-specification hardware since sequence alignment is performed to all protein sequences but not only to protein sequences that indicate higher similarity, (2) dependent on RDBMS which require the user to setup the RDBMS software and to import the data or sources into the RDBMS format, and (3) partially based on the GO data which requires the user to download the GOA data or protein sequence data sets from several sources.

Therefore, in this chapter, a new way of applying the GO terms to annotate anonymous protein sequences is introduced. The GO-based method consists of three main components. In the first component, the single monolithic GO RDF/XML file is split into smaller files. It is carried out to avoid dependency on RDBMS format, to provide all-in-one source by adding protein sequences and IEA evidence associations into the files since they are not included in the original GO RD/XML file, and to make the GO data easily accessible and processable. In the second component, the main focus of this chapter, semantic similarity search is performed over the smaller GO RDF/XML files. The target is to find a group of semantically similar GO terms with higher term similarity score to a GO term which is foreseen to have higher relationship with the query protein sequence. Lastly, the results obtained from the second component are verified by computing sequence alignment score between the query protein sequence and all protein sequences attached to those GO terms. With this GO-based method, sequence alignment is carried out only to protein sequences with higher outguessed similarity. Hence, demand for high computational facilities and execution time can be reduced. A GO-based tool named *extended* UTMGO is developed to demonstrate the GO-based method. The *extended* UTMGO employs a GO browser named *basic* UTMGO (refer to Chapter 5) for implementing the second component. The JAligner engine (http://jaligner.sourceforge.net/) that uses the Smith-Waterman algorithm has been integrated and modified to perform the sequence alignment and to comply with the *extended* UTMGO. The flow of the *extended* UTMGO can be summarized as shown in Figure 6.1.

The rest of this chapter is organized as follows. Section 6.2 presents existing tools for annotating anonymous protein sequences. Section 6.3 gives the step-by-step description of the *extended* UTMGO. Section 6.4 explains the testing environment and evaluation measures used to validate the *extended* UTMGO. Section 6.5 presents experimental results and discussion and is followed by summary in Section 6.6.

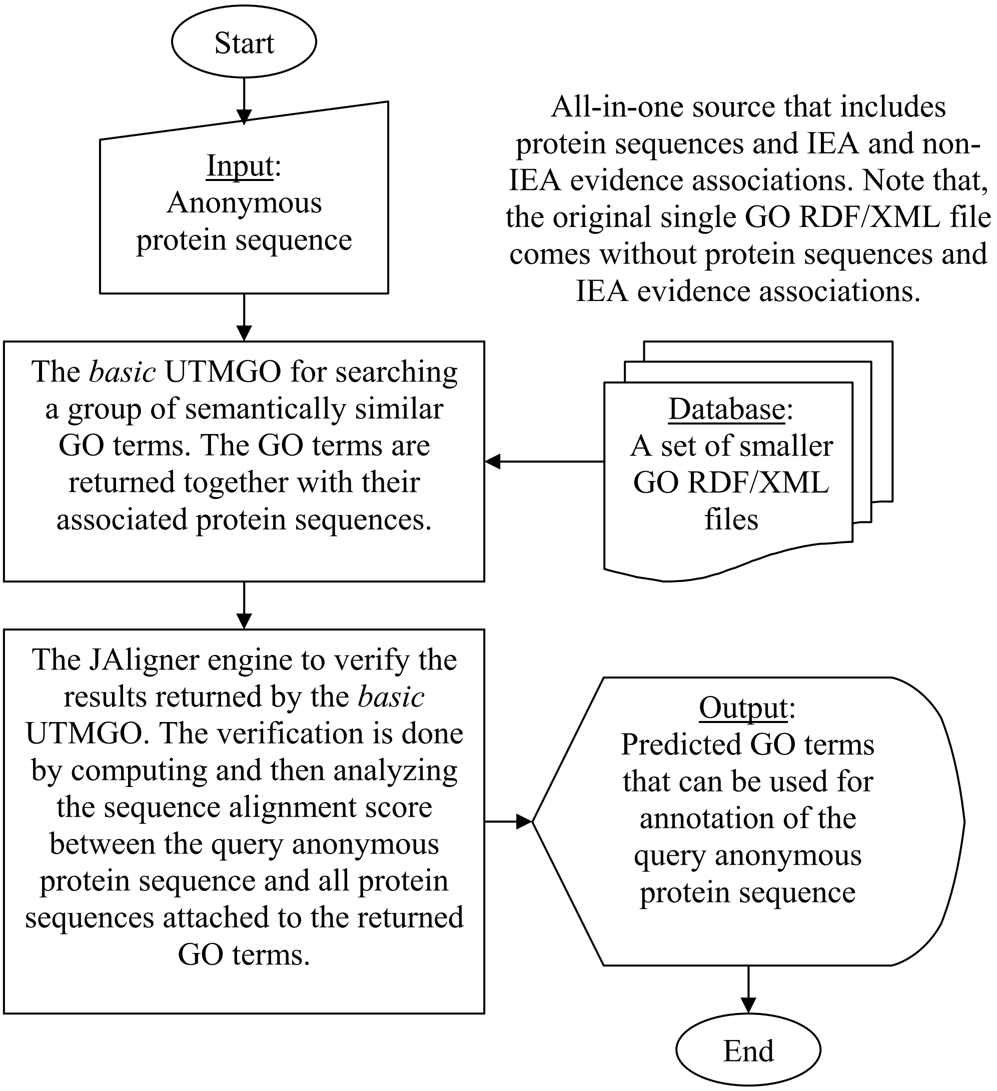

Start
Input:
Anonymous protein sequence
All-in-one source that includes protein sequences and IEA and non-IEA evidence associations. Note that, the original single GO RDF/XML file comes without protein sequences and IEA evidence associations.
The *basic* UTMGO for searching a group of semantically similar GO terms. The GO terms are returned together with their associated protein sequences.
Database:
A set of smaller GO RDF/XML files
The JAligner engine to verify the results returned by the *basic* UTMGO. The verification is done by computing and then analyzing the sequence alignment score between the query anonymous protein sequence and all protein sequences attached to the returned GO terms.
Output:
Predicted GO terms that can be used for annotation of the query anonymous protein sequence
End

Figure 6.1: The flowchart of the *extended* UTMGO.

6.2 Related Work

Several tools have been developed in recent years to annotate anonymous protein sequences in accordance with the GO terms. The generally used tools include:

(i) GoFigure [171] is a tool that accepts an unknown DNA or protein sequence as an input and then uses BLAST to predict the GO terms by identifying homologous sequences in the GO annotated databases.

(ii) GOtcha [172] is a tool that provides a prediction of a set of GO terms for a given query sequence (DNA or protein). BLAST is used to get the initial score of each GO term and the scores are calibrated against term-specific probability (P-score) to give higher accuracy.

(iii) GOPET [173] is an automated annotation tool for assigning the GO terms to cDNA or protein query sequences. It uses BLAST to perform homology searches against GO-mapped protein databases, and support vector machines for the prediction and the assignment of confidence values.

(iv) JAFA [174] is a meta-server that uses several function prediction programs such as GoFigure, GOtcha, GOblet [175], Phydbac [176], and InterProScan [177]. It accepts a protein sequence and returns the predicted GO terms with prediction score that is based on the ratio of agreeing servers.

However, as mentioned earlier in the previous section, for offline usage, these tools are difficult to configure and use, especially by bioscientists. The tools also require an expensive high performance computing environment. Whereas, for online usage, they depend on internet stability and speed.

6.3 The *extended* UTMGO

The operation of the *extended* UTMGO is divided into two cases: with (Option 1) or without (Option 2) a GO term entered by the user as shown in Figure 6.2 and 6.3 respectively. An example of the query anonymous protein sequence used to demonstrate the *extended* UTMGO is as follows:

```
MVRGKTQMKRIENPTSRQVTFSKRRNGLLKKAFELSVLCDAEVALIV
FSPRGKLYEFASASTQKTIERYRTYTKENIGNKTVQQDIEQVKADADG
LAKKLEALETYKRKLLGEKLDECSIEELHSLEVKLERSLISIRGRKTKL
LEEQVAKLREKEMKLRKDNEELREKCKNQPPLSAPLTVRAEDENPDR
NINTTNDNMDVETELFIGLPGRSRSSGGAAEDSQAMPHS
```

This protein sequence belongs to “MADS50” (*MADS-box transcription factor 50*, GR:Q9XJ60), an *Oryza sativa* species obtained from the Gramene database. The *extended* UTMGO, as shown in Figure 6.1, consists of the following steps:

(i) Get an anonymous protein sequence, the number of GO terms to be returned N_t, a term similarity threshold, the number of protein sequences associated with each GO term to be returned N_s, and optionally a GO term from the user.

(ii) If the GO term is null, then go to step (iii), otherwise, go to step (vi).

(iii) Get the input from the user for appropriate species, matrix type either Blocks Substitution Matrix (BLOSUM) or Point Accepted Mutations (PAM), and open and extend gap penalties to restrict the search.

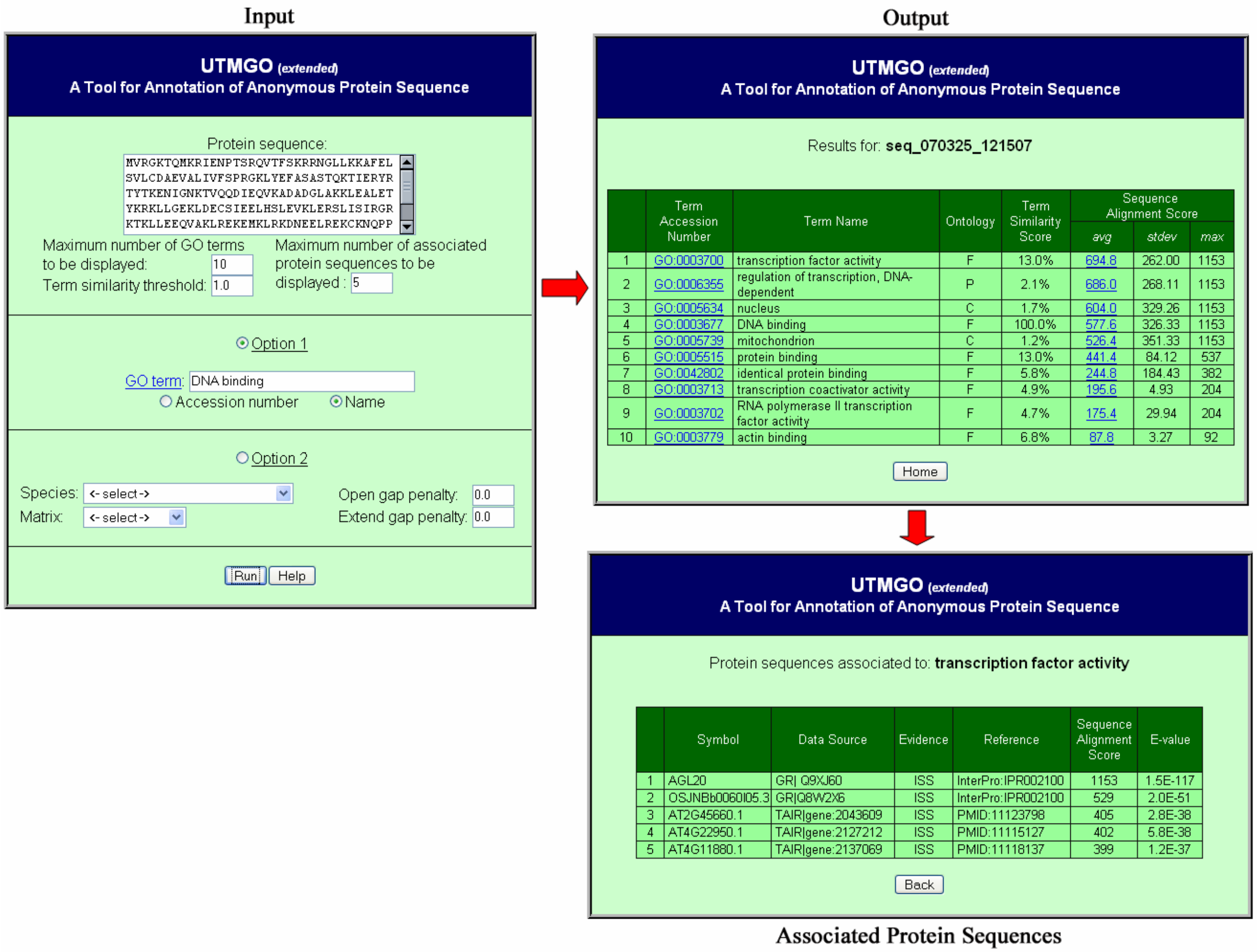
(iv) Perform the sequence similarity search for the query anonymous protein sequence from step (i). The search is carried out for protein sequences from the fragmented GO RDF/XML files that are related to the molecular function terms. The output is a protein sequence with the highest sequence alignment score. The JAligner engine is used to perform the sequence similarity search.

(v) Select a molecular function term with the highest association with the protein sequence obtained in step (iv) for the next step. If there is more than one term, the user has to make the selection.

(vi) Submit the GO term either from step (i) or step (v) to the *basic* UTMGO and then perform semantic similarity search.

(vii) Return N_t GO terms with the term similarity score higher than the term similarity threshold, as set in step (i), together with protein sequences associated with them.

(viii) Calculate sequence alignment score between the query anonymous protein sequence and all protein sequences for each GO term obtained from the previous step using the JAligner engine. The information displayed to the user is the same as in the *basic* UTMGO: the GO term accession number and its short description, category, and term similarity score. Additional information given is arithmetic mean, standard deviation, and the largest value of the sequence alignment score of N_s number of protein sequences with higher sequence alignment score that is attached to the GO term.



Input

UTMGO *(extended)*
A Tool for Annotation of Anonymous Protein Sequence

Protein sequence:

```
MVRGKTQMKRIENPTSRQVTFSKRRNGLLKKAFEL
SVLCDAEVALIVFSPRGKLYEFASASTQKTIERYR
TYTKENIGNKTVQQDIEQVKADADGLAKKLEALET
YKRKLLGEKLDECSIEELHSLEVKLERSLISIRGR
KTKLLEEQVAKLREKEMKLRKDNEELREKCKNQPP
```

Maximum number of GO terms to be displayed: 10

Term similarity threshold: 1.0

Maximum number of associated protein sequences to be displayed : 5

● Option 1

GO term: DNA binding

○ Accession number ● Name

○ Option 2

Species: <-select->

Matrix: <-select->

Open gap penalty: 0.0

Extend gap penalty: 0.0

Output

UTMGO *(extended)*
A Tool for Annotation of Anonymous Protein Sequence

Results for: **seq_070325_121507**

	Term Accession Number	Term Name	Ontology	Term Similarity Score	Sequence Alignment Score		
					avg	*stdev*	*max*
1	GO:0003700	transcription factor activity	F	13.0%	694.8	262.00	1153
2	GO:0006355	regulation of transcription, DNA-dependent	P	2.1%	686.0	268.11	1153
3	GO:0005634	nucleus	C	1.7%	604.0	329.26	1153
4	GO:0003677	DNA binding	F	100.0%	577.6	326.33	1153
5	GO:0005739	mitochondrion	C	1.2%	526.4	351.33	1153
6	GO:0005515	protein binding	F	13.0%	441.4	84.12	537
7	GO:0042802	identical protein binding	F	5.8%	244.8	184.43	382
8	GO:0003713	transcription coactivator activity	F	4.9%	195.6	4.93	204
9	GO:0003702	RNA polymerase II transcription factor activity	F	4.7%	175.4	29.94	204
10	GO:0003779	actin binding	F	6.8%	87.8	3.27	92

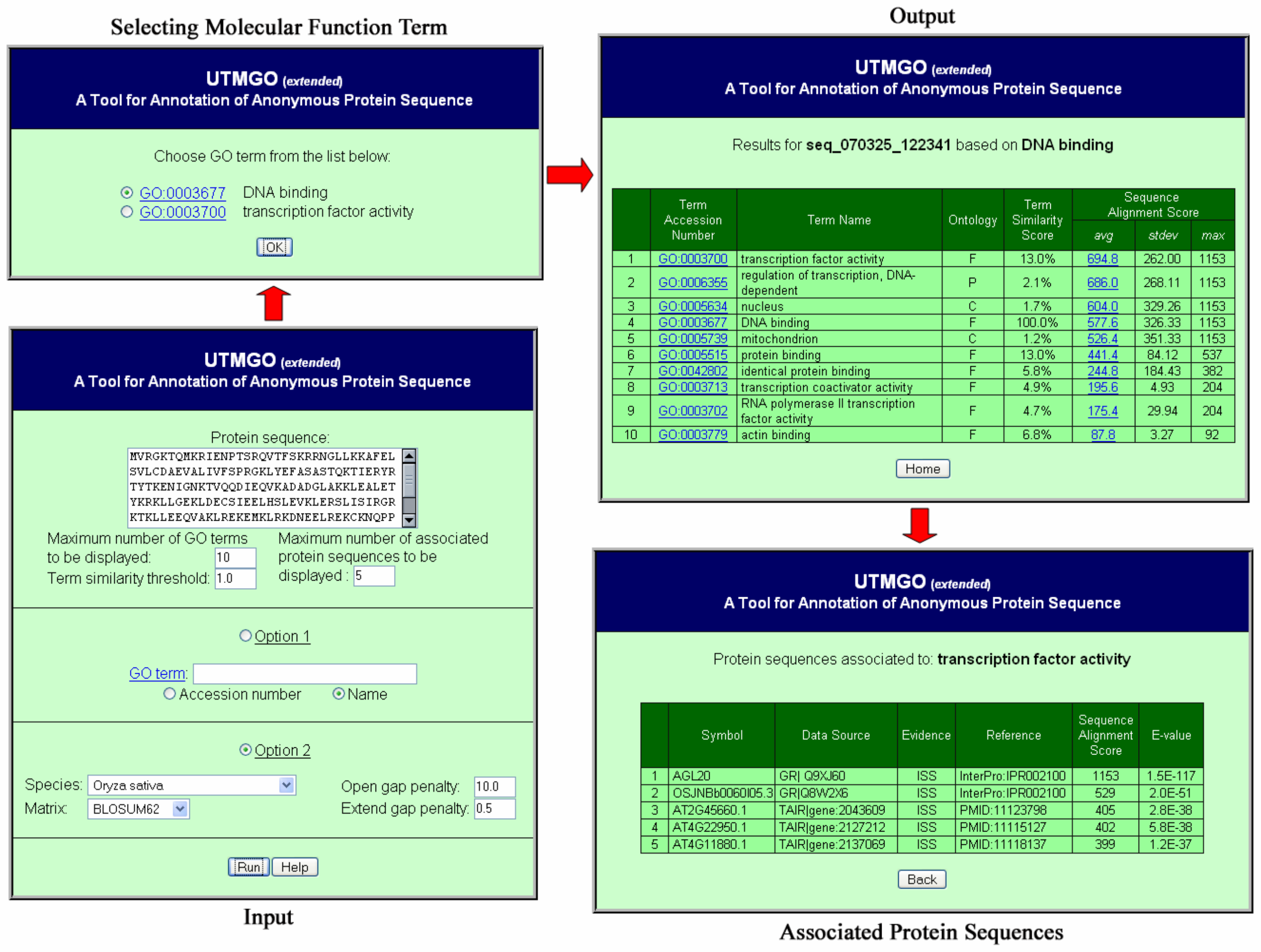
UTMGO *(extended)*
A Tool for Annotation of Anonymous Protein Sequence

Protein sequences associated to: **transcription factor activity**

	Symbol	Data Source	Evidence	Reference	Sequence Alignment Score	E-value
1	AGL20	GR| Q9XJ60	ISS	InterPro:IPR002100	1153	1.5E-117
2	OSJNBb0060I05.3	GR|Q8W2X6	ISS	InterPro:IPR002100	529	2.0E-51
3	AT2G45660.1	TAIR|gene:2043609	ISS	PMID:11123798	405	2.8E-38
4	AT4G22950.1	TAIR|gene:2127212	ISS	PMID:11115127	402	5.8E-38
5	AT4G11880.1	TAIR|gene:2137069	ISS	PMID:11118137	399	1.2E-37

Associated Protein Sequences

Figure 6.2: A screenshot of the *extended* UTMGO with a GO term entered by the user (Option 1).



Selecting Molecular Function Term

UTMGO *(extended)*
A Tool for Annotation of Anonymous Protein Sequence

Choose GO term from the list below:

● GO:0003677 DNA binding
○ GO:0003700 transcription factor activity

Input

UTMGO *(extended)*
A Tool for Annotation of Anonymous Protein Sequence

Protein sequence:

```
MVRGKTQMKRIENPTSRQVTFSKRRNGLLKKAFEL
SVLCDAEVALIVFSPRGKLYEFASASTQKTIERYR
TYTKENIGNKTVQQDIEQVKADADGLAKKLEALET
YKRKLLGEKLDECSIEELHSLEVKLERSLISIRGR
KTKLLEEQVAKLREKEMKLRKDNEELREKCKNQPP
```

Maximum number of GO terms to be displayed: 10
Term similarity threshold: 1.0
Maximum number of associated protein sequences to be displayed : 5

○ Option 1

GO term:
○ Accession number ● Name

● Option 2

Species: Oryza sativa
Matrix: BLOSUM62
Open gap penalty: 10.0
Extend gap penalty: 0.5

Output

UTMGO *(extended)*
A Tool for Annotation of Anonymous Protein Sequence

Results for **seq_070325_122341** based on **DNA binding**

	Term Accession Number	Term Name	Ontology	Term Similarity Score	Sequence Alignment Score		
					avg	*stdev*	*max*
1	GO:0003700	transcription factor activity	F	13.0%	694.8	262.00	1153
2	GO:0006355	regulation of transcription, DNA-dependent	P	2.1%	686.0	268.11	1153
3	GO:0005634	nucleus	C	1.7%	604.0	329.26	1153
4	GO:0003677	DNA binding	F	100.0%	577.6	326.33	1153
5	GO:0005739	mitochondrion	C	1.2%	526.4	351.33	1153
6	GO:0005515	protein binding	F	13.0%	441.4	84.12	537
7	GO:0042802	identical protein binding	F	5.8%	244.8	184.43	382
8	GO:0003713	transcription coactivator activity	F	4.9%	195.6	4.93	204
9	GO:0003702	RNA polymerase II transcription factor activity	F	4.7%	175.4	29.94	204
10	GO:0003779	actin binding	F	6.8%	87.8	3.27	92

UTMGO *(extended)*
A Tool for Annotation of Anonymous Protein Sequence

Protein sequences associated to: **transcription factor activity**

	Symbol	Data Source	Evidence	Reference	Sequence Alignment Score	E-value
1	AGL20	GR| Q9XJ60	ISS	InterPro:IPR002100	1153	1.5E-117
2	OSJNBb0060I05.3	GR|Q8W2X6	ISS	InterPro:IPR002100	529	2.0E-51
3	AT2G45660.1	TAIR|gene:2043609	ISS	PMID:11123798	405	2.8E-38
4	AT4G22950.1	TAIR|gene:2127212	ISS	PMID:11115127	402	5.8E-38
5	AT4G11880.1	TAIR|gene:2137069	ISS	PMID:11118137	399	1.2E-37

Associated Protein Sequences

Figure 6.3: A screenshot of the *extended* UTMGO without a GO term entered by the user (Option 2).

6.4 Testing Preparation and Evaluation Measures

The GO data released in January 2007 as shown in Table 2.1 is used to test the *extended* UTMGO. The computer used is a low-cost PC cluster, HP d530 with 25 processors. The low-cost PC cluster is implemented using MPICH2 libraries under Fedora Core 2 running on Pentium IV 2.8GHz of processor, 512MB of memory, and 100Mbps of network speed. This setup is the minimum requirement for offline usage if the user wants to install and use the *extended* UTMGO locally. However, for online usage, the *extended* UTMGO can be accessed remotely via the internet like other online bioinformatics tools. But currently these tools are not ready for online usage and will be opened for public soon.

In case of data sets, a total of 200 protein sequences from the GO annotated databases were used as input. These protein sequences were selected randomly with 50 protein sequences from Gramene, a database of *Oryza sativa*; 50 protein sequences from Ensembl, a database of *Homo sapiens*; 50 protein sequences from SGD, a database of *Saccharomyces cerevisiae*; and 50 protein sequences from TAIR, a database of *Arabidopsis thaliana*. Same as with the *basic* UTMGO, the *extended* UTMGO uses recall (refer to Equation 5.15) and precision (refer to Equation 5.16) to validate its effectiveness.

6.5 Results and Discussion

The comparison between the *extended* UTMGO and the other GO-based protein sequence annotation tools such as GoFigure [171], GOtcha [172], GOPET [173], and JAFA [174] is shown in Table 6.1. The comparison is based on the average results of the 200 query protein sequences that are selected randomly from the GO annotated databases as mentioned earlier in Section 6.4. Thus, the *extended* UTMGO provides a better precision (90.32%) and the JAFA offers a better recall

(88.80%) which is just 0.87% higher than the *extended* UTMGO. However, the JAFA provides the slowest processing time (518.22 seconds) and its precision is 3.55% lower than the *extended* UTMGO. The best processing time is 163.79 seconds that is taken by the *extended* UTMGO. An example query that is based on "MADS50" (*MADS-box transcription factor 50*, GR:Q9XJ60) as the input protein sequence is shown in Table 6.2 (for the top 10 predicted GO terms). The average and the maximum values of the sequence alignment score (*avg* and *max*) for the protein sequences associated with the predicted GO terms are used as an indicator to assess these tools, because quality of the results depends on the sequence alignment score between the query anonymous protein sequence and the protein sequences associated with the predicted GO terms. Thus, higher is better. As depicted in Table 6.2, all the GO terms with the average sequence alignment score equal or higher than "RNA polymerase II transcription factor activity" (GO:0003702, *avg* = 175.4) are returned by the *extended* UTMGO. However, even though the average sequence alignment scores for "flower development" (GO:0009908, *avg* = 113.0) and "cytoplasm" (GO:0005737, *avg* = 93.8) are higher than "actin binding" (GO:0003779, *avg* = 87.8), they are out of the *extended* UTMGO radar since their term similarity scores are 0.9% and 0.6% respectively. These term similarity scores are lower than the term similarity threshold (1.0%) set for this testing session. Moreover, as shown in Table 6.2, all GO terms with the highest value of the maximum of sequence alignment score (1,153) are returned by the *extended* UTMGO. Note that although GO terms such as "positive regulation of transcription from RNA polymerase II promoter" (GO:0045944, *max* = 153), "DNA bending activity" (GO:0008301, *max* = 153), and "regulation of transcription from RNA polymerase II promoter" (GO:0006357, *max* = 151) have the maximum of sequence alignment score higher than "actin binding" (GO: 0003779, *max* = 92), but they are not ranked as the predicted GO terms by the *extended* UTMGO. The reason is that their average sequence alignment score is lower than the value for "actin binding" (GO: 0003779).

The experimental results have shown that the *extended* UTMGO has the capability of annotating anonymous protein sequences with higher precision and recall with quicker processing time as compared to other GO-based protein sequence annotation tools such as GoFigure [171], GOtcha [172], GOPET [173], and JAFA [174]. The protein sequences associated with the predicted GO terms that are returned by the *extended* UTMGO also have higher sequence alignment score to the query anonymous protein sequence. In addition, the *extended* UTMGO does not depend on BLAST and RDBMS and is fully based on the GO data.

Table 6.1: Comparison of performance between *extended* UTMGO and other GO-based protein sequence annotation tools.

GO-based protein sequence annotation tool	**Recall**	**Precision**	**CPU time**
extended UTMGO	87.93	90.32	163.79
GoFigure	83.15	84.09	195.48
GOtcha	83.62	84.63	302.11
GOPET	86.39	85.31	270.82
JAFA	88.80	86.77	518.22

6.6 Summary

The GO terms have been actively used to annotate various protein sets. SWISS-PROT, TrEMBL, and InterPro are protein databases that are annotated according to the GO terms. However, direct implementation of the GO terms for annotation of anonymous protein sequences is not easy, especially for species not commonly represented in biological databases. Therefore, in this chapter, the structure of the *basic* UTMGO is extended to develop a GO-based protein sequence annotation tool named *extended* UTMGO. The objective of developing the *extended* UTMGO is to provide a simple and practical tool that is capable of producing better results and requires a reasonable amount of running time with low computing cost specifically for offline usage. The *extended* UTMGO uses the GO terms together with protein sequences associated with the terms to perform the annotation task. Other GO-based protein sequence annotation tools such as GoFigure, GOtcha, GOPET, and JAFA have been used to compare the performance of the *extended* UTMGO.

Table 6.2: An example of comparison between *extended* UTMGO and other GO-based protein sequence annotation tools.

Rank	***extended* UTMGO**		**GoFigure**		**GOtcha**	
	GO term accession number	**Sequence alignment score**	**GO term accession number**	**Sequence alignment score**	**GO term accession number**	**Sequence alignment score**
1	GO:0003700	*avg* = 694.8 *max* = 1,153	GO:0003700	*avg* = 694.8 *max* = 1,153	GO:0003677	*avg* = 577.6 *max* = 1,153
2	GO:0006355	*avg* = 686.0 *max* = 1,153	GO:0003677	*avg* = 577.6 *max* = 1,153	GO:0030528	*avg* = 0.0 *max* = 0
3	GO:0005634	*avg* = 604.0 *max* = 1,153	GO:0007275	*avg* = 0.0 *max* = 0	GO:0003700	*avg* = 694.8 *max* = 1,153
4	GO:0003677	*avg* = 577.6 *max* = 1,153	GO:0009908	*avg* = 113.0 *max* = 565	GO:0006139	*avg* = 0.0 *max* = 0
5	GO:0005739	*avg* = 526.4 *max* = 1,153	GO:0006350	*avg* = 0.0 *max* = 0	GO:0006350	*avg* = 0.0 *max* = 0
6	GO:0005515	*avg* = 441.4 *max* = 537	GO:0006355	*avg* = 686.0 *max* = 1,153	GO:0006355	*avg* = 686.0 *max* = 1,153
7	GO:0042802	*avg* = 244.8 *max* = 382	GO:0005634	*avg* = 604.0 *max* = 1,153	GO:0005622	*avg* = 38.6 *max* = 101
8	GO:0003713	*avg* = 195.6 *max* = 204	-	-	GO:0008233	*avg* = 29.6 *max* = 148
9	GO:0003702	*avg* = 175.4 *max* = 204	-	-	GO:0005215	*avg* = 17.0 *max* = 85
10	GO:0003779	*avg* = 87.8 *max* = 92	-	-	GO:0005737	*avg* = 93.8 *max* = 134

Rank	***extended* UTMGO**		**GOPET**		**JAFA**	
	GO term accession number	**Sequence alignment score**	**GO term accession number**	**Sequence alignment score**	**GO term accession number**	**Sequence alignment score**
1	GO:0003700	*avg* = 694.8 *max* = 1,153	GO:0006355	*avg* = 686.0 *max* = 1,153	GO:0045944	*avg* = 86.4 *max* = 153
2	GO:0006355	*avg* = 686.0 *max* = 1,153	GO:0003677	*avg* = 577.6 *max* = 1,153	GO:0006657	*avg* = 0.0 *max* = 0
3	GO:0005634	*avg* = 604.0 *max* = 1,153	GO:0003700	*avg* = 694.8 *max* = 1,153	GO:0004402	*avg* = 0.0 *max* = 0
4	GO:0003677	*avg* = 577.6 *max* = 1,153	GO:0006139	*avg* = 0.0 *max* = 0	GO:0008362	*avg* = 0.0 *max* = 0
5	GO:0005739	*avg* = 526.4 *max* = 1,153	GO:0006350	*avg* = 0.0 *max* = 0	GO:0007144	*avg* = 0.0 *max* = 0
6	GO:0005515	*avg* = 441.4 *max* = 537	GO:0045944	*avg* = 86.4 *max* = 153	GO:0007129	*avg* = 36.2 *max* = 92
7	GO:0042802	*avg* = 244.8 *max* = 382	GO:0006357	*avg* = 85.2 *max* = 151	GO:0007020	*avg* = 19.0 *max* = 95
8	GO:0003713	*avg* = 195.6 *max* = 204	GO:0003936	*avg* = 0.0 *max* = 0	GO:0007004	*avg* = 0.0 *max* = 0
9	GO:0003702	*avg* = 175.4 *max* = 204	GO:0008301	*avg* = 57.4 *max* = 153	GO:0007015	*avg* = 20.2 *max* = 101
10	GO:0003779	*avg* = 87.8 *max* = 92	-	-	GO:0006430	*avg* = 16.6 *max* = 83

CHAPTER 7

CONCLUSION

7.1 Concluding Remarks

Protein sequence annotation is pivotal for the understanding of its function. Accuracy of manual annotation provided by curators is still questionable by having lesser evidence strength and yet a hard task and time consuming. A number of computational methods including tools have been developed to tackle this challenging task. However, particularly for offline usage, these tools are difficult to configure and use, especially by bioscientists. The tools also require an expensive high performance computing environment, require the user to setup the RDBMS software and to import the data or sources into the RDBMS format every time the data is updated, require the user to download relevant data from multiple sources, and depend on time intensive and blind sequence similarity search like BLAST. Whereas, for online usage, they depend on internet stability and speed. Therefore, the goal of this study is to introduce a new computational method for assigning highly correlated GO terms of annotated protein sequences to partially annotated or newly discovered protein sequences. This computational method is fully based on GO data and annotations. Two problems were identified to achieve this GO-based method. The first problem relates to splitting the monolithic GO RDF/XML file into a set of smaller files that can be easy to assess and process. Thus, these files can be enriched with protein sequences and IEA evidence associations. This automatic clustering problem has been solved by the genetic split-merge algorithm. The second problem involves searching for a group of semantically similar GO terms that match to the query GO term. The genetic similarity algorithm has been proposed to resolve this semantic similarity searching problem. The GO-based protein sequence annotation tool namely *extended* UTMGO has been introduced to demonstrate the capabilities of the proposed GO-based method. Furthermore, its basic version which is a GO browser that is based on semantic similarity search has also been introduced.

As mentioned earlier in Chapter 3, the methodology of this study is structured in three phases. In the first phase, as discussed in Chapter 4, the monolithic GO RDF/XML file is split into smaller files in order to reduce difficulties in maintaining, publishing, validating, and processing the file. To split the GO RDF/XML file, the GO terms have been grouped into a number k of clusters. Thence, this study has shown that clustering the GO terms can be modeled as the GPP. The GPP has been solved by the genetic split-merge algorithm that combines the parallel genetic algorithm and the split-and-merge algorithm. The parallel genetic algorithm has been used to find the best combination of node-cluster. On the other hand, the split-and-merge algorithm has been applied to identify the best number k of clusters k_{best}. During the clustering process, the genetic split-merge algorithm has employed cohesion-and-coupling metric as a criterion to measure the goodness of the generated clusters. The dependency index γ has been introduced to avoid the genetic split-merge algorithm from producing problematic clusters with either undersized or oversized number of elements. Unlike any other clustering algorithm, the proposed algorithm with the split-and-merge strategy can automatically find the best number k of clusters k_{best}. Compared to other automatic clustering algorithms, the genetic split-merge algorithm is capable of producing balanced clusters. The experimental results have shown that the genetic split-merge algorithm requires reasonable amount of execution time and the generated clusters have better DBI and F-measure values compared to the existing algorithms. Furthermore, the users are allowed to set the minimum number k of clusters k_{min} they wish to maintain.

In the second phase, as discussed in Chapter 5, the *basic* UTMGO is based on the genetic similarity algorithm. It is a combination of genetic and semantic similarity search, and has been presented as an alternative way of searching the GO terms. The search is done by determining a group of semantically similar GO terms that are related to the query GO term. The semantic similarity search is not based on keyword matching but is based on the degree of relationships between the GO terms. A gene product that is associated with one or more GO terms is used as a foundation to compute the amount of information the GO terms share in common that gives the degree of relationships. In the meantime, the genetic search plays the main role in finding a set of GO terms from the large GO graph. The search results have indicated that the *basic* UTMGO is able to find a group of semantically similar GO terms with higher recall and precision and reasonable processing time as compared to other existing GO browsers.

Lastly, in the third phase, as discussed in Chapter 6, the usefulness of the *basic* UTMGO has been shown by its extended version. The *extended* UTMGO has the capability of annotating anonymous protein sequences with higher precision and recall with quicker processing time. The protein sequences associated with the predicted GO terms that are returned by the *extended* UTMGO also have higher sequence alignment score to the query anonymous protein sequence. In addition, the *extended* UTMGO does not depend on BLAST and RDBMS and is fully based on the GO data.

7.2 Contributions

As described earlier in the previous section, the contributions of this study can be summarized as follows:

(i) In Chapter 4, the genetic split-merge algorithm has been introduced as an automatic clustering algorithm. The algorithm is specifically designed for ontology clustering by combining the parallel genetic algorithm with the split-and-merge algorithm.

(ii) In Chapter 5, the genetic similarity algorithm has been introduced as a semantic similarity searching algorithm. The algorithm is specifically designed for ontology searching by combining the parallel genetic algorithm with the semantic similarity measure algorithm.

(iii) In Chapter 5, the *basic* UTMGO has been developed as a semantic similarity-based GO browser. The tool is specifically developed for finding a group of semantically similar GO terms for a given query GO term.

(iv) In Chapter 6, the *extended* UTMGO has been developed as a GO-based protein sequence annotation tool. The tool is specifically developed for finding a group of GO terms which are predicted to have higher relationship with the query anonymous protein sequence that can be used for annotation of the query anonymous protein sequence.

7.3 Future Works and Constraints

Future work for the genetic split-merge algorithm and the genetic similarity algorithm is to develop an adaptive mechanism that is capable of automatically determining the optimal values of genetic algorithm parameters such as crossover probability, mutation probability, and replacement percentage. This is due to the fact that the most suitable combination of parameters for one problem or data set is not always optimal for others. Therefore, these parameters should be tuned whenever the problem or data set changes. Particularly for the genetic split-merge algorithm, an improvement to be considered is to use semantic similarity measure during the calculation of the degree of interaction between GO terms by the cohesion-and-coupling metric. On the other hand, for the genetic similarity algorithm, further improvement includes taking the known correlations among GO terms into consideration in the calculation of the conceptual distance.

Future improvements in the *basic* and *extended* UTMGO are to provide the user with free text typing during entering the GO term and to develop a thesaurus for the user to check the predicted annotation. Specifically for the *basic* UTMGO, future development direction is to implement it to predict protein function and protein-protein interactions. For *extended* UTMGO, additional enhancement includes the ability to support more than one protein sequence per query and to accept DNA sequence as an input.

Some constraints identified in this study are as follows: Determining which GO terms are relevant using Equations 5.15 and 5.16 from over 20 thousand GO terms is not an easy task to execute, especially when what is relevant can be very subjective. A ranking function that determines the ordering of the query results, in order to determine how relevant a GO term is, is required for a basic calculation to accurately estimate the recall and precision. In the meantime, as the size of the GO increases, additional computing resources are required to provide faster results. Understanding of the GO terms and their properties by the users is also required in order for them to use the *basic* and the *extended* UTMGO efficiently.

7.4 Summary

In this chapter, we concluded our study and presented the contributions to solve the problems of browsing the GO terms and annotating the protein sequence. The chapter ended with proposing some directions for further research works.

LIST OF RELATED PUBLICATIONS

No	Journal	Related Chapters
1	Othman R. M., Deris S., and Illias R. M. Computational Method for Annotation of Protein Sequence According to Gene Ontology Terms. *International Journal of Biomedical Sciences*. 2006. 1(3): 186-199. [a GO Bibliography]	Chapters 1-3
2	Othman R. M., Deris S., Illias R. M., Zakaria Z., and Mohamad S. M. Automatic Clustering of Gene Ontology by Genetic Algorithm. *International Journal of Information Technology*. 2006. 3(1): 37-46. [a GO Bibliography]	Chapter 4
3	Othman R. M., Deris S., and Illias R. M. A Genetic Split-Merge Algorithm for Splitting the Monolithic Gene Ontology RDF/XML File. *Journal of Biomedical Informatics*. Accepted and revision under review. [impact factor of the journal: 2.388]	Chapter 4
4	Othman R. M., Deris S., Illias R. M., Alashwal H. T., Hassan R., and Mohamed F. Incorporating Semantic Similarity Measure in Genetic Algorithm: An Approach for Searching the Gene Ontology Terms. *International Journal of Computational Intelligence*. 2006. 3(3): 257-266. [a GO Bibliography]	Chapter 5
5	Othman R. M., Deris S., and Illias R. M. A Genetic Similarity Algorithm for Searching the Gene Ontology Terms and Annotating Anonymous Protein Sequences. *Journal of Biomedical Informatics*. DOI: 10.1016/j.jbi.2007.05.010. [impact factor of the journal: 2.388]	Chapters 5-6

No	Journal	Related Chapters
6	Othman R. M., Deris S., and Illias R. M. UTMGO: A Tool for Searching a Group of Semantically Related Gene Ontology Terms and Application to Annotation of Anonymous Protein Sequence. *International Journal of Biomedical Sciences*. 2006. 1(2): 111-119. [a GO Bibliography]	Chapters 5-6

REFERENCES

[1] Chinnasamy A., Mittal A., and Sung W. K. Probabilistic Prediction of Protein-Protein Interactions from the Protein Sequences. *Computers in Biology and Medicine*. 2006. 36(10): 1143-1154.

[2] Pireddu L., Szafron D., Lu P., and Greiner R. The Path-A Metabolic Pathway Prediction Web Server. *Nucleic Acids Research*. 2006. 34(Web Server Issue): W714-W719.

[3] Acquaah-Mensah G. K., Leach S. M., and Guda C. Predicting the Subcellular Localization of Human Proteins Using Machine Learning and Exploratory Data Analysis. *Genomics Proteomics Bioinformatics*. 2006. 4(2): 120-133.

[4] Whitfield E. J., Pruess M., and Apweiler R. Bioinformatics Database Infrastructure for Biotechnology Research. *Journal of Biotechnology*. 2006. 124(4): 629-639.

[5] Brooksbank C., Cameron G., and Thornton J. The European Bioinformatics Institute's Data Resources: Towards Systems Biology. *Nucleic Acids Research*. 2005. 33(Database Issue): D46-D53.

[6] Apweiler R., Bairoch A., and Wu C. H. Protein Sequence Databases. *Current Opinion in Chemical Biology*. 2004. 8(1): 76-80.

[7] Kretschmann E., Fleischmann W., and Apweiler R. Automatic Rule Generation for Protein Annotation with the C4.5 Data Mining Algorithm Applied on SWISS-PROT. *Bioinformatics*. 2001. 17(10): 920-926.

[8] Wu C. H., Huang H., Yeh L. S., and Barker W. C. Protein Family Classification and Functional Annotation. *Computational Biology and Chemistry*. 2003. 27(1): 37-47.

[9] Gattiker A., Michoud K., Rivoire C., Auchincloss A. H., Coudert E., Lima T., Kersey P., Pagni M., Sigrist C. J., Lachaize C., Veuthey A. L., Gasteiger E., and Bairoch A. Automated Annotation of Microbial Proteomes in SWISS-

PROT. *Computational Biology and Chemistry*. 2003. 27(1): 49-58.

[10] Fleischmann W., Moller S., Gateau A., and Apweiler R. A Novel Method for Automatic Functional Annotation of Proteins. *Bioinformatics*. 1999. 15(3): 228-233.

[11] Apweiler R., Bairoch A., Wu C. H., Barker W. C., Boeckmann B., Ferro S., Gasteiger E., Huang H., Lopez R., Magrane M., Martin M. J., Natale D. A., O'Donovan C., Redaschi N., and Yeh L. S. UniProt: The Universal Protein Knowledgebase. *Nucleic Acids Research*. 2004. 32(Database Issue): D115-D119.

[12] Wu C. H., Apweiler R., Bairoch A., Natale D. A., Barker W. C., Boeckmann B., Ferro S., Gasteiger E., Huang H., Lopez R., Magrane M., Martin M. J., Mazumder R., O'Donovan C., Redaschi N., and Suzek B. The Universal Protein Resource (UniProt): An Expanding Universe of Protein Information. *Nucleic Acids Research*. 2006. 34(Database Issue): D187-D191.

[13] Bairoch A., Apweiler R., Wu C. H., Barker W. C., Boeckmann B., Ferro S., Gasteiger E., Huang H., Lopez R., Magrane M., Martin M. J., Natale D. A., O'Donovan C., Redaschi N., and Yeh L. S. The Universal Protein Resource (UniProt). *Nucleic Acids Research*. 2005. 33(Database Issue): D154-D159.

[14] Snyder K. A., Feldman H. J., Dumontier M., Salama J. J., and Hogue C. W. Domain-Based Small Molecule Binding Site Annotation. *BMC Bioinformatics*. 2006. 7: 152.

[15] Koski L. B., Gray M. W., Lang B. F., and Burger G. AutoFACT: An Automatic Functional Annotation and Classification Tool. *BMC Bioinformatics*. 2005. 6: 151.

[16] Jones C. E., Baumann U., and Brown A. L. Automated Methods of Predicting the Function of Biological Sequences Using GO and BLAST. *BMC Bioinformatics*. 2005. 6: 272.

[17] Prlic A., Domingues F. S., Lackner P., and Sippl M. J. WILMA-Automated Annotation of Protein Sequences. *Bioinformatics*. 2004. 20(1): 127-128.

[18] Yuan X., Hu Z. Z., Wu H. T., Torii M., Narayanaswamy M., Ravikumar K. E., Vijay-Shanker K., and Wu C. H. An Online Literature Mining Tool for Protein Phosphorylation. *Bioinformatics*. 2006. 22(13): 1668-1669.

[19] Chiang J. H. and Yu H. C. Literature Extraction of Protein Functions Using Sentence Pattern Mining. *IEEE Transactions on Knowledge and Data*

Engineering. 2005. 17(8): 1088-1098.

[20] Sigrist C. J. A., Castro E. D., Langendijk-Genevaux P. S., Saux V. L., Bairoch A., and Hulo N. ProRule: A New Database Containing Functional and Structural Information on PROSITE Profiles. *Bioinformatics*. 2005. 21(21): 4060-4066.

[21] Yu G. X. Ruleminer: A Knowledge System for Supporting High-Throughput Protein Function Annotations. *Journal of Bioinformatics and Computational Biology*. 2004. 2(4): 595-617.

[22] Morbach J., Yang A., and Marquardt W. OntoCAPE—A Large-Scale Ontology for Chemical Process Engineering. *Engineering Applications of Artificial Intelligence*. 2007. 20(2): 147-161.

[23] Williams R. J., Martinez N. D., and Golbeck J. Ontologies for Ecoinformatics. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*. 2006. 4(4): 237-242.

[24] Naphade M., Smith J. R., Tesic J., Chang S., Hsu W., Kennedy L., Hauptmann A., and Curtis J. Large-Scale Concept Ontology for Multimedia. *IEEE Multimedia*. 2006. 13(3): 86-91.

[25] Köhler J., Philippi S., Specht M., and Rüegg A. Ontology Based Text Indexing and Querying for the Semantic Web. *Knowledge-Based Systems*. 2006. 19(8): 744-754.

[26] Hess C. and Schlieder C. Ontology-Based Verification of Core Model Conformity in Conceptual Modeling. *Computers, Environment, and Urban Systems*. 2006. 30(5): 543-561.

[27] Pérez-Rey D., Maojo V., García-Remesal M., Alonso-Calvo R., Billhardt H., Martin-Sánchez F., and Sousa A. ONTOFUSION: Ontology-Based Integration of Genomic and Clinical Databases. *Computers in Biology and Medicine*. 2006. 36(7-8): 712-730.

[28] Camon E., Magrane M., Barrell D., Lee V., Dimmer E., Maslen J., Binns D., Harte N., Lopez R., and Apweiler R. The Gene Ontology Annotation (GOA) Database: Sharing Knowledge in Uniprot with Gene Ontology. *Nucleic Acids Research*. 2004. 32(Database Issue): D262-D266.

[29] Lewin A. and Grieve I. C. Grouping Gene Ontology Terms to Improve the Assessment of Gene Set Enrichment in Microarray Data. *BMC Bioinformatics*. 2006. 7: 426.

[30] Wu X., Zhu L., Guo J., Zhang D. Y., and Lin K. Prediction of Yeast Protein-Protein Interaction Network: Insights from the Gene Ontology and Annotations. *Nucleic Acids Research*. 2006. 34(7): 2137-2150.

[31] Cai Z., Mao X., Li S., and Wei L. Genome Comparison using Gene Ontology (GO) with Statistical Testing. *BMC Bioinformatics*. 2006. 7: 374.

[32] Zheng B., McLean D. C., and Lu X. Identifying Biological Concepts from a Protein-Related Corpus with a Probabilistic Topic Model. *BMC Bioinformatics*. 2006. 7: 58.

[33] The Gene Ontology Consortium. The Gene Ontology (GO) Project in 2006. *Nucleic Acids Research*. 2006. 34(Database Issue): D322-D326.

[34] Lomax J. Get Ready to GO! A Biologist's Guide to the Gene Ontology. *Briefings in Bioinformatics*. 2005. 6(3): 298-304.

[35] Harris M. A., Lomax J., Ireland A., and Clark J. I. The Gene Ontology Project. In: Subramaniam S. ed. *Encyclopedia Genetics, Genomics, Proteomics and Bioinformatics: Volume 4*. New York: John Wiley & Sons. g408202; 2005.

[36] Bada M., Stevens R., Goble C., Gil Y., Ashburner M., Blake J. A., Cherry J. M., Harris M. A., and Lewis S. A Short Study on the Success of the Gene Ontology. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*. 2004. 1(2): 235-240.

[37] The Gene Ontology Consortium. The Gene Ontology (GO) Database and Informatics Resource. *Nucleic Acids Research*. 2004. 32(Database Issue): D258-D261.

[38] The Gene Ontology Consortium. Creating the Gene Ontology Resource: Design and Implementation. *Genome Research*. 2001. 11(8): 1425-1433.

[39] The Gene Ontology Consortium. Gene Ontology: Tool for the Unification of Biology. *Nature Genetics*. 2000. 25(1): 25-29.

[40] Kanungo T., Mount D. M., Netanyahu N. S., Piatko C. D., Silverman R., and Wu A. Y. An Efficient k-Means Clustering Algorithm: Analysis and Implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2002. 24(7): 881-892.

[41] Cheng C. H., Lee W. K., and Wong K. F. A Genetic Algorithm-Based Clustering Approach for Database Partitioning. *IEEE Transactions on Systems, Man and Cybernetics, Part C*. 2002. 32(3): 215-230.

[42] Günter S. and Bunke H. Self-Organizing Map for Clustering in the Graph Domain. *Pattern Recognition Letters*. 2002. 23(4): 405-417.

[43] Hathaway R. J. and Bezdek J. C. Fuzzy c-Means Clustering of Incomplete Data. *IEEE Transactions on Systems, Man and Cybernetics, Part B*. 2001. 31(5): 735-744.

[44] Chen C. Y. and Ye F. Particle Swarm Optimization Algorithm and its Application to Clustering Analysis. *Proceedings of the 2004 IEEE International Conference on Networking, Sensing, and Control*. March 21-23, 2004. Taipei, Taiwan: IEEE. 2004. 789-794.

[45] Jain A. K., Murty M. N., and Flynn P. J. Data Clustering: A Review. *ACM Computing Surveys*. 1999. 31(3): 264-323.

[46] Berkhin P. *Survey of Clustering Data Mining Techniques*. Technical Report. Accrue Software Inc.; 2002.

[47] Kotsiantis S. and Pintelas P. Recent Advances in Clustering: A Brief Survey. *WSEAS Transactions on Information Science and Applications*. 2004. 1(1): 73-81.

[48] Pelleg D. and Moore A. X-Means: Extending k-Means with Efficient Estimation of the Number of Clusters. *Proceedings of the 17th. International Conference on Machine Learning*. June 29-July 2, 2000. Stanford, California: Morgan Kaufmann. 2000. 727-734.

[49] Hamerly G. and Elkan C. Learning the k in k-Means. *Proceedings of the 17th. Conference on Neural Information Processing Systems*. July 20-24, 2003. Vancouver, Canada: NIPS Foundation. 2003. http://books.nips.cc/nips16.html.

[50] Tseng L. Y. and Yang S. B. A Genetic Approach to the Automatic Clustering Problem. *Pattern Recognition*. 2001. 34(2): 415-424.

[51] Kuo R. J., Chang K., and Chien S. Y. Integration of Self-Organizing Feature Maps and Genetic-Algorithm-Based Clustering Method for Market Segmentation. *Journal of Organizational Computing and Electronic Commerce*. 2004. 14(1): 43-60.

[52] Dutt S. and Deng W. Probability-Based Approaches to VLSI Circuit Partitioning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. 2000. 19(5): 534-549.

[53] Walshaw C. and Cross M. Parallel Optimisation Algorithms for Multilevel

Mesh Partitioning. *Parallel Computing*. 2000. 26(12): 1635-1660.

[54] Shi J. and Malik J. Normalized Cuts and Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2000. 22(8): 888-905.

[55] Getz G., Gal H., Kela I., Notterman D. A., and Domany E. Coupled Two-Way Clustering Analysis of Breast Cancer and Colon Cancer Gene Expression Data. *Bioinformatics*. 2003. 19(9): 1079-1089.

[56] Walshaw C. and Cross M. Mesh Partitioning: A Multilevel Balancing and Refinement Algorithm. *SIAM Journal on Scientific Computing*. 2000. 22(1): 63-80.

[57] D'Amico S. J., Wang S. J., Batta R., and Rump C. M. A Simulated Annealing Approach to Police District Design. *Computers and Operations Research*. 2002. 29(6): 667-684.

[58] Saab Y. G. An Effective Multilevel Algorithm for Bisecting Graphs and Hypergraphs. *IEEE Transactions on Computers*. 2004. 53(6): 641-652.

[59] Wolfe G., Wong J. L., and Potkonjak M. Watermarking Graph Partitioning Solutions. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. 2002. 21(10): 1196-1204.

[60] Elsner U. *Graph Partitioning: A Survey*. Technical Report. Technische Universitat Chemnitz; 1997.

[61] Fjällström P. O. Algorithms for Graph Partitioning: A Survey. *Linköping Electronic Articles in Computer and Information Science*. 1998. 3(10): http://www.ep.liu.se/ea/cis/1998/010.

[62] Bui T. N. and Moon B. R. Genetic Algorithm and Graph Partitioning. *IEEE Transactions on Computers*. 1996. 45(7): 841-855.

[63] Kaveh A. and Bondarabady H. A. R. A Hybrid Graph-Genetic Method for Domain Decomposition. *Finite Elements in Analysis and Design*. 2003. 39(13): 1237-1247.

[64] Kohmoto K., Katayama K., and Narihisa H. Performance of a Genetic Algorithm for the Graph Partitioning Problem. *Mathematical and Computer Modelling*. 2003. 38(11-13): 1325-1332.

[65] Stuckenschmidt H. and Klein M. Structure-Based Partitioning of Large Concept Hierarchies. *Proceedings of the 3rd. International Semantic Web Conference*. November 7-11, 2004. Hiroshima, Japan: Springer. 2004. 289-

303.

[66] Eilbeck K., Lewis S. E., Mungall C. J., Yandell M., Stein L., Durbin R., and Ashburner M. The Sequence Ontology: A Tool for the Unification of Genome Annotations. *Genome Biology*. 2005. 6(5): R44.

[67] Bard J., Rhee S. Y., and Ashburner M. An Ontology for Cell Types. *Genome Biology*. 2005. 6(2): R21.

[68] Feldman H. J., Dumontier M., Ling S., Haider N., and Hogue C. W. CO: A Chemical Ontology for Identification of Functional Groups and Semantic Comparison of Small Molecules. *FEBS Letters*. 2005. 579(21): 4685-4691.

[69] Thompson J. D., Holbrook S. R., Katoh K., Koehl P., Moras D., Westhof E., and Poch O. MAO: A Multiple Alignment Ontology for Nucleic Acid and Protein Sequences. *Nucleic Acids Research*. 2005. 33(13): 4164-4171.

[70] Grenon P., Smith B., and Goldberg L. Biodynamic Ontology: Applying BFO in the Biomedical Domain. *Studies in Health Technology and Informatics*. 2004. 102: 20-38.

[71] Ratsch E., Schultz J., Saric J., Lavin P. C., Wittig U., Reyle U., and Rojas I. Developing a Protein-Interactions Ontology. *Comparative and Functional Genomics*. 2003. 4(1): 85-89.

[72] Grabowski J. and Pempera J. The Permutation Flow Shop Problem with Blocking: A Tabu Search Approach. *Omega*. 2007. 35(3): 302-311.

[73] Sun M. Solving the Uncapacitated Facility Location Problem Using Tabu Search. *Computers and Operations Research*. 2006. 33(9): 2563-2589.

[74] Tiwari M. K., Kumar S., Prakash, and Shankar R. Solving Part-Type Selection and Operation Allocation Problems in an FMS: An Approach Using Constraints-Based Fast Simulated Annealing Algorithm. *IEEE Transactions on Systems, Man and Cybernetics, Part A*. 2006. 36(6): 1170-1184.

[75] Attiya G. and Hamam Y. Task Allocation for Maximizing Reliability of Distributed Systems: A Simulated Annealing Approach. *Journal of Parallel and Distributed Computing*. 2006. 66(10): 1259-1266.

[76] Moz M. and Pato M. V. A Genetic Algorithm Approach to a Nurse Rerostering Problem. *Computers and Operations Research*. 2007. 34(3): 667-691.

[77] Toroslu I. H. and Arslanoglu Y. Genetic Algorithm for the Personnel Assignment Problem with Multiple Objectives. *Information Sciences*. 2007.

177(3): 787-803.

[78] Zecchin A. C., Simpson A. R., Maier H. R., Leonard M., Roberts A. J., and Berrisford M. J. Application of Two Ant Colony Optimisation Algorithms to Water Distribution System Optimization. *Mathematical and Computer Modelling*. 2006. 44(5-6): 451-468.

[79] Yin P. Y. and Wang J. Y. Ant Colony Optimization for the Nonlinear Resource Allocation Problem. *Applied Mathematics and Computation*. 2006. 174(2): 1438-1453.

[80] Heo J. S., Lee K. Y., and Garduno-Ramirez R. Multiobjective Control of Power Plants Using Particle Swarm Optimization Techniques. *IEEE Transactions on Energy Conversion*. 2006. 21(2): 552-561.

[81] Jacobson S. H., McLay L. A., Hall S. N., Henderson D., and Vaughan D. E. Optimal Search Strategies Using Simultaneous Generalized Hill Climbing Algorithms. *Mathematical and Computer Modelling*. 2006. 43(9-10): 1061-1073.

[82] You L. and Wood S. Assessing the Spatial Distribution of Crop Areas Using a Cross-Entropy Method. *International Journal of Applied Earth Observation and Geoinformation*. 2005. 7(4): 310-323.

[83] Arostegui Jr. M. A., Kadipasaoglu S. N., and Khumawala B. M. An Empirical Comparison of Tabu Search, Simulated Annealing, and Genetic Algorithms for Facilities Location Problems. *International Journal of Production Economics*. 2006. 103(2): 742-754.

[84] Kannan S., Slochanal S. M. R., and Padhy N. P. Application and Comparison of Metaheuristic Techniques to Generation Expansion Planning Problem. *IEEE Transactions on Power Systems*. 2005. 20(1): 466-475.

[85] Elbeltagi E., Hegazy T., and Grierson D. Comparison among Five Evolutionary-Based Optimization Algorithms. *Advanced Engineering Informatics*. 2005. 19(1): 43-53.

[86] Kumar S., Ong S. H., Ranganath S., and Chew F. T. A Luminance- and Contrast-Invariant Edge-Similarity Measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2006. 28(12): 2042-2048.

[87] Clayden J. D., Bastin M. E., and Storkey A. J. Improved Segmentation Reproducibility in Group Tractography Using a Quantitative Tract Similarity Measure. *NeuroImage*. 2006. 33(2): 482-492.

[88] Paclik P., Novovicova J., and Duin R. P. W. Building Road-Sign Classifiers Using a Trainable Similarity Measure. *IEEE Transactions on Intelligent Transportation Systems*. 2006. 7(3): 309-321.

[89] Peng Y. and Ngo C. W. Clip-Based Similarity Measure for Query-Dependent Clip Retrieval and Video Summarization. *IEEE Transactions on Circuits and Systems for Video Technology*. 2006. 16(5): 612-627.

[90] van der Meer F. The Effectiveness of Spectral Similarity Measures for the Analysis of Hyperspectral Imagery. *International Journal of Applied Earth Observation and Geoinformation*. 2006. 8(1): 3-17.

[91] Chen X., Tian J., and Yang X. A New Algorithm for Distorted Fingerprints Matching Based on Normalized Fuzzy Similarity Measure. *IEEE Transactions on Image Processing*. 2006. 15(3): 767-776.

[92] Lee S. and Crawford M. M. Unsupervised Multistage Image Classification Using Hierarchical Clustering with a Bayesian Similarity Measure. *IEEE Transactions on Image Processing*. 2005. 14(3): 312-320.

[93] Moghaddam B., Nastar C., and Pentland A. A Bayesian Similarity Measure for Deformable Image Matching. *Image and Vision Computing*. 2001. 19(5): 235-244.

[94] Skerl D., Likar B., and Pernus F. A Protocol for Evaluation of Similarity Measures for Rigid Registration. *IEEE Transactions on Medical Imaging*. 2006. 25(6): 779-791.

[95] Núñez H., Sànchez-Marrè M., Cortés U., Comas J., Martínez M., Rodríguez-Roda I., and Poch M. A Comparative Study on the Use of Similarity Measures in Case-Based Reasoning to Improve the Classification of Environmental System Situations. *Environmental Modelling and Software*. 2004. 19(9): 809-819.

[96] Kirac M., Ozsoyoglu G., and Yang J. Annotating Proteins by Mining Protein Interaction Networks. *Bioinformatics*. 2006. 22(14): e260-e270.

[97] Ray S. and Craven M. Learning Statistical Models for Annotating Proteins with Function Information Using Biomedical Text. *BMC Bioinformatics*. 2005. 6(Suppl 1): S18.

[98] Ponomarenko J. V., Bourne P. E., and Shindyalov I. N. Assigning New GO Annotations to Protein Data Bank Sequences by Combining Structure and Sequence Homology. *Proteins*. 2005. 58(4): 855-865.

[99] Salasznyk R. M., Westcott A. M., Klees R. F., Ward D. F., Xiang Z., Vandenberg S., Bennett K., and Plopper G. E. Comparing the Protein Expression Profiles of Human Mesenchymal Stem Cells and Human Osteoblasts Using Gene Ontologies. *Stem Cells and Development*. 2005. 14(4): 354-366.

[100] Basu S., Bremer E., Zhou C., and Bogenhagen D. F. MiGenes: A Searchable Interspecies Database of Mitochondrial Proteins Curated Using Gene Ontology Annotation. *Bioinformatics*. 2005. 22(4): 485-492.

[101] Lu P., Szafron D., Greiner R., Wishart D. S., Fyshe A., Pearcy B., Poulin B., Eisner R., Ngo D., and Lamb N. PA-GOSUB: A Searchable Database of Model Organism Protein Sequences with their Predicted Gene Ontology Molecular Function and Subcellular Localization. *Nucleic Acids Research*. 2005. 33(Database Issue): D147-D153.

[102] Camon E., Magrane M., Barrell D., Binns D., Fleischmann W., Kersey P., Mulder N., Oinn T., Maslen J., Cox A., and Apweiler R. The Gene Ontology Annotation (GOA) Project: Implementation of GO in SWISS-PROT, TrEMBL, and InterPro. *Genome Research*. 2003.13(4): 662-672.

[103] Ye J., Fang L., Zheng H., Zhang Y., Chen J., Zhang Z., Wang J., Li S., Li R., Bolund L., and Wang J. WEGO: A Web Tool for Plotting GO Annotations. *Nucleic Acids Research*. 2006. 34(Web Server Issue): W293-W297.

[104] Lee H. K., Braynen W., Keshav K., and Pavlidis P. ErmineJ: Tool for Functional Analysis of Gene Expression Data Sets. *BMC Bioinformatics*. 2005. 6: 269.

[105] Liu H., Hu Z. Z., and Wu C. H. DynGO: A Tool for Browsing and Mining Gene Ontology and its Associations. *BMC Bioinformatics*. 2005. 6: 201.

[106] Aitken S., Korf R., Webber B., and Bard J. COBrA: A Bio-Ontology Editor. *Bioinformatics*. 2005. 21(6): 825-826.

[107] Fu L. and Medico E. FLAME, A Novel Fuzzy Clustering Method for the Analysis of DNA Microarray Data. *BMC Bioinformatics*. 2007. 8: 3.

[108] Kim S. Y., Lee J. W., and Bae J. S. Effect of Data Normalization on Fuzzy Clustering of DNA Microarray Data. *BMC Bioinformatics*. 2006. 7: 134.

[109] Zhong W., He J., Harrison R., Tai P. C., and Pan Y. Clustering Support Vector Machines for Protein Local Structure Prediction. *Expert Systems with Applications*. 2007. 32(2): 518-526.

[110] Huang J., Tzeng G., and Ong C. Marketing Segmentation Using Support Vector Clustering. *Expert Systems with Applications*. 2007. 32(2): 313-317.

[111] Papamichail G. P. and Papamichail D. P. The k-Means Range Algorithm for Personalized Data Clustering in e-Commerce. *European Journal of Operational Research*. 2007. 177(3): 1400-1408.

[112] Chan Z. S. H., Collins L., and Kasabov N. An Efficient Greedy k-Means Algorithm for Global Gene Trajectory Clustering. *Expert Systems with Applications*. 2006. 30(1): 137-141.

[113] Martinez-Estudillo A. C., Hervas-Martinez C., Martinez-Estudillo F. J., and Garcia-Pedrajas N. Hybridization of Evolutionary Algorithms and Local Search by Means of a Clustering Method. *IEEE Transactions on Systems, Man and Cybernetics, Part B*. 2006. 36(3): 534-545.

[114] Gesu V. D., Giancarlo R., Bosco G. L., Raimondi A., and Scaturro S. GenClust: A Genetic Algorithm for Clustering Gene Expression Data. *BMC Bioinformatics*. 2005. 6: 289.

[115] Meunier B., Dumas E., Piec I., Bechet D., Hebraud M., and Hocquette J. F. Assessment of Hierarchical Clustering Methodologies for Proteomic Data Mining. *Journal of Proteome Research*. 2007. 6(1): 358-366.

[116] Li Y. Bayesian Model Based Clustering Analysis: Application to a Molecular Dynamics Trajectory of the HIV-1 Integrase Catalytic Core. *Journal of Chemical Information and Modeling*. 2006. 46(4): 1742-1750.

[117] Zeng Y. and Garcia-Frias J. A Novel HMM-Based Clustering Algorithm for the Analysis of Gene Expression Time-Course Data. *Computational Statistics and Data Analysis*. 2006. 50(9): 2472-2494.

[118] Torres H. M., Gurlekian J. A., Rufiner H. L., and Torres M. E. Self-Organizing Map Clustering Based on Continuous Multiresolution Entropy. *Physica A: Statistical Mechanics and its Applications*. 2006. 361(1): 337-354.

[119] Mitra S., Banka H., and Pedrycz W. Rough–Fuzzy Collaborative Clustering. *IEEE Transactions on Systems, Man and Cybernetics, Part B*. 2006. 36(4): 795-805.

[120] Peters G. Some Refinements of Rough k-Means Clustering. *Pattern Recognition*. 2006. 39(8): 1481-1491.

[121] Zio E. and Baraldi P. Evolutionary Fuzzy Clustering for the Classification of Transients in Nuclear Components. *Progress in Nuclear Energy*. 2005. 46(3-

4): 282-296.

[122] Datta S. and Datta S. Evaluation of Clustering Algorithms for Gene Expression Data. *BMC Bioinformatics*. 2006. 7(Suppl 4): S17.

[123] Mingoti S. A. and Lima J. O. Comparing SOM Neural Network with Fuzzy c-Means, k-Means and Traditional Hierarchical Clustering Algorithms. *European Journal of Operational Research*. 2006. 174(3): 1742-1759.

[124] Thalamuthu A., Mukhopadhyay I., Zheng X., and Tseng G. C. Evaluation and Comparison of Gene Clustering Methods in Microarray Analysis. *Bioinformatics*. 2006. 22(9): 2405-2412.

[125] Guldemır H. and Sengur A. Comparison of Clustering Algorithms for Analog Modulation Classification. *Expert Systems with Applications*. 2006. 30(4): 642-649.

[126] Ma P. C. H., Chan K. C. C., Yao X., and Chiu D. K. Y. An Evolutionary Clustering Algorithm for Gene Expression Microarray Data Analysis. *IEEE Transactions on Evolutionary Computation*. 2006. 10(3): 296-314.

[127] Laszlo M. and Mukherjee S. A Genetic Algorithm Using Hyper-Quadtrees for Low-Dimensional k-Means Clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2006. 28(4): 533-543.

[128] Sheng W., Swift W., Zhang L., and Liu X. A Weighted Sum Validity Function for Clustering with a Hybrid Niching Genetic Algorithm. *IEEE Transactions on Systems, Man and Cybernetics, Part B*. 2005. 35(6): 1156-1167.

[129] Aykanat C., Cambazoglu B. B., Findik F., and Kurc T. Adaptive Decomposition and Remapping Algorithms for Object-Space-Parallel Direct Volume Rendering of Unstructured Grids. *Journal of Parallel and Distributed Computing*. 2007. 67(1): 77-99.

[130] Duarte A., Sánchez Á., Fernández F., and Montemayor A. S. Improving Image Segmentation Quality through Effective Region Merging Using a Hierarchical Social Metaheuristic. *Pattern Recognition Letters*. 2006. 27(11): 1239-1251.

[131] Mitchell B. S. and Mancoridis S. On the Automatic Modularization of Software Systems Using the Bunch Tool. *IEEE Transactions on Software Engineering*. 2006. 32(3): 193-208.

[132] Salim N. and Mohemad R. Compound Selection for Drug Lead Identification

Using Genetic Algorithm (GA). *Journal of Advancing Information and Management Studies*. 2005. 2(1): 46-55.

[133] Garai G. and Chaudhuri B. B. A Novel Genetic Algorithm for Automatic Clustering. *Pattern Recognition Letters*. 2004. 25(2): 173-187.

[134] Takashima E., Murata Y., Shibata N., and Ito M. Techniques to Improve Exploration Efficiency of Parallel Self-Adaptive Genetic Algorithms by Dispensing with Iteration and Synchronization. *Systems and Computers in Japan*. 2006. 37(14): 25-33.

[135] Rahul, Chakraborty D., and Dutta A. Optimization of FRP Composites against Impact Induced Failure Using Island Model Parallel Genetic Algorithm. *Composites Science and Technology*. 2005. 65(13): 2003-2013.

[136] Katayama K., Hirabayashi H., and Narihisa H. Analysis of Crossovers and Selections in a Coarse-Grained Parallel Genetic Algorithm. *Mathematical and Computer Modelling*. 2003. 38(11-13): 1275-1282.

[137] Gropp W., Lusk E., Ashton D., Buntinas D., Butler R., Chan A., Ross R., Thakur R., and Toonen B. *MPICH2 User's Guide Version 1.0.4*. MPICH2 Documentation. Argonne National Laboratory; 2006.

[138] Wall M. *GAlib: A C++ Library of Genetic Algorithm Components*. GAlib Documentation. Massachusetts Institute of Technology; 1996.

[139] Günter S. and Bunke H. Validation Indices for Graph Clustering. *Pattern Recognition Letters*. 2003. 24(8): 1107-1113.

[140] Maulik U. and Bandyopadhyay S. Performance Evaluation of Some Clustering Algorithms and Validity Indices. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2002. 24(12): 1650-1654.

[141] Bandyopadhyay S. and Maulik U. Nonparametric Genetic Clustering: Comparison of Validity Indices. *IEEE Transactions on Systems, Man and Cybernetics, Part C*. 2001. 31(1): 120-125.

[142] Cui X., Gao J., and Potok T. E. A Flocking Based Algorithm for Document Clustering Analysis. *Journal of Systems Architecture*. 2006. 52(8): 505-515.

[143] Watts R. J. and Porter A. L. R&D Cluster Quality Measures and Technology Maturity. *Technological Forecasting and Social Change*. 2003. 70(8): 735-758.

[144] Carroll S. and Pavlovic V. Protein Classification Using Probabilistic Chain Graphs and the Gene Ontology Structure. *Bioinformatics*. 2006. 22(15):

1871-1878.

[145] Zhou G. P. and Cai Y. D. Predicting Protease Types by Hybridizing Gene Ontology and Pseudo Amino Acid Composition. *Proteins*. 2006. 63(3): 681-684.

[146] Scheer M., Klawonn F., Munch R., Grote A., Hiller K., Choi C., Koch I., Schobert M., Hartig E., Klages U., and Jahn D. JProGO: A Novel Tool for the Functional Interpretation of Prokaryotic Microarray Data Using Gene Ontology Information. *Nucleic Acids Research*. 2006. 34(Web Server Issue): W510-W515.

[147] Guo X., Liu R., Shriver C. D., Hu H., and Liebman M. N. Assessing Semantic Similarity Measures for the Characterization of Human Regulatory Pathways. *Bioinformatics*. 2006. 22(8): 967-973.

[148] Tang Y. and Zheng J. Linguistic Modelling Based on Semantic Similarity Relation among Linguistic Labels. *Fuzzy Sets and Systems*. 2006. 157(12): 1662-1673.

[149] Steichen O., Bozec C. D., Thieu M., Zapletal E., and Jaulent M. C. Computation of Semantic Similarity Within an Ontology of Breast Pathology to Assist Inter-Observer Consensus. *Computers in Biology and Medicine*. 2006. 36(7-8): 768-788.

[150] Maki W. S., Krimsky M., and Munoz S. An Efficient Method for Estimating Semantic Similarity Based on Feature Overlap: Reliability and Validity of Semantic Feature Ratings. *Behavior Research Methods*. 2006. 38(1): 153-157.

[151] Leacock C. and Chodorow M. Combining Local Context and WordNet Similarity for Word Sense Identification. In: Fellbaum C. ed. *WordNet: An Electronic Lexical Database*. Cambridge: MIT Press. 265-283; 1998.

[152] Lin D. An Information-Theoretic Definition of Similarity. *Proceedings of the International Conference on Machine Learning*. July 24-27, 1998. Madison, Wisconsin: Morgan Kaufmann. 1998. 296-304.

[153] Jiang J. J. and Conrath D. W. Semantic Similarity Based on Corpus Statistics and Lexical Taxonomy. *Proceedings of the International Conference on Research in Computational Linguistics*. August 22-24, 1997. Taipei, Taiwan: Morgan Kaufmann. 1998. 19-33.

[154] Resnik P. Using Information Content to Evaluate Semantic Similarity in a

Taxonomy. *Proceedings of the International Joint Conference on Artificial Intelligence*. August 22-25, 1995. Montreal, Canada: Morgan Kaufmann. 1995. 448-453.

[155] Budanitsky A. and Hirst G. Evaluating WordNet-Based Measures of Lexical Semantic Relatedness. *Computational Linguistics*. 2006. 32(1): 13-47.

[156] Lord P. W., Stevens R. D., Brass A., and Goble C. A. Investigating Semantic Similarity Measures across the Gene Ontology: The Relationship Between Sequence and Annotation. *Bioinformatics*. 2003. 19(10): 1275-1283.

[157] Popescu M., Keller J. M., and Mitchell J. A. Fuzzy Measures on the Gene Ontology for Gene Product Similarity. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*. 2006. 3(3): 263-274.

[158] Sevilla J. L., Segura V., Podhorski A., Guruceaga E., Mato J. M., Martínez-Cruz L. A., Corrales F. J., and Rubio A. Correlation Between Gene Expression and GO Semantic Similarity. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*. 2005. 2(4): 330-338.

[159] Mitra S. and Hayashi Y. Bioinformatics with Soft Computing. *IEEE Transactions on System, Man and Cybernetics, Part C*. 2006. 36(5): 616-635.

[160] Kushchu I. Web-Based Evolutionary and Adaptive Information Retrieval. *IEEE Transactions on Evolutionary Computation*. 2005. 9(2): 117-125.

[161] Pal S. K., Talwar V., and Mitra P. Web Mining in Soft Computing Framework: Relevance, State of the Art and Future Directions. *IEEE Transactions on Neural Networks*. 2002. 13(5): 1163-1177.

[162] Chen L., Luh C., and Jou C. Generating Page Clippings from Web Search Results Using a Dynamically Terminated Genetic Algorithm. *Information Systems*. 2005. 30(4): 299-316.

[163] Paul T. K. and Iba H. Gene Selection for Classification of Cancers Using Probabilistic Model Building Genetic Algorithm. *Biosystems*. 2005. 82(3): 208-225.

[164] Rodriguez M. A. and Jarur M. C. A Genetic Algorithm for Searching Spatial Configurations. *IEEE Transactions on Evolutionary Computation*. 2005. 9(3): 252-270.

[165] Tamine L., Chrisment C., and Boughanem M. Multiple Query Evaluation Based on an Enhanced Genetic Algorithm. *Information Possessing and Management*. 2003. 39(2): 215-231.

[166] Couto F., Silva M., and Coutinho P. Semantic Similarity over the Gene Ontology: Family Correlation and Selecting Disjunctive Ancestors. *Proceedings of the ACM Conference on Information and Knowledge Management*. October 31-November 5, 2005. Bremen, Germany: ACM Press. 2005. 343-344.

[167] Leung A. K., Trinkle-Mulcahy L., Lam Y. W., Andersen J. S., Mann M., and Lamond A. I. NOPdb: Nucleolar Proteome Database. *Nucleic Acids Research*. 2006. 34(Database Issue): D218-D220.

[168] Winter C., Henschel A., Kim W. K., and Schroeder M. SCOPPI: A Structural Classification of Protein-Protein Interfaces. *Nucleic Acids Research*. 2006. 34(Database Issue): D310-D314.

[169] Gao G., Zhong Y., Guo A., Zhu Q., Tang W., Zheng W., Gu X., Wei L., and Luo J. DRTF: A Database of Rice Transcription Factors. *Bioinformatics*. 2006. 22(10): 1286-1287.

[170] Flores S., Echols N., Milburn D., Hespenheide B., Keating K., Lu J., Wells S., Yu E. Z., Thorpe M., and Gerstein M. The Database of Macromolecular Motions: New Features Added at the Decade Mark. *Nucleic Acids Research*. 2006. 34(Database Issue): D296-D301.

[171] Khan S., Situ G., Decker K., and Schmidt C. J. GoFigure: Automated Gene Ontology Annotation. *Bioinformatics*. 2003.19(18): 2484-2485.

[172] Martin D. M., Berriman M., and Barton G. J. GOtcha: A New Method for Prediction of Protein Function Assessed by the Annotation of Seven Genomes. *BMC Bioinformatics*. 2004. 5: 178.

[173] Vinayagam A., del Val C., Schubert F., Eils R., Glatting K. H., Suhai S., and König R. GOPET: A Tool for Automated Predictions of Gene Ontology Terms. *BMC Bioinformatics*. 2006. 7: 161.

[174] Friedberg I., Harder T., and Godzik A. JAFA: A Protein Function Annotation Meta-Server. *Nucleic Acids Research*. 2006. 34(Web Server Issue): W379-W381.

[175] Groth D., Lehrach H., and Hennig S. GOblet: A Platform for Gene Ontology Annotation of Anonymous Sequence Data. *Nucleic Acids Research*. 2004. 32(Web Server Issue): W313-W317.

[176] Enault F., Suhre K., Poirot O., Abergel C., and Claverie J. M. Phydbac (Phylogenomic Display of Bacterial Genes): An Interactive Resource for the

Annotation of Bacterial Genomes. *Nucleic Acids Research*. 2003. 31(13): 3720-3722.

[177] Quevillon E., Silventoinen V., Pillai S., Harte N., Mulder N., Apweiler R., and Lopez R. InterProScan: Protein Domains Identifier. *Nucleic Acids Research*. 2005. 33(Web Server Issue): W116-W120.