POLITECNICO DI MILANO
V Facoltà di Ingegneria
Laurea Ingegneria Informatica
Dipartimento di Elettronica e Informazione

A flexible and context-independent software architecture for constructing, composing and validating complex networks

Relatore: Prof. Marco Masserolii

Tesi di Laurea di:
Francesco Gini Matr. 765058

Anno Accademico 2012-2013

Chapter 1

ABSTRACT

Complex networks have proved to be a powerful modelling tool to analyze systems whose relations are difficulty representable by means of more traditional techniques, such as mathematical equations. They have been already used in many fields, ranging from economics and sociology to computer science and bioinformatics. The research work carried out for this Master's thesis pertains to the first step involved in a typical network analysis workflow, namely, the building network task. This is a fundamental part of the whole workflow, as, generally, analysis results highly depend on the structure of the model to which they are applied. Therefore, this thesis is aimed at designing and implementing a generic and reusable software infrastructure which can perform and automatize the task of network construction - a laborious and tedious step often slowing down the set up time of experiments based on complex network analysis. The software architecture is conceived not only to work with data organized in different ways but also to orchestrate a composition workflow that can be used to integrate more networks into a single one. This thesis is structured as follows. Chapter 1 gives an introduction to the biomedical context, providing the reader with information about the analysed data types related to this field. Chapter 2 'Theory of Complex Networks' presents the basic theoretical notions necessary to fully understand the network concepts used throughout the research. Moreover, it provides the reader with examples of analysis applied to real systems - mainly focused on the application of complex network theory results to the biomolecular domain - in order to give a hint of the potentialities associated to modelling systems by means of networks. As the objectives of the work are defined in Chapter 3, Chapter 4 'Algorithms and Software Libraries' presents the materials and methods adopted to build the software architecture. Chapter 5 constitutes the heart of the thesis dealing with the whole software architecture, its motivations and the associated issues. Chapter 6 and Chapter 7 both discusses the validation of the project's results. The former reports statistics mostly about the time responsiveness and the memory consumption associated to use cases run on the platform. The latter, 'Testing and Validation of composition workflow', exploits biomolecular use cases in order to validate the network composition workflow, which was implemented with the architecture. In Chapter 8, the conclusions of the project are outlined, while Chapter 9 deals with potential software developments and applications in which the framework

architecture could be exploited. The document is closed by the appendix A which includes the technical documents developed for the software architecture.

Chapter 2

SOMMARIO

Le reti complesse costituiscono un potente strumento di modellazione nell'analisi di sistemi le cui relazioni sono difficilmente rappresentabili attraverso tecniche più tradizionali, quali le equazioni matematiche. Esse hanno già trovato applicazione in svariati campi, dalle scienze sociali ed economiche, alle scienze dell'informazione, compresa la bioinformatica. Il lavoro di ricerca, svolto nell'ambito di questa tesi di Laurea Magistrale, si concentra sulla prima fase tipica di *workflow* di analisi delle reti, ossia la fase di costruzione delle reti. Questa rappresenta una parte fondamentale dell'intero *workflow*, dal momento in cui, generalmente, i risultati dell'analisi dipendono fortemente dalla struttura del modello a cui sono applicati. Per questo motivo la ricerca mira a creare ed implementare un'infrastruttura *software* generica e riutilizzabile, in grado di svolgere ed automatizzare la fase di costruzione delle reti – fase particolarmente laboriosa e spesso alla base del rallentamento di esperimenti basati sul'analisi di reti complesse. L'architettura del *software* è concepita non solo per gestire diversi formati dati, ma anche per dirigere un flusso di operazioni tali da poter integrare più reti all'interno di un unico modello, esso stesso una rete. La tesi si articola nel modo seguente. Il Capitolo 1 introduce concetti relativi al contesto biomedico, fornendo al lettore informazioni riguardo ai tipi di dati analizzati nel lavoro e appartenenti a questo ambito. Il Capitolo 2, dal titolo 'Theory of Complex Networks' - (Teoria delle reti complesse), presenta le nozioni teoriche di base necessarie alla piena comprensione di nozioni legate alle reti e ricorrenti nella tesi. Inoltre, fornisce una serie di esempi di analisi accostate a sistemi reali – riguardanti principalmente l'applicazione di risultati teorici delle reti complesse al dominio biomolecolare – allo scopo di offrire una prospettiva sulle potenzialità associate a sistemi di modellazione attraverso le reti. Mentre gli obiettivi del lavoro sono descritti nel Capitolo 3, il Capitolo 4 'Algorithms and Software Libraries' - (Algoritmi e Librerie Software) illustra i materiali e i metodi adottati per costruire l'architettura del *software*. Il Capitolo 5, dunque, rappresenta il cuore della tesi e tratta l'architettura del *software*, le motivazioni alla base della sua strutturazione e altre questioni ad essa relative. Il Capitolo 6 ed il Capitolo 7 si occupano della validazione dei risultati del progetto, concentrandosi rispettivamente su parametri legati ai tempi di esecuzione e ai consumi di memoria delle operazioni lanciate sulla piattaforma il primo, e sui casi d'uso biomolecolari al fine di validare il processo di composizione di reti im-

plementato all'interno dell'architettura, il secondo, intitolato 'Testing and Validation of composition workflow' - (Test e validazione del workflow di composizione). Nel Capitolo 8 vengono delineate le conclusioni tratte in seguito alla realizzazione del progetto e il Capitolo 9, infine, illustra i potenziali sviluppi futuri del *software* ed le eventuali applicazioni per cui l'architettura, così strutturata, potrebbe essere sfruttata. In calce al lavoro, l'Appendice A contiene tutti i documenti tecnici sviluppati per l'architettura del *software*.

Contents

1 ABSTRACT **1**

2 SOMMARIO **3**

3 INTRODUCTION **13**

3.1 Genes and Proteins 13

3.2 Genomic and Proteomic Research 16

3.3 Biomedical-Molecular Terminologies and Ontologies 17

3.4 Biomolecular Databanks 19

3.5 Issues Related to an Effective Use of Available Biomolecular Information 24

3.6 Complex Networks Analysis 24

4 THEORY OF COMPLEX NETWORKS **29**

4.1 Graph Definition 29

4.2 Networks Properties 32

4.2.1 Topology of Real Networks 38

4.3 Main Network Investigations 41

4.3.1 Centrality Measures 41

4.3.2 Community Structure 47

5 OBJECTIVES **51**

6 ALGORITHMS AND SOFTWARE LIBRARIES **55**

6.1 Similarity Metrics 55

6.2 Network Composition Algorithm 57

6.3 Validation Algorithm 59

6.4 Techniques of Network Representation 60

6.4.1 Sparse Matrices 62

6.5 Software Libraries 64

6.5.1 Mathematical Kernels 64

6.5.2 Multithreading 69

7 DESIGN AND IMPLEMENTATION RESULTS **71**

7.1 System Requirements 71

7.2 Workflow Definition 72

7.2.1 Use Cases of the System 72

7.2.2 Toy Example of a Co-Autorship Network 74
7.2.3 Simple Network Construction Workflow 78
7.2.4 Composite Network Construction Workflow 81
7.3 Software Architecture 82
7.3.1 Preliminary Choices 83
7.3.2 Java Subsystem 84
7.3.2.1 Input and Configuration 84
7.3.2.2 Adding a New Functionality (a) 86
7.3.2.3 Data Retrievers 90
7.3.2.4 The Roles of Networks within the System 92
7.3.3 Native Code Subsystem 93
7.3.3.1 The Session Mechanism 93
7.3.3.2 Strategy Pattern for a Single Network Construction . 96
7.3.3.3 Adding a New Functionality (b) 96
7.3.3.4 Local and Global Methods Implementation 98
7.3.3.5 Composing Input Networks 98
7.3.4 Validation Workflow 99
7.4 Implementation Issues and Solutions 100
7.4.1 Computation of Networks from Annotation Profiles 101
7.4.2 Computational Efficiency 102
7.4.2.1 The Resnik Similarity Computation Case 104
7.4.3 Functions Included in the Software Architecture 107

8 PERFORMANCE OF THE IMPLEMENTED SOFTWARE SYSTEM 111
8.1 Comparison of Performances for the Single Network Construction Cases 111
8.1.1 Networks Constructed from Node Associations 112
8.1.2 Networks Constructed from Flat Annotation Profiles 113
8.1.3 Networks Constructed from Ontological Annotation Profiles . . 114
8.2 Performance of Composition Workflow 115
8.3 Parameters Incidence on Performance 117

9 TESTING AND VALIDATION OF COMPOSITION WORKFLOW 123
9.1 Gene Ontology Annotation Prediction for the Gallus gallus Species . . 123
9.1.1 Evaluation of Parameter Impact on the Prediction Task 125
9.1.2 Comparison of GO Ontology Dependencies 128
9.2 Gene Function Prediction with Heterogeneous Data 131

10 CONCLUSIONS 135

11 FUTURE WORKS 137

Bibliography **138**

A SOFTWARE ARCHITECTURE **147**

A.1 Java Subsystem 147

A.1.1 Package InputTypes 149

A.1.2 Package Configuration 150

A.1.3 Package DescriptionCreator 152

A.1.4 Package MethodDescriptors 154

A.1.5 Package Retriever 154

A.2 C++ Subsystem 155

List of Figures

3.1 DNA double helical structure . . . 13
3.2 Transcription and translation processes . . . 14
3.3 Transcription from nuclear RNA to mRNA . . . 15
3.4 Standard genetic code . . . 15
3.5 DAG structure and Gene Ontology . . . 19
3.6 Growth of biological databank number . . . 22
3.7 Genomic data file formats . . . 23
3.8 Tabular data file formats . . . 23

4.1 Undirected, directed, weighted graphs . . . 30
4.2 Bipartite network . . . 31
4.3 Degree distributions for real networks . . . 34
4.4 Local clustering coefficient . . . 36
4.5 Community structures . . . 37
4.6 Internet network structure . . . 39
4.7 Examples of real world networks . . . 40
4.8 Betweenneess measure . . . 46
4.9 Overlapping community structures . . . 49

6.1 Complexity of operations for various network representations . . . 61
6.2 The structure of a binary heap . . . 62
6.3 Eigen matrix vector product benchmark . . . 68

7.1 Use cases for the software architecture . . . 73
7.2 Direct network representation . . . 73
7.3 Examples of annotation profiles and their meaning . . . 75
7.4 Workflow of simple network construction . . . 79
7.5 Unfolding mechanism . . . 80
7.6 Workflow of the network composition process . . . 81
7.7 Main view of the system architecture . . . 83
7.8 Schema of the generic adapter pattern . . . 84
7.9 Xml file of a request to the software architecture . . . 85
7.10 Xml description of a sparsification method . . . 86
7.11 Sequence diagram associated to a network construction request . . . 87
7.12 Class diagram for the inclusion of a new method . . . 89

7.13 Xml description of the sparsification method to be used 90
7.14 Xml description of the annotation retriever to be used 92
7.15 Xml description of the inputs of the network composition process . . . 93
7.16 Activity diagram associated to a user request 94
7.17 Class diagram of Session and SessionRegistry 95
7.18 Class diagram of the method registry 97
7.19 Steps performed to find the maximum information content ancestor . . 105

8.1 Graphs of cache performances 115
8.2 Percentage of time saved with annotation filtration 120
8.3 Percentage of time saved with a smaller sparsification threshold . . . 121

9.1 Bar diagram of the annotated gene set frequencies 124
9.2 ROC curves at different ranges of feature selection 127
9.3 ROC curve for GO branches comparisons 130
9.4 ROC curves for GO BP predictions from single networks 133
9.5 ROC curves for GO BP predictions from composite networks 134

A.1 Class diagram of the package cnw. 148
A.2 Class diagram of the package inputTypes. 149
A.3 Class diagram of the package Configuration. 151
A.4 Class diagram of the package DescriptionCreator 153
A.5 Class diagram of the package methodDescriptor 154
A.6 Class diagram of the retriever package. 155
A.7 Class diagram of the main classes of the native code subsystem . . . 156

List of Tables

8.1 Task performances for network from node associations 112
8.2 Task performances for network from flat annotations 113
8.3 Task performances for network from ontological annotations 114
8.4 Table of the number of nodes and edges of the tested networks 116
8.5 Table of performances without annotation filtration 118
8.6 Table of performances with annotation filtration 119
8.7 Table of performances with less sparsified networks 120
8.8 Table of performances with different filtration and sparsification setting 121

9.1 Data dimensions for the Gallus gallus species 124
9.2 AUC results for different pruning configurations 126
9.3 AUC results for different sparsity level 127
9.4 AUC results for GO annotation predictions in Gallus gallus 128
9.5 Table of annotations for the Arabidopsis thaliana organism 131
9.6 AUC results for GO BP annotation predictions Arabidopsis thaliana - single networks 132
9.7 AUC results for GO BP annotation predictions Arabidopsis thaliana - composite networks 134

Chapter 3
INTRODUCTION

This section wants to provide a brief overview of the context in which this thesis is collocated. The main definitions, necessary for a correct understanding of this script, as well as descriptions of data types involved in the bioinformatics domain, and their related issues are provided.

3.1 Genes and Proteins

DNA (deoxyribonucleic acid) is a macromolecule which is present in each cell of almost all known living organisms. Its function is to code the instructions needed for cells development and functioning. The DNA structure is composed by two strands which run in opposite directions to each other; as discovered by James Watson and Francis Crick in the 1953, the two strands are arranged in two helical chains each coiled round the same axis (figure 3.1).

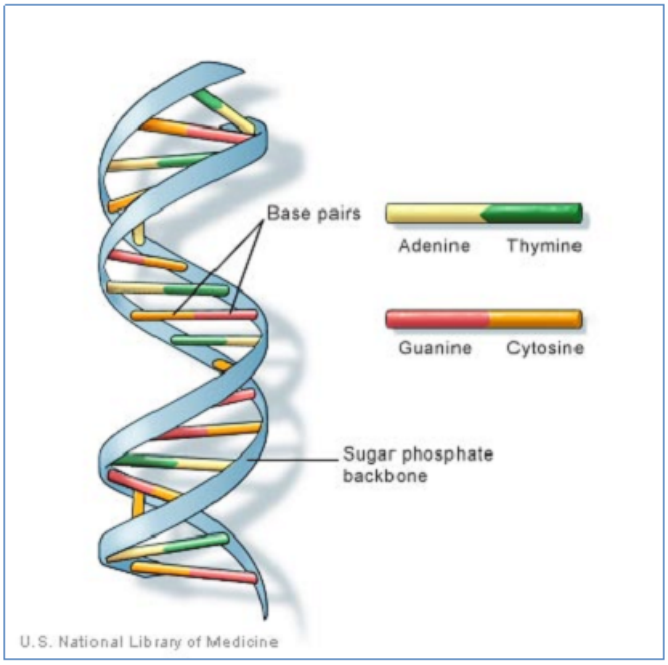


Figure 3.1 – DNA double helical structure. (Figure taken from http://ghr.nlm.nih.gov/handbook/basics/dna)

Each chain is composed by a sequence of simple units called nucleotides. Segments of these nucleotide chains are responsible for carrying the genetic informa-

tion, they are named genes. Genes represent just a little part of the whole DNA in a cell, for example genes constitutes just the 1,5% of the human DNA whereas the remaining part has still an unknown function.

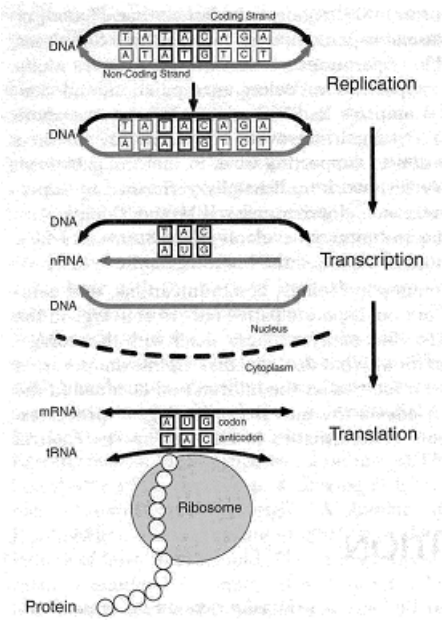


Figure 3.2 – Transcription and translation processes from DNA to proteins. During the transcription phase, DNA is transcripted into mRNA which then leaves the cell nucleus towards cytoplasm. Here ribosomes drive the translation phase producing proteins. (Figure taken from [1])

Genetic information are encoded in the DNA through its structure, which is a long sequence of nucleotides, each one appearing in one of the four forms: (A)Adenine, (T)Thymine, (G)Guanine, (C)Cytosine. The ”instructions” coded in genes are used by cells for protein synthesis during the transcription and translation processes (figure 3.2). At first each gene is transcribed into a macromolecule, called RNA (Ribonucleic acid), that is a nucleic acid as well; however its structure is composed by a single strand and the Thymine nucleobase is replaced by (U)Uracil. RNA is then used to transport the coding information outside the nucleus of the cell towards the cytoplasm. Here is where ribosomes translate RNA into a chain of amino acids. There exist different types of RNA, the one which is in charge of the genetic information transportation outside the nucleus of the cell is called mRNA. While in prokaryotes mRNA is directly derived from DNA, in eukaryotes the transcript needs to be processed to produce a mature mRNA, this is because eukaryotes genes have both coding (exons) and non-coding parts (introns). In order to produce mature mRNA the introns are removed and the exons are joined together, this process is called splicing (figure 3.3). The splicing process is often non deterministic and different mRNAs can be generated from the same pre-mature RNA (alternative splicing). The genetic information that are coded in the DNA are read (translated) by

means of the genetic code which associates RNA triples to amino acids; each triple of nucleobase codes forms an amino acid, and sequences of amino acids form proteins. Since there are four different possible nucleobases in the DNA, the number of different generable triples is 64. However, the number of known amino acids is 20, hence the mapping between triples and amino acids is not univocal. The genetic code express such mappings between nucleotide triples and amino acids (figure 3.4).

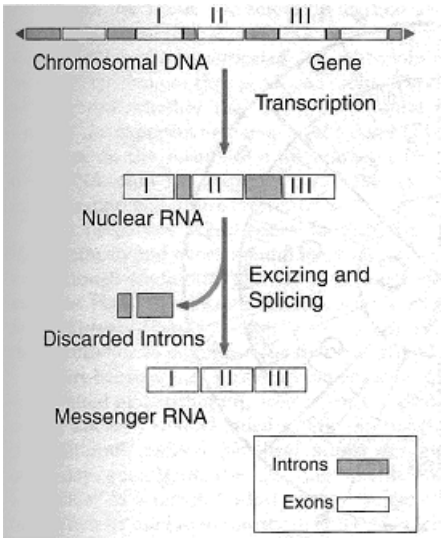


Figure 3.3 – Transcription from nuclear RNA to messenger RNA (mRNA). In this phase introns are discarded while exons are joint together. (Figure taken from [1])

nonpolar polar basic acidic (stop codon)

1st base	2nd base								3rd base
	U		C		A		G		
U	UUU	(Phe/F) Phenylalanine	UCU	(Ser/S) Serine	UAU	(Tyr/Y) Tyrosine	UGU	(Cys/C) Cysteine	U
	UUC		UCC		UAC		UGC		C
	UUA	(Leu/L) Leucine	UCA		UAA	Stop (*Ochre*)	UGA	Stop (*Opal*)	A
	UUG		UCG		UAG	Stop (*Amber*)	UGG	(Trp/W) Tryptophan	G
C	CUU		CCU	(Pro/P) Proline	CAU	(His/H) Histidine	CGU	(Arg/R) Arginine	U
	CUC		CCC		CAC		CGC		C
	CUA		CCA		CAA	(Gln/Q) Glutamine	CGA		A
	CUG		CCG		CAG		CGG		G
A	AUU	(Ile/I) Isoleucine	ACU	(Thr/T) Threonine	AAU	(Asn/N) Asparagine	AGU	(Ser/S) Serine	U
	AUC		ACC		AAC		AGC		C
	AUA		ACA		AAA	(Lys/K) Lysine	AGA	(Arg/R) Arginine	A
	AUG[A]	(Met/M) Methionine	ACG		AAG		AGG		G
G	GUU	(Val/V) Valine	GCU	(Ala/A) Alanine	GAU	(Asp/D) Aspartic acid	GGU	(Gly/G) Glycine	U
	GUC		GCC		GAC		GGC		C
	GUA		GCA		GAA	(Glu/E) Glutamic acid	GGA		A
	GUG		GCG		GAG		GGG		G

Figure 3.4 – Standard genetic code.(Figure taken from [2])

The other fundamental entities componing the cell are proteins. They are the base elements of every biological structure, either plant or animal, and they participate in almost every process within the cell. They may have several functions according to their type: the most common ones are structural or mechanical functions, or else catalysing biochemical reactions. The main characteristic of proteins is their ability to bind other molecules, this allows proteins to carry out the set of functions they are known for. The binding feature is determined by the tertiary structure of proteins which, in turn, determines the presence of binding sites on the protein surface. A protein 3-dimensional structure can be described in four different aspects: the first one (the primary structure) is the amino acid sequence of the protein; the second aspect (secondary structure) is the presence of regularly repeated local structures; the third one (tertiary structure) expresses the overall shape of the protein by defining how secondary structures are combined together; the last aspect (quaternary structure) is the structure composed by several proteins to form a protein complex.

3.2 Genomic and Proteomic Research

Genomics is a discipline in genetics concerned with the study of the genome of organisms. The term genome refers to the entirety of an organism's hereditary information, including both genes and non-coding sequences of DNA/RNA. In particular, genomics focuses on the structures, contents, functions and evolution of the genome. While genomics aims at studying the genome of organisms, proteomics, which is a scientific discipline that may be seen as the step further with respect of genomics, studies the proteome of organisms. More precisely, its aim is to characterise the structures, functions, activities and quantities of proteins in a species. Differently from the genome, the proteome is dynamic in time and its composition can vary due to external factors and/or kinds of cell or tissue. On the other hand, genome is nearly a constant entity; the interaction between genome and the environmental conditions, in addition to a particular phase of the cell life cycle and a specific organism cell or tissue, determines the proteomic expression in a given state. Due to the dynamic nature of proteome, proteomics is a discipline more complex than genomics.

The previously discussed process of protein biosynthesis is not error free, in fact the information coded by genes may be altered by external problems (i.e. adverse environmental conditions, such as electromagnetic radiation with high intensity, excessive heat, etc.) and/or internal problems (e.g. unsolicited mutation, noise over the information transmission, etc.). These variations may cause problems to cells, diseases and, in the worst hypothesis, they can lead to the cell's death. The proteomics and genomics research areas are concerned with the molecular investigation of transcripts and proteins in order to gain a better understanding of biomolecular

mechanisms and interactions occurring inside a cell, and to provide the scientific community with new models describing the complex biology system of cells. However, these field of studies are strongly multi disciplinary and involves disciplines such as biology, medicine, chemistry, electronics, statistics and informatics. In particular, the last one targets the realization of tools and methods for the storage, management, analysis and visualization of a huge, every-day augmenting volume of data that are continuously generated by high-throughput experiments.

3.3 Biomedical-Molecular Terminologies and Ontologies

In the continuation of the thesis, the terms protein and gene will be often replaced by the term biomolecular entity due to the fact that the topics that will be exposed hold for both terms.

A controlled vocabulary provides a methodology to neatly organize knowledge in a way that it is easily readable and unambiguous, so that it can be correctly parsed by machines in a short time. A controlled vocabulary is composed by a set of terms, which are selected and authorized by the designer itself. This is diverse from the natural language, where the used vocabulary might allow for synonyms and polysemes; these factors, which enhance the information ambiguity, are totally or nearly absent in a controlled vocabulary. Each term in a controlled vocabulary usually represents a particular concept or characteristic, and it is univocally identified by an alphanumeric code. An example of controlled vocabulary is the Medical Subject Heading (MeSH) [3], its objective is indexing magazine articles and books to foster the categorisation and the research of scientific documentation within the medical domain. In the context of genomic annotations, there is a large number of controlled vocabularies which can be divided into two main categories:

- Flat controlled vocabularies: terms in these vocabularies are not linked to each other through relations;
- Structured controlled vocabularies or ontologies: terms in these vocabularies are hierarchically structured through relations, which are further characterized by a specific semantic meaning. The typical structure adopted for ontologies is the Direct Acyclic Graph (DAG)[1].

While in the past the adoption of ontologies was rare, at present the number of tools which are working with ontologies have been increasing exponentially, hence their structures and properties are becoming of fundamental importance. However, there

[1]The word DAG stands for Directed Acyclic Graph. DAG is a mathematical structure represented by means of a graph having directed links and no cycles. Hence, for each vertex there exists no path that starts from a node and goes back to the node itself. DAG structures are often exploited to represent and describe ontologies.

are still important controlled vocabularies which are flat; for example, this is the case of OMIM (Online Mendelian Inheritance in Man) [4], which is a vocabulary used to describe genetic pathologies in human organism, or the Reactome [5] vocabulary used for the description of pathways[2]. Among ontological controlled vocabularies, it is possible to cite ChEBI (Chemical Entities of Biological Interest) [6] focused on chemical components, the already cited MeSH [3], SBO (System Biology Ontology) [7] whose focus is on computational modelling within the biological field, or also GO (Gene Ontology) [8].

Nowadays, an always larger number of organizations are adopting controlled vocabularies to describe biomolecular entity knowledge. The mappings generated between biomolecular entities and specific terms, belonging to a controlled vocabulary, are called annotations. Hence, a certain entity is described by the meaning associated to the term that is annotating the entity itself.

Gene Ontology

The controlled vocabulary GO is part of the Gene Ontology project, which is perhaps the most important and used bioinformatics initiative, on a world wide scale. The final goal of the project is to standardize the way biomolecular entities are described, using methods that are independent of the particular species and/or type of biomolecular entities being described. As a matter of fact, the main outcome of the project is the GO ontology, however, besides that, the project supports also the implementation of tools that use the GO ontology for executing text mining tasks [9] or for clinical applications [10].

The GO ontology is a composition of three different ontological vocabularies, each one targeting a particular biological aspect of the cell; they are:

- Biological Process (BP): the biological process to which the biomolecular entity contributes (about 17000 terms);
- Molecular Function (MF): the biochemical function of the biomolecular entity (about 8600 terms);
- Cellular Component (CC): the location of the biomolecular entity in the cell (about 2500 terms).

Each ontology can be represented by means of a DAG structure, the vocabulary terms are the nodes of the DAG, whereas the relations among terms (is_a and part_of relations) are encoded through links connecting two nodes (figure 3.5). All the three ontologies have just one root node each one, and the three root GO categories (MF, BP and CC) have been designed to be independent ontologies.

[2] Pathways are series of chemical reactions occurring within a cell

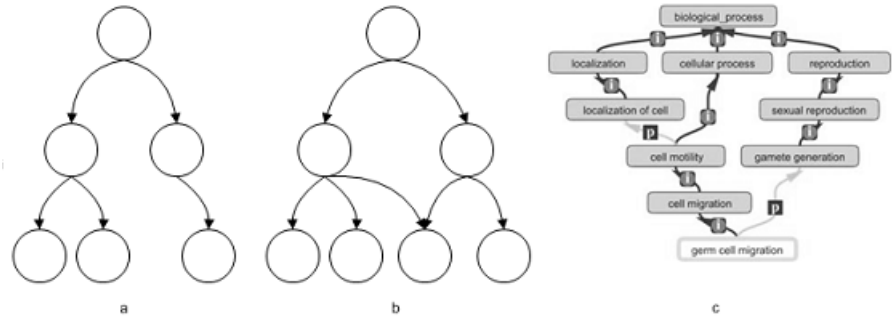


Figure 3.5 – DAG structure and Gene Ontology. a) A simple tree, b) A direct acyclic graph (DAG). The third leave node has two parent nodes c) Fragment of the BP ontology. The term ”gem cell migration” has two parent nodes: it is part_of ”gamete generation” and it is a subtype of ”cell migration”

Besides defining the three ontologies, the GO Consortium freely provides also annotations to the biomolecular entities of forty seven different organisms. The association between a biomolecular entity and a specific vocabulary controlled term is usually performed by experts of the sector, called curators. Such a mapping (annotation) is derived through experimental results and/or literature reviewing. However, there are also cases in which annotations are not suggested by curators, instead they are computed electronically based on other annotations. In the GO ontologies this situation can be distinguished by looking at the Evidence Code associated to annotations; when a certain annotation has been computed electronically its Evidence code is IEA (Inferred from Electronic Annotation). The IEA annotations in GO are the only ones that are not reviewed by curators, as such they might be less accurate. Another important descriptor of an annotation to a GO term is the field Qualifier; in particular, the qualifier add information about the meaning of the association, for example a qualifier with a value equal to NOT implies that the negate of the relation occurring between the term and the biomolecular entity has to be considered. This is specially useful to indicate when an association between a term and a biomolecular entity should be avoided.

It is worthy to note that, depending on the meaning of the relations associated to the DAG structure representing an ontology, an annotation of a biomolecular entity to a specific controlled term can be ”unfolded” according to the DAG. The unfolding process consist in annotating the biomolecular entity also to the ancestors of the given term. On the GO ontologies, this operation is usually performed for links having a semantics of type is_a or part_of.

3.4 Biomolecular Databanks

Before personal computer’s coming, information was stored on physical support such as papers. Data storing was an expensive task due to the difficulties en-

countered in managing and ordering large quantities of data, thus the economic, time and human resources needed for these tasks were considerable. With birth of informatics and its quick evolution, new ways of data management have been proposed and developed. In particular, databases turned out to be an effective and efficient solution to manage data. A database, or databank, is a set of information where data are typically stored and organized in a neat and structured way. The core part of a database, which is responsible for these operations, is its DBMS (database management system). By putting structural and logical constraints upon the information representation, the DBMS allows other applications to efficiently and effectively manage and query data. The recent techniques used in biomolecular experiments have been generating an unprecedented quantity of new genetic data (e.g microarray experiment [11, 12]), therefore one of the most important bioinformatics goals is to find efficient solutions to manage experimental data. Databases are the natural way to face this problem, hence it is not surprising that they cover a central role in bioinformatics. However, the problem is far to be completely resolved, mainly due to further issues that are intrinsic to the scientific research nature of the problem. In fact, many research groups are involved in genomic and proteomic projects, and often each of them is independently carrying out its experiments, obtaining its own results; the scientific community needs to have access to a global view of the knowledge disclosed from single experiments, thus the efforts of bioinformatics are not just seeking methods to manage large dimension of data, but they are also concerned with the devising of new techniques to integrate data from different sites. The final objective is to create simple interfaces, usable by researches, which permit to share all the information among the scientific community. In this scenario, the integration of data from several databases is of fundamental importance, but unfortunately this task is not simple and it is hampered by the lack of standards within the biomedical community. Data are often stored using different technologies, formats and terminologies, hence there is not a simple and common solution to this problem, instead many approaches to tackle these issues have been proposed, each one with its own feedbacks and drawbacks. The project GPDW [13], carried out by our laboratory, is collocated in this respect and it faces the problem by suggesting a data warehouse approach.

There exist several technologies available for databases [14, 15], the most common used technologies in bioinformatics domain are:

- flat file: all data are stored in a unique file. All the records are stored sequentially, separated by special characters; usually records are arranged in tabular like structures, where each record is a row and each column is a property that describes the record;

- relational database: data are conceptually arranged in tables which are linked together through relationships. The DBMS is in charge of resolving those relationships and for the addition, modification and deletion of tuples (records). [14, 15]

Biological databanks contain data coming from a wide spectrum of molecular biology sectors and they can be divided in two groups:

- primary databanks: they store information and annotations about DNA sequences, proteins and their structures. The stored biomolecular sequences, and their respective annotations, should be enough to identify a particular sequence along with its function. A few examples of primary databanks are GenBank [16], EMBL [17] and DDBJ [18];

- secondary databanks: they contain enriched information derived from primary databanks. Examples of information stored in secondary databanks are RNA, pathways, data from microarray experiments [11, 12], pathologies etc. Perhapts, the most known secondary databanks are OMIM [4] and PUBMED [19].

Since there exist many dimensions through which is possible to describe biomolecular entities, specialized content databases have been proposed, such as:

- databanks for scientific literature;

- databanks for taxonomies;

- databanks for nucleotides;

- databanks for genomes;

- databanks for proteins;

- databanks for microarray data.

As showed in figure (3.6), the number of biological databanks has been subject of a tremendous increment over the time.

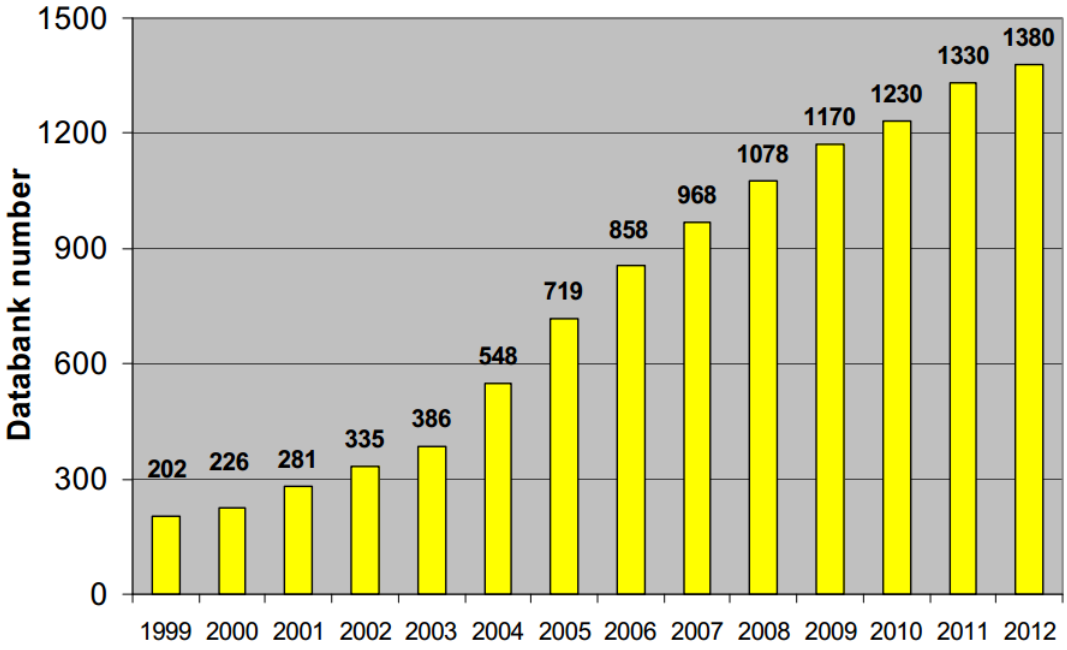


Figure 3.6 – Growth of biological databank number over the time. (Figure taken from [1])

Another distinction among existing databases can be arranged according to the way they are accessed. The plurality of possible database access methods is another layer of complication that must be taken into account when integrating different databanks. Typical way for accessing databanks are:

- Access through web interfaces (HTML or XML pages): information provided is not structured and data are showed by means of heterogeneous interfaces. Customized parser are necessary to retrieve data;
- Access through FTP servers: this method requires to have human and technological resources to re-implementing a local databank;
- Access through web services: this service is made available by just a few databanks; executable queries are limited to the ones offered by services;
- Direct accesses: it is almost never available due to the potential damage to which databanks would be exposed.

Data File Formats

Databanks may provide the same data in different formats and a unique standard to represent and share information related to biomolecular entities, and their annotations, is missing [20]; figure (3.7) shows the most common cases of data file formats. A flat format is a text file containing data values in which the structure of data is defined by rows and special characters. Usually, within the genomic context, every row of a flat file is associated with a specific biomolecular entity, and the row

meaning is defined by one or more labels, sometimes having hierarchical relationships with each others. Generally speaking, when data values are organized in rows and columns, a flat file can be seen as a textual file in a tabular-like form where the semantics of data depends on the their position in the table. The existent variations of tabular file are schematically illustrated in figure (3.8).

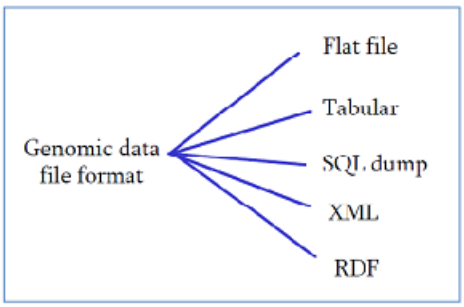


Figure 3.7 – Genomic data file formats. (Figure taken from [20])

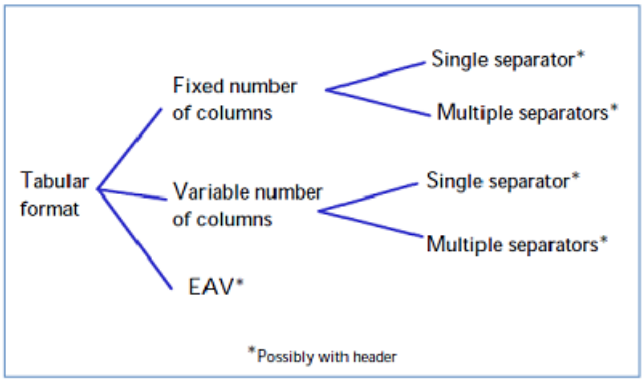


Figure 3.8 – Tabular data file formats. (Figure taken from [20])

Other databanks provide their data by means of SQL dumps. Since dumps are DBMS specific, even importing data with the precedent versions of the right DBMS might be problematic. Examples of databanks which make available SQL dumps of their data are GO (Gene Ontology) [8] and InterPro [21], this latter one is a database that stores information about protein family domains and functional sites.

XML (eXtensible Markup Language), is a metalanguage created and managed by the World Wide Web Consortium (W3C). Its purpose is to define languages for describing semi structured documents but, due to its flexibility, it has been used also for communication of data between heterogeneous sources and destinations. For these reasons many databanks provide their data in a XML format. The structure of a XML document can be described using the language ”Document Type Definition” (DTD) or by means of XML-Schema which, compared to DTD, has the advantage of being a XML derived language itself [14].

Another format, that sometimes is made available by biological databanks, is RDF (Resource Description Framework). RDF is the standard proposed by W3C to describe structured metadata in order to enhance interoperability among applications which communicate through the Web. Using RDF it is possible to specify a RDF model which describes the meaning of the data by defining meta data and relationships among them; a particular RDF model is the RDF Schema, which defines syntax and constructors to specify the structure of the metadata, thus defining the semantics of a RDF model. [22]

3.5 Issues Related to an Effective Use of Available Biomolecular Information

From what it has been said so far, it is possible to summarize the issues that come up when dealing with biomolecular information: First of all, there are a lot of databanks geographically distributed, this fact introduces problems related to the integration of data from heterogeneous databases and the resolution of duplicated and inconsistent data. The work of the last years has been focused on the integration of biomolecular information in a unique central structure, thus easing the updating and management tasks of data, this was also one of the goal of GPDW [13] and other integration systems; another issue is represented by controlled vocabularies. In fact, even tough they claim to be orthogonal to each others, there are relations among terms of one vocabulary with terms of other vocabularies, specially if they are managed by different organizations; finally, there is a problem with the goodness of annotations included in databanks [23] and the difficulties faced by curators in validating, maintaining and managing new annotations. For example, the Go Consortium has annotated just 19920 genes out of the about 25000 present in human beings from the completion of human genome sequencing [24] finished in 2003 by Celera-Genomics and Homo Genome Project (HGP) to nowadays. Furthermore about the 58% of the current annotations are electronically inferred, this means that they have not been confirmed by any curators, so they might have a low accuracy.

3.6 Complex Networks Analysis

The other hot topic related to biomolecular data is the correct interpretation of the data themselves. In fact, the recent explosion of the amount of experimental data, derived from large-scale experiments, has brought new challenges when trying to analyse them. High-throughput techniques in molecular biology generate a huge amount of data which can potentially reflect the interplay between biomolecules on a global scale, however the complexity of the mechanisms governing the cell life and the volume of the available data make hard to characterize roles and relations of biomolecules. What is needed are models, frameworks and techniques which

can accommodate and exploit the richness of information available from the scientific community to discover new knowledge about cells and their members. To this end, network models provide a natural, flexible and expressive formalism to model and analyse the cell environment and its complex interconnections. The representation of complex cellular networks by means of graphs has made it possible to study topological and functional features of the cells using well understood graph theoretical concepts that can be used to predict the structural and dynamical properties of the underlying network. Using graph driven methods for analyzing complex cellular network it is a rather recent field of studies, nevertheless several methods and approaches can be found in the literature. Some works have been done (e.g. [25, 26]) trying to review and organizing the constantly growing amount of proposed approaches, methods and tools, whose common aim is to extract information from biological data using a graph-based framework.

Basically, two major categories of biomolecular networks can be distinguished: networks which reflect the biomolecular physical interconnections that play some role within the cell environment, and the one whose links are logical associations among biomolecular entities (i.e. genes and/or proteins). In both cases the nodes of the networks are typically associated to cellular components, whereas the edges change accordingly to the meaning they express.

The most common biological networks belonging to the first group are:

- Transcriptional gene regulatory networks: they are represented by graphs with two sets of vertices. One set represents transcription factors and the other represents genes. Each edge symbolizes a binding of a transcription factor to a gene;
- Protein-Protein Interaction networks (PPI): they consist of the physical interactions (edges) among an organism's proteins (nodes);
- Metabolic networks: each vertex represents a metabolite and each edge represents a biochemical reaction.

On the other side, the second category of networks is composed by networks whose edges semantics is not constrained to represent physical phenomena occurring within the cell. For example, two genes might be linked together if they are known to have the same function within the cell. Typical instances of this class of networks are:

- Co-Expression networks: they describe expression patterns among genes. Nodes are genes, and two genes linked together with an edge means they are expressed under the same conditions;

- Functional networks: they represent functional associations among nodes (i.e. proteins or genes) by means of edges;
- Disease-Gene networks: they are represented by bipartite graphs with two sets of vertices. One set represents diseases of a given organism, while the other represents the organism's genes. A gene and a disease are connected by an edge if the gene is involved in the disease.

Throughout this thesis particular attention was paid to the class of functional networks, as a matter of fact the tests performed in this project all concern functional networks computed from data derived from the GPDW data warehouse of Politecnico di Milano[13].

Biological Network Reconstruction

There are two main approaches for network reconstruction differing in the underlying data types and interpretation. On the one hand, networks are reconstructed based on genome information, well-curated databases and primary research literature; on the other hand, networks are inferred, usually from gene expression experimental data. With the former approach, semantically controlled extraction of database instances leads to networks based on carefully organized knowledge gathered in many in vitro and in vivo experiments. However, data collected from databases usually lack of tissue and/or disease specific information (they are concerned with a theoretical whole genome stem cell) and information about temporal processes, thus if the analysis focus is constrained to certain tissue specific or temporal aspects, networks reconstructed from databases may present undesired links (e.g. molecules can be connected although they are no simultaneously highly expressed in any tissue). The second approach involves the prediction of interactions starting from a set of experimental data, usually gene expression profiles. This method is more suitable for those analysis which want to study cell aspects under certain conditions and/or dynamical perturbations of the environment. Also in this case, networks might show incorrect interactions due to the nature of the reconstruction mechanism itself, which is based on prediction of interactions.

Several resources, depending on the network interaction types that have to be modelled, can be exploited for network construction. Maps of interactions are already present in the literature (e.g.[27, 28, 29, 30] are maps of protein interaction networks; also a metabolic network reconstruction for S. cerevisiae is available at [31]), these maps are usually concerned about a single organism and a single interaction type (i.e. Protein-protein interactions, gene-gene regulatory interactions or metabolic interactions). The interactions among genes/proteins are collected by integrating the results of several experiments and those data are usually stored in

specialized databases which provide the information necessary to build networks (e.g. DIP [32], BioGRID [33] or STRING [34] for protein-protein interactions, KEGG [35] and BioCyc [36] for metabolic networks, and RegulonDB [37] and EcoCyc [38] for transcriptional regulatory networks).

Besides directly collecting network interactions from data, one might use the data available in the literature and/or in data sources to infer networks. Network inference is the inference of the topology, that is the prediction of the "wiring diagram" of the network. Usually the inference of a biological network structure is done by using the growing sets of high-throughput expression data profiles. However, other promising approaches can be found in the literature, they usually try to infer an a-priori determined type of related links; for instance, focusing on functional networks, the assumption behind the presence of a connection between two network entities is that they share some common cellular functions. These type of edges are often derived by learning from a so called golden standard [39, 40]; in this respective, the Gene Ontology's[8] Biological process branch is probably the most common used source of functional information.

This thesis tries to answer in an efficient and effective way to some of the problems arising in the biomedical context by adopting a network approach to model the data taken from the GPDW data warehouse developed at Politecnico di Milano.

Chapter 4
THEORY OF COMPLEX NETWORKS

Biological systems, neural networks, scientific co-authorship, transportation networks are just a few examples of systems that can be expressed by means of graphs. These systems are characterized by the presence of discrete units and interconnections among them. Representing a system with a graph allows to specify and investigate its global and local properties; in order to describe a system with a graph, one needs to specify its nodes (discrete units) and edges (interconnections) among them. The following sections want to provide basic notions about graph theory and complex system properties, as well as metrics which will be necessary to understand the work discussed throughout this script.

4.1 Graph Definition

The study of networks has been part of a branch of discrete mathematics for decades, this discipline, in the mathematics domain, is called Graph Theory. However this field of studies was confined to the study of small networks which have a formal mathematics description. With an always larger computational power made available by recent progress in discipline such as computer science, electronics and informatics, a new movement of interest and research is appealing the scientific community: Complex Networks. Unlike graph theory, the discipline of complex networks analyse networks whose structure is irregular, complex and evolving in time. Moreover, the networks under investigation often encompass thousands or millions of nodes, thus moving the attention from properties of individual edges or nodes to the large scale statistical properties of graphs. Here, the formal definition of a graph, taken from the graph theory area, is given.[41] Since a complex network can formally be represented as a graph, providing the reader with a mathematical description of what a graph is might be useful to better understand the next chapters.

Basically a graph $G = (\mathscr{N}, \mathscr{L})$ is composed by two sets $\mathscr{N}$ and $\mathscr{L}$, where $\mathscr{N} \neq \emptyset$ and $\mathscr{L}$ is a set of pairs of elements of $\mathscr{N}$. A graph can be directed or undirected, in the former case elements of $\mathscr{L}$ are ordered pairs while in the latter one there is no specific direction in relationships between the two elements of the pair. The elements of $\mathscr{N} = \{n_1, n_2, ..., n_N\}$ are nodes (or vertices) of the graph G, while links (or edges) of G are the elements of $\mathscr{L} = \{l_1, l_2, ..., l_k\}$. The number of elements in $\mathscr{N}$

and $\mathscr{L}$ are indicated with N and K, respectively. In the following sections a graph will be denoted as $G(N,K) = (\mathscr{N}, \mathscr{L})$ or simply $G_{N,K}$. Usually a specific node is denoted with its order i in the set $\mathscr{N}$, hence a natural way to refer to a determined link is to specify the nodes it connects together. $l_{i,j}$ is the link which couples nodes i and j, the link is said to be *incident* in nodes i and j; whereas the two nodes are referred to as the *end-nodes* of link $l_{i,j}$. Two nodes joined by a link are referred as *adjacent* or *neighboring*. Figure (4.1)

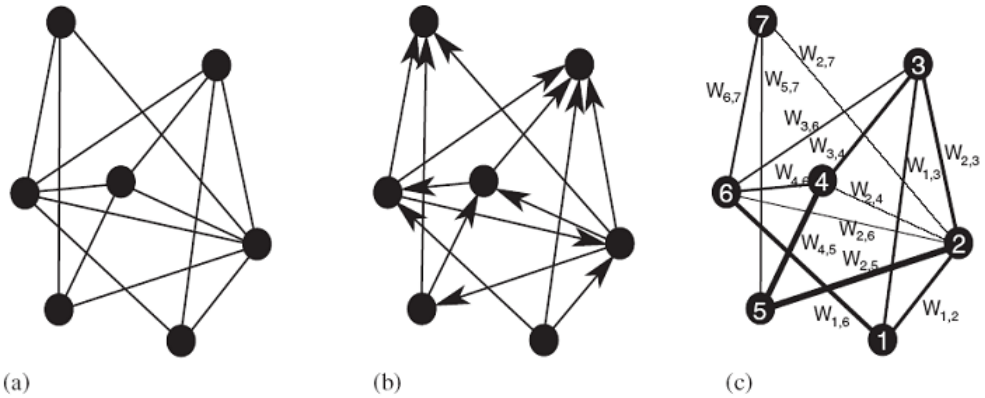


Figure 4.1 – Graphical representation of an (a) undirected, (b) directed, (c) weighted undirected graph. (figure taken from [41]).

shows the usual representations of graphs. As it is possible to observe from the figure (4.1), links in directed graphs are represented with arrows which draw the direction of the relationship between two nodes. Labels $W_{i,j}$ of graph (c) represent weights of links joining node i and j; weights are graphically denoted with the thickness of edges. Weighted networks provide a model to describe the strength of connection; in many real network examples this is a desirable feature, for instance it is possible to represent unequal traffic on the Internet [42], strong and weak ties between individuals in social networks [43, 44] or different capabilities of transmitting electric signals in neural networks [45] and many other scenarios. The graph definition exposed above can be further enriched to encompass domain specific detailed. For instance there may be the necessity to separate nodes and/or edges according to their characteristics, in other words, sometimes we would like to represent different types of nodes and edges. This could be the case of a social network of people, the vertices may represent men or women, people of different nationalities, locations, ages and so on. Edges may represent friendship, animosity, professional acquaintance and many others. A special case of graphs formed by two different types of nodes are *bipartite networks* (figure 4.2)

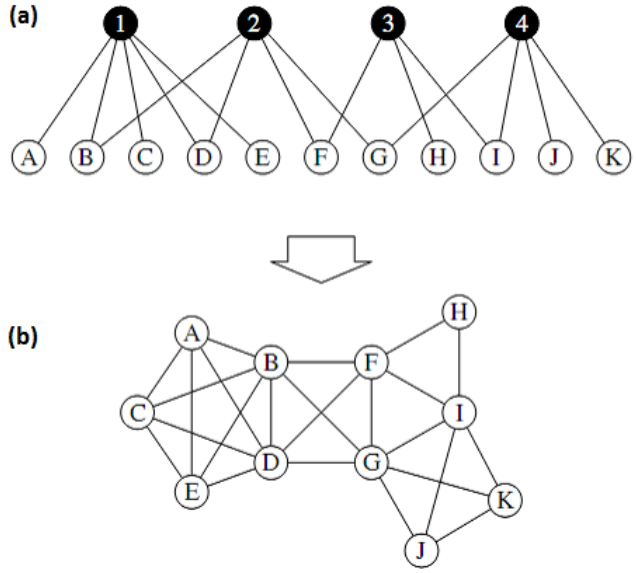


Figure 4.2 – (a) is an example of bipartite network, nodes denoted with letter belong to the same class, whereas nodes referred with numbers belongs to another one. (b) represents the projection of the bipartite network onto the class of nodes indicated by letters (a). (figure taken from [46]).

. A bipartite graph $B = (\mathscr{V}, \mathscr{E})$ is composed by two sets of nodes $\mathscr{V}_1$, $\mathscr{V}_2$ such that $\mathscr{V}_1 \cap \mathscr{V}_2 = \emptyset$ and one set of links $\mathscr{E}$ where if the link $e_{i,j} \in \mathscr{E}$ then $i \in \mathscr{V}_1$and $j \in \mathscr{V}_2$, or the reverse. Hence in a bipartite networks links can join just nodes belonging to different classes. Examples of systems where this formalism could be useful are the real world associations papers/authors, boards/directors or actors/movies. From a bipartite network is possible to obtain an ordinary graph by projection (figure 4.2), in the projected network the link $l_{i,j}$ will exists iff the the two nodes i and j have a common neighbor in the bipartite network, so if exist a node z such that $e_{i,z}$ and $e_{j,z}$ are links of the bipartite graph. Finally, another aspect that complicates the formalism used to describe graph, but at the same time allows for more powerful descriptions of systems, is the time. In fact many system are not static and their component may evolve over time, hence vertices or edges may appear or disappear, as well as values defined on those vertices and edges may change.

Graph Notations

The maximum number of edges for a graph G with N nodes is $N(N-1)/2$ (N(N-1) for directed graph), when this is the case the number of edges $K = \vartheta(N^2)$ and the graph is said to be *dense,* whereas when $K \ll \vartheta(N^2)$ the graph is called *sparse*. It is often desirable to analyse only portions of a graph, which are properly called *subgraphs*; a subgraph $G' = (\mathscr{N}', \mathscr{L}')$ of a graph $G = (\mathscr{N}, \mathscr{L})$ is a graph such that $\mathscr{N}' \subseteq \mathscr{N}$ and $\mathscr{L}' \subseteq \mathscr{L}$. If G' contains all links of G that join two nodes in $\mathscr{N}'$, then

G' is said to be the *subgraph induced* by $\mathcal{N}'$. Other terms that are recurrently used in graph theory refer to the concept of reachability among nodes; two nodes that are not adjacent may be nevertheless reachable from one to the other. A *walk* from node i to j is defined as a sequence of adjacent nodes that starts with i and ends with j. A walk is often associated with a number which specified the walk length and it is computed as the number of edges in the sequence. Other three notations are derived from the concept of walk: the first one is *trail*, which is a walk without edge repetition; the second one is *path* that is a walk in which no node is visited more than once; the last one is *cycle* which is a closed walk, of at least three nodes, in which no edge is repeated. One property that is often analysed in a graph is the *shortest path* or, also called, *geodesic,* that is the walk of minimal length between two nodes. Finally, a graph is defined as *connected* if for every pair of distinct nodes there exists at least one path between the two nodes; if a graph is not connected then it is said do be *unconnected* or *disconnected*. A *component* of the graph is a maximally connected induced subgraph, where maximally means that it is impossible to find a connected induced subgraph which includes all the nodes of the component; a *giant component* is a component whose size is of the same order of N.

Adjacency Matrix

A common representation of a graph is the *adjacency matrix*. Given a graph $G_{N,K}$ its adjacency matrix is a square matrix $\mathscr{A}_{NxN}$ whose entry $a_{i,j}$ is equal to 1 when the link $l_{i,j}$ exists, and zero otherwise. A *weights matrix* may also be defined for weighted networks, in this case the entry $w_{i,j}$ will take the value of the weight corresponding to the link $l_{i,j}$. Further possible representation will be discussed later on in the chapter "Materials and Methods"

4.2 Networks Properties

Since the beginning of interests and researches in complex networks, many networks have been analysed and compared. Perhaps the first observation that came out from these comparative efforts is that real world networks are non random in some features. A first issue that research had to face was how to compare networks. For this reason a serie of new concepts and measures were proposed to characterize real network topologies. Global measures such as degree distribution, diameter, characteristic path length, clustering coefficient and modularity have been also widely discussed in the context of cellular networks [47, 48]. This section presents the unifying principles and the statistical properties common to most real networks, maintaining a special attention towards biomolecular data applications; the reader interested in a more detailed discussion of these properties could consult

[41, 49, 46].

Node Degree

The degree k_i of a node i is the number of links incident with the node:

$$k_i = \sum_{j \in \mathcal{N}} a_{i,j} \tag{4.1}$$

where $a_{i,j}$ is the term of the adjacency matrix in position (i, j). The degree node must be divided in two distinct measures in case of directed networks, these two measures are called *out-degree* k_i^{out} of the node and *in-degree* k_i^{in} of the node and they count just the outgoing and ingoing edges of the node, respectively. The total degree k is obtained by summing up in-degree and out-degree. In a weighted graph is possible to re-define the concept of node degree taking into account weights of the links incident to the node. In this case the metric adopted is called *strength* s_i of the node and it is computed by summing up the weights of incident edges:

$$s_i = \sum_{j \in \mathcal{N}} w_{i,j} \tag{4.2}$$

Degree Distribution

The degree distribution $P(k)$ is the probability that a node chosen uniformly at random has degree k. It is the most basic topological characterization of a graph. As it was for the degree node in directed networks, the degree distribution is also split in two distributions $P(k^{in})$ and $P(k^{out})$. Degree distributions are often plotted (figure 4.3)

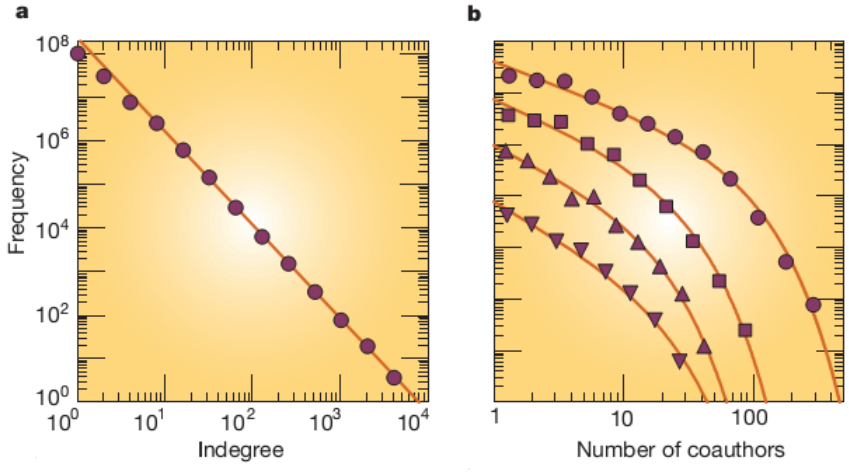


Figure 4.3 – Degree distributions for real networks. a)World-Wide Web. Nodes are web pages; links are directed URL hyperlinks from one page to another. The log–log plot shows the number of web pages that have a given in-degree (number of incoming links). b) Co-authorship networks. Nodes represent scientists; an undirected link between two people means that they have written a paper together. The data are for the years 1995–1999 inclusive and come from three databases: arxiv.org (preprints mainly in theoretical physics), SPIRES (preprints in high-energy physics) and NCSTRL (preprints in computer science). Symbols denote different scientific communities: astrophysics (circles), condensed-matter physics (squares), high-energy physics (upright triangles) and computer science (inverted triangles). (Image taken from [50])

to gain an idea on how the degrees are distributed among the nodes. Without plotting the distribution it is still possible to summarize its characteristics by using the n-moment of $P(k)$:

$$< k^n >= \sum_k k^n P(k) \tag{4.3}$$

The 1-moment of $P(k)$ is the mean degree of G.

In a weighted network, given a node i, two different situations can occur concerning its weights: all weights $w_{i,j}$ can be of the same order $\frac{s_i}{k_i}$, or only a few weights dominates over the others. This disparity on weights of a node i can be evaluated by the measure Y_i [51]:

$$Y_i = \sum_{j \in \mathcal{N}_i} (\frac{w_{i,j}}{s_i})^2 \tag{4.4}$$

where $\mathcal{N}_i$ is the set of first neighbor of node i. Similarly to the degree distribution for degree nodes the *strength distribution* $R(s)$ defines the probability that a vertex has strength s.

Shortest Path Lengths

Shortest path is another basic measure used to characterize network topologies. This feature plays an important role in those networks where communication and

transport problems are important. To summarize the length paths of a network an *average shortest path length* L, also known as *characteristic path length*, is often calculated:

$$L = \frac{1}{N(N-1)} \sum_{i,j \in \mathcal{N}, i \neq j} d_{i,j} \tag{4.5}$$

where $d_{i,j}$ denotes the geodesic distance between node i and node j. The problem of 4.5 is that it diverges if two nodes are not connected in any way ($d_{i,j} \to \infty$). To avoid this circumstances one can limit the summation in formula 4.5 to the nodes belonging to the largest component, another alternative is to consider the harmonic mean of geodesic lengths, calculating the *efficiency* of G:

$$E = \frac{1}{N(N-1)} \sum_{i,j \in \mathcal{N}, i \neq j} \frac{1}{d_{i,j}} \tag{4.6}$$

When using the efficiency, instead of the characteristic path length, the nodes that are not connected contribute with zero value in the summation. The efficiency is an indicator of the traffic capacity of the network. In case of weighted networks, lengths can be associated to each edge (the edge length in unweighted networks is assumed as equal to 1), for instance one could define the length of edge which goes from i to j as $l_{i,j} = \frac{1}{wi,j}$. Having length associated to edges the definition of geodesic distances changes for taking into account this fact, the quantity $d_{i,j}$, called *weighted shortest path*, is now defined as the the smallest sum of edge lengths throughout all the possible paths in the graph from i to j. A diameter of a network is defined as the maximum value over every possible geodesic distance/weighted shortest path of the network.

By observing at the diameters and average path lenghts of biomolecular networks [52, 53, 54], it has been observed that these two values are often 'small' in comparison to the network size. This property is known as small-world property and is not peculiar of only biomolecular networks its usual interpretation in this context is that only a small number of intermediate reactions are necessary for any one protein/gene/metabolite to interact with another one.

Clustering

Clustering is a measure used to compare the *transitivity* T of a network, where the transitivity refers to the relative number of transitive triples:

$$T = \frac{3 \times \#ofTrianglesInG}{\#ofConnectedTriplesOfVericesInG} \tag{4.7}$$

This is the ratio of triangles in the network over the whole number of triads. The factor 3 in formula 4.7 accounts for the fact that a triangle can be seen as three

different triads. Another metric proposed to evaluate the transitivity property of a network is the *clustering coefficient* C. Differently from 4.7 C is calculated starting from a local property of the graph: the local clustering coefficient of a node c_i (figure 4.4)

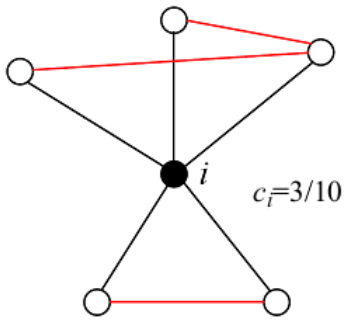


Figure 4.4 – Example of local clustering coefficient of node i in an undirected, unweighted network. (figure taken from [46])

. This quantity is defined as:

$$c_i = \frac{2e_i}{k_i(k_i-1)} = \frac{\sum_{j,m} a_{i,j}a_{j,m}a_{m,i}}{k_i(k_i-1)} \tag{4.8}$$

The local clustering coefficient states how likely two nodes j,m that share a common neighbor i are connected to each other. In 4.8 e_i refers to the number of edges which are present in the subgraph induced by the neighbors of i. The clustering coefficient of the whole graph is then given by the mean of all local clustering coefficients:

$$C = \frac{1}{N}\sum_{i\in\mathcal{N}} c_i \tag{4.9}$$

It is possible to investigate the variation of local clustering coefficients according to the node degrees by defining the quantity $c(k)$ as the average of local clustering coefficients of nodes having degree equal to k. Besides the clustering coefficients discussed above many other metrics have been proposed over the years, Barrat et al. [55] proposed a cluster coefficient that takes into account weights of a weighted network:

$$c_i^W = \frac{1}{s_i(k_i-1)}\sum_{j,m}\frac{(w_{i,j}+w_{i,m})}{2}a_{i,j}a_{j,m}a_{m,i} \tag{4.10}$$

The weighted clustering coefficient accounts for the number of closed triangles in the neighborhood of node i as well as their relative weight with respect to the vertex strength.

One example of a clustering analysis study is the work of Ravasz et al. [56]; in this work the average clustering coefficients C for the metabolic networks of 43 organisms were compared with the average clustering coefficient of a random network with the same degree distribution. In all the 43 cases the average clustering coefficient of the metabolic network was at least one order of magnitude higher than the one of the random network implying that in metabolic networks edges tend to be clustered together.

Communities

Many complex networks can be characterized by their attitude to be divided into communities. The intuitive behind the idea of community concept is that nodes tend to be more connected within the community they belong to. There is not a clear definition of community in the literature but the two features that are used the most to identify communities are: 1) groups of vertices that have a high density of interconnections within them, with a lower density of edges among different groups (figure 4.5)

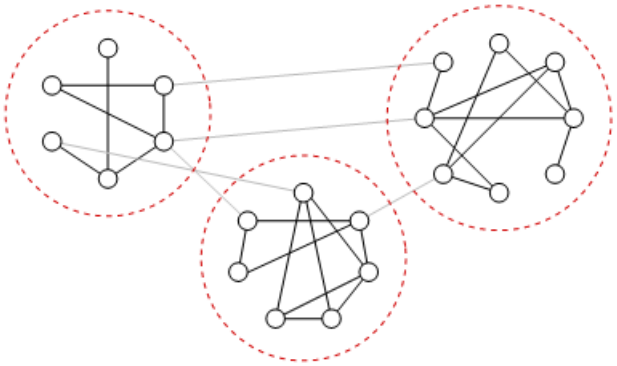


Figure 4.5 – Three communities with high density of intra-community connections and low density of inter-community connections. (figure taken from [41])

; or 2) groups of nodes which are fully, or nearly fully, connected within the same groups. The second feature brings to the definition of *cliques, k-plex* and *n-clique*. A clique is a maximal subgraph of G in which each node of the clique is connected to every other node belonging to the same clique. The definition of clique can be weakened requiring that the maximum number of missing links for each node, with respect to a maximal complete subgraph (clique), is k, in this case the maximal subgraph obtained takes the name of k-plex. A different definition for characterizing communities, which is derived from the clique definition as well, is the n-clique. A n-clique adopts the same definition of a clique with the exception that the adjacency of nodes is substituted with a reachability requirement of at most n. More details on

community structures, definitions and algorithms to deal with them are provided in the next sections.

Graph Spectra

The set of eigenvalues of a graph adjacency matrix $\mathscr{A}$, the so called *graph spectra*, is another mechanism to characterize graph. A graph $G_{N,K}$ has N eigenvalues and N corresponding eigenvectors. The graph spectra present some features depending on the topology of the graph; undirected graphs, without loops, have real eigenvalues and the corresponding eigenvectors are orthogonal to each other, whereas imaginary eigenvalues can arise from directed graphs. From the Perron-Frobenius derives the fact that directed and undirected graphs have a real eigenvalue μ_N which is associated to a real non negative eigenvector, in addition the absolute value of all the other eigenvalues is less or equal than μ_N. If G is connected then μ_N has multiplicity 1 and $|\mu| < \mu_N$ where $\mu$is a generic eigenvalue of $\mathscr{A}$ different from μ_N . The value of μ_N decreases with edges and nodes removing, in a connected undirected graph this means that μ_N is not degenerate and every component of the corresponding eigenvector is non-negative, since the other eigenvectors are orthogonal to it their components are mixed in signs. Eigenvalues and eigenvectors of an adjacency matrix have properties strictly related to network topology characteristics, such as diameter, connectivity of graph and cycles.

4.2.1 Topology of Real Networks

Nowadays, the availability of large databases and high computing power has permitted to construct large real network and to analyze their properties. Many real systems can be modeled by using complex networks, this in turn, allows to investigate topological and dynamical features of systems. Examples of systems that have been modeled by using complex networks, thus identifying edges and nodes for the systems, are numerous. The World Wide Web (WWW) is a network of websites (figure 4.6), the brain is a network of neurons, an organization is a network of people and also food webs and metabolic pathways can all be represented by networks (figure 4.7)[57]; a first stage in complex networks researches was to analyse, compare and characterize large real network topologies. This process led to interesting discoveries, finding that most of real network topologies share common features which are not present in regular graphs (those that had been studied by mathematician in graph theory), and they are not even in completely randomized graphs.[41] The two main topological characteristics that have been found in real networks are the so called *small-world* and *scale-free* properties.

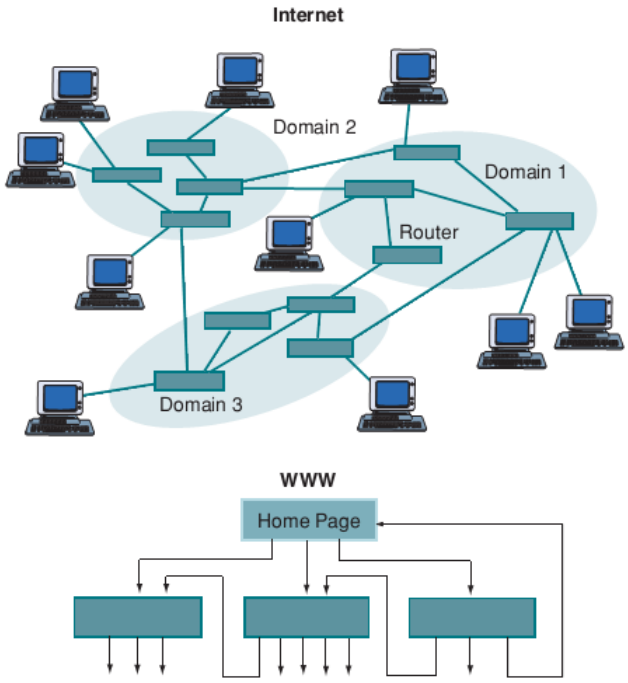


Figure 4.6 – Network structures of the Internet and the WWW. On the Internet, nodes are routers (or domains) connected by physical links. The nodes of the WWW are web pages connected by directed hyperlinks. (figure taken from [57])

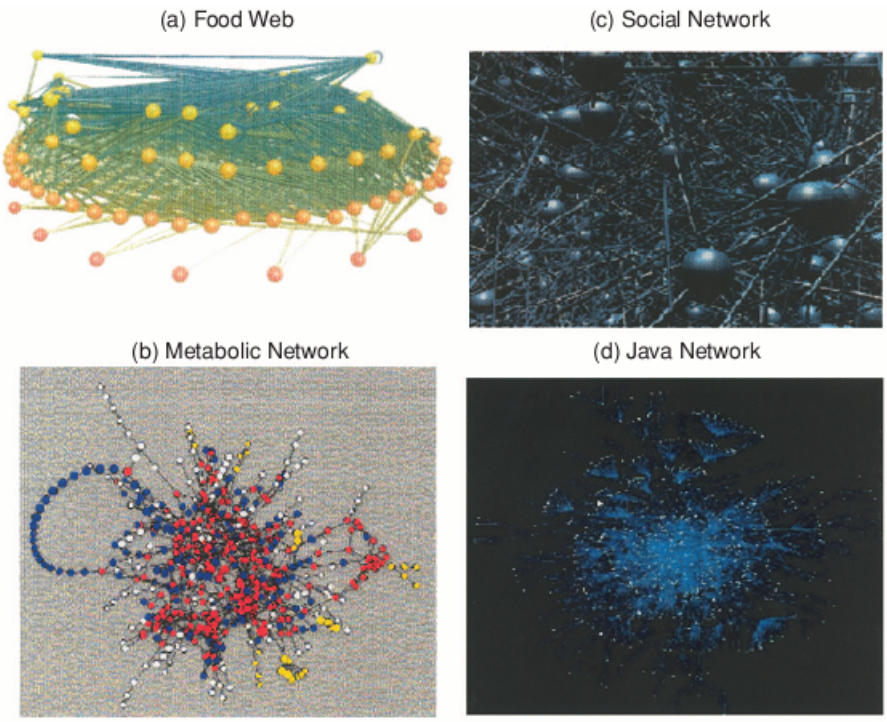


Figure 4.7 – (a) Food web of the Little Rock Lake shows "who eats whom" in the lake. The nodes are functionally distinct "trophic species". (b) The metabolic network of the yeast cell is built up of nodes - the substrates that are connected to one another through links, which are actual metabolic reactions. (c) A social network that visualizes the relationship among different groups of people in Canberra, Australia. (d) The software architecture of the Java Development kit 1.2. The nodes represents different classes and a link is set if there is some relationship (use, inheritance, or composition) between two corresponding classes. (figure taken from [57])

Small-World Property

The small-world property of a network refers to the fact that there is a relatively short path between any two nodes of the network. Its characterization is given by an average shortest path length L (see Eq. 4.5) that depends at most logarithmically on the network size N.[41] Many real networks have been shown to have the small-world property, as well as an high value of the clustering coefficient (see Eq. 4.9). This fact is at the basis of the proposal made by Watts and Strogatz [58], they define a small-world network as a network having a small value for the average shortest path and high value for the clustering coefficient. In addition to define the small-world property, the authors also proposed a network model which generates network with such characteristics.

Scale-Free Property

Another features that a lot of large-scale complex networks have in common is the scale-free property. This property refers to the fact that a network display power law shaped degree distribution $P(k) \sim Ak^{-\gamma}$. The name scale-free is due to the fact

that power-laws have the property of having the same functional form at all scales. Hence a change of scale of the independent variable x will result in a function with the same shape of $f(x)$, apart from a multiplicative factor, where $f(x)$ is the original power-law function (i.e $f(\alpha x) = \beta f(x)$). For large-scale complex networks, the typical range of values that the exponent of the power-law shaped degree distribution takes is $2 < \gamma < 3$. A direct consequence of the degree distribution shape in scale-free networks is the presence of a few nodes (the *hubs*) connected to many other nodes, and a large number of poorly connected elements. The most famous model of these type of networks is the one proposed by Barabási and Albert in the paper [59]. This model (BA model) is said to be dynamic due to the fact that the number of nodes is not fixed, in fact it is based on an attachment rule called preferential attachment rule. Following this rule, the new node to be added is connected to other nodes by following a probability distribution which is not homogenous, instead the new node is more likely to be connected with higher degree nodes.

Several types of biological networks, such as PPI, regulatory and metabolic networks in different organisms are found to have scale-free topologies. Yook et al. [54] studied the protein interaction network of S. cerevisiae using data from four different sources, in all four cases the degree distribution appeared to be broad tailed and to be best described by some form of modified power law. References [53] and [52] provide the first reports on the large-scale organization of metabolic networks, showing its scale-free nature. General properties of scale-free networks suggested further investigation of those properties within the biological context, for instance hubs are known to play a key role for the scale-free networks' robustness, removing an hub can disconnect the network; it has been proposed that genes and proteins corresponding to hubs are more important for an organism's survival than of those with lower degree.

4.3 Main Network Investigations

This section presents two of the most common analysis performed with network models. Knowing the structure of a network, it is possible to derive a variety of quantities which can describe peculiar characteristics or pattern for that network. Here the centrality and community structure problems are presented.

4.3.1 Centrality Measures

Centrality researches aim at discovering those nodes/edges that play an important role within the network structure, for this reason they are referred as central entities. When these researches are applied to biomolecular entities, the term centrality is often substituted by the term essentiality. A gene or a protein is said to be essential for

an organism cell if the cell can't survive without it. There exist several interpretations of centrality, each of them introduce its own measure of centrality. In the following subsections, the most famous centrality measures are briefly discussed, for a more detailed explanation of these measures one might want to refer to [60].

Degree Centrality

This centrality measure correspond to the node degree measure 4.1. The intuition behind its application is that nodes with many links could be more relevant within the network structure. For instance, in a network of friendships, those people who have connections to many others, might have more influence or more prestige than those who have fewer connections. In directed networks the degree centrality is usually split in in-degree centrality and out-degree centrality, accounting for incoming and outgoing arrows respectively.

The degree centrality measure for the PPI network in S. cerevisiae was investigated against the essentiality of proteins in the works of Jeong et al. [61] and Yu et al. [62]. These works showed that the degree of essential proteins is higher than the degree of non-essential one, claiming that there exists a possible relation between node degree and essentiality of a protein in a protein interaction network. However, the precise relationship between them is still far to be completely understood.

Eigenvector Centrality

While degree centrality is a rough measure, in the sense that each node neighbor gives a contribution of one to the node centrality, the eigenvector centrality measure tackles this aspect by giving a greater reward to those nodes that are linked to nodes which are themselves important. Hence eigenvector centrality gives each node a score proportional to the sum of the scores of its neighbors, in this way nodes which are connected to few but important nodes gain a score higher than the score they would get using degree centrality. More precisely an eigenvector centrality score x'_i for a node i is defined as:

$$x'_i = \sum_j A_{ij} x_j \tag{4.11}$$

Or written in matrix notation:

$$x' = Ax \tag{4.12}$$

where x is the vector of centrality scores and A is the adjacency matrix. The calculation of vector x is repeated at each time step, enhancing the estimate of the vector itself. After t time steps the vector of centralities is given by:

$$x(t) = A^t x(0) \tag{4.13}$$

Writing $x(0)$ as a linear combination of eigenvectors v_i of the adjacency matrix ($x(0) = \sum_i c_i \mathbf{v}_i$) the formula 4.13 is equivalent to:

$$x(t) = A^t \sum_i c_i v_i = \sum_i c_i k_i^t v_i = k_1^t \sum_i c_i \left[\frac{k_i}{k_1}\right]^t v_i \tag{4.14}$$

where the k_i are the eigenvalues of A and k_1 is the largest of them. Since k_1 is the largest eigenvalues, the vector of centrality scores is dominated by the first term of the summation since the other terms decay exponentially for $t \to \infty$ ($x(t) \to c_1 k_1^t v_1$). These observations lead to the fact that the limiting vector of centralities is proportional to the leading eigenvector of the adjacency matrix therefore the centrality x satisfies:

$$Ax = k_1 x \tag{4.15}$$

In case of directed networks is still possible to calculate the eigenvector centrality but the process is more fragile. First of all, due to the asymmetry of the adjacency matrix in directed networks, the problem of finding the leading eigenvector must be specified for the special case of left eigenvectors or right eigenvectors.

The other issue, that doesn't occur when dealing with eigenvector centrality in undirected network, is that in directed network there might be nodes which have just outgoing (incoming) links. In this case the score of these nodes will be zero, moreover if a node has just incoming (outgoing) links from (to) nodes with a zero centrality score then it will have a zero score too; this behavior is propagated over the entire network. With this definition of centrality the only nodes which may have a non-zero centrality score are those encompassed in a strongly connected component (i.e. a maximal subset of vertices such that there is a directed path in both directions between every pair in the subset) or the out-component nodes, however it is often desirable to have a centrality measure that can assign high values to nodes that are not either in a strongly connected component nor in the out-component set, for instance if a node has a high in-degree.

Katz Centrality

A possible solution to the issue coming with the eigenvector centrality applied to directed network is represented by the Katz centrality. The simple idea behind Katz centrality is to give to each vertex a small amount of centrality regardless the network structure:

$$x_i = \alpha \sum_j x_j A_{ji} + \beta \tag{4.16}$$

Here α and β are simple constants. The first part of 4.16 is the normal eigenvec-

tor centrality, whereas the second term β is an extra score that all nodes receive. In this way nodes that have a zero in-degree will have a non-zero left eigenvector centrality score, at the same time vertex that are pointed to by those nodes will receive some advantages from them.

PageRank

The PageRank centrality measure, which is the trade name given it by Google, is based on the same mechanisms of Katz centrality but in addition it accounts for the fact that a node with high Katz centrality which is linked to many others nodes will give potentially high centrality to each of them. This feature is often considered undesirable as being a node among many others should prevent the node to stand out from the rest. PageRank centrality tackles this situation by diluting the centrality which a node gains adopting Katz centrality. Technically this is achieved by dividing the Katz centrality score of a node by the degree of neighbor nodes that contribute to the score. Hence the score of a node is expressed as:

$$x_i = \alpha \sum_j \frac{x_j}{k_j^{out}} A_{ji} + \beta \tag{4.17}$$

where the new term k_j^{out} is simply the outgoing degree of node j. In matrix terms the PageRank centrality is:

$$x = (I - \alpha A^T D^{-1})^{-1} \beta 1 = D(D - \alpha A^T)^{-1} \beta 1 \tag{4.18}$$

where D is defined as the diagonal matrix with elements $D_{ii} = max(k_i^{out}, 1)$. The term 1 in the definition of the element D_{ii} is given by the fact that the formula 4.17 has an indeterminate value if a node j doesn't have outgoing links. This unwanted behavior is corrected by assigning $k_j^{out} = 1$ to those nodes which don't have any outgoing edge. In principle one could assign any value to those nodes and the final result won't change due to the fact that the nodes respective rows of the adjacency matrix are fulfilled with zero-values.

Hubs and Authorities

Another way to evaluate the centrality of nodes in directed networks is by means of hub and authority. The general idea is to divide the importance of a node in two distinct centrality measures: hub centrality and authority centrality. A node is intended to be a hub if it has information about where useful nodes are, for instance a review page which points to authoritative sources is an important resource to discover informative page. Hence, hub centrality is related to the out-degree of a node and its capacity to point to important nodes. Here it comes the definition of authority centralities, in fact a node hub centrality score is high if the node has outgoing edges to

nodes with high authority centrality. The authority score concerns the in-degree of a node, the larger the number of links which point to that node is the greater is authority score is. As it was for the hub centrality score, the authorities centrality of a node is influenced by the importance of the nodes connected to that one. In this case the higher the hub score of incoming neighbors is the greater the authority score of node is. In mathematical terms it is evident the circular dependence between hubs and authorities scores:

$$x = \alpha A^T y, \quad y = \beta A x \tag{4.19}$$

x and y refer to authority and hub centrality scores respectively. Combining the two expressions:

$$A^T A x = \lambda x, \quad A A^T y = \lambda y \tag{4.20}$$

where $\lambda = (\alpha\beta)^{-1}$. Thus the authority and hub centralities are the $A^T A$ and AA^T eigenvectors with the same eigenvalue λ. It turns out that all the eigenvalues of the two matrices are the same and, similar to what was done for the eigenvector centrality, the eigenvalue which should be taken is the largest one. The proof that all the eigenvalues of the two matrices are the same is straightforward, in fact considering λ an eigenvalue of $A^T A x = \lambda x$ and multiplying both terms by A, the equation results to be $AA^T A x = \lambda A x$; then looking combining this equation with the second one of 4.20 it is easy to see that $y = Ax$ is the eigenvector for AA^T with eigenvalue the same λ.

Betweenness Centrality

Betweenness centrality measures the extent to which a vertex lies on paths between other vertices [60]. To understand what betweenness centrality is let us think at the internet network and packets which flow on it. Supposing that each pair of nodes exchanges a packet with equal probability per unit time and that every packet takes one of the shortest paths between nodes. After a suitably long time the number of packets which have passed through each vertex is proportional to to the number of geodesic paths the vertex lies on. This is number is what is referred to as betweenness centrality. Vertices with high betweenness centrality may be important in the network for their central role in controlling the traffic communication. Moreover, the removal of high betweenness vertices might disrupt communications among other node because of their property to lie on a large number of geodesic paths. The expression in mathematical terms of betwenneess centrality is usually given by:

$$x_i = \sum_{st} \frac{n_{st}^i}{g_{st}} \tag{4.21}$$

where x_i is the betweenness centrality of node i, n_{st}^i is the number of geodesic paths from s to t passing through i and g_{st} is a term that accounts for the fact that there might be more than one geodesic path between two vertices s and t, g_{st} is exactly the number of geodesic paths existing between the two nodes. Betweennes centrality differs from the other centrality measures for the fact that it doesn't provide an indication of how well connected a node is, for instance its degree, the degree of its neighbors or the distance from other vertices whereas it states how much a vertex falls between others (figure 4.8).

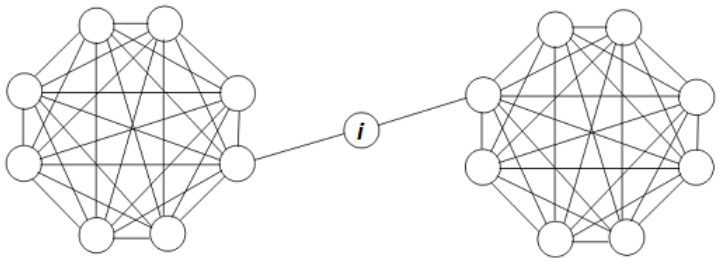


Figure 4.8 – Node i is the only way to pass from vertices of the left full connected group to vertices of the right full connected group. Hence its betweennes centrality is very high even if it is not well connected. (figure taken from [46])

However both measures fails when one would like to consider all the paths passing through two nodes. Another betweenness metric, which does count all paths, is the *random-walk betweenness* defined as:

$$x_i = \sum_{st} n_{st}^i \tag{4.22}$$

where, at this time, the term n_{st}^i refers to the number of time that the random walk from s to t passes through i on its journey, averaged over many repetitions of the walk. This metric includes contributions from all paths due to the fact that the random walk visit every path from s to t with some probability (even though a very small one).

Depending on the context in which this metrics are applied one might seem more appropriate than the others. For instance random-walk betweenness could be the right betweenness centrality measure for network where traffic moves randomly, whereas short paths betweenness centrality should perform better in networks where traffic flows knowing the paths which is taking.

A measure that has been introduced in the previous sections and is related to betweenness centrality is local clustering 4.8. Local clustering can also be thought as a centrality metric which measures control over flows between just the immediate neighbors of a vertex, hence it can be seen as a local version of betweenness centrality. Due to the local nature of the clustering centrality its computation is far less

expensive with respect to the betweenness centrality and, given that the two metrics are strongly correlated, the local clustering is often preferred as a metric measuring the control over information flows.

4.3.2 Community Structure

Communities, also called clusters or modules, are groups of vertices which probably share common properties and/or play similar roles within the graph.[63] The concept of community is a fundamental aspect for many real systems since it offers a way to describe and summarize them by means of coarse grained descriptions. Due to their importance, communities analysis have been applied in many fields; examples of communities can be found in several contexts, for example, we can find a lot of group organizations in society such as families, friendship and working circles, small town, villages etc etc., indeed the term community itself belongs to a social context. Other typical examples of communities can be taken from the biological and internet domains: within a protein protein interaction network communities analysis are often exploited to find group of proteins likely to share the same function; in the the graph of the World Wide Web, where pages (nodes) are linked together depending on the hyperlink structure of the web, communities might reveal group of pages which treat the same topic.

Grouping objects by their similarity is the main application of community analysis, nevertheless a series of related analysis can be conducted starting from the concept of community. For instance, one could study the roles played by nodes within the same community. Nodes that are central to the community, meaning that have many links towards nodes of the same module, might play a central role for the community existence and stability, whereas vertices lying at the boundaries between communities play an important role for the communication between modules. Another analysis which may be applied starting from a set of communities is the transformation of the original graph to a graph whose nodes represent communities, and links among them are drawn if there exist connections among nodes belonging to different groups, and/or overlapping nodes.

Another significant concept, related to the community one, is the hierarchical organization displayed by most networked systems in the real world. Many real systems present a modular organization of their elements where nodes are first grouped into small communities, these small communities are, in turn, combined into larger modules and so on. The hierarchical organization of elements is recurrent in many real systems, for instance the human body is composed by cells, which are grouped into tissues, which form organs, etc. The topic of communities is popular in computer science too. For example, task allocation is a crucial aspect in parallel computing. In this respect, one try to minimize the communication between processors so that the

calculation is speeded up. This can be done by splitting the computer cluster into groups with about the same number of processors in such a way that the number of connections among processors belonging to different groups are minimized.

Community Detection Problem

First of all, it necessary to distinguish two main classes of algorithms based on their final outputs: algorithms which look for partitions of the graph and algorithms which allow for overlapping modules[64]. A partition is a division of a graph such that each vertex belongs to exactly one cluster. Finding overlapping communities is quite a new problem compared to the problem of graph partitioning, nevertheless it has been attracting a lot of attentions due to the fact that many networked real systems present natural divisions of their elements in overlapping modules. For instance, in a social network, where nodes are people and linkages constitute acquaintance relationships, one may find the same person involved in the community of his/her colleagues and also in the community of his/her friends. Another example is taken from the biomolecular domain: in a protein protein interaction network, communities may reveal groups of proteins involved in the same processes. In this respect, it becomes extremely important for an algorithm to allow for overlapping modules since many proteins are involved in more than one process.

A common shared definition of a community structure doesn't exist. In the previous section a community has been described as a group of vertices which probably share common properties and/or play similar roles within the graph. The aim of community detection in graphs is to identify the modules and, possibly, their hierarchical organization, by only using the information encoded in the graph topology [63], so it makes sense to characterize a community as a group of vertices densely connected to each other, while presenting a small number of edges leaving the community towards nodes which are left out of the group. This is the common idea behind the concept of community, however lacking of a formal, well defined definition introduces some degree of arbitrariness resulting in the existence of a plurality of legitimate ways of resolving the community detection problem. A direct consequence of relating the community concept with a notion of density is that communities can be identified just on sparse network. In fact, if the network weren't sparse then the degree distribution would be too homogeneous for communities to make sense. This is not necessarily true for weighted networks, as long as the weight distribution is heterogeneous.

Another requirement that a community is often requested to have is that every pair of nodes inside the same group must be connected by some path lying entirely inside the community. This feature ease the community detection problem on disconnected graph since one can just focus on connected sub-part of the network.

It is worh mentioning that the last few years have seen an increasing interest toward techniques able to detect overlapping structures. Perhaps the first and most known work which addressed this issue is the Clique Percolation Method (CPM) of Palla et al.[64]. CPM is an algorithm based on the assumption that intra-cluster edges are likely to form cliques, whereas inter-community edges have little chance to be involved into cliques. Starting from this assumption, Palla and colleagues' method [64] looks for *k-cliques* (in this context the term k-clique is used to refer to a complete graph with k nodes) and then try to merge k-cliques together into a *k-clique chain*. The condition for merging two k-cliques together is that they have k-1 nodes in common. The authors also introduced the concept of *connected k-cliques*; two k-cliques are connected if exists a k-clique chain which encompass both k-cliques. Finally, a *k-clique community* is a maximal subgraph obtained by the union of a k-clique and all k-cliques connected to it (figure 4.9).The CPM techniques has been also expanded for analysing weighted and directed networks.[65, 66]

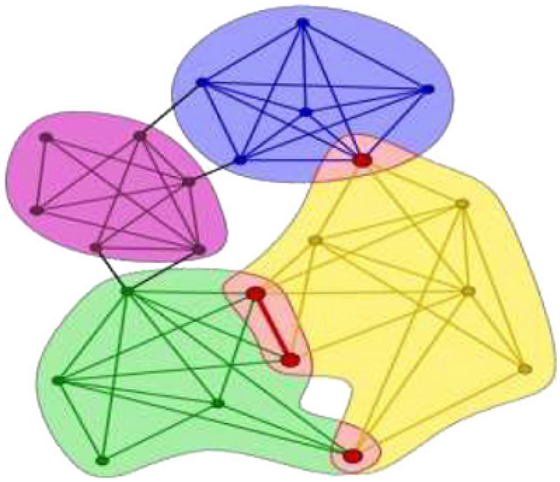


Figure 4.9 – The example shows communities spanned by adjacent 4-cliques. Overlapping vertices are shown by the bigger dots. (figure taken from [64])

Chapter 5
OBJECTIVES

The second chapter of this thesis gave a brief introduction of the field of complex networks, as well as some examples showing real cases where this discipline has been applied to. Even with these small set of cases, that were kept simple enough to ease the discussion of network properties, one can perceive the strong impact that complex network theory and graph analysis may have upon the studying and characterization of systems. And, in fact, this field has been subject of an enormous attention during the last few years, from both the academic world and the business executives. As an example, it is possible to cite the new Facebook's service *"People you may know*" that uses social graph concepts to suggest new friendships. Or also, the more traditional example of PageRank (section 4.3.1), which is a well known algorithm that Google is using for computing the importance of a web-page, based on other pages that are linking to it . The focus on this emerging area is specially emphasized within the bioinformatics community and, while the second chapter of this thesis provided examples of what has been done inside the biomedical context, the graph analysis-related literature is constantly growing.

This thesis finds its place in the field enlightened above. The thesis main goal is to design and implement a general software that allows to easily build large network models, which can be subsequently used for further analysis. In a typical network analysis experiment, unless the data set used is directly derived from another graph investigation problem, the network set-up process is usually long and tedious to carry on. This thesis aims at lightening such a network constructing task. The final outcome of the project wants to be a software architecture that can be used to construct biomolecular networks computed in different ways. The software system to be designed and implemented has the objective of being flexible enough to satisfy different scenarios of usage, depending on the final user needs.

This research work has also the goal of investigating the existent solutions, that can be found within the scientific literature, which may be adopted to construct networks by integrating heterogeneous data. From this research activity, one particular approach will be selected as a reference for the network composition method to be implemented in the software architecture of this thesis; the purpose of the composition method is to aggregate different heterogeneous networks into a unique network model. The major motivation behind composing networks is that, the analysis tech-

niques which can be applied to a composite network[1] exploit the integrated view of the different heterogeneous data sets being composed. In this respect, a part of this thesis will be dedicated to the evaluation of the results provided by a gene function prediction method when applied to composite networks computed from different data sets.

It is worth noting that the software system proposed will be context-independent, and its application can be extended to data not derived from the biological domain.

The general objectives reported above can be decomposed into sub-objectives as follows:

1. Definition of the main actions performed when constructing networks; identification and modelling of the principal categories of data that are used to build networks, with particular attention to the biological case.

2. Definition of the workflow for the network composition task; adaptation and extension of a composing network algorithm, taken from the literature, that will be implemented in the software architecture.

3. Designing of a general infrastructure to build large network models; the major objective of the design phase is to make the infrastructure as much extensible and customizable as possible. In particular, the designed architecture must be able to handle the categories of data modelled while performing the activities stated in the point number 1.

4. Efficient implementation of algorithms that perform the main procedures identified in points number 1 and 2; the crucial aspects to be considered when implementing these algorithms will be their performances in terms of memory consumption and time-responsiveness.

5. Execution of the developed methods within an automatic workflow which produces a network representation of the input data; given a set of data, that in the biological case are biomolecular entities and their annotations, the software architecture will be able to build networks from them and, when it is required, to compose them into a single network.

6. Implementation of algorithms, that have been proposed in the literature, to predict gene functions from a network of gene associations.

[1] In the remainder of the thesis the word “composite network” is used to refer to a network computed by combining $n > 1$ input networks; whereas networks obtained without performing any composition process are referred as “simple/single network”

7. Validation of the method implemented to derive a composite network from multiple input networks; this phase will evaluate the possible benefits that might occur when predicting gene functions from an data-integrated network model.

Chapter 6

ALGORITHMS AND SOFTWARE LIBRARIES

This chapter describes the methods and the technologies that were applied throughout this thesis. The first three sections present the algorithms and metrics used to compute edge weights of simple networks, to construct composite networks and to predict annotations of biomolecular entities. The fourth section discusses the commonest approaches that are adopted to represent a network inside the computer memory. Finally, the last section gives a brief description of the software libraries that were used for this project.

6.1 Similarity Metrics

The software architecture that was developed in this thesis defines the concept of edge weight metric (see section 7.2) as a metric used to compute the weights of network links when a network is being constructed. To this end, two different metrics were included in the software architecture as default metrics: Pearson Correlation Coefficient (PCC) and Resnik Similarity. The former one is intended to be used with flat annotation profiles, while the latter one can be used with ontological annotations. Throughout this thesis the term annotation profile is used to indicate a set of terms associated to a certain element, this terminology is taken from the biomedical context. In this context, the element is a biomolecular entity, while the terms are usually taken from a controlled vocabulary and they are used to characterize the entity. As it was explained in the Introduction of the thesis (see section 3.3), these mappings are named as annotations. It is worth noting that, even though the terminology adopted belongs to the biomedical area, the concept represented by the word annotation profile is easily portable to other area, being it just a series of mappings element-features.

Pearson Correlation Coefficient (PCC)

The Pearson Correlation Coefficient is a measure of the linear relationship between two variables. Its formulation, when applied to a sample of paired data (X_i, Y_i) with means $\overline{X}$ and $\overline{Y}$ respectively, is the following:

$$r = \frac{\sum_{i=1}^{n}(X_i - \overline{X})(Y_i - \overline{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \overline{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \overline{Y})^2}} \tag{6.1}$$

The value of r ranges from -1 to 1. Correlations of value 1 or -1 correspond to data points lying on the same line, whereas a zero value correlation means that there is no linear correlation between the two variables. An interesting property of the Pearson correlation coefficient is that it is invariant, up to a sign, to linear transformations of data. Hence, the two variables X and Y may be transformed respectively into a + bX and c + dY, where a, b,c and d are constants, while the absolute value of the PCC on new data remains the same as the one computed from the original data.

Resnik Similarity

The Resnik similarity[67] is a metric which gives an estimation of the similarity of two terms belonging to a set of terms arranged in a IS-A structure. Although it was originally conceived for working with IS-A terminologies, its application has been extended to terminologies including other types of relationships, for instance the GO ontology (see section 3.3). Resnik's metric computes the similarity of two terms based on the maximum information content (IC) of their common ancestors. The ancestor of two concepts in a terminology is a term which subsumes both concepts, whereas, following the standard argumentation of information theory, the information content of a concept c can be quantified as its negative log likelihood, -log p(c). The Resnik's similarity in formula is:

$$Sim_{Resnik}(a,b) = \max_{c \varepsilon A(a,b)} \{-log(p(c))\} \tag{6.2}$$

where $A(a,b)$ denotes the set of ancestors which are in common between a and b. Even if the Resnik metric was originally designed to compute the similarity between two terms, its application has been extended to the computation of similarities between pairs of annotated profiles. To this purpose, a common approach consist in calculating the Resnik similarities of each term in the first profile with the annotating terms of the second profile, next, per each term, just the maximum value, among the ones calculated, is retained. The similarities are then summed up and divided by the number of annotating terms existing in the two profiles. The mathematical formulation of this procedure is:

$$Sim(S,T) = \frac{\sum_{s \varepsilon S} \max_{t \varepsilon T}\{Sim_{Resnik}(s,t)\} + \sum_{t \varepsilon T} \max_{s \varepsilon S}\{Sim_{Resnik}(t,s)\}}{|S| * |T|} \tag{6.3}$$

where S and T are the sets containing the annotating terms of the two profiles.

6.2 Network Composition Algorithm

The intuition for combining networks into a single one is that aggregating heterogeneous information, also coming from different sources, into a unique model of the data may bring benefits to the analysis being conducted. Depending on the algorithm adopted to mesh the networks together, this integrated view may be used, for instance, to identify the strongest common interactions among nodes; thus trying to mitigate the problem of false positive, that arises in many contexts. Or else, to include heterogeneous relationships among nodes, resulting in a model which can better describe the domain of usage. This is specially true for the cell machinery, here gene and protein associations are derived from heterogeneous experiments, each one focusing on a particular aspect of the cell machinery. Another motivation for composing networks is to infer new links given a background assumption. This latter point is the key idea which is exploited by the program solution implemented for the software architecture of this thesis.

The method used to assemble a composite network is based on the technique discussed in the work of Mostafavi and colleagues [39]. In this paper, the composition technique is used to compute a functional network of genes (i.e. a network whose connections express to which extent the nodes are similar with respect to the functions the genes are known to have in the cell) by weighting the coherence of networks calculated from heterogeneous biomolecular data (e.g. protein to protein interactions, gene expression profiles, protein domain co-occurrences etc.) against a network computed from GO annotations. This last network is called the target network and it is the subject which drives the learning phase of the algorithm. Extrapolating these concepts from the biomolecular context, one can identify the main subjects involved in the procedure:

- n>0 input networks: the networks to be combined together;
- a target network: the network which instructs the algorithm on how to combine the input networks;
- a composite network: the output of procedure.

The composite network is defined as a weighted sum of the input networks. Calling W_d the input network denoted by the subscript d, the composite network $W*$ is given by:

$$W* = \sum_{1}^{n} \mu_d W_d \tag{6.4}$$

where μ_d is the weight associated to the network W_d. A linear regression problem is solved in order to find $W*$; the quantity to be minimized is the least squares error

between the composite network and the target network T. The vector $\overrightarrow{\mu^*}$ containing the optimal weights per each network is the one minimizing the equation:

$$\overrightarrow{\mu^*} = argmin_{\overrightarrow{\mu}} trace((T - W*)^T (T - W^*)) \tag{6.5}$$

Since the $trace(WT)$ is equal to $vec(W^T)vec(T)$, where vec is an operator which stacks the columns of a matrix one on top of the other, the equation 6.2 can be rewritten as:

$$\overrightarrow{\mu^*} = argmin_{\overrightarrow{\mu}} (\overrightarrow{t} - \Omega\overrightarrow{\mu})^T (\overrightarrow{t} - \Omega\overrightarrow{\mu}) \tag{6.6}$$

where $\overrightarrow{t} = vec(T)$, $\Omega = [vec(W_1), ..., vec(W_n)]$ and $\vec{\mu} = [\mu_1, ..., \mu_n]^T$. This last formulation of the problem coincides with finding the the vector $\overrightarrow{\mu^*}$ which minimizes the least squares error between the edge weights in $\vec{t}$ and the edge weights of the composite network. While the columns of T are stacked one on top of the other, the corresponding interactions in the input networks are selected to compose the matrix Ω. In practice, an additional weight μ_0 is added to the vector of network weights, and a corresponding one valued column is added to the matrix Ω. This avoids the vector solution $\overrightarrow{\mu^*}$ to anchored to the origin.

Even if this technique is quite simple, it has some nice properties. First of all, performing the evaluation of the coherence of all the input networks against the target in a unique common task appeared to be robust to the presence of redundant and noisy networks [39]. On the other hand, techniques that compare an input network at time against the target, such as [68], often need special attention to the problem of redundant data set. For example, in the work of Lee et al. [68] this issue was faced by including tunable parameters into the model, then the parameters are optimized for a given gene function prediction task. Instead, it is worth noting that the method of Mostafavi and colleagues lacks of parameters, property which makes the algorithm application automatic and effortless. However, the drawback related to the lack of tunable parameters is that the method might result limitative for specific scenarios. For instance, if one would like to take into account constraints and requirements which are particular of a given context, then there is little room for customization of the algorithm. Last but not the least point to highlight is that the given least squares task can be solved efficiently with existing algorithms which are also developed to deal with sparse matrices, hence allowing for managing large dimension of sparse data.

6.3 Validation Algorithm

The validation procedure that is followed to assess the validity of a composite network strictly depends on the final goal the network was built for. Typically, the composite network is constructed having in mind a subsequent goal (analysis) which is achieved (performed) by exploiting the structure of the composite network. One of the objectives of the thesis was to investigate the possible benefits that might appear when predicting the functions of biomolecular entities from a composite network computed from heterogeneous data. Given these premises, the algorithm that was chosen to perform the validation task was the GRF algorithm of Zhu et al. [69]. This method is a semi-supervised learning approach based on a Gaussian random field model. A minor modification of the algorithm was used also in [39] for the same prediction problem, i.e predicting functions of biomolecular entities.

Labeled and unlabeled data are represented as vertices in a weighted graph, with edge weights encoding the similarity between instances. In particular, given a label vector y where y_i represents the prior evidence for node i of having the function of interest, the GRF algorithm assigns a discriminant score $f_i \in [-1,1]$ to each node i in the network.

The GRF algorithm can be written in the following general form:

$$\vec{f}^* = argmin_{\vec{f}} \sum_{i=1}^{n} (y_i - f_i)^2 + \sum_{i,j=1}^{n} w_{ij}^* (f_i - f_j)^2 \tag{6.7}$$

where n is the number of nodes in the network and w_{ij}^* is the weight of the edge linking node i to node j. The quantity above, to be minimized, ensures that the discriminants f remain close to their initial labels and that nodes linked by an edge with a high similarity weight are assigned discriminants with similar values. The GRF algorithm has intimate connections with random walk; imagine a particle walking along the graph G. Starting from an unlabeled node i , it moves to a node j with probability p_{ij} after one step. The walk continues until the particle hits a labeled node. Then f_i is the probability that the particle, starting from node i, hits a labeled node with label l.[69]

The problem above can also be written in matrix form as:

$$\vec{f}^* = argmin_{\vec{f}} (\vec{f} - \vec{y})^T (\vec{f} - \vec{y}) + \vec{f} L \vec{f} \tag{6.8}$$

and solved efficiently by computing the solution of a linear system of equations $My = f$, where $M = (I + L)$, and L is the Laplacian of the network. Moreover, the system solution might be found by exploiting iterative techniques such as the conjugate gradient (CG). It must be noted that most of the techniques that are usually used to efficiently solve system of equations put some restrictions on the form of the matrix

M. For instance, in the case of CG, the matrix of equations has to be symmetric and positive-definite. These considerations have to be taken into account when the composite network is generated.

6.4 Techniques of Network Representation

There exist many ways to store networks into computer main memory; the choice of one approach instead of another has repercussions on the speed of the program and the amount of used memory. The most common ways to store network data or parts of their characterizations are:

- Adjacency matrices;
- Adjacency lists;
- Adjacency trees;
- Heaps.

For a detailed review of the pros and cons of each method one may consult the book of Newman [60]. Briefly, I can point out the most salient features of each approach. Adjacency matrices allow to encode the edges of a network into a two dimensional matrix. The number of rows and the number of columns of the matrix coincide, and they are equal to the number of nodes in the network. The matrix cell denoted by indexes i and j contains the value of the weight of the edge linking node i to node j. Storing a network in the form of an adjacency matrix is convenient in many ways. First of all, many of the algorithms devised to work on networks are expressed in mathematical terms by using matrices, hence their conversion to code is straightforward, if one represents the network in an adjacency matrix. An adjacency matrix can be advantageous also for fast direct accesses to the elements, which require constant time O(1) in this case; if the algorithms that are run over the network make use of direct accesses to the edges of the network then an adjacency matrix might be a proper choice to store the network. On the other hand, the adjacency matrix is not always a convenient network representation; for instance running through the neighbors of a node takes time O(n), where n is the number of nodes, even if the network is sparse (i.e. the number of nodes linked to a certain node is far less than n). The major drawback, at least for the purposes of the software developed in this thesis, would be the amount of memory required to store an adjacency matrix, which is O(n^2); in fact, the usual use case considered in this work is the one involving sparse networks (i.e. m << n^2,where m is the number of links in the network).

An alternative to the adjacency matrix representation is the adjacency list structure. An adjacency list is a set of lists, one for each vertex i. Each list contains

labels of the node attached to vertex i. In case the network to be stored is sparse, the amount of memory saved by representing the network with an adjacency list, with respect to an adjacency matrix representation, is considerable. For instance, supposing to store a network whose weights are simply integers which count the number of links connecting two vertices, and that the network has n=10.000 nodes and m=100.000 directed edges; in this scenario, on a machine where each integer occupies 4 byte, an adjacency matrix representation of the network would occupy about 400MB as opposed to the 440KB required by an adjacency list representation. The adjacency list structure allows also to iterate directly over the neighbors of a nodes i, without having to check all possible pair of nodes involving i. One of the cons of an adjacency list representation is that it is not directly usable with existing mathematical libraries which work on matrices. Moreover, even if the algorithm that has to be applied over the network does not make use of matrices, but it requires many direct accesses to edges, then one would probably consider to use an adjacency matrix to represent a network. In fact, as it was explained before, adjacency matrices allows for constant time direct accesses to links, whereas to seek a particular edge in an adjacency list one potentially has to pass by all the links connected to a node, which on average requires time O(m/n).

The other data structure that is sometimes adopted to represent networks is an adjacency tree. Tree structures have most of the advantages of the adjacency list, while being considerably faster for some operations and consuming just a little bit more memory. For instance, a direct access to a certain link on a tree scales logarithmically in the number of edges, the time required is O(log(m/n)) instead of O(m/n) . An adjacency tree structure is organized in the same fashion of an adjacency list, with the only difference that this time each vertex of the network is the root of a tree and its neighbors are arranged in the tree based on the order imposed by their labels. The figure 6.1 presents the complexity of some common operations when performed on different representation of the network.

Operation	Adjacency matrix	Adjacency list	Adjacency tree
Insert	$O(1)$	$O(1)$	$O(\log(m/n))$
Delete	$O(1)$	$O(m/n)$	$O(\log(m/n))$
Find	$O(1)$	$O(m/n)$	$O(\log(m/n))$
Enumerate	$O(n)$	$O(m/n)$	$O(m/n)$

Figure 6.1 – The leading-order time complexity of four operations for various representations of a network of n vertices and m edges. The operations are adding an edge to the network (insert), removing an edge from the network (delete), testing whether a given pair of vertices are connected by an edge (find), and listing the neighbors of a given vertex (enumerate). (Figure taken from [60])

The last data structure mentioned before is the heap. Heaps are not usually used to store a network representation but sometimes they are useful to organize values lying on the network. Their definitive property is to arrange values in such a way that the retrieval of the minimum or maximum value is quick. A binary heap is built upon a binary tree[1]. Each of the item stored in a heap is characterized by two elements: a label that identifies the item and a value associated to the item (figure 6.2). There are two important features about heap structures: first the heap is filled row by row, hence the tree is always completely packed and its depth is logarithmic in the number of items in the tree; second, the values associated with the items in the heap are partially ordered, meaning that each value is greater than (less than) or equal to the one above it in the tree and less than (greater than) or equal to both of the two below it. As it will be explained later on, in the chapter related to the performance, this structure has been adopted to speed-up the ontological similarity computation performance.

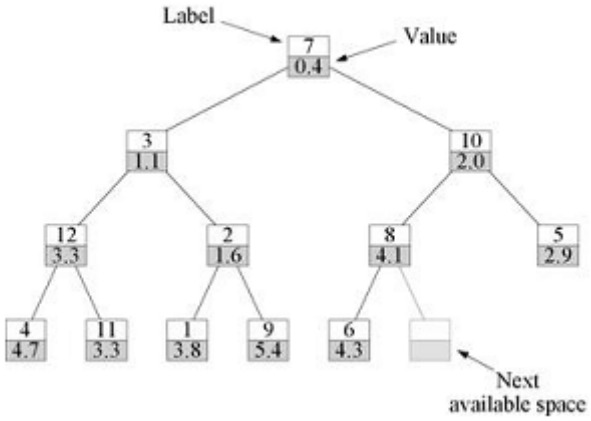


Figure 6.2 – The structure of a binary heap. Each of the nodes of the tree contains a label and a numerical value. The tree is partially ordered so that the numerical value at each node is greater than or equal to the value above it in the tree and less than or equal to both of the values below it. (Figure taken from [60])

6.4.1 Sparse Matrices

Given the pros and cons of each network representation that has been presented above, the network representation chosen to be implemented in the software architecture developed is the sparse matrix. This decision allows, on the one side, to take advantage of robust and efficient mathematical libraries and algorithms already present in the literature, which are often implemented in matricial terms, on the other side a sparse matrix representation has not the drawback of memory wasting that would come out using a dense adjacency matrix for representing a sparse network.

[1] A binary tree is a tree where each node has at most two children.

A sparse matrix is simply a matrix with a large number of zero values. In contrast, a matrix where many or most entries are non-zero is said to be dense. There exists several formats to store sparse matrices into computer memory.[70] The most known ones are:

- Compressed Row Storage;
- Compressed Column Storage;
- Block Compressed Row Storage;
- Compressed Diagonal Storage;

The first two formats are the ones generally used by mathematical libraries which perform operations on sparse matrices that have no specific sparsity pattern. Their storage formats are the most general and they do not store any unnecessary elements. They are one the reverse of the other; here the Compressed Column Storage (CCS) is explained, that is also the format that has been used in the software architecture developed. CCS, as well as CRS (Column Row Storage), works in a similar fashion with respect to an adjacency list, where every node of the network has its neighbors specified in a list. Bringing this idea to a network represented by means of a matrix corresponds to compress the columns (the rows in case of CRS format) of the matrix, hence the nodes of the network, by listing just those entries that coincide to non-zero values. As a matter of fact, the compressed column storage (CCS) format puts the subsequent non-zeros of the matrix columns in contiguous memory locations. The data structures used to manage a CCS format are simply three arrays; two arrays have a length that is equal to the number of non zero elements in the matrix, and they are used to store respectively the values and the row indexes of those elements. The third array has a dimension equals to the number of columns plus one, the $i-th$ entry of this array contains the index associated to the first non zero value of the column i, with respect to the other two arrays. So, for example, the unsymmetric matrix A:

$$A = \begin{matrix} 10 & 0 & 0 & 0 & -2 & 0 \\ 3 & 9 & 0 & 0 & 0 & 3 \\ 0 & 7 & 8 & 7 & 0 & 0 \\ 3 & 0 & 8 & 7 & 5 & 0 \\ 0 & 8 & 0 & 9 & 9 & 13 \\ 0 & 4 & 0 & 0 & 2 & -1 \end{matrix}$$

is represented in memory by the three vectors outerInd, innerInd and values:

$$values = \begin{matrix} 10 & 3 & 3 & 9 & 7 & 8 & 4 & 8 & 8 & 7 & 7 & 9 & -2 & 5 & 9 & 2 & 3 & 13 & -1 \end{matrix}$$

$$innerInd = \begin{matrix} 0 & 1 & 3 & 1 & 2 & 4 & 5 & 2 & 3 & 2 & 3 & 4 & 0 & 3 & 4 & 5 & 1 & 4 & 5 \end{matrix}$$

$$outerInd = \begin{matrix} 0 & 3 & 7 & 9 & 12 & 16 & 19 \end{matrix}$$

If the matrix is symmetric, it is necessary only to store the upper (or lower) triangular portion of the matrix. The tradeoff is a more complicated algorithm with a somewhat different pattern of data access. All the other formats listed above work by making assumptions about the sparsity structure of the matrix, so, for instance, the Block Compressed Row Storage is based on the fact that a certain matrix is composed of square dense blocks of non-zeros in some regular pattern, or else, the Compressed Diagonal Storage format assumes that a matrix is banded with bandwidth that is fairly constant from row to row. These pattern specific formats permits to efficiently perform some operations which can take advantage of the sparsity patterns. However, for the problems being treated in this project, these formats are not necessary since no assumptions can be made a-priori on the structure of the matrices that the system will manage.

The software architecture that was implemented in this thesis uses the C library SVDLibC implementation of the format CCS. The library was used in order to be as coherent as possible to the decisions already made in previous works of the lab. Basically, the SVDLibC library defines a simple container with the three arrays characterizing a CCS format. Accesses to the internal structures of a CCS format are not performed through calls to dedicated functions, whereas the internal representation of the sparse matrix is directly managed by the user. On the other hand, the library provides nice utilities to manage the serialization of sparse matrices onto files, as well a function to compute the transpose of a sparse matrix.

6.5 Software Libraries

6.5.1 Mathematical Kernels

This section describes the mathematical kernels being used to implement vector and matrix operations. BLAS and LAPACK are interfaces that have become standards de-facto for performing basic and high level linear algebra operations respectively. ACML and Eigen instead provide the actual implementation of linear algebra operations and they were used to develop parts of the software architecture. The former one is conforming to the API exposed by BLAS and LAPACK, while EIGEN defines its own API to perform linear algebra operations.

BLAS

BLAS (Basic Linear Algebra Subprograms) [71] is an API supported by the largest number of mathematical libraries aiming at performing vector and matrix operations. There exists both open source (OpenBLAS[72], ATLAS[73], etc. etc.) and vendor(ACML[74] from AMD, MKL[75] from Intel etc. etc.) implementations of it which are targeting the maximization of linear algebra operations performance. The library is divided into levels: Level1, Level2, Level3 and Sparse-BLAS. The first level is concerned to operations between vectors, the second level to operations between matrices and vectors, while the third level address operations between matrices. The Sparse-BLAS level is the last added and, as the name suggests, it contains operations involving sparse matrices. It is also subdivided into three levels, resembling the organization of the dense operations, however there are not operations that involve just sparse entities; at least one of the two subjects of an operation has to be in dense form. The library defines four type of data which it is possible to use:

- float - 4 bytes;
- double - 8 bytes;
- complex - 8 bytes;
- doublecomplex - 16 bytes.

Most of the operations are available in all the four type versions, a prefix before the method name specifies which type of operands the method is expecting.

LAPACK

LAPACK (Linear Algebra PACKage) is a software package provided by Univ. of Tennessee; Univ. of California, Berkeley; Univ. of Colorado Denver; and NAG Ltd.. which is written in Fortran 90. It includes routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems, and singular value problems. The original goal of the LAPACK project was to make the widely used EISPACK and LINPACK libraries run efficiently on shared-memory vector and parallel processors. On these machines, LINPACK and EISPACK are inefficient because their memory access patterns disregard the multi-layered memory hierarchies of the machines, thereby spending too much time moving data instead of doing useful floating-point operations. LAPACK addresses this problem by reorganizing the algorithms to use block matrix operations, such as matrix multiplication, in the innermost loops. LAPACK routines are written so that as much as possible of the computation is performed by calls to the Basic Linear Algebra Subprograms (BLAS). Though the original LAPACK package is written in Fortran

90, there exist also extensions written in different languages, such as clapack (C), lapack++ (C++) and JLAPACK (Java) and optimized LAPACK libraries are provided by the computer vendor (e.g. ACML[74], IntelMKL[75], etc. etc.). For the purposes of the project of this thesis it important to specify that the LAPACK functions address just operations performed on dense or banded matrices.

ACML

ACML (AMD Core Math Library) [74] is a library which provides a free set of thoroughly optimized and threaded math routines for high-performance computing, scientific engineering and related compute-intensive applications. ACML consists of the following main components:

- A full implementation of Level 1, 2 and 3 Basic Linear Algebra Subroutines (BLAS);
- A full suite of Linear Algebra (LAPACK) routines;
- A comprehensive suite of Fast Fourier Transforms (FFTs) in both single, double , single complex and double complex data types;
- Random Number Generators in both single and double precision.

The library can be compiled both for 32-bit machine and 64-bit machine; the last versions of the ACML library support SSE[2] and SSE2[3] instruction sets. The ACML libraries, even if not belonging to an open source project, are freely available. ACML uses the OpenMP API to parallelize vector and matrix operation by exploiting all the core that the users allows it to use.

EIGEN

Eigen [76] is a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms. It supports all matrix sizes, from small fixed-size matrices to arbitrarily large dense matrices, and it also include a module for sparse matrices. It supports all standard numeric types, including std::complex. It supports various matrix decompositions and geometry features. Moreover, its ecosystem of unsupported modules provides many specialized features such as non-linear optimization, matrix functions, a polynomial solver, FFT, and much more. Eigen exploits

[2]SSE (Streaming SIMD Extensions) is an SIMD instruction set designed by Intel. SIMD (Single instruction, multiple data) instructions can greatly increase performance when exactly the same operations are to be performed on multiple data objects.

[3]SSE2 is the expansion of the SSE processor supplementary instruction sets first introduced by Intel

the concept of expression templates to carry on its computation, this allows to intelligently remove temporaries and enable lazy evaluation of operations, when that is appropriate; explicit vectorization is performed for SSE 2/3/4, ARM NEON[4], and AltiVec[5] instruction sets, with graceful fallback to non-vectorized code. For large matrices, special attention is paid to cache-friendliness. The API is clean and expressive while feeling natural to C++ programmers, thanks to expression templates. A plus of Eigen library is that it doesn't have any dependencies other than the C++ standard library and there is no binary library to link to, and no configured header file; Eigen is a pure template library defined in the headers so its application is straightforward. The choice of using Eigen was mainly dictated by the presence of a well documented sparse module offering many functions. For instance, there are six built-in sparse solvers for solving sparse linear system. However, there are also wrappers to the main lower-level libraries addressing the same kind of problems (e.g. SuiteSparse[6][77] and SuperLu[7][78]). Even if Eigen library is written at an higher level of abstraction with respect to its counterparts (e.g. OpenBLAS, ACML, ATLAS etc. etc.), that makes Eigen easier to use, Eigen conserves good performances when performing linear algebra operations. The links [8] and [9] point to pages reporting benchmarks for dense and sparse linear algebra operations respectively. The results of one of the test that has been performed can be seen in figure 6.3. Last but not the least, Eigen is free software, it is licensed under the MPL2, which is a simple weak copyleft license.

[4]The Advanced SIMD extension (aka NEON or "MPE" Media Processing Engine) is a combined 64- and 128-bit single instruction multiple data (SIMD) instruction set, designed by British company ARM Holdings, that provides standardized acceleration for media and signal processing applications.

[5]AltiVec is a floating point and integer SIMD instruction set designed and owned by Apple, IBM and Freescale Semiconductor.

[6]SuiteSparse is a single archive that contains all packages that professor Tim Davis has authored or co-authored that are available at the web-site: http://www.cise.ufl.edu/research/sparse/SuiteSparse/. This research is supported by the National Science Foundation, Sandia National Laboratories, Google, and NVIDIA.

[7]SuperLU is a general purpose library for the direct solution of large, sparse, nonsymmetric systems of linear equations on high performance machines. The library is written in C and is callable from either C or Fortran.

[8]http://eigen.tuxfamily.org/index.php?title=Benchmark-March2009

[9]http://eigen.tuxfamily.org/dox-devel/group__TopicSparseSystems.html

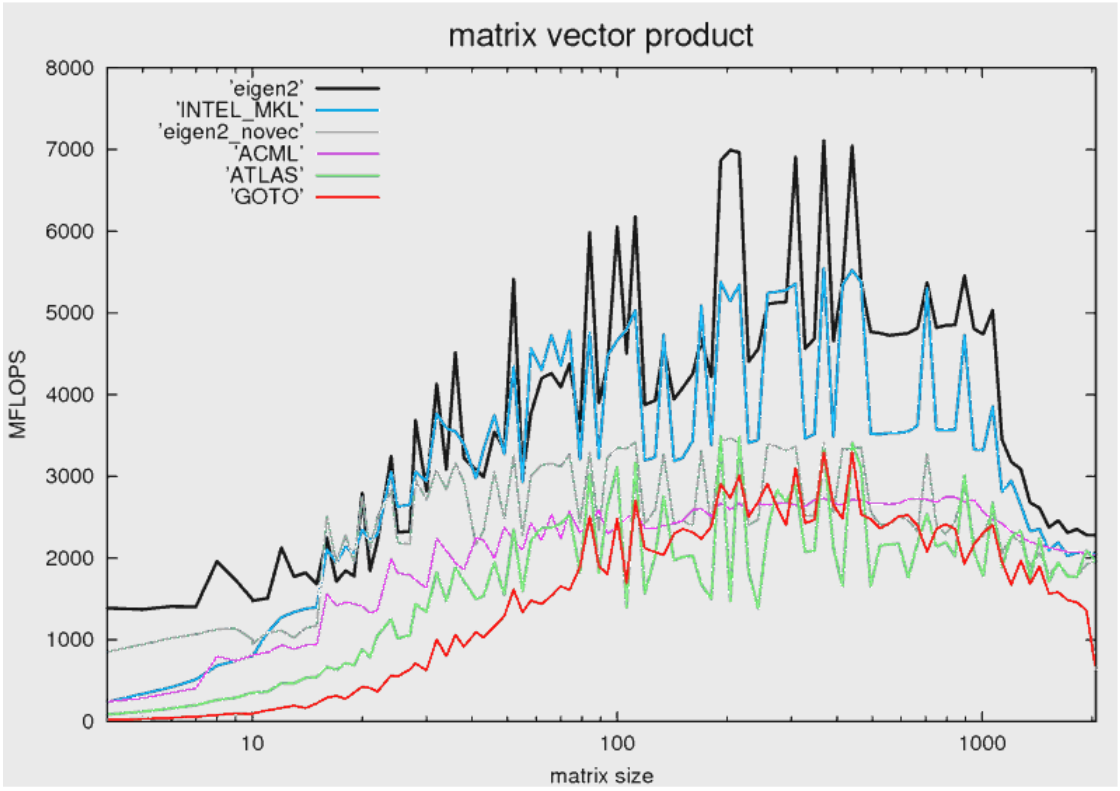


Figure 6.3 – Eigen matrix vector product benchmark. (Figure taken from: http://eigen.tuxfamily.org/index.php?title=Benchmark-March2009)

LSQR: Sparse Equations and Least Squares

As it was mentioned earlier, the algorithm used to build a composite network consist in solving a least squares error minimization. Even though least squares error is a well know and used method to solve linear regression problem, there are a few libraries which offer an implementation of it working with sparse matrices. The code that was used in this project is the C implementation by Michael Friedlander of the method LSQR [79], devised by Paige and Saunders. This code is freely available at the web site of the LSQR method[10]. One nice characteristic of this implementation is that some of the functions used by the library implements BLAS interfaces. Hence one can adapt the code to work with the best BLAS implementation for his/her machine.

The LSQR method at the basis of the library solves or minimizes:

$$solve : Ax = b$$

$$minimize : ||Ax - b||^2$$

[10] http://www.stanford.edu/group/SOL/software/lsqr.html

$$minimize : \|Ax - b\|^2 + \lambda^2 \|x\|^2$$

The scalar λ is a damping parameter. If λ>0, the solution is "regularized" in the sense that a unique solution always exists, and $\|x\|$ is bounded. The method is based on the Golub-Kahan bidiagonalization process and is particularly adapted for matrix $A_{m,n}$ where $m \gg n$ as in the least squares problem faced by this thesis.

6.5.2 Multithreading

The two mathematical libraries that have been used in this thesis (i.e. ACML and EIGEN) both support multithreading in performing algebra operations. These libraries, as well most of the libraries performing high performance computing on shared memory machines, adopt the OMP[80] (OpenMP) API. Even though OMP is already exploited by the selected mathematical kernels, its application would be limited to the linear algebra operations that are performed by the software architecture of this thesis through the mathematical kernels adopted. This turned out to be not enough for boosting up the performance of the software implemented, hence portions of code were explicitly parallelized by using OMP directives.

OMP

The OpenMP Application Program Interface (API) supports multi-platform shared-memory parallel programming in C/C++ and Fortran on all architectures, including Unix platforms and Windows NT platforms. Jointly defined by a group of major computer hardware and software vendors, OpenMP is a portable, scalable model that gives shared-memory parallel programmers a simple and flexible interface for developing parallel applications for platforms ranging from the desktop to the supercomputer. OMP is defined by a collection of compiler directives, library routines, and environment variables which provide a model for parallel programming that is portable across shared memory architectures from different vendors. The directives extend the C, C++ and Fortran base languages with single program multiple data (SPMD) constructs, tasking constructs, worksharing constructs, and synchronization constructs, and they provide support for sharing and privatizing data. The functionality to control the runtime environment is provided by library routines and environment variables. Compilers that support the OpenMP API often include a command line option to the compiler that activates and allows interpretation of all OpenMP directives.

Chapter 7
DESIGN AND IMPLEMENTATION RESULTS

This chapter describes the work that was carried out throughout this project. All the decisions that were made, the methods applied, the artifacts produced share the common goal of achieving the objectives stated in section 4.3.1. The chapter is subdivided into five main sections. The first section states which were the requirements considered for the project. The second section discusses the modelling choices that were done in order to model the workflows for the software architecture being developed. The categories of data the system is built to work on, and the workflows executed by the software architecture are defined in this part of the chapter. The third section provides a description of the architecture design, highlighting its fundamental aspects. The fourth section is a short report pertaining the most interesting issues tackled during the implementation phase, and the solutions that were implemented to face them or the considerations that were derived from them. The last section of the chapter gives a short overview of the functionalies implemented in the system.

7.1 System Requirements

Before entering into the discussion of the designed software architecture, it is worthy to state which were the requirements considered during the conception, the design and the implementation of the system.

1. Modularity: the major requirement, the system wants to be compliant to, is the modularization of its sub-parts. Since there exists a multitude of approaches for constructing networks from raw data, it would be extremely limiting developing a software system with a fixed narrow set of available methodologies, without considering possible extensions of the platform. Thus the idea of extensibility played a central role in the project design phase;

2. Computational efficiency: the computational issues, often involved when managing large amount of data, as it is the case for the planned use case of the project, constitutes another key point of the system. Constructing networks with thousands of nodes requires careful usage of computer memory, as well as the adoption of ad-hoc structures to represent them, such as sparse matrices. Moreover, the computation of interactions among nodes is generally an

intensive task, thus techniques such as multithreading and parallel programming are essential in order to get acceptable building times;

3. Generalization: this requirement might be seen as an implication of the modularity. However, here it is explicitly stated to underline the fact that, even if the system is tested against a bioinformatics use case, the modelling, the procedures and the system infrastructure can be applied to generic use cases;

4. Usability: along with the two principal concepts of extensibility and performance, the system has to be easy to configure and to use and must not be bounded to a specific user interface;

7.2 Workflow Definition

One of the main goal of this thesis was to provide a software system which allows for building networks belonging to different contexts and that was modular enough to be easily expandable to meet user changing requirements. To achieve these results, the system was built upon macro entities and steps that are often involved in network construction processes. For these reasons, a first important stage of the whole project was the definition of the workflows, and the tasks they are composed of, that are executed when building networks.

7.2.1 Use Cases of the System

A first observation that came out from the first analysis phase of the problem is that the tasks performed on raw data, which are then used to derive the network edges and nodes, depend on the structure and the meaning of the data themselves. Four different scenarios were considered, each one with different input data; these are (figure 7.1):

1. building a network directly from interactions;

2. building a network from data annotated to flat features;

3. building a network from data annotated to ontological features;

4. building a composite network.

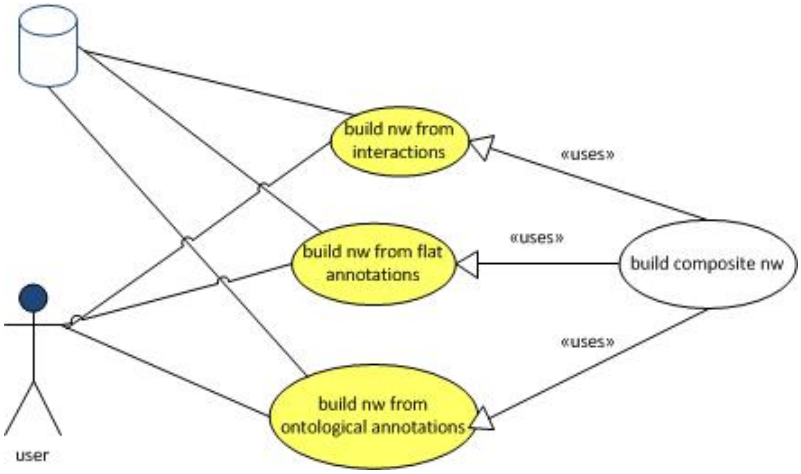
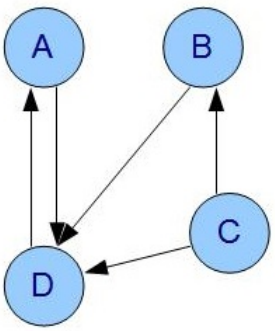


Figure 7.1 – Use cases for the software system to be designed.

The four points are listed in ascendant order of complexity. In fact, while in the first use case the network interactions are explicitly given as input of the system (figure 7.2), the other use cases require some extra processing; in particular, for the second and the third points, specific metrics are needed to compute network interactions. These metrics are the ones that were defined as edge weight metrics, since their application to the input data allow to compute the weighted links for the network to be constructed. For instance, the PCC and Resnik coefficients (see Material and methods 6.1) are examples of metrics that may be used to compute edge weights expressing the similarity of nodes.



A	B	C	D	
-	-	-	1	A
-	-	-	1	B
-	1	-	1	C
1	-	-	-	D

Figure 7.2 – Example of a direct network. The edges in the graph are derived by the interactions provided in matricial form. A one denotes the presence of a link going from the element associated to the cell's row, to the element associated with its column.

The inputs for the second and the third use cases are annotation profiles. They can be seen as binary vectors which encode the associations between an element (i.e. this will be a node of the network) and a set of terms, or features, describing that element (see rows of table a, figure 7.3). The set of terms are ordered in a list with some logic; the position of a certain term in the ordered list is used to access

the information pertaining the annotation of that term to the network node. Values in the annotation profile that are equal to one correspond to associations between the corresponding terms in the vector and the element the profile refers to. A vector entry set to zero means that there is no known association between the term and the element, while a negative value, typically minus one, states that term and element are known for not being associated to each other. The difference between annotation profiles of flat featured and and annotation profiles of ontological features is the nature of the data passed into the system. In both cases, the data do not represent network interactions explicitly, instead they describe the nodes of the network by means of annotations to features. Given the set of vectors which describe the nodes of the network (i.e. annotation profiles), the graph is built by adopting an edge weight metric which allows to compute edge weights among vertices based on some function over the node profiles. What do change in these two situations is that, differently from network derived from flat features, a network constructed from features having an ontological structure may have its edge weights computed from metrics rely on the structure of the terminology the features belong to. In practice, instead of considering the features of a node just as simple flat labels, the ontological situation allows to add extra information that can be used to better describe the context of data.

7.2.2 Toy Example of a Co-Autorship Network

The following example discussion should clarify the concept explained above. Let's set the goal of a certain experiment to ranking nodes of a network according to the probability of having a given property, which, in turn, is known for being associated to just a fraction of them. Thus, the desired output is a list of nodes sorted by the likelihood of having a particular property. This is a common task for graph analysis, in fact various approaches have been defined to determine importance of nodes. For instance, one might apply one of the centrality measures discussed in section 4.3, such as node degree centrality, betweenneess centrality or, also, eigenvector based centrality [81]. A different possible approach might consist in using an iterative label propagation algorithm or random walk based methods, such as the one discussed in the work of Li and Patra [82] to prioritize gene disease candidates, or the validation algorithm implemented in this thesis [69]. For the toy example being discussed here, the network elements are research papers of a certain university, while the properties to be predicted are paper topics; such an application may arise, for example, in scenarios of automatic categorization of documents. For each paper the only data available are its authors; on the other hand, there is another source of information, which is related to the authors, that is the research areas and departments the authors belong to. The situation is represented in figure 7.3; table a), the

one with papers (A,B,C,D) on the rows and authors on the columns (John, Anne, David, Linda, Susan and Carol), is the starting point where to build the network from. This table (a.k.a. annotation matrix) is composed by the annotation profiles (i.e. the rows) associated to the network nodes (A,B,C,D). In this specific case, the one values of the table cells specify which authors wrote a certain paper. Being the papers the nodes of the network, what is missing to entirely define the network are its edges. They are calculated by applying an experiment-specific metric to the annotation profiles.

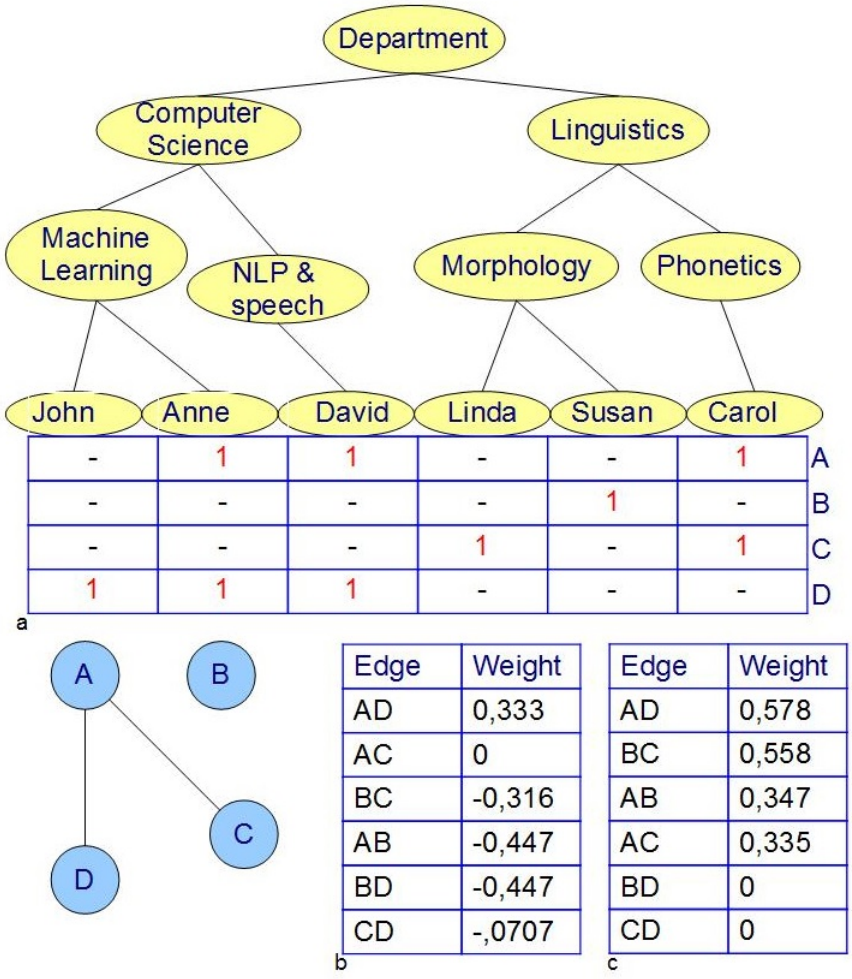


Figure 7.3 – Figure of the co-authorship network example. Table a) gives the associations among research papers (A,B,C,D) and authors (column headers). Cells with 1 values state that the associated column header is an author of the paper corresponding to the cell's row. Each row is the annotation profile for the respective node. On top of the table a), there is a graph (yellow circles) denoting the organization of authors within the university departments. Just below table a) there are the networks built with the three different approaches explained in the text. The first network, built using the co-occurrence metric, is shown by means of a graph (light blue circles). Tables b) and c) represent edge weights of networks computed from PCC similarity and Resnik similarity respectively.

A simple metric that is often used to construct co-authorship network [83], works as follow: two papers are linked together if and only if they share at least n authors.

The result of this metric applied to the toy example, setting n to a value of one, is drawn in figure 7.3. Being simple this metric has some limitations; in fact, given a specific topic and the subset of nodes associated to that topic, every algorithm that tries to predict the topic affiliation for the remained vertices, sees the association between paper A and C and the association between A and D having exactly the same meaning being them both unweighted edges. Looking back to the profile table in figure 7.3, one can observe that paper D is written by authors who all belong tho the computer science department, hence the topic of the paper is likely to be Computer Science related. The same reasoning can be applied to paper C, but this time the authors all belong to the Linguistic department. On the other hand, paper A is written by Carol (Linguistic department) and, David and Anne (Computer Science department), hence its topic is likely to involve both subjects. The metric of co-occurrence, applied to this simple example, draws a link from A to both B and C, however one may claim that the contribute of computer science authors to the paper A has been larger than the linguistics ones, thus the two edges should not be weighted equally (note that in an unweighted network an implicit weight of one can be given to each edge in the network). Whether this last statement is true or not it depends on many factors, such as the final goal of the experiment, its context and also the algorithm that will be used to predict the paper-topic associations. Going on with the example, let's consider the case in which one would prefer to assign different weights to edges AC and AD. In this respective, we can apply the Pearson Correlation Coefficient (PCC) to the university papers example. The results are shown in table b) of figure 7.3. Edges AC and AD are the top weighted ones, but this time the linkage between A and D is stronger than AC. So far, the edge weights were calculated without using any notion about the organization of the authors within the departments. However, sometimes it is desirable to have such an additional context information to assemble the network. This extra knowledge pertains the features used to describe the elements that will constitute the network nodes, not directly the nodes themselves. These observations were the central motivations for including the possibility of expressing feature relationships, by means of DAG structures, within the software system being developed. Since the system offers such an opportunity of exploiting DAG structures to encompass extra context information, the set of usable edge weight metrics is enlarged. To conclude the example that has been carried on during this section, the computation of the weights between papers is performed applying the Resnik similarity to the paper profiles (table c) of figure 7.3). The DAG structure involved in this specific scenario encodes the author-department organization within the university. It is worth noting that the introduction of feature domain context information causes the edge weight of BC to move up to the second position, in the descendent list of network weights. This result has a simple interpre-

tation, in fact it is evident from figure 7.3 that paper B and paper C were written by authors all belonging to the Linguistic area (Susan for paper B, Linda and Carol for paper C), hence, by including the information encompassed in the DAG structure, that concerns departments and research areas, it is likely that topics of B and C are more similar than others.

The example above showed simple cases of networks that the system has to be able to deal with. The three main categories of networks, that are those constructed from explicit interactions, those built from flat feature profiles and the ones computed from ontological feature profiles were distinguished. Moreover, the central role of the edge weight metric used to compute network interactions was discussed. Given that, one of the main goals of the system was to remain as general and extensible as possible, and that the choice of metrics for computing network interactions is highly bounded to the specific contexts and/or analysis to be performed, it was of vital importance to design a software which is modular with respect to the metric that will be used. Such an heterogeneity is manifested even between experiments dealing with the same type of data and having the same final goal; for instance, both the works of Mostafavi et al. [39] and the one of Lee and colleagues [40] aimed at integrating heterogeneous networks for predicting gene functions. However, even if they both dealt with protein to protein interaction (PPI) networks, they used different approaches to construct the networks from protein interaction data sets. In Mostafavi's work the PPI networks were derived by applying PCC to the protein interaction profiles, whereas the latter work made use of the hypergeometric distribution to compute the interactions among proteins.

The last use case, which remained out of the discussion, consist in composing different networks into one composite network (figure 7.1). In this case the inputs for the composition workflow are already networks, as such this use case includes the functionalities of the others. Moreover, trying to formalize the process to construct composite network is really hard; in fact, how to perform the combination of networks is highly algorithm dependent and there exists a wide spectrum of methods suitable for this purpose, each one with its pros and cons. The software system has been equipped with one these methods, as described in the Materials and methods section 6.2.

The identification of the four main use cases discussed so far (figure 7.1), allowed me to distinguished two different workflows:

- Simple network construction;
- Composite network construction.

The former defines the tasks that are performed when building a network from either node associations or node profiles. The latter specifies the processes executed

when composing networks.

7.2.3 Simple Network Construction Workflow

While it was difficult to define a fixed process for composing network, it was possible to define the most common tasks that are often involved in a general network construction workflow. The macro steps that were identified in this project are the sparsification, symmetrization and normalization of the networks to be constructed. The sparsification of a network is often requested when the network has a large number of nodes, and each one has weighted edges towards every other node. In these cases, a full representation of the network is often impossible, due to the amount of memory needed to store all the interactions. An approach that is frequently adopted to overcome this problem consists in discarding some network interactions based on their weights. Interactions are usually removed by setting their values to zero, this way a representation of the network through sparse matrices can reduce the memory consumption. Another operation generally requested is the symmetrization of an adjacency matrix; many network analysis are confined to just undirected networks, due to the fact that they are derived from theories which can be applied to just symmetric matrices. Since the input data of an experiment are seldom symmetric by their nature, sometimes it is necessary to make them symmetric by means of 'artificial' corrections. The last operation that was mentioned is the normalization. This operation is fundamental whenever one wants to compare two different networks, or wants to constrain the interactions weights of a network within a certain interval of admissible values. The process of normalization relates to the adjustment of values measured on different scale to a common scale. As it was the case for the metrics used to compute edge weight metrics, also these latest operations have not a common standard way to be performed; indeed, the same operation may be executed following different approaches. For instance, the normalization might be a column normalization, that constrains the columns of the matrix to sum up to one, or else it might be a normalization of the type $\tilde{W} = D^{-1/2}WD^{-1/2}$ where D is the diagonal matrix having diagonal values equals to the sum of values of the respective rows[84]. Given the variety of approaches that can be used to solve the same conceptual task, it was, once again, of fundamental importance that the software being developed allows to accommodate the user specific implementation for a particular operation, besides offering the default ones.

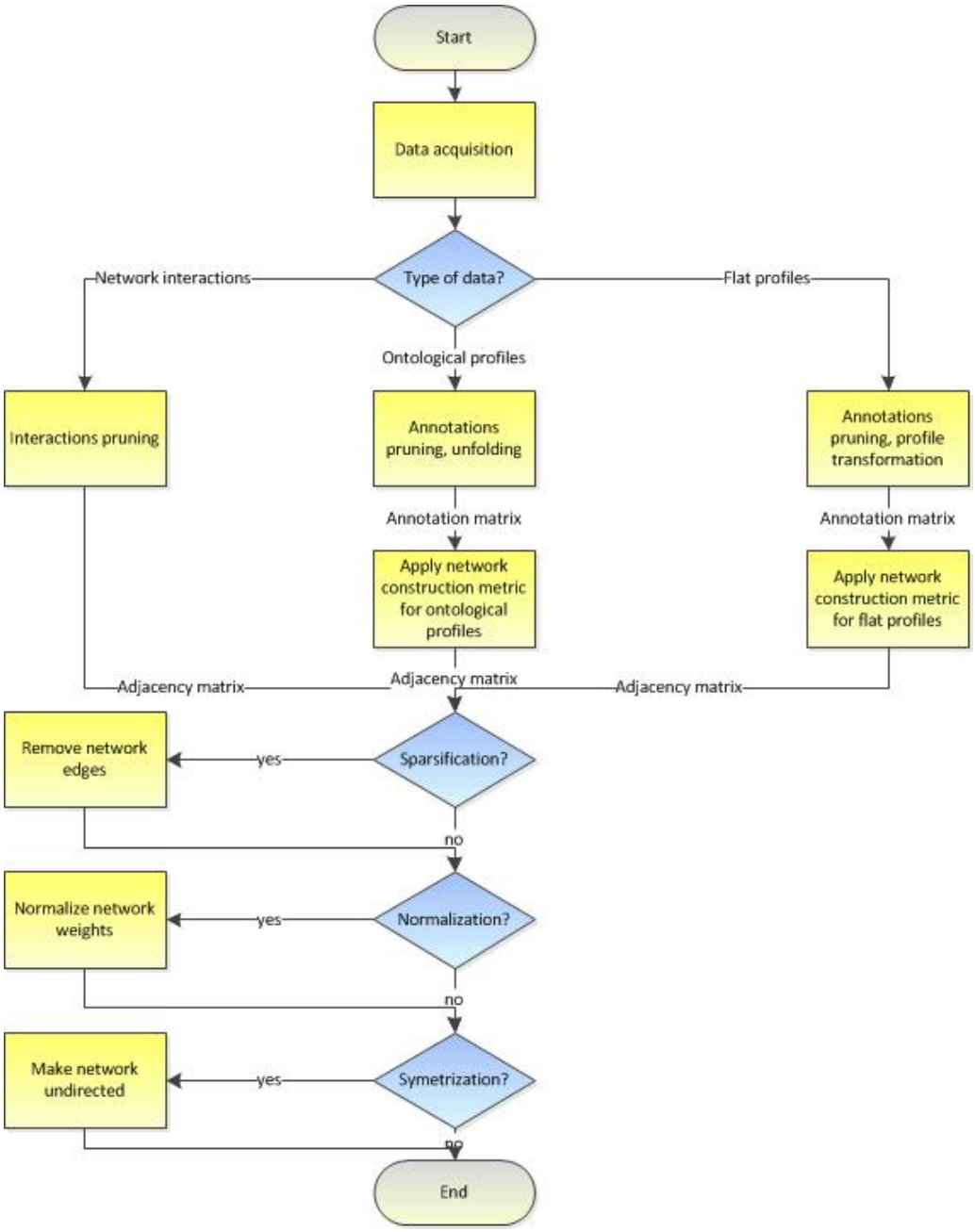


Figure 7.4 – Workflow of simple network construction. The yellow box are the tasks performed in network construction processes; blue diamonds are choices over the tasks.

The figure 7.4 summarizes all the points discussed above. This is the coarse grained workflow followed by the software architecture proposed in this project. After the first step, concerning the data acquisition, the flow of operations splits up into one of the three branches identified based on the type of data being acquired. The particular branch which is followed determines which operation might be performed on the data, however, in any case, there is a shared step that pertains the pruning of data. Even though this task logically performs the same operation in all the three cases, it is worth noting that its results are different. As when data pruning is applied to the most-left branch in figure 7.4, what is filtered out are network interac-

tions, when pruning is applied to the other two cases the elements being removed are annotations, i.e. associations between nodes and their properties. Moreover, depending on the type of data, other data manipulation techniques can be applied before computing the network interactions. In case the features of nodes are arranged in a DAG structure, the system allows to choose whether performing the unfolding of the annotations or not. The unfolding of annotations consists in propagating an annotation of a certain feature c to all the parents of c. For instance, the profile of node A (figure 7.5), that is annotated to term c, changes when unfolding is enabled. After unfolding the annotation profile of A include also the annotation to term f.

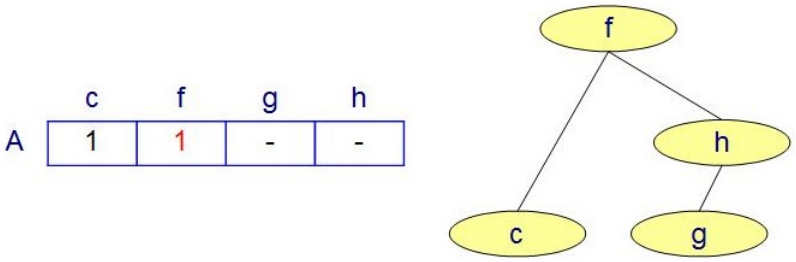


Figure 7.5 – Unfolding. Node A is originally annotated to feature c; after the unfolding has been performed, the annotations is propagated to feature f.

The other task that is data-type dependent is the transformation of data; this operation was thought for flat profiles and it offers a possibility to manipulate and change data before passing the profiles to the network construction metric. For example, an operation that is frequently performed is the log transformation of annotations based on the frequency of a given feature in a corpus of annotated data. This is performed by changing the ones in a profile with the information content of the features and the zeros with the logarithm of one minus the frequency of the features. The transformation step might result essential also when the profiles are not binary; so far the annotations were considered as binary flags indicating whether an element is associated with a feature or not. However, sometimes the properties of a node are numeric and, in this cases, the rescaling and/or normalization of values might be a desired option.

If the input data are annotation profiles, the rows of the annotation matrix are used to compute the weights of the network links, accordingly to the desired metric. The output of this step is an adjacency matrix that encode the weights associated to the network edges. At this point (figure 7.4), the output of all the three branches consist of the same network representation structure. The next steps are the three main operations that have been discussed above and that are performed, when requested, on the adjacency matrix.

7.2.4 Composite Network Construction Workflow

As discussed in the Material and methods section 6.2, the algorithm that was selected to perform the composition of networks defines three main subjects:

- one target network;
- a set of $n > 0$ input networks;
- one composite network.

It is possible to say that the composition workflow (figure 7.6) starts where the simple network construction (figure 7.4) workflow ends. In fact, the inputs of the composition process are one target network and one or more input networks. The software architecture developed offers a unique flow of execution which is able to orchestrate all the tasks performed while composing networks, including the construction of the target and the input networks. This property makes the whole process highly automatized and easy to perform.

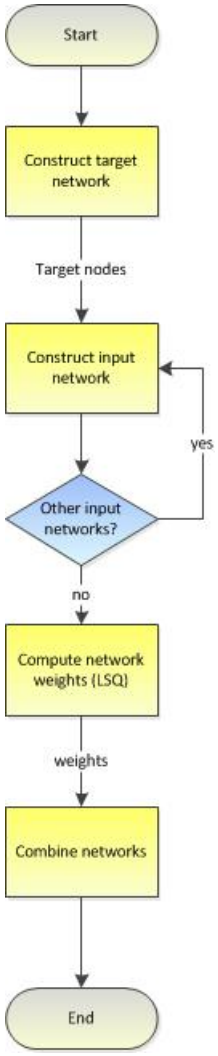


Figure 7.6 – Workflow of the network composition process. The main macro steps involved in the network composition process are put in yellow boxes.

As one can see from the picture 7.6, the first step of the workflow consists in computing the target network following one of the procedure explained in the previous section (figure 7.4). Having identified the nodes participating in the target network, the next step concerns the calculation of the n>0 input networks. The same methodologies applied for building the target network can be applied in this case, with the only limitation that the nodes of input networks are filtered based on the nodes of the target. The workflow proceeds by starting the training phase, each input network is assigned with a weight describing the coherence of the network with respect to the target. After all weights have been obtained, the composite network is constructed by computing a weighted sum of the input networks interactions. The idea behind this process is that the input networks and the target network encode in their links a shared property, but the manifestation of this property is not entirely included in the target network, or it might be not very accurate. Hence, the contribution of the input networks may help to determine, to a larger extent, the property characterization. For instance,i n the work of Mostafavi et al. [39] the property shared among networks is a functional relationship among nodes; the ground truth in this sense is known to be the GO BP ontology. However, the annotations to the BP ontology are not complete and some of them may also be false positive. In this respect, a composite network built by exploiting other sources of information, which may also include a notion of functional relationship among their elements (e.g. this is the case for protein to protein interactions data), can be further analysed to seek new functional associations among genes or to strengthen the confidence upon the ones already discovered.

7.3 Software Architecture

The main purpose of the designed and implemented software architecture is to ease and automatize all those processes and steps that are necessary to construct complex networks, ranging from the acquisition of the input data to the composition of constructed networks. The desired outcome of this work wanted to be a platform that can be exploited to set up and speed up general complex network analysis, by automatizing the network construction and composition tasks. The main steps and entities involved in these processes have been covered in the previous section (figures 7.4 and 7.6 show the general workflows of operations that are required to this end). Besides the network building and network composition functions, the system provides a set of classes that can be used to validate the network composition result; this last point is extensively discussed in the section Testing and Validation of Composition Workflow 9.

7.3.1 Preliminary Choices

The system architecture was decomposed in two main blocks, the first one was written in Java language while the second one, the computation engine of the system, was written in C++. This choice was motivated by the fact that the system is designed to manage large dimension of data and processor intensive computations, hence a core part of the program written in native code is needed to guarantee efficiency in memory consumption and time responsiveness. These two requirements would have been hardly met using a language, such as Java, that is executed inside a managed environment, the Java virtual machine (JVM). On the other hand, Java was used to develop the part pertaining the acquisition of data and the rendering of the results by means of ROC curves[1], discussed in the chapter 9. The division of the architecture into these two sub-parts allowed to reuse pieces of software already developed at the Bioinformatics and Web Engineering lab. of Politecnico di Milano. In particular, the functionalities exported from existing code regards the connection to the Genomic and Proteomic Data Warehouse (GPDW) by means of a connection pooling manager and the plots of ROC curve, which was developed by Sarati in his thesis[1].

The two subsystems communicate with each other by means of JNI - Java Native Interface (figure 7.7). The communication between the C++ layer and the JNI layer was handled using the Adapter design pattern [85], this way the complexity of accessing and converting data defined in the Java layer through the JNI interface results to be hidden to the C++ layer (figure 7.8).

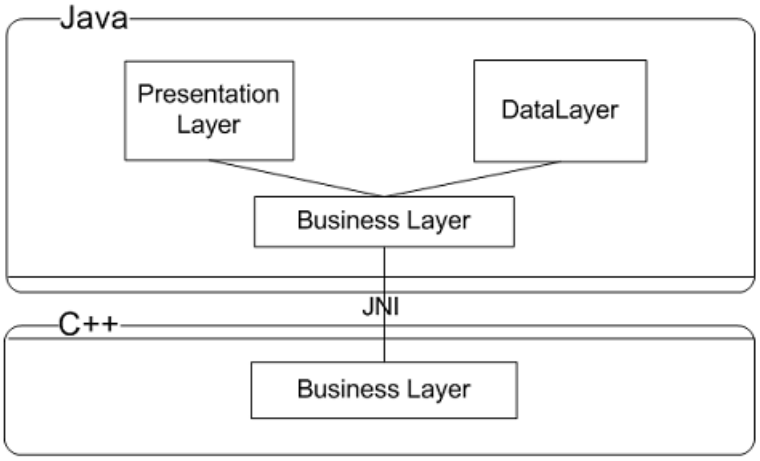


Figure 7.7 – Main view of the system architecture.

[1] ROC stands for Receiver Operating Characteristic and it is a graphical plot which illustrates the performance of a binary classifier system as its discrimination threshold is varied. In this thesis the two axes of the graph refers to the False positive rate and to the False negative rate of predictions for the x-axis and the y-axis respectively.

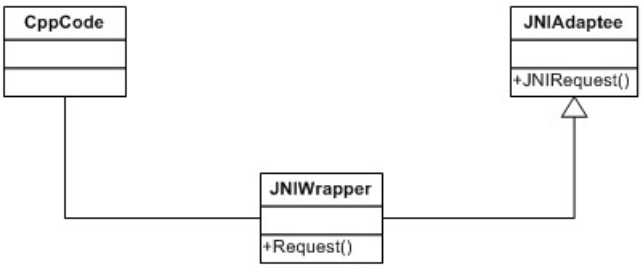


Figure 7.8 – Schema of the generic adapter pattern used to hide the complexity of JNI interface to the CppCode

The characterizations of the Java subsystem and the native code subsystem are now discussed separately.

7.3.2 Java Subsystem

The Java component of the architecture is in charge of parsing and validating a request to the system, collecting the input data for building complex networks and rendering the results in case the issued request is a validation request. Since two of the principal requirements of the architecture were the modularity of its sub-parts and the possibility of extending the approaches that the system provides for building networks, the design phase of the architecture was characterized by such guidelines.

7.3.2.1 Input and Configuration

The starting point of the whole system is a user request which states how the software should build the networks, and eventually compose them. Due to the multitude of options and combinations that the user can choose the requests are encoded into XML files. This choice allowed to be compliant with the modularization requirement, in fact, given that XML files provide a structured organization of documents by means of tags, they can easily express the modularization of a request. Moreover, adopting XML files to formulate requests made possible to parse and validate the inputs exploiting robust and free technologies such as JAXP and the java.xml package. In particular, an XML schema was defined, which is used to validate the particular XML instance requests. Figure 7.9 shows the simplest request that can be submitted to the software.

```
<?xml version="1.0"?>
<!-- this file match the structure of the input interface belonging to the inputTypes package-->
<cnwInput xmlns="http://www.polimi.it.bio.cnw"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.polimi.it.bio.cnw ../schemas/input.xsd">     1. xml schema location
    <target>
        <flatNetwork>
            <name>HomoSapiensNetworkGOBP</name>
            <minAnnotationCount>3</minAnnotationCount><!--pruning of annotations based on the frequency of terms-->
            <maxAnnotationCount>200</maxAnnotationCount><!--pruning of annotations based on the frequency of terms-->
            <sparsity idMethod="cnw_AbsoluteAdaptiveThreshold">     2. method to be used for sparsification
                <params>
                    <param>200</param><!--top interactions to keep-->
                    <param>50</param><!--low interactions to keep-->
                    <param>true</param><!--keep interactions -->
                    <param>false</param><!--removeNegative-->
                </params>
            </sparsity>
            <symmetrization>false</symmetrization>
            <normalization>false</normalization>
            <nonNegative>false</nonNegative><!--enable negative annotations-->
            <annotationRetriever idMethod="cnw_GPDWAnnotationRetriever">     3. method to be used for data retrieval
                <params>
                    <param>db_config.xml</param>
                    <param>cnwGPDW</param>
                    <param>cnw</param>
                    <param>query</param>
                </params>
            </annotationRetriever>
            <save>true</save>     4.save flag
            <similarity idMethod="cnw_PCCSimilarity">
            </similarity>
        </flatNetwork>
    </target>
</cnwInput>
```

Figure 7.9 – Xml file of a request to the software architecture. This is the simplest request that may be submitted, here the software is requested to build a network starting from flat data. The metric used to combine profiles is the PCC similarity (cnw_PCCSimilarity); the data are retrieved from the GPDW datawarehouse (cnw_GPDWAnnotationRetriever).

At this time, I have to specify that throughout this section the architecture will be explained by means of examples of possible requests that the system is able to process. Only the main relevant aspects of the architecture will be treated, a more technical description, accompanied with the complete set of UML diagrams is provided in the appendix A. Coming back to figure 7.9, the first thing to note is that an xml schema location is specified (red text number 1); the schema located at the provided path is used to validate the xml request. The careful reader is likely to have also noted that in the xml request one can find most of the entities that were discussed in the second section of this chapter. In fact, those fundamental blocks are fixed conceptualization of the program, the user can play around with these abstractions by either enabling or disabling them or changing the way they are performed, including the possibility of providing the user specific implementation of a certain task. In the software documentation, and sometimes in this script, the general name associated to these building blocks is “method”, since they all denote the concept of a general method used to perform a given task. All the tasks that can be performed to construct a network are defined with Java interfaces (see appendix A.1.1, Package inputTypes), thus the choice of using Xml files to express a software request is just an arbitrary decision dictated by the pros of having a textual description which resemble the hierarchical composition of a request. However, one can easily im-

plements a graphical interface to the user, as long as the the software is accessed through these interfaces.

Another important Xml file that has a central role in the software architecture is the configuration file. It provides information about which implemetations of the tasks defined in the previous section are available in the system. It also allows to specify further settings such has the ideal number of threads to be used, whether to store the network in main memory or to serialize them, and which format must be used to serialize the network structures. As for the xml input request file, also the configuarion file validity is checked with the xml technologies mentioned above.

7.3.2.2 Adding a New Functionality (a)

This subsection of the thesis shows the mechanism that was designed to add new implementations of the tasks defined in the previous section (figure 7.4), it also provides an idea on how the software architecture works. To make the discussion smoother I use one concrete example of the sparsification algorithm that was implemented for the project. Let's suppose we would like to add the sparsification method that was requested in figure 7.9 (red text number 2).

Th sparsification method requested is identified by an attribute idMethod, this must be an id that uniquely identify a method within the software architecture. Apart of being identified by its id, a method is also characterized by the interface that it implements, so, in this specific case, the xml parser will instantiate an object of id "cnw_AbsoluteAdaptiveThreshold" whose interface is "SparsificationMethod" (see appendix A.1.1, Package inputTypes). All the methods supported by the architecture must appear in the configuration file, which is again an xml file with associated its own xml schema. The configuration file is used to store the information about available methods and it provides information concerning which general operation a method performs; in this case a sparsification operation. Moreover, the configuration file contains the information about which class must be used to instantiate a method description (figure 7.10).

```
<sparsificationMethods>
    <method idMethod="cnw_AbsoluteAdaptiveThreshold" type="Local">
        <methodDescriptorFactory>
            <path>cnw.descriptionCreator.DefaultMethodDescriptorCreator</path>
        </methodDescriptorFactory>
    </method>
</sparsificationMethods>
```

Figure 7.10 – Piece of the xml configuration file related to the sparsification method being requested.

As the figure 7.10 shows, a method has a factory class associated to itself which is identified by a Java path. This path is used to create a factory object through the Java reflection mechanism (a technique that allows to examine or modify the

run-time behavior of applications); the factory created is, in turn, used to instantiate the specific method description. It is worthy to detail more this approach, but first a clarification is needed: the java subsystem instantiates “method description”, not the real method performer. This is because the methods performing operations on the input data are written in native code (C++) in order to meet the software’s time and space requirements. Taking our example, what is instantiated by the program is an object of type “SparsificationMethodDescriptor” which, in this specific case, is sub-classed by “AdaptiveSparsification” (see appendix A.1.4 Package MethodDescriptors, for further details), a class that contains the description on how to perform the “cnw_AbsoluteAdaptiveThreshold” sparsification. The sequence diagram (figure 7.11) depicts the flow of function calls undertaken by the program to build the “cnw_AbsoluteAdaptiveThreshold” sparsification method description.

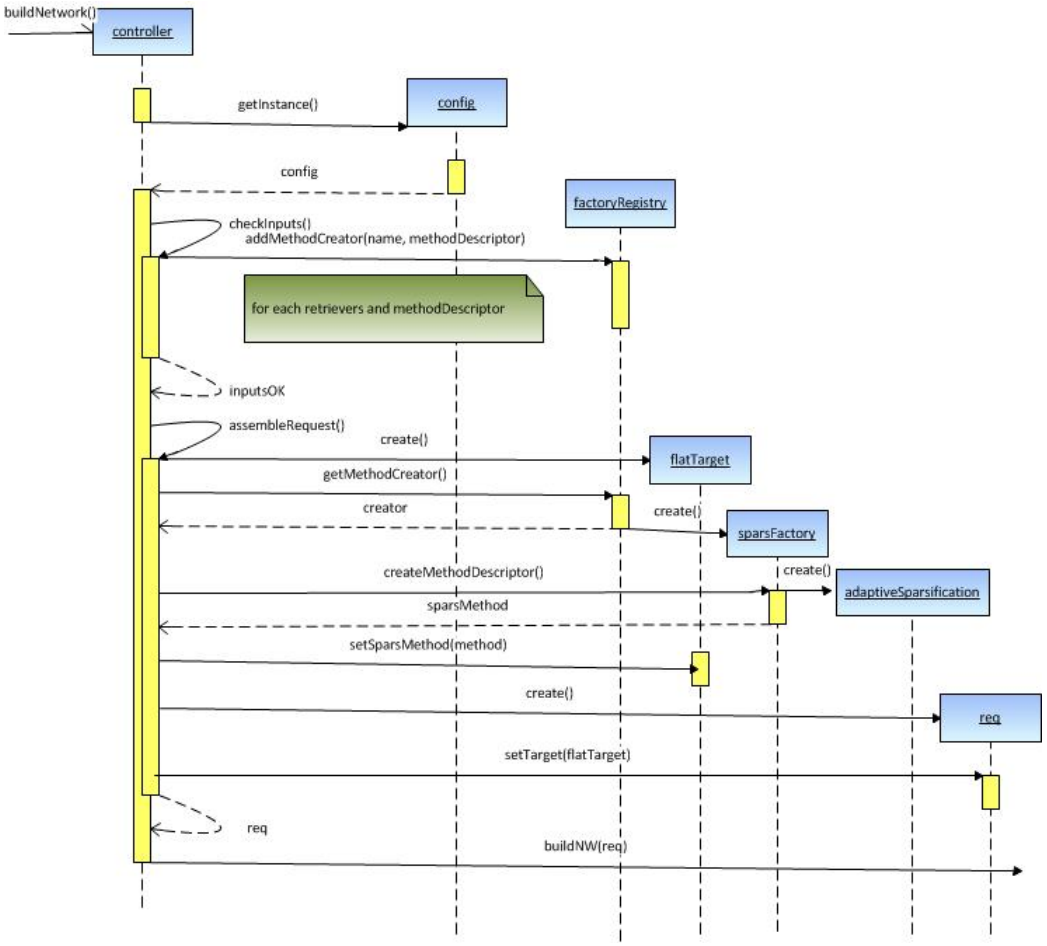


Figure 7.11 – Sequence diagram associated to a network construction request.

As the figure shows, the flow starts by launching a new buildNetwork() request to the controller of the Java subsystem. The first time the system receives a request it also reads the xml configuration file (done by the config object in the sequence diagram) and populates a registry with the factory found in the configuration file, instantiated through Java reflection. Next, the request is controlled, checking that the

methods requested and data retrievers were declared in the configuration file. If this is the case, the system proceeds by assembling a request description that will be sent to the native code subsystem. In this specific example, the network to be built is derived from flat annotations, hence the corresponding network description for this type of data is instantiated and populated. Since the example wants to build a network that is sparsified by means of the “cnw_AbsoluteAdaptiveThreshold” method, the controller will pass this id to the factoryRegistry, that has been previously populated from the configuration file with the id of methods and their factories. By matching the id, the registry returns the factory associated to the “cnw_AbsoluteAdaptiveThreshold method. At this point, the controller gets an “adaptiveSparsification” description by asking to the factory (Abstract factory pattern[85]), the description is then set into the request which will be sent to the native code subsystem. This mechanism of instantiating and setting descriptions into the request is adopted for all the entities of the system that can be customized with user-specific implementations (e.g. the “cnw_PCCSimilarity” specified in the request of figure 7.9). The two key points of this approach are: the usage of the Java reflection mechanism, that can potentially allow to add a functionality to the program at run-time, and the adoption of the Abstract Factory pattern. This last point permits the controller to be independent from the particular factory provided by the user and also to the particular method description. In this way every new method that is conform to the interfaces exposed by the architecture can be plugged into the architecture. As a matter of fact, the steps needed for a new method to be correctly loaded into the system are:

1. Developing a description for the method which is compliant to the interface exposed by the system;

2. Developing a factory class for the above description;

3. Register the association between the method id and its factory class into the configuration file.

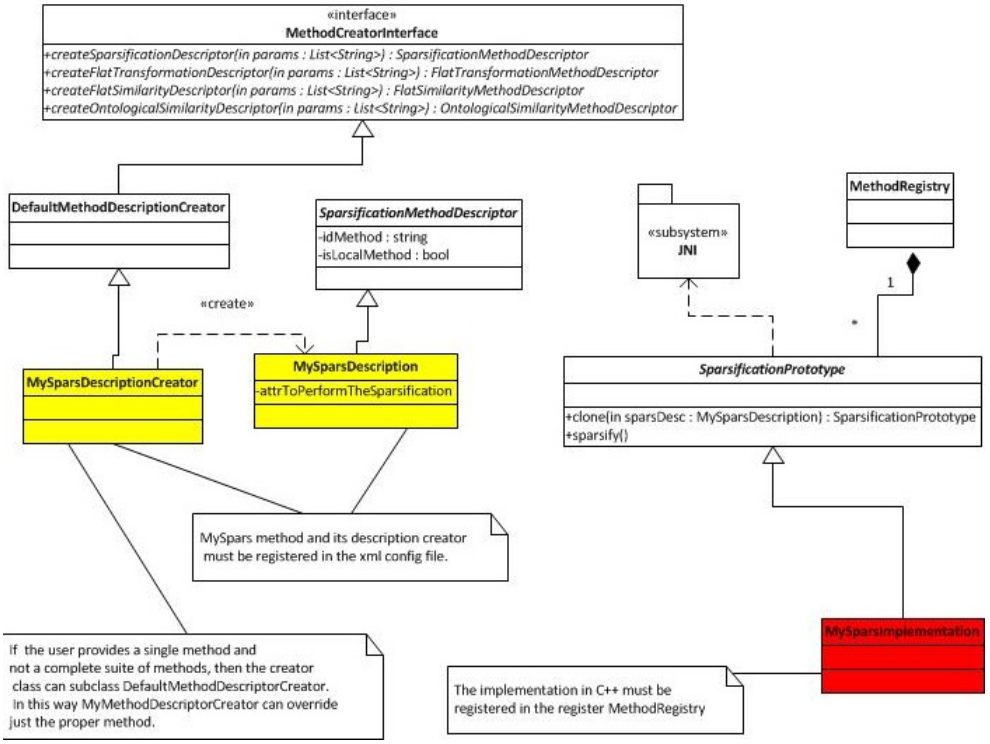


Figure 7.12 – Class diagram of the classes involved in the addition of a new sparsification method to the system. The left-hand side of the diagram refers to the classes belonging to the Java subsystem, whereas the right-hand side regards classes of the native code subsystem

The class diagram 7.12 shows the interfaces that must be implemented by a sparsification method that wants to be added into the system. The yellow classes “MySparsDescriptionCreator” and “MySparsDescription” are the factory class and the description associated to the new method, respectively. “MethodCreatorInterface” is the interface that the factory class must implement, whereas the “SparsificationMethodDescriptor” is the interface that the sparsification method descriptions of the system must implement (other method description interfaces, such as the one for flat similarity methods are reported in appendix A.1.4, Package MethodDescriptors). As one can see from the figure 7.12, all the inputs to the factories are lists of strings, this choice guarantees that the mechanism of inclusion of new functionality is as general as possible. For instance, the piece of the xml input file concerning the sparsification method (figure 7.13), is composed by the id of the specific method to be used and a list of parameters that are specific to the method that is selected. In this case, the first parameter indicates the number of top interaction weights to be retained per each node, the second parameter does the same but with the lowest interaction weights, counting from the lowest valued interaction.

```
<sparsity idMethod="cnw_AbsoluteAdaptiveThreshold">     2. method to be used for sparsification
    <params>
        <param>200</param><!--top interactions to keep-->
        <param>50</param><!--low interactions to keep-->
        <param>true</param><!--keep interactions -->
        <param>false</param><!--removeNegative-->
    </params>
</sparsity>
```

Figure 7.13 – Piece of the xml input request regarding the sparsification method to be used.

The third and fourth parameter are flag used to indicate whether to keep the interactions in between or outside the interval specified and whether to conserve negative interactions or not. Here the important fact is that the system can host new implementations of a given task, such as the sparsification task or the profile similarity computation, without posing any limitations upon the number and types of parameters that the new implementation can have. In fact, all the parameters specified are passed to the implementation specific factory as a list of strings; hence, it is a factory responsibility to convert the parameters from a string type to the desired input types and to put them into the method description objects.

7.3.2.3 Data Retrievers

As explained so far, the Java subsystem is responsible for assembling a request of the user; the actual computation of the network takes part at the the native code side. For this reason, when a new method is added to the system, also its actual implementation must be provided; this topic will be discussed in the next sub-section, as it pertains the native code subsystem. The only exceptions to this statement are the data retriever objects of the system. As already mentioned, the data layer part of the system resides into the Java subsystem. This choice was made mostly to reuse the GPDW data warehouse connection software already developed by the lab. Inside the system, from a conceptual point of view, the data retriever are treated as all the customizable methods (i.e. sparsification methods, flat similarity methods, transformation methods etc.). As a consequence, many different data formats can be exploited to build networks, each one associated to its own data retriever. As it was the case for the previous example of the AdaptiveThreshold sparsification method, also the data retrievers must implement the interfaces that the system exposes (see appendix A.1.5, Package Retriever). Three different data retriever interfaces were defined, depending on the meaning of the data retrieved. These are:

- NodesRetriever
- AnnotationRetriever
- RelationshipRetriever

NodesRetriever, as the name suggests, is used to take the elements that will be subsequently set as nodes of the network; the annotation retriever is used with two different semantics. In case the network that one wants to build is derived directly from interactions, the AnnotationRetriever must provide the node associations, on the other hand, if the network has to be built starting from flat annotations or ontological ones, then the AnnotationRetriever is supposed to retrieve association between nodes and their features; the last retriever, RelationshipRetriever, is expected to be used in case of network constructed from ontological data., its purpose is to supply associations parent-child among features. At the time of writing, the only retrievers developed are the ones to connect to the GPDW data warehouse, however including new retrievers into the software architecture should be effortless. The procedure that has to be followed is the same as the one explained above for the sparsification method. The only difference is that, this time, the object instantiated by the factory is not a description of a method, instead it is the implementation of the method itself, which is written in Java language. The fact that all the classes involved in the instantiation and implementation of the data retrievers reside entirely in the Java subsystem, allows to fully take advantage of the Java reflection functionality. Hence, including new data retrievers to the system can be done directly at run-time. The possibility of adding new data retrievers to the system is specially important when one would like to reuse data taken from other experiments; this is often the case of a typical biomedical scenario, where the presence of a large multitude of file formats (see the introduction 3.4) requires a software architecture which is flexible enough to accommodate future needs. The same reasoning, about generic parameters that can be passed as inputs to the specific method implementations, still applies to the data retrievers. As an example, the figure 7.14 shows the piece of xml code used in an input xml request to indicate which annotation retriever has to be used and with what parameters it should be called. The first three parameters are information needed to set up a connection with the GPDW data warehouse, whereas the remaining two parameters express the query that must be submitted in order to retrieve the annotations. The first one is the main query; within the main query, it is possible to insert placeholders (i.e. the %s token specified in the CASE clause) that will be substituted with the outputs of the queries appearing below the main query, following the same order of placeholders insertion.

```
<annotationRetriever idMethod="cnw_GPDWAnnotationRetriever">
    <params>
        <param>db_config.xml</param>
        <param>cnwGPDW</param>
        <param>cnw</param>
        <param>
        select distinct g2bff.biological_function_feature_oid, g2bff.gene_oid, f.feature_type,
        (CASE WHEN (g2bff.qualifier IN %s) THEN true  ELSE false END) AS is_negate
        from gene2biological_function_feature as g2bff
        join (select gene_oid from gene
            where taxonomy_id in (9606) and (is_obsolete is null or is_obsolete = false)) as g on g2bff.gene_oid = g.gene_oid
        join (select biological_function_feature_oid, feature_type from biological_function_feature
            where feature_type in (6) and (is_obsolete is null or is_obsolete = false)) as f on f.biological_function_feature_oid = g2bff.biological_function_featu
        where (g2bff.evidence not in (1,2) or g2bff.evidence is null);
        </param>
        <param>
        SELECT id FROM flag.qualifier_flags WHERE UPPER(name) LIKE 'NOT%';
        </param>
    </params>
</annotationRetriever>
```

Figure 7.14 – Piece of xml input request regarding the definition of the annotation retriever to be used.

7.3.2.4 The Roles of Networks within the System

So far the discussion of the Java subsystem has been confined to the case of a request for building a single network, however this use case has been meaningful enough to show all the main points of the software architecture at Java side. As a matter of fact, a request for building a composite network adds just a few elements to the Java part of the program. These are the distinction of networks into target and input networks and the possibility of directly specifying what elements will be the nodes of the composite networks. If the nodes of the composite network are not specified, then the program assumes that the vertices to be used are the ones of the target network, as a consequence all the data related to the input networks are filtered with respect to this set of nodes. The role of a network within the composition process is defined by the type of the interface implemented by the network description provided by the user (see appendix A.1.1, Package InputTypes). This property is also reflected into the xml input file by the two tags “<target>” and “<inputNetwork>”. Both the nodes and the input networks are optional entites that may be left out of the request. As said before, if the nodes are not directly specified in a separate tag “<nodes>”, the target vertices are used as nodes for the composite network. If in a request the input networks are not given, the process stops when it has built a simple network, which must be defined as the target network. The figure 7.15 displays the xml file associated with a request for constructing a composite network; here, the nodes were specified at the top of the file.

```
<?xml version="1.0"?>
<!-- this file match the structure of the input interface belonging to the inputTypes package-->
<cnwInput xmlns="http://www.polimi.it.bio.cnw"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.polimi.it.bio.cnw ../schemas/input.xsd">
    <nodes>
        <nodeRetriever idMethod="cnw_GPDWNodeRetriever">
            <params>
                <param>db_config.xml</param>
                <param>cnwGPDW</param>
                <param>cnw</param>
                <param>
                select gene_oid
                from gene as g
                where g.taxonomy_id = 9606 and (g.is_obsolete is null or g.is_obsolete = false) and g.gene_type = 2
                </param>
            </params>
        </nodeRetriever>
    </nodes>
    <target>
        <flatNetwork>
            ...
            ...
        </flatNetwork>
    </target>
    <inputNetwork>
        <ontoNetwork>
            ...
            ...
        </ontoNetwork>
    </inputNetwork>
    <inputNetwork>
        <directNetwork>
            ...
            ...
        </directNetwork>
</cnwInput>
```

Figure 7.15 – Xml associated to a composite network building request. The ellipsis are in place of the network descriptions

7.3.3 Native Code Subsystem

The Java subsystem architecture was introduced in the previous section, its main responsibility is to parse the request of the user and to assemble a new request, expressed in a proper format, that is then passed to the native code subsystem. Here, the object representing the assembled request is undisclosed and investigated with the support of the JNI interface. All the networks specified in the request are constructed following the user indications, which consist of a selection of specific methods and data retrievers to be used to solve the main tasks identified in the first section of this chapter.

7.3.3.1 The Session Mechanism

A single user request generates more than one interactions between the two main components of the system (figure 7.16). There is a concept of session which is shared among all the interactions sparked from a single user request. As figure 7.16 shows, the flow of activities starts by creating a new session, which is a structure that is created and maintained within the native code subsystem.

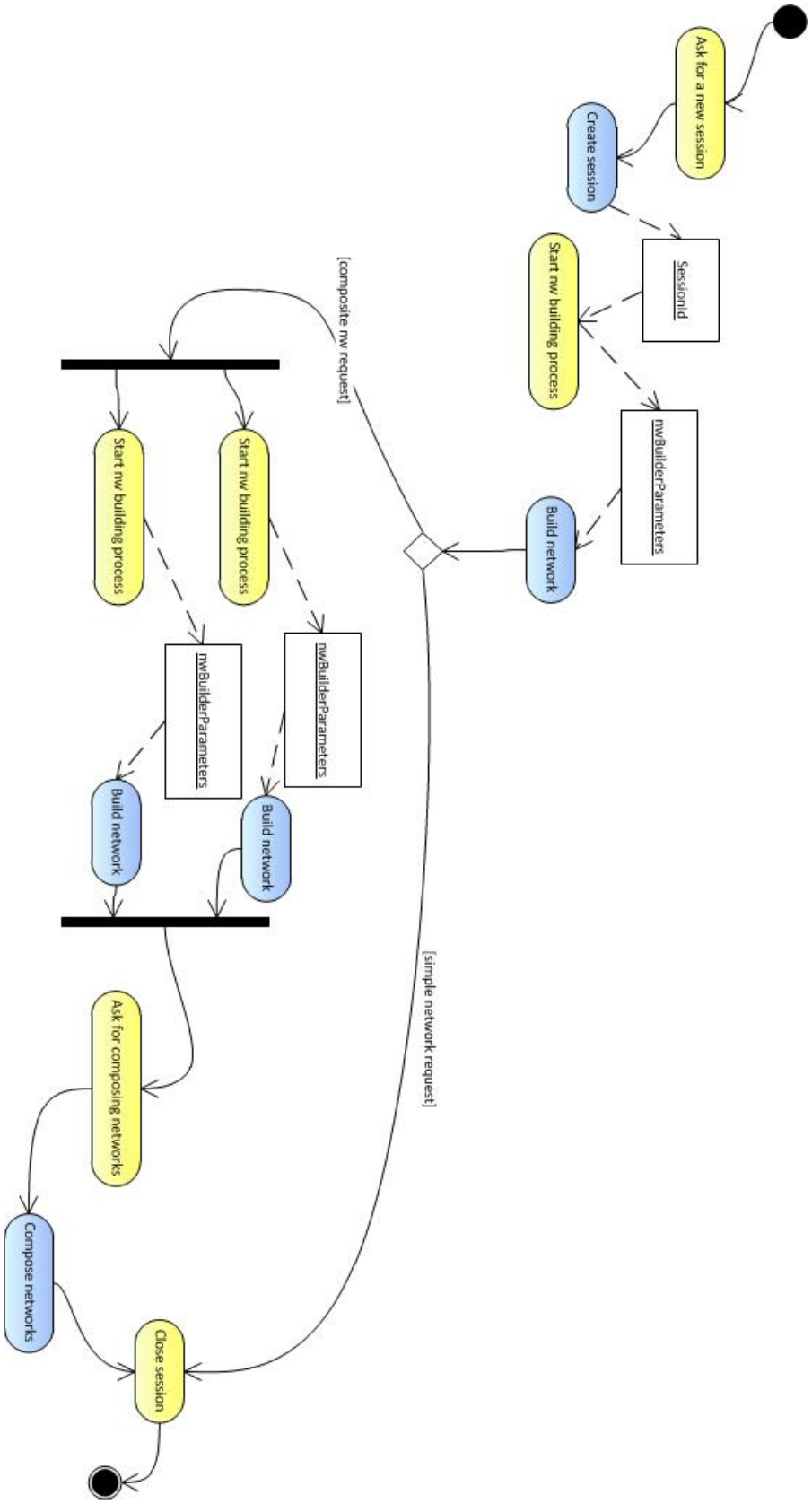


Figure 7.16 – Activity diagram of the activities undertaken by the Java subsystem and the native code subsystem to deliver a user request. The activities in yellow boxes take part within the Java subsystem; usually, they are requests sent to the C++ subsystem. Light blue boxes denote activities carried on by the native code subsystem. White rectangular boxes represent the object flows between activities.

The class “SessionRegistry”, which implements the Singleton pattern [85], is in charge of managing the life cycle of sessions.

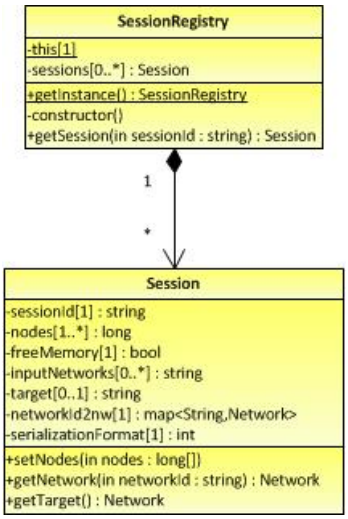


Figure 7.17 – Class diagram of the Session and SessionRegistry structures

The figure 7.17 shows the class diagram of the two classes involved in the realization of the session mechanism. As one can see from the attribute of the class Session, objects belonging to this class are used to store all the intermediate products generated by a user request, for instance the target network and the input networks when the desired output is a composite network. Besides request-related products, a session contains also information about general settings of the program that will be applied to all the tasks performed within the context of that session. For example, the values of the flag “freeMemory” and “serializationFormat” (attributes of class “Session”, figure 7.17) are specified whenever a session is created; their values are read by the Java subsystem, from the xml configuration file cited in the previous section, and then they are passed as inputs of a new session creation call. The first flag, “freeMemory”, gives and indication to the program about whether to keep the middle results (e.g. annotation matrix, target networks and input networks) in main memory or to serialize them into permanent memory and then reload them when they are requested. This mechanism of writing to files and reading from them every time the data are requested might be necessary in case all the data do not fit into main memory. For instance, assuming that the program is at the point of summing up input networks to derive the composite one, in some scenarios it might be that the size of each input network is quite large, thus having all the input networks loaded into main memory is unfeasible. In this case, by enabling the flag “freeMemory”, the program keeps in memory only those networks that are currently involved in the summation, that is a pair of networks; the result is then summed up to the next input network that will be loaded from main memory only when needed. On the other hand, if there is no problem of memory space for the given scenario, avoiding

to writing and loading to file can speed up the entire process. The other flag that was mentioned earlier is the “serializationFormat”; every time a network or some of its products are written to file this flag is checked, its value determines the format used to write files (textual or binary and dense or sparse).

7.3.3.2 Strategy Pattern for a Single Network Construction

Once a session has been created the first network can be constructed. In order to do that, the Java subsystem sends a building network request to the native code subsystem (figure 7.16), specifying the description on how the building process should be performed. The description (i.e. nwBuilderParameters in the figure 7.16) is analysed by accessing its fields through the JNI interface. At first, the category of the data (i.e. network interactions, flat annotations or ontological annotations) to be used is checked. Depending on this, the program follows one of the branches of the workflow discussed in the first section (figure 7.4). The entire workflow was implemented in source code by adopting the Strategy pattern [85]; this pattern allows to define a family of algorithms, encapsulate each one and make them interchangeable. The abstract blocks of the tasks involved in the workflow were defined following this pattern, each abstract block can be implemented by different concrete objects -method performers- which execute the task following their own strategy. The method performers implement the interfaces exposed by the software architecture, moreover their prototypes are stored into a registry of methods (this is explained in the next subsection). Thus, the implementation of the workflow defined results to be really modular. To change one of the step performed in the workflow, it is enough to develop a new method performing that precise step and implementing the right interface. The method is then automatically retrieved from the registry based on its uniqe id.

The input data necessary to build the network are retrieved by using the data retriever that has been attached to the network description, hence the native code calls back the Java side executing the specific implementation of the data retriever that has been requested by the user. As explained before, the difference between methods used to retrieve data and methods used to process data is that the former are implemented in Java language, while the latter are implemented in native code to have a more direct control over the computational time and the memory consumption of the methods.

7.3.3.3 Adding a New Functionality (b)

At this point, it is possible to come back to the discussion pertaining the procedure that must be followed in order to extend the methods supported by the software architecture. The previous section discussed the steps that have to be taken for

including a new method into the system (figure 7.12); one thing was left out of the explanation. In fact, since the data processing methods are implemented in native code, there is still one step needed to fully enable the new method inside the system, this is the method registration within the registry residing in the native code subsystem.

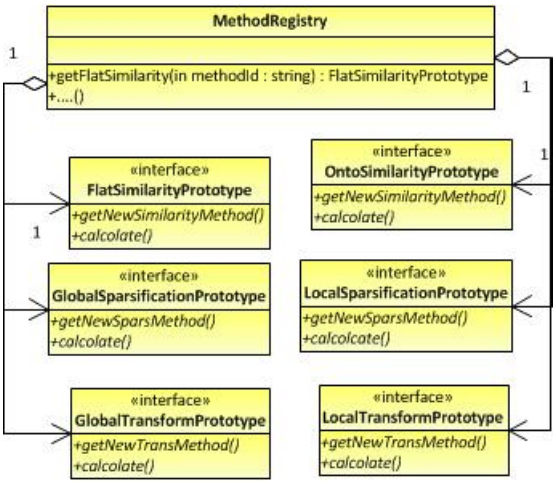


Figure 7.18 – Class diagram of the MethodRegistry class

The figure 7.18 shows such a structure; the first thing to note is that the same main tasks that have been identified throughout this chapter are present also in the native code subsystem. They were defined by means of interfaces that are implemented by the specific methods which perform the general tasks using their own strategies. What happens inside the software system, when the program is analysing a network building description object, is that the method registry is queried for a particular task implementation. If a prototype of the method implementation, as long as its unique id, were previously stored into the registry, then an instance of an object performing the desired method is instantiated by its prototype; this instantiation mechanism is suggested by the Prototype pattern [85]. Once the performers of the given tasks have been obtained, the flow of operations needed to build the network (figure 7.4) can be resumed; the macro steps modeled in the first section of this chapter are performed by the specific method implementations retrieved from the method registry. The execution of tasks are initiated by calling the function provided by the objects obtained from the registry; among the arguments of these functions there is the method description that was built in the Java subsystem and which gives information to the method about how it should perform its task. To complete the description of the figure 7.12 regarding the inclusion of a new sparsification method into the system, one can see that the “MySparsificationImplementation” class implements the interface “SparsificationPrototype”, which is designed exactly for the sparsification task.

7.3.3.4 Local and Global Methods Implementation

The discussion about the interfaces of the objects stored in the method registry let me introduce the last concept that characterize the architecture model. This conceptualization can be seen in the class diagram of the method registry (figure 7.18) and it concerns the differentiation between local methods and global methods. From the figure 7.18, one can note that the sparsification task and the transformation task of the flat annotations can be performed either in a local version or in global one. This differentiation is very important when the network to be built has a large number of nodes. For instance, supposing a network of 50K nodes, the adjacency matrix that stores all its interactions has 50K * 50K cells. If each weight of the network is saved using a type double of 64 bit, the entire space occupation of this matrix is about 19073MB. The first section of this chapter has already suggested that one technique to reduce the space taken by a network is the sparsification of the network itself. This consists in removing some “unimportant” edges, thus saving the space that was required to store those links. However, in the workflows that have been defined so far (figure 7.4), the sparsification task comes after that the adjacency matrix has been built, hence the system still requires enough memory to accommodate all the matrix, at least in a first stage. What was done by defining local versions of some tasks (i.e. sparsification and transformation) was to allow those tasks to be performed on a reduced part of the data. In case of a lacal version of a sparsification method the subset of data are the edges connected to a single node (i.e. a row or a column of the adjacency matrix). In this way, the edge weights of a given node are removed as soon as the complete set of its links has been computed, without waiting for having the entire set of network interactions available. A similar reasoning is applied to the transformation of flat annotation data. The difference is that in this case the task works on the annotation matrix (figure 7.4), while the sparsification process is done on the adjacency matrix. As a consequence the memory saturation problem can arise when the annotation matrix has large dimension, both in term of annotated elements and annotating features, and the transformation process of the annotation profiles transform the profiles from sparse to dense. It is worthy to note that the global versions of the tasks implemented may still result useful in some applications, in fact global versions have the complete vision of the entire dataset they work on.

7.3.3.5 Composing Input Networks

If the original user’s request was to build a composite network (figure 7.16) the same process explained above, applied when constructing a single network, is repeated for all the other input networks and the target included in the request. Once they are computed, the Java subsystem calls the native function that compose the input

networks. At first the target is retrieved from the session, if the target is not in memory then it is loaded from the file it was serialized to. The adjacency matrix associated to the target is then scanned and all the indexes of the non zero values are conserved. After this phase, the matrix needed for the training phase of the composition algorithm (see section 6.2) is computed by selecting the elements of the input networks - previously retrieved from the session - that have a matching with the conserved indexes of the target. This operation can be parallelized by using the OMP directives; whether to execute it with or without multithreading it depends on the configuration file settings. The matrix derived in such a way, is then used with the LSQR algorithm (see section 6.5.1) to compute the weights to be assigned to each input network. The composite network is obtained by summing up the adjacency matrix representations of the input networks, weighting each networks with the learnt coefficient.

7.3.4 Validation Workflow

The validation workflow of the system is executed to validate the composite network construction procedure. To validate the results obtained by composing input networks (i.e. a composite network) the GRF label prediction algorithm (see section 6.3) is run on the composite network. The results are then assessed by plotting ROC curves reporting False Positive (FP) and False Negative (FN) rate on the axis. Each ROC curve is also characterized by an AUC value (Area Under the Curve) which summarizes the prediction accuracies obtained. The entry point, in the software architecture, for the validation process is again the Java subsystem. This time the interface implemented was a Graphical User Interface, realized through the Swing libraries. The software for plotting ROC curves was already available on the lab. server, so the interface developed was just a connection to the available code, in addition to some specific fields and boxes needed to customize the validation task. From the interface, the user can select the xml request input file that is used to start a new entire workflow composed by the following activities:

- construction of the target network;
- construction of the the input networks;
- composition of the networks into the composite network;
- validation of the composite network performing a label prediction task.

In alternative to provide an xml file request, the user can select networks already built that will be just composed together and validated. Before starting the whole process the user is supposed to select the desired value for the number of k associated to

the k-cross validation to be performed. Based on this number, the annotation matrix of the target - which is the source of the prior evidence of a certain feature - is split into k portions. K-1 parts of the annotation matrix are selected to be used for constructing the target network, while the remaining portion of the matrix is hidden to the process. The composite network is then built against the target network obtained in this way. On the composite network is then run the GRF algorithm, which was also implemented within the software architecture; this was done by using the conjugate gradient solver for systems of linear equations which is available with the Eigen libraries (see section 6.5.1). It must be noted that the conjugate gradient algorithm works with symmetric and positive-definite matrices. As a consequence, the input networks used to compute the composite network must be made symmetric and all their negative entries must be removed[2]. The inputs of the GRF algorithm are the association between network nodes and a certain feature, its output is a vector of discriminants which indicates the likelihood of each node of being associated with the feature. For the validation workflow, all the annotations residing in the k-th part of the matrix that was hidden during the composite network construction are set to zero, hence every feature in the annotation matrix is predicted through the composite network and the predictions of the values that were set to zero are compared with the prior evidence. This process is then repeated by swapping the k-th hidden part of the matrix with another portion still untested. At the end of the day, the ROC curves are plotted by computing the FP and FN rate while varying the threshold value which determines when a discriminant is considered as a positive prediction instead of a negative one.

7.4 Implementation Issues and Solutions

During the software architecture implementation, particular attention was paid to the aspect of computational efficiency of the code. As a consequence, a careful selection of the libraries adopted was made. These libraries have been discussed in the section 6.5 and their applications are treated later on in this section. However, to fully understand the issues related to the computational efficiency of the code, one should know what metrics were implemented in the system and what are the possible bottlenecks of the architecture. The construction of a simple network directly from node associations it is not a relevant case for this topic, as it shows no particular issue. Instead, what is important to know here is how the network constructed from annotation profiles are computed. The next subsection treats this topic.

[2] $M = M^T$ with $m_{ii} > 0$ and $m_{ii} > \sum_{j \neq i} |m_{ij}|$ is positive definite. See section 6.3 for how the matrix M is constructed

7.4.1 Computation of Networks from Annotation Profiles

The software system provides two default ways of computing similarity edge weights from annotation profiles. These are PCC correlation and Resnik similarity (see section 6.1); the former is used for flat profiles, while the latter applies to ontological profiles. Independently from the specific metric used, the computation of networks from annotation profiles resulted to be the most time consuming task among the ones performed while constructing simple networks. The reason is that a weight value must be computed for each pair of possible nodes (i.e. potential edges of the network), thus this operation has complexity at least $O(n^2)$. In addition to this, the way the weight is computed might add other complexity to the task. For instance, to get the PCC coefficient value, one has also to compute the dot product of the normalized profiles (i.e. the annotation profiles minus their averages), this add a factor k to the complexity of the task $O(n^2k)$, where k is the number of features in the annotation matrix (i.e. the dimensions of the annotation profiles). The Resnik metric is even more demanding and a way to face this problem is presented later on in this section. However, what it is important to know, for the right interpretation of the Performance results discussed in the next chapter, is the presence of the factor n^2 in the complexity of the computation of the weights, which, in source code, is translated into two nested loops.

```
for i=1 to number(nodes)
    for j=1 to number(nodes)
        apply_metric(ann_profile(i),ann_profile(j)
```

It is worth noting that some assumptions could reduce the factor n^2, for instance if the metric applied in the inner most loop is symmetric one would need to compute $n^2/2$ weights. Nevertheless, this may result in a loss of generality of the program and some situations could not be handled any more. Let's assume that the user have requested a specific local sparsification method to sparsify the network being constructed. As it was stated before, the local version of the sparsification works by sparsifying the network using the local information consisting of the weights of a node. This avoids of having a dense adjacency matrix allocated in memory. In fact, as soon as the inner loop - the one indexed by j in the pseudo code above - terminates the computation of the weight vector v_i (i.e. the vector of all the edge weights of node i), the vector can be sparsified. Now, if one use the assumption that the metric is symmetric to avoid of calculating the lower (or upper) triangular part of the adjacency matrix, there must be enough memory to allocate the intermediate results associated to incomplete weight vectors. In other words, when the inner loop terminates its execution, all the weights associated to the links of node i have been computed, thus this vector can be locally sparsified; on the other hand, the elements e_{zi}, where $i < z < number(nodes)$, are intermediate values of incomplete

weight vectors v_z. These values must be stored until the corresponding vectors are complete, only then they can be sparsified.

7.4.2 Computational Efficiency

A considerable portion of the time dedicated to this thesis was spent by investigating and developing solutions which are able to efficiently manage the tasks performed during the workflows associated to a network construction request. Two major parameters were taken into consideration when trying to optimize algorithms and code: the memory that is consumed by a task and the time required to perform a task. These two measures often conflicts, in the sense that when one tries to reduce the space needed to execute a certain task, it is frequently true that the time requested to complete it rises, the same the reverse. For this reason, a good solution is the one that finds the suitable trade-off between time and memory, depending on the user needs. It must be noted that the software architecture proposed is designed to work with large dimension of data, hence concepts as memory consumption and time responsiveness become key points of the whole architecture. This section wants to illustrate the technologies and solutions that have been applied to face those issues.

Some of the design and implementation choices that were adopted to address the requirements above, have been already mentioned. The one that has the greatest impact on the structure of the whole architecture is the decision of implementing the mathematical core part of the program with a native language, that is C++. The drawback of this solution is that the code loses the machine-independent feature typical of Java. In fact, the Java platform is a programming environment consisting of the Java virtual machine (JVM) and the Java Application Programming Interface(API). Java applications are written in the Java programming language, and compiled into machine-independent binary class format. The direct consequence of having this sort of middle layer in between the source code and the machine code is that Java applications are portable to different machines, indeed they are executed inside implementation of the Java virtual machine. By developing a central part of the system using the C++ programming language, the code needs to be recompiled if run on different machines. On the other hand, a native code language such as C++ permits to have a wider control over the memory allocation strategies to be put in place, moreover the fact that the code is directly compiled into machine code allows the compilers to optimize the code itself for being run on a specific machine. The other strategy being discussed earlier, that avoids to retain allocated large portions of the memory, even if they are not currently used, is the mechanism of serialization and loading implemented in the “Session” class (figure 7.17). If the user enable this function, all the networks and structures that have been already computed, and that will be accessed again only in a second moment, are temporarily serialized in files

onto the hard disk. When one of these structures is required for completing a task, it is reloaded into main memory. This process is completely transparent to the other classes of the system, indeed, being the “Session” class the central point where middle results are collected, all the classes that need a certain structure that is known to be already computed (e.g. an input network) ask to the “Session” class for it. Hence, the unique component in the system that knows whether a structure is allocated in main memory or it has been previously serialized onto a file is the “Session” class. The last trick discussed in the previous sections, which tries to tackle situations of memory exhaustion is the definition of local version of operations that otherwise are executed on the entirety of a subject. The example that was done before concerns the sparsification operation: instead of deciding which edges of the network must be removed by looking at the entire network, one might think at a procedure that is applied to every node of the network iteratively; at each node the procedure selects the node to be removed.

The three main approaches presented above, that try to reduce memory consumption and to enhance time responsiveness, may be seen as solutions that were planned at design time, meaning that they have some repercussions on the design of the software architecture. However, these strategies turned out to be not enough to achieve the results desired; besides those mechanisms, other technologies and solutions were applied in this project.

First of all, many of the implemented software routines were parallelized using the OMP API. So, for instance, the normalization of a network, the computation of network weights by means of PCC coefficient or Resnik similarity, the validation workflow, the construction of the matrix model for the least squares error minimization problem, they all exploit multithreading and parallel programming techniques to perform their tasks. Then, a careful selection of the C++ standard library structures was made depending on the operations being performed. So for instance, unordered_set and unordered_map were used when direct accesses to elements were the core part of the routines being implemented; they allow for constant time direct accesses to the elements they store. On the other hand, ordered maps and sets where chosen when preserving the order of the elements, while still having a low time direct access (logarithmic in the dimension of the structure), was important. Throughout the whole program, matrices were handled by means of sparse matrix formats (see section 6.4.1). This choice allows to reduce the memory allocated for the process, as the zero elements of a matrix are not explicitly put in memory. Eventually, the ACML library, which provides a free set of thoroughly optimized and threaded math routines for high-performance computing, was used to implement basic linear algebra operations, such as the dot product involved in the PCC coefficient computation.

7.4.2.1 The Resnik Similarity Computation Case

The metric implemented to compute network weights exploiting the DAG structure of ontological data was the Resnik similarity applied to annotation profiles (eq. 6.3). Bringing this computation to an acceptable time responsiveness has been, from a computational performance point of view, the most challenging task of this thesis. Besides the usual two nested loops, that iterate over the n nodes of the network to calculate one weight per each pair of node profiles, there is also one more inner loop. It computes, for each feature annotating one network node, the maximum information content among all the common ancestors of features annotating the other network node, thus the complexity of this task is O(n^2kq) where n is the number of nodes in the network, k is the number of features in a node profile and q is the depth of the DAG that must be investigated to find the common ancestors. At the end of the day, the strategy adopted was to parallelize the external loop over the n nodes thanks to the OMP directives, pre-computing the ancestors of each feature appearing in the annotation matrix and pre-computing, for each profile, the union of all ancestors of the features composing that profile. At this point, given a feature f and a set Q of all ancestors of a profile p, the maximum information content among the common ancestors between f and Q is found by moving upwards over the DAG from f, until f itself or one of its ancestors is found also in Q and there exist no other common ancestor with higher information content. The figure 7.19 represents this situation; given the two profiles p_1 and p_2, a single step of the Resnik similarity computation extended to profiles consist in seeking the common ancestor with maximum information content between the feature f_1 and all the features in profile p_2. As said, the strategy adopted is to pre-compute the set of all the ancestors of features in p_2, this is Q_2 in the figure 7.19, and to find the intersection between Q_2 and the iteratively selected ancestors of f_1. If the unfolding of the annotations was performed accordingly to the organization of the features into the DAG, then the following property holds: a generic parent node annotates a number of elements that is greater or equal to the maximum number of elements which are annotated by its children. This is translated into the fact that a parent node cannot have an information content greater than the information content of any of its children. This property is used as a stopping criteria when seeking for the common ancestor with maximum information content (see figure 7.19 description for an example). Besides trying to implement a clever solution for seeking the common ancestors, without comparing the ancestors of each pair of possible features, another trick was used to speed up the computation. It consists in caching some intermediate results that otherwise would be calculated ex-novo every time. The products cached are the values of maximum information content that are computed for pair composed by a feature and

an entire annotation profile; basically these values are the results coming out from the computation performed in the example of figure 7.19.

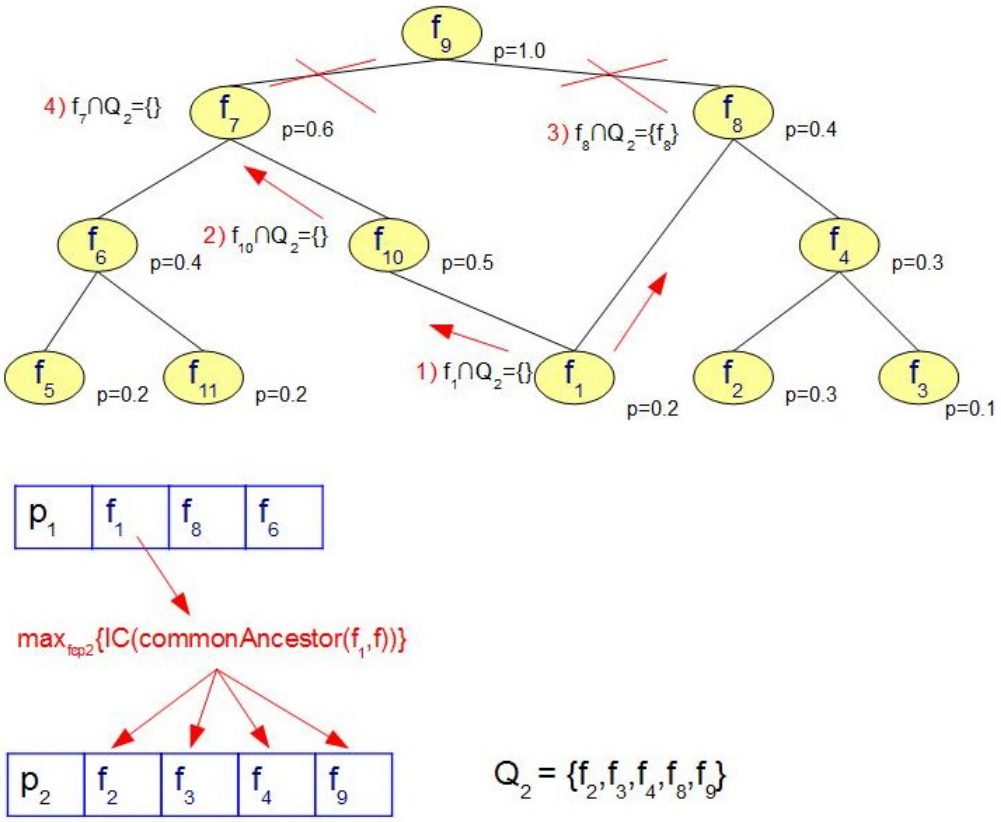


Figure 7.19 – Steps performed to find the maximum information content ancestor. $p_1$and p_2 are two annotations profiles. The picture represents the steps taken to seek the maximum information content common ancestor $max-a$ between a feature f_1 and an entire profile p_2. First of all, all the ancestors of features in p_2 are collected in the set Q_2. To find $max-a$ the algorithm starts from f_1(red point number 1 in the figure); since f_1 is not in Q_2,its parents are investigated (i.e. f_{10} and f_4); the first parent checked is f_{10}, it is also not present in Q_2, hence the research goes on by analysing f_4. f_4 is in Q_2, thus it is a candidate for being the sought $max-a$, moreover the branches starting from f_4 and going upwards are not further investigated because the probabilities of the features connected to them is greater or equal to the probability of f_4 (note that this is true only if the annotations were unfolded in a pre-processing step). A feature probability with a value greater than another one implies that the information content of the first feature is smaller than the IC of the second one. The next step is to analyse the branches that have not been removed, in this case the only one is the edge going from $f_{10}$to f_7. Since f_7 has a probability (p=0.6) greater than the one of the common ancestor already found (f_8, prob=0.4), we can stop the research here without moving upwards from f_7 to find the common ancestor deriving from this branch (i.e. f_9). The $max-a$ that has been found is f_8.

Furthermore, the fact that each feature has an associated frequency - stating how often the particular feature is used to annotate elements of the network (i.e. network nodes) - is used to drive the population phase of the cache. The concept is simple, given a fixed size cache, the values that are put into the cache are the one referring to features with the greatest frequencies among the annotated corpus set, hence they are likely to be requested more often than the others. Two structures

were used to implement this pattern: the first one is a simple C++ unordered_map that guarantees constant time access to data by using an hash function for arranging the data within the map. The keys of the map are composed by pairs of values, each on constituted by a feature label and an annotation profile index; the values inside the map are the similarities computed between a certain feature and an annotation profile. The second structure used to implement the desired behavior of the cache is a binary heap (figure 6.2), this allowed to retrieve the minimum frequency value in the cache in a constant time. Moreover, the addition of a new element is done in a logarithmic time with respect to the cache size, which is faster than keeping an array totally ordered (i.e. on addition of a new element O(n) moves are required in the worst case). A generic heap's element contains the pair feature-annotation profile it refers to and the frequency associated to the feature. Every time a similarity between a feature and an annotation profile has to be computed, its value is first sought inside the cache. If it is not present there, then it is computed following the procedure described above. The final move is to check whether the new value has to be inserted into the cache, this is done by comparing the frequency of the feature with the frequency of the value stored in the root of the heap. If the new value is more frequent than the old one, and the cache has already reached its maximum size, the old root is removed from the heap and the new element is added by preserving the partial order of the heap, meaning that the new value is pushed down until its frequency is not greater than the frequencies of both its children. The steps performed can be summarized by the following piece of pseudo-code:

```
Given a feature f and an annotation profile p,
the similarity s is found as:
if(<f,p> in cache)
    return cache.map.value(<f,p>)
else
    tmp_s = compute_similarity(f,p)
    if(cache.map.size < max_size)
        cache.map.insert(<f,p>,tmp_s)
        cache.heap.insert(<f,p>,f.frequency)
    else
        if(f.frequency > cache.heap.root.frequency)
            pair = cache.heap.removeRoot()
            cache.map.remove(pair)
            cache.map.insert(<f,p>,tmp_s)
            cache.heap.pushDown(<f,p>,f.frequency)
        else
            //tmp_s must not be cached
    return tmp_s
```

It is worth noting that, when multithreading is enabled, the cache mechanism introduced above must be protected by locks. This is necessary to prevent situations of data race. In particular, both the operations of testing and updating on the cache were implemented within a critical section, as defined in OMP.

7.4.3 Functions Included in the Software Architecture

As said earlier, one of the objectives of the software architecture was to provide a software which can be adapted to several situations that may arise in different contexts. The previous sections were mainly focused on the general architecture of the system, presenting its peculiar aspects which make it suitable for the given requirements. Besides having implemented the general infrastructure of the system, which can be subsequently extended, some default functions were developed in order to test the whole infrastructure and to pursue the objectives stated in the chapter 5. In particular, one of the goals of the project was to implement a method to derive a composite network from multiple input networks and to validate this network executing gene function prediction tasks. For this reason, the default metrics implemented in the software system were chosen to accomplish this last point, however, as it was explained before, the inclusion of a new functionality into the system has been thought at design time, hence it is an easy and effortless operation. The remainder of this section lists the default functions included in the system and the operations that can be performed on input data and networks:

Data Acquisition and Filtering

The first set of functions provided by the system pertains to the acquisition and filtration of input data. In case of a simple network derived from annotation profiles, the annotations given as input of the program are first checked for duplicates. Whenever duplicates are found, just one annotation among all the duplicates is retained. The annotations can also be filtered based on the fact that they are positive or negative annotations. If the negative annotations are enabled by the user, a consistency check is performed, that is: if there exist both negative and positive annotations for a given pair entity-feature, then all the annotations involving that pair are discarded. The annotations can also be pruned by specifying an interval of minimum and maximum number of entity annotated to a given feature. Assuming to specify an interval 3 - 300, all the features annotating less than 3 or more than 300 entities are discarded.

Annotations Unfolding

Specific to ontological annotation profiles, it is the possibility of performing the unfolding of the annotations based on the DAG structure retrieved from feature relationships.

Local Sparsification

The software system provides a version of local sparsification which is based on absolute adaptive thresholds. The user can suggest two thresholds defining an interval to be taken into account for the sparsification. Besides the threshold values, it is also possible to indicate whether to keep the interactions in between the interval or outside. The threshold values denote the place where to set the placeholders of the interval on the ordered list of weights of a given nodes; starting from the lowest weight, the first placeholder is placed after counting n weights, where n is the value of the minimum threshold. The second placeholder is placed counting downward from the top valued weight, its location is found after counting k (i.e. the value of the maximum threshold) weights. The two placeholders identify an interval over the ordered list of weights, a flag specified by the user indicates whether to keep the weights in between the placeholders or outside. An additional setting of the implemented local sparsification logic is represented by a flag which activate or inactivate the selection of negative weights.

Local Log Transformation

This functions is available just for flat annotation profiles. The operation performed is to transform a sparse flat annotation profile into a dense one by changing the one values with the information content of the feature associated to that annotation and the zero values with the logarithm of one minus the frequency of the feature in the annotated corpus set.

PCC Correlation

This function was implemented as default function for computing the similarity weights of nodes starting from their flat annotation profiles. How the operation is performed is describe in section 6.1. It is worthy to specify that, in case of dense vectors, this function was implemented using the ACML library utilities. Another implementation of the PCC correlation coefficient was implemented for working with sparse vectors.

Resnik Similarity and its Modification

The default edge weight computation metric for the ontological annotation profiles is the Resnik similarity. How this measure was implemented in the software architecture has been already discussed. The user can also activate a modification of this measure that was implemented when testing the gene function prediction task. It consist in defining a threshold which represent a level of DAG depth (or a value of information content). Pair of features which have the maximum information content

common ancestor with depth (or information content) lower of this threshold are assigned a negative similarity calculated by performing the logarithm of one minus the frequency of the common ancestor.

Normalization

The standard procedure of normalization that the system uses is based on the following formula:

$$\tilde{W} = D^{-1/2} W D^{-1/2}$$

where D is the diagonal matrix having diagonal values equals to the sum of values of the respective columns.

Symmetrization

The symmetrization function implemented within the software infrastructure adopts the following logic: if the elements a_{ij} and a_{ji} of the matrix A have the same sign, then their new values are the average of the two original values. If they are of opposite sign, they are both set to zero. Otherwise, if one of the two elements is equal to zero, its value is changed to the one of the other element.

Composition through LSQR

The composition task implemented is based on a method which minimizes the least squares error between the target network and the input networks. For details relating to this method see section 6.2.

Validation through GRF

Finally, the validation method that was implemented is the GRF algorithm [69] (see section 6.3).

Chapter 8

PERFORMANCE OF THE IMPLEMENTED SOFTWARE SYSTEM

The following chapter is dedicated to the assessment of the system's performance in terms of amount of memory and run time requested by the tasks performed during the construction of networks. All the tests here reported uses data retrieved from the GPDW[13] data warehouse of the Bionformatics and Web Engineering lab. of the Politecnico di Milano. While the next chapter try to validate the network composition methodology with biological use cases, this part of the thesis concerns just the performances of the software implemented, as a result some tests might seem logically incorrect if seen under a biological perspective.

The test are run on a machine with a processor i7-2670QM, 2,2GHz (4 cores, 8 threads); Level 1 Cache 256 KB, Level 2 Cache 1024 KB, Level 3 Cache 6144 KB. The amount of RAM memory available is 4GB. The data are retrieved from an external hard disk connected through a USB 3.0 port. All the tests were performed by enabling the multithreading code that was implemented.

8.1 Comparison of Performances for the Single Network Construction Cases

The first set of tests aimed at comparing the different tasks performed when executing one of the three branches of the workflow run for building complex networks (figure 7.4). Which specific branch is performed is determined by the type of data used to construct the network. All the input data used for this set of tests pertain the genes of the Homo Sapiens specie (taxonomy id 9606 within the GPDW data warehouse).

The test results presented throughout this section are shown in tabular form. For each network the following parameters are provided:

1. Number of nodes (#nodes, as named in table headers);
2. Number of edges (#edges);
3. Number of features; this voice is present just if the network is computed from annotations (#features);
4. Number of annotations; this voice is present just if the network is computed from annotations (#annotations);

5. Coefficient specifying the sparsity level of the matrix; this is calculated with the formula $\frac{\#edges}{\#nodes * \#nodes}$ (s)
6. Time spent on single tasks executed to build the network; all the times are provided in milliseconds (t-nameOfTheTask)
7. Peak of memory allocated by the process; memory data are provided in MegaByte (pm)

8.1.1 Networks Constructed from Node Associations

The first case considered is the simplest one, that is when the data retrieved from the external source - the GPDW data warehouse in this case - already define the network nodes associations. The table 8.1 reports the results of the test executed by retrieving gene associations stored in the table gene_relationship of GPDW.

#nodes(genes)	#edges	s	t-data_acq	t-network_b	t-normaliz	t-symmetriz	pm
99703	73203	7,45E-006	1321	7,565	4,758	133,124	16

Table 8.1 – Table of results obtained when constructing the network directly from network interactions. t-data_acq is the time for data acquisition and filtration; t-network_b is the time for network construction; t-normaliz is the time for normalization of the adjacency matrix; t-symmetriz is the time for symmetrization of the matrix.

As the table shows, for this simple case the greatest part of the time was spent by retrieving the data from the data warehouse (i.e. t-data acq in the table). The column header named as t-network_b refers to the time that it took to create and populate the sparse matrix representing the network. The two last time-related columns (i.e. t-normaliz and t-symmetriz) are associated to the final tasks common to all the branches of the network construction workflow (figure 7.4). The normalization processes performed throughout this section divide each element ij of the sparse matrix for the product of the sum of elements in column i and elements in column j. Since sparse matrices are stored in a Compressed Column Storage format, this operation can be performed very quickly; moreover this method was implemented using the OMP library, and so all the eight processor threads were exploited. Apart from the data acquisition task, the symmetrization is the computation which dominates the total time of the process. One of the reason is that the symmetrization procedure implemented is not yet parallelized. However, the main cause of its high computation time, with respect to the other tasks performed on the sparse matrix, is the format of the sparse matrix used. In fact, the procedure that was implemented to symmetrize a matrix consist in substituting the values of elements in position ij and ji with their average, if both elements are non-zero values; otherwise, the value different from zero is used in place of the zero value. Adopting a Compressed Column Storage format it is impossible to have a direct access to an element lying at

a given row, hence one should build a transposed copy of the sparse matrix to gain accesses by rows. This also implies a growth of the memory allocated to complete the task.

8.1.2 Networks Constructed from Flat Annotation Profiles

The next test is about building a network from flat annotations to a certain set of features. For this test, I chose to set the network nodes to the group of protein coding genes of Homo sapiens, while the features annotating those genes were retrieved from the pathway table of the GPDW. The results are shown in table 8.2. As one can see, this test involves the execution of a larger number of tasks with respect to the previous case. Moreover two different matrices are constructe; the first one (i.e. the annotation matrix), characterized by the nodes, the features and the annotations, is used to store the annotation profiles of each node; the second one is the matrix representing the network.

#nodes	#features	#annotations	#edges	s	t-data_acq	t-annMat_b	t-network_b	t-normaliz	t-symmetriz	pm
20254	229	16388	533512	0,0013	1494,96	1,623	5,78E+004	94,839	738,95	30

Table 8.2 – Table of results obtained when constructing the network from flat annotations. t-data_acq is the time for data acquisition and filtration; t-annMat_b is the time for annotation matrix construction; t-network_b is the time for network construction; t-normaliz is the time for normalization of the adjacency matrix; t-symmetriz is the time for symmetrization of the matrix.

Before speaking of the central computation of this test (i.e. the computation of network links starting from the annotation profiles), it is worth specifying that a local data transformation and a local sparsification algorithms were applied to data; their computation time is included under the t-network_b voice. Given that the network being constructed had 20254 nodes, without applying a local sparsification algorithm the peak of memory would have been at least 20254 x 20254 x sizeof(dataType) $\approx$1,53 GB(using 4byte dataType), much more than the actual 30 MB (table 8.2).

It is evident from the data in the table that the bottleneck of the whole process is the voice t-network_bu, which corresponds to the computation of the network edge weights starting from the annotation profile of each node. In this specific case the metric that was used to compare annotation profiles was the PCC correlation (eq. 6.1). Let's now look under the hood what happens when computing network weights. For each network its profile is transformed by the local transformation method, next, starting from this new profile, the PCC correlations with all the other transformed profiles are calculated. Roughly speaking, the PCC correlation computation requires to calculate the averages of the profiles, the sum of squares of their errors and the dot product of the errors of the two profiles. For this reason, and considering that the number of nodes in the network is 20254, the time of about 58 seconds is well

acceptable. After all, the flexibility and modularization of the system put some limitations onto the optimizations that would have been possible if the system had been designed for specific tasks. For instance, a generic implementation of a metric which computes network weights sees the local transformation method of the profiles as an external agent. It only knows its interface, and the fact that it is supposed to call the transformation method on the annotation profiles before computing the weight from two profiles. Hence, it is not possible to do specific assumptions about the transformed profiles; if it would have been possible to assume that the returned transformed profiles were sparse, then a generic method that computes the network weights could have cached all the transformed profiles without worrying so much about the space that this operation had required. The final benefit would have been transforming n profiles instead of n^2 , where n is the number of nodes of the network.

8.1.3 Networks Constructed from Ontological Annotation Profiles

The last case of single network construction is the one related to ontological data. This is the most time consuming scenario (table 8.3), due to the metric that was implemented and used to generate network weights from annotation profiles (i.e. the weights computation takes about 55 minutes) .

#nodes	#features	#annotations	#edges	s	t-data_acq	t-annMat_b	t-network_b	t-normaliz	t-symmetriz	pm
20254	5899	172844	894500	0,0021	5654,44	39,456	3,30E+006	197,814	1409,42	200

Table 8.3 – Table of results obtained when constructing the network from ontological annotations. t-data_acq is the time for data acquisition and filtration; t-annMat_b is the time for annotation matrix construction; t-network_b is the time for network construction; t-normaliz is the time for normalization of the adjacency matrix; t-symmetriz is the time for symmetrization of the matrix.

It must be said that, computationally speaking, the data used for this test represent one of the worst situation. In fact, the ontological annotations were retrieved from the GO BP ontology (excluding the annotations with code IEA, Inferred from Electronic Annotation), which is the largest and deepest ontology in the GPDW data warehouse; moreover, the Homo Sapiens species is one of the most annotated species with respect to the GO BP ontology. It is also worthy to note that the time of about 55 minutes was taken without using the caching technique discussed in section 7.4.2.1; when caching is enabled, the run time drops to less than an half of the original time.

The benefits in terms of required computation time, at different levels of cache size, are shown in figure 8.1 together with the respective dimensions of allocated memory. As one can see, the time taken by the process changed from a value of about 55 minutes - when no cache was used - to nearly 20 minutes, when the cache

size was set to its maximum. The figure 8.1 also shows that after a certain cache size the advantages brought by augmenting the dimension of the cache were not so important. In fact, due to the cache population mechanism, the greater is the dimension of the cache, the larger is the number of 'infrequent' features that fit into the cache. The limit situation is when the cache is large enough to accommodate all the values, this makes the threads to spend more time waiting for gaining access to the cache (due to the frequent execution of the single-threaded code for inserting new elements into the cache itself).

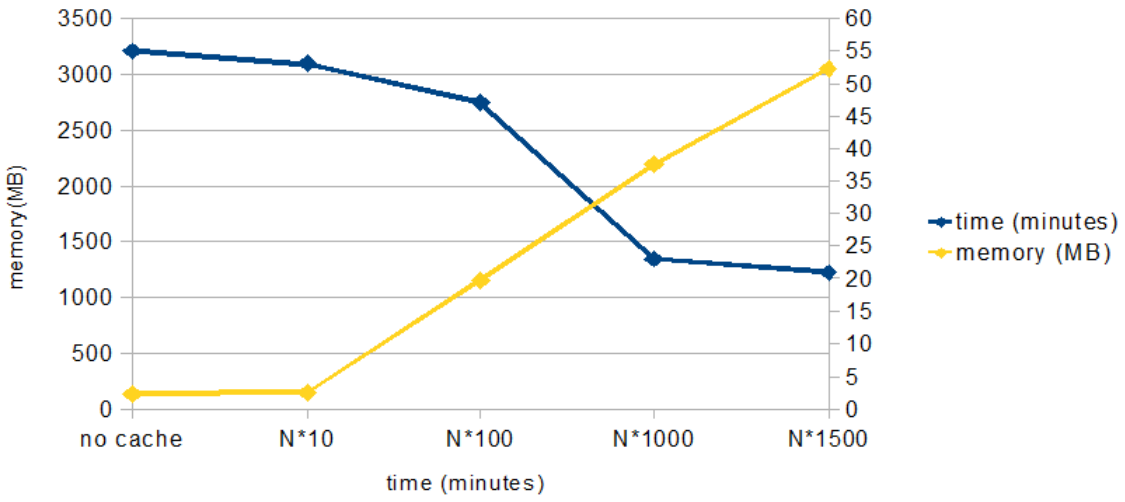


Figure 8.1 – Graphs of cache performances. The time (blue line) required to perform the edge weights computation task, at different degrees of cache size. The memory allocated to store the cache structure is reported on the yellow line. The dimension of the cache is given on the x-axis, it is expressed by factors of the number network nodes (N), that in this specific case is equal to 20254.

8.2 Performance of Composition Workflow

This section reports the time spent by the steps performed while composing input networks with respect to a target network. The composition task was set up with the following parameters:

- Nodes of the networks: protein coding genes of Homo Sapiens species;
- Target network: network constructed from the ontological annotations between genes and GO[8] Cellular component features. The sparsification threshold was set to 100, meaning that only the top 100 links per each node were retained;
- Input network 1: network built from flat annotations between genes and pathways. The sparsification threshold was set to 100;

- Input network 2: network built from flat annotations between genes and genetic disorders. The sparsification threshold was set to 100;

- Input network 3: network built from flat annotation between genes and publication references. The sparsification threshold was set to 100.

The dimensions of the networks constructed for this test are given in the table 8.4

role	#nodes	#edges
target	20254	508200
input network 1	20254	533512
input network 2	20254	7932
input network 3	20254	1909255

Table 8.4 – Table of the number of nodes and edges of the networks used in the test.

. The process for composing networks, as it was described in the section 6.2, may be divided into three main steps: the composition of the model training matrix by stacking the input network adjacency matrices columns one on top of the other (model derivation), the resolution of the least square problem (learning), the computation of the composite network given the weights calculated in the previous step (network composition). The times spent in each phase are:

- model derivation = 5012.13 ms;

- learning = 20.774 ms;

- network composition = 3873.76 ms.

These results show that the composition process is not the bottleneck of the whole workflow. In fact, these times are two orders of magnitude smaller than the times required to build a network from ontological annotation profiles of GO BP. Another interesting observation is that the composition workflow can be split from the single network construction workflow. This way, it is possible to pre-compute the input networks for a certain number of data sources. Then, only in a second moment, the networks can be integrated together, on the fly, according to the user needs. A basic implementation of this pattern was implemented for the testing and validation phase. Before validating, the single networks constructed from a user request can be saved onto files by enabling a 'save' flag, which is part of the network descriptions, in the xml file. Later on, the user can start a composition workflow by specifying the files where the networks were serialized, instead of describing, with the input xml file, how they should be constructed from ex-novo.

8.3 Parameters Incidence on Performance

The previous sections have reported the performances of the system by analysing a single use case at time. The data used were the protein coding genes of the Homo Sapiens and their annotations to different type of features. As said before, that case was one of the most demanding from a computation point of view, due to the high number of annotating features available for that organism (this is specially true for the case of GO BP annotations). In this section, I performed another set of tests in a similar fashion of the ones executed before; however, what changes is the goal of the experiments. While the sections before presented the main performance aspects associated to single experiments, this section wants to compare the time performances achieved on different realistic use cases when changing the parameters of the software and the data sets used. The biological results obtained changing the system parameters are discussed in the next chapter.

The series of tests being reported here was performed by using the protein coding genes of the Arabidopsis thaliana organism. In the last two decades, Arabidopsis thaliana has gained much interest from the scientific community as a model organism for the genetics. Its genome was the first to be completely sequenced within the plant kingdom. Since it was considered a model organism, its genome has been intensively studied and characterized, hence it is a good choice for validation and testing phase. The number of protein coding genes of this organism, as retrieved from the GPDW data warehouse, is 27937.

In this section too, the results are displayed in tabular form. The column headers of tables have the following semantics, in order of appearance:

1. The type of data (Type; this flag can be 'flat' for flat annotation profiles and 'onto' for ontological annotation profiles; when the flag is onto the annotation profiles were unfolded according to the DAG structure of the source);

2. The source data set of the features. The data were always collected from the GPDW data warehouse; here source refers to the feature type definition given within the GPDW; Annotations from the Gene Ontology with the code IEA were not taken into account (source);

3. The minimum and maximum thresholds used to filtrate the annotations (min-max);

4. The number of annotated nodes (#nodes);

5. The number of annotating features (#features);

6. The number of annotation (#annotations);

7. The maximum depth of the annotating features; this information is present only if the source is treated as an ontological source (depth);
8. The time for data acquisition, and eventually unfolding (t-data_acq);
9. The time for computing network weights, including the phases of sparsification and profile transformation (t-network_b);
10. The time spent to normalize the adjacency matrix (t-normaliz);
11. The time spent to make the adjacency matrix symmetric (t-symmetriz).

As it was for the previous sections, the time provided throughout these tests are in milliseconds.

Performances when no Annotation Filtration Is Performed

To begin with the analysis of the impacts that changing in parameters values have on the time responsiveness of the system, I constructed a series of networks without performing any filtration of the annotations. The sparsification thresholds of all the networks were set to 30; this way just the top 30 interactions per each node are kept (table 8.5).

type	source	#genes	#features	#annotations	depth	t-data_acq	t-network_b
Onto	BP	10537	3765	255042	18	2835	1,49E+006
Onto	MF	13074	2219	93958	14	2001	653297
Onto	CC	9757	569	136129	14	1733	449989
Flat	Pathway	3543	123	7713		832	8077
Flat	Pub Ref	27397	10622	65703		20813	37880

Table 8.5 – Table of the performances related to the construction of different networks. No annotation filtration was performed on the input data; the sparsification value was set to 30, meaning that each node of the network has at maximum 30 links to other nodes

Looking at the table, it is possible to see that the construction of the network from annotation data retrieved from GO BP was the most time consuming task. This issue was already noted in the first section, for the Homo Sapiens species, and here this result was confirmed. However, it must be noted that, constructing networks from the other two ontologies of GO (i.e. MF and CC) took less than half of the time spent for the BP case. Networks built from flat annotations were computed in really short time. One thing is worth mentioning, the building process for the network constructed from publication references (Pub ref in the table) did not include any transformation of the annotation profiles, whereas the annotation profiles derived from Pathways were transformed using the default transformation function implemented in the system, see section 7.4.3. This choice was made due to the size of the annotating feature set of Publication references (i.e. 10622 as reported

in the table). It was possible to save a lot of time by avoiding the transformation of annotation profiles into dense real valued vectors. As a matter of fact, in the system architecture, it was also implemented a function that can calculate the PCC correlation between sparse vectors. This is much faster than the dense calculation. The repercussions on the gene function prediction task, that this different approach may have, is discussed in the next chapter. To have an idea of the difference between the time spent for the dense computation instead of the sparse one, the network from Pathway was constructed with the same settings as above, except the fact that the sparse implementation of the PCC correlation was used. These are the two times for the voice t-network_b:

- PCC applied to transformed dense vectors, t-network_b = 8077 ms;
- PCC applied to original sparse annotation profiles, t-network_b = 836 ms.

The two times are about one the tenth of the other.

Performances on Filtered Annotations

The next table, number 8.6, reports the time spent on single tasks executed for constructing the same network as before, with the only difference that some annotations were filtered out from the input data. All the annotations of the different sources used before were filtered by keeping the terms that annotate at least 3 genes and at most about one percent of the number of genes that were annotated in the previous tests. This information is given in the column 'min-max' of the table. The table 8.6 is composed also of two additional columns, representing the times spent for the normalization and the symmetrization of the adjacency matrix, respectively.

type	source	min-max	#features	#annotations	depth	t-network_b	t-normaliz	t-symmetriz
Onto	BP	3-105	2230	41281	15	200827	37	248
Onto	MF	3-130	1063	17857	13	87402	32	237
Onto	CC	3-97	321	5900	14	7716	16	52,96
Flat	Pathway	3-35	62	1054		499	11	14
Flat	Pub Ref	3-273	2069	18596		4405	65	254

Table 8.6 – Table of the performances related to the construction of different networks. Annotations were filtered based on the number of genes they annotate; the values min-max specify the interval used to select annotations, the ones annotating a number of genes which lies between this interval are retained; the sparsification value was set to 30, meaning that each node of the network had at maximum 30 links to other nodes

As the table shows there was an evident improvement of time responsiveness for the software system. On average, the percentage of time saved during the edge weights computation phase (t-network_b in the table), with respect to the previous data, was about the 90,7% (see figure 8.2). This is an important result, however its implications on the effectiveness of the composite network, in performing the task

it was built for, should be investigated; this is done in the next chapter, where the composite network is used to predict gene functions.

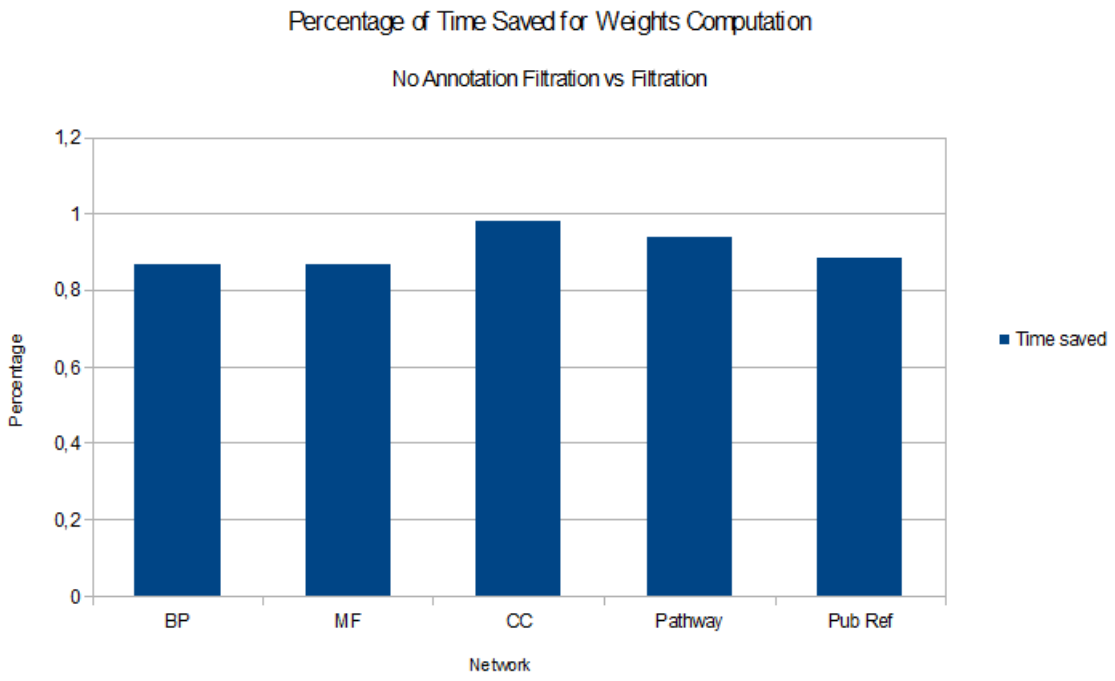


Figure 8.2 – Bar diagram of the percentage of time saved during the computation of network edge weights when annotation are filtered,with respect to the case without any filtration. The five bars refer to the five network constructed from different data, see table 8.6

Performances on Less Sparsified Networks

At this point, I investigated the changes in time responsiveness when the networks constructed are less sparsified. In these experiments, the sparsification thresholds of the five networks being constructed were set to a value of 150 instead of the old value equals to 30. The statistics collected are reported in table 8.7.

type	source	min-max	t-network_b	t-normaliz	t-symmetriz
Onto	BP	3-105	200827	284	1440
Onto	MF	3-130	87402	257	1440
Onto	CC	3-97	7716	78	352
Flat	Pathway	3-35	499	6	16
Flat	Pub Ref	3-273	4405,33	203	1027

Table 8.7 – Table of the performances related to the construction of different networks. Annotations were filtered based on the number of genes they annotate; the values min-max specify the interval used to select annotations, the ones annotating a number of genes which lies between this interval are retained; the sparsification value was set to 150, meaning that each node of the network has at maximum 150 links to other nodes

The table presents only the times for the normalization and symmetrization tasks because changes in the values of the sparsification threshold have no effects on the time spent to perform operations that stay before the sparsification phase, in the

network construction workflow. The percentage of time saved with a sparsification threshold of 30 with respect to a sparsification threshold equals to 150 is plotted in figure 8.3. Also this time the relative improvements on the performance were significant. Nevertheless, the times associated to the normalization and symmetrization operations are much lower than the times requested to compute network weights. As such, the benefits, in terms of the total amount of reduced time over the whole process, when passing from a sparsification threshold of value 150 to one of value 30 are negligible.

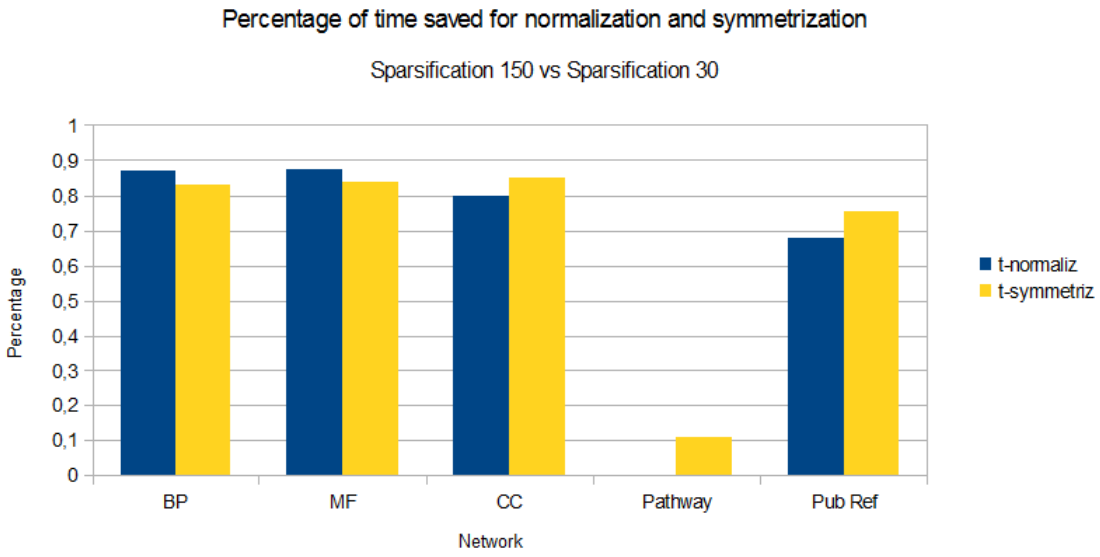


Figure 8.3 – Bar diagram of the percentage of time saved during the normalization (blue bars) and symmetrization (yellow bars) when the top 30 interactions per node are retained, instead of a value of 150. The five bars refer to the five network constructed from different data, see8.7

Comparison of Parameters Changing on the Composition Process Performance

To conclude this chapter I report also the times spent executing the three phases of the composition workflow when the system parameter changes. The composite network was computed from the five networks used in the previous series of experiments. The network built from BP annotations was used as target of the algorithm. The table 8.8 shows the results.

filtration	sparsification	model derivation	learning	network composition
no	30	151,093	5,585	701,45
yes	30	94,098	3,35	144,99
yes	150	336,447	20,101	541,28

Table 8.8 – Table of performances with different filtration and sparsification setting.

The composition process took less time when both filtration of the data and sparsification were activated. The filtration of the annotations used to build the input

networks were the same applied in the previous set of tests. Even if a fall in the execution time was achieved by filtration and sparsification, it must be said that the times obtained without filtration or with a sparsification threshold equals to 150 were also good results.

Chapter 9

TESTING AND VALIDATION OF COMPOSITION WORKFLOW

This chapter of the thesis presents and discusses the results obtained by applying the composition workflow to data retrieved from the GPDW[13] data warehouse. The first set of tests pertains to a small group of data from the Gallus gallus species; the tests performed aimed at characterizing the role played by the free variables of the composition workflow. The second series of experiments targets the evaluation of the benefits that an integration of multiple data sets into a unique composite network might have on a gene function prediction task.

9.1 Gene Ontology Annotation Prediction for the Gallus gallus Species

As discussed in the introduction of this thesis the Gene Ontology is perhaps the most famous and used ontology to annotate new biomolecular entities. Its branches are designed in order to encode orthogonal aspects of the biological features related to biomolecular entities. However, the patterns of associations between biomolecular entities and features of a certain GO subontology (i.e. Biological Process (BP), Molecular Function (MF) and Cellular Component (CC)) may be present also within the other subontlogy.

While investigating the repercussions that changes of network parameters have on the prediction results, these experiments also compares the relations intervening between annotations to terms of the different parts of GO. More precisely, given a target network constructed from a given GO subontology the question issued was: to which extent is it possible to predict annotations - of a given corpus of elements - towards a GO branch X, using annotations of other GO subontology?

The data involved in this set of tests are the genes protein coding of the Gallus gallus species and all the annotating terms of the Gene Ontology, except for the Inferred from Electronic Annotation ones (code IEA). Both these types of data are available from the GPDW data warehouse of the Bioinformatics and Web Engineering lab. of the Politecnico di Milano, as such I used this source of data for my experiments. The table 9.1 shows the dimensions of the data; Max-depth is the max depth among the terms that annotate a protein coding genes for Gallus gallus. As one can see from the table, the data source which have the larger number of

annotations towards the considered genes is BP. This is the usual case, in fact the Biological Process ontology is the most used of the three GO ontology, and it is also the one appearing more frequently in the research literature.

Ontology	Max-Depth	Positiv. Ann	Negativ ann	After unfolding	# Genes	#Annotated Features
BP	18	2575	17	32057	17246	3488
MF	14	886	10	4310	17246	728
CC	14	1038	4	8015	17246	439

Table 9.1 – Data dimensions for the Gallus gallus species. The columns reports, in the order, the branch of GO; the max depth among terms annotating genes of Gallus gallus; Positive ann and Negativ ann are the positive and negative annotations, respectively; After unfolding is the number of annotations after that unfolding has been performed; #Genes is the number of protein coding genes for Gallus gallus species; #Annotated features is the number of features of the respective GO ontology which have at least one annotations with a Gallus gallus genes.

It is also worthy to provide the distribution of the frequencies of the GO terms with respect to the size of the set of genes they annotate (figure 9.1). From the figure it is possible to observe that the major part of the features annotate just a small portion of the all genes considered here; while the features annotating large portion of data are a few. Another observation is that the DAG structure of GO is reflected on the graph by the increasing distance between bars, in fact, when the unfolding of annotations is performed, the more one moves upwards to the DAG structure the larger is the set of genes annotated by the encountered terms.

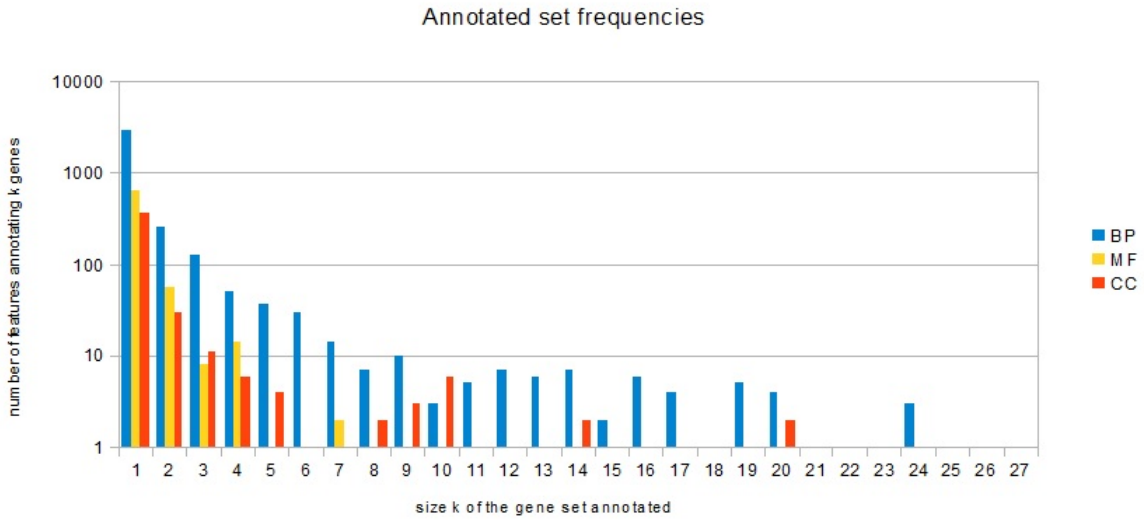


Figure 9.1 – Graph of the GO annotated gene set frequencies. The x-axis of the graph indicates the dimension of the annotated set of genes associated to a term. Each bar denotes an interval of of length 10, the interval is found with the following relation: (index-1)*10 +1<=k<index*10. The y-axis reports the number of terms that annotate a corpus with size included in the respective x-axis interval.

9.1.1 Evaluation of Parameter Impact on the Prediction Task

The first network parameters that were considered in this series of tests were the minimum and maximum thresholds used for filtering the annotations (i.e. the pruning phase). The values of the two thresholds denote the minimum and the maximum number of genes that an annotating term has to annotate in order to be included in the set of annotations that will be used to construct a network. The goal of these experiments was to assess the impact that different values of filtration thresholds had on a gene prediction task, which was performed on the network constructed from the annotations themselves. At first, the prediction algorithm (see section 6.3) was run on a simple network (computed without composing multiple networks) trying to predict the annotations of genes to GO BP terms. The network was obtained using the Resnik similarity to compute network edges from annotations to GO MF terms. The method used to validate the results was the k-cross validation technique, with a valued of k equals to 4. Basically, the annotation matrix (genes x terms) constructed from GO BP was divided into 4 subparts, after having shuffled the order of its genes (i.e. row of the matrix). One of this subpart is retained as test set, as a consequence all its annotations were set to zero. The rest of the matrix was used to predict the removed annotations. The predictions were performed one term at time, by providing the prediction algorithm with the column of the annotation matrix corresponding to that term. When all the terms were predicted, the procedure started again by swapping the test set with one of the subparts of the matrix still untested, this was repeated until all the matrix annotations were predicted. The prediction accuracy was evaluated by means of ROC curves and AUC (Area under the curve). On the x-axis of the graph there was the False Positive rate (FP) - the rate at which an error, associated to the event of a label prediction where it wasn't originally present, occurs - instead, the y-axis reported the False Negative rate (FN); false negative means that an annotation is missing with respect to the original annotation profile. Since the graphs plotted have FP and FN rate on the axis, the smaller their AUC values are the better the predictions. The results obtained by varying the set of annotations used to construct the network from MF annotations are presented in table 9.2 The annotations considered for the tests were selected by specifying the values of the filtration thresholds. In the table 9.2, these values are displayed in the column 'Pruning'

Type	Source	Pruning	Sparsification	#Genes	#Features	#Annotations	AUC	Line
Annotation Profiles	BP	5-80		610	874	15454		
Input network	MF	0-50	10	365	385	783	0,06731	green
Input network	MF	3-50	10	233	61	400	0,07315	blue
Input network	MF	0-10	10	336	378	668	0,06751	red
Input network	MF	0-20	10	352	383	736	**0,06675**	violet

Table 9.2 – AUC results for different pruning configuration. Type denotes the role of the subject in the prediction task; Source indicates where the annotations come from; Pruning defines the interval for the terms to be considered: The terms that annotate a number of genes within this interval were kept to construct the network; Sparsification is the maximum number of links per node; #Genes (#Features) is the number of annotated genes (annotating features) taken from the set of genes annotated in the Annotation matrix (that is the set of Gallus gallus genes annotated to BP terms, which is constituted by 616 genes); Line refers to the associated line in the graph 9.2

The annotation profile of each gene was built by considering just those terms that annotate a number of genes in between the values 5 and 80, in order to avoid to include in the evaluation process terms that are too specific or too generic, respectively. As the table shows, the best result was achieved with a pruning range set from 0 to 20. In particular, it is interesting to note that, by removing the terms that annotate a few genes, the performances are significantly worse, whereas, by including a wide range of annotations (i.e. 0-50), the performance are quite similar to the best AUC obtained. Since the prediction algorithm (see section 6.3) uses also the network structure to perform prediction of node labels, it should be important how the edge weights of the networks are computed. In this test the weights were computed by calculating the Resnik similarity between annotation profiles of nodes. In this respect, one could explained the results obtained (table 9.2) as follows:

- Removing the most informative terms (i.e. the terms that annotate a few genes) is counterproductive;

- Keeping the most general terms in the annotation profiles produce similar results than dropping them; however, one should consider also the time performance issues related to the usage of all the available set of annotating features (see previous chapter).

Calculating the edge network weights using also the most informative annotating terms helped the prediction task. This may have a logical interpretation when thinking at how the Resnik metric works. Since the Resnik metric is based on the maximum information content of common ancestors between terms, including terms with high DAG depth into the annotation profiles does not hurt the calculation.

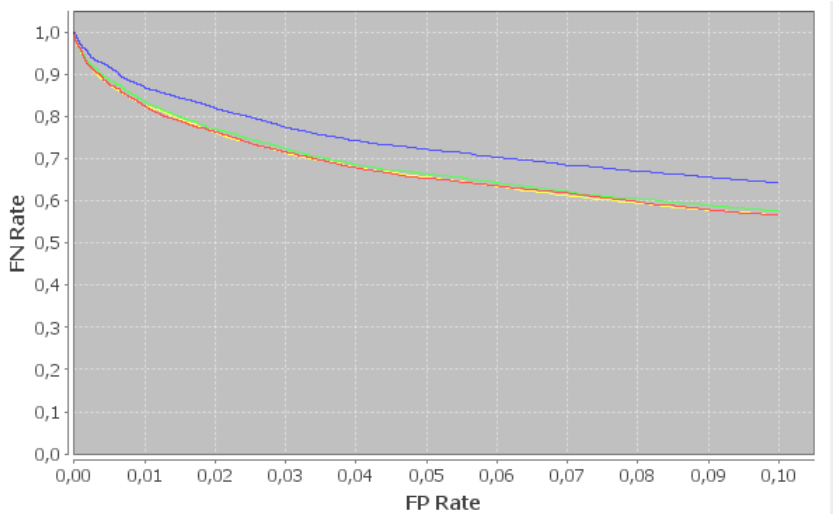


Figure 9.2 – ROC curves at different ranges of feature selection. Refer to the table 9.2 for a complete description of the data

Another parameter that can influence the prediction algorithm is the sparsity of the network, that is a threshold on the maximum value of links per node. Different AUC values for different sparsity levels were computed (table 9.3); this level appeared to be important for the prediction of annotations, a low level of sparsity, such as the value 3 in table 9.3, does not guarantee enough information for the predictor. On the other hand, a value too high, as it can be 50 for a network of 616 genes, is likely to bring in the network a consistent portion of uninformative links. A more detailed analysis of this parameter would probably help the prediction task performed over the network. One simple alternative to an adaptive threshold can be obtained by using a fixed value to perform the selection of links, in this way it is also possible to have networks that do not present an homogeneous degree distribution. However, such a technique depends on the characteristics of the metric used to compute weights and on the data themselves, in fact a threshold value should be sought for every specific case. The experiments performed in this section all used an adaptive threshold sparsification level, however it will be worthy to investigate cleverer solutions as future works.

Type	Source	Pruning	Sparsification	AUC
Annotation Profiles	BP	5-80		
Input network	MF	0-50	3	0,07897
Input network	MF	0-50	10	**0,06697**
Input network	MF	0-50	20	0,06765
Input network	MF	0-50	50	0,07034

Table 9.3 – AUC results for different sparsity level.

9.1.2 Comparison of GO Ontology Dependencies

For the remaining tests, being performed in this section of the chapter, I set a pruning range of 0-20 and a sparsity level of 20; with these values I tested the effectiveness of simple networks, computed from a given branch of GO, when they are used to predict annotations belonging to another branch. Moreover, the composite network obtained by integrating two different networks was also used to predict annotations of the GO subontology left out of the composition. The AUC results are reported in table 9.4

Type	Experiment #1			Experiment #2			Experiment #3		
	Source	AUC	Line	Source	AUC	Line	Source	AUC	Line
Target	MF			CC			BP		
Input network	CC	0,06885	a)blue	MF	0,06927	b)red	CC	0,0769	c)blue
Input network	BP	0,04478	a)green	BP	0,0519	b)blue	MF	0,06765	c)red
Input network	CC			MF			CC		
Input network	BP	**0,04375**	a)red	BP	0,0494	b)green	MF	0,6429	c)green

Table 9.4 – AUC results for different prediction tasks of GO annotations. In each experiment the target is an annotation matrix derived from a different ontology of GO. The first two lines reports prediction calculated upon a single network, the last line reports the prediction computed using a composite network assembled from the annotations of two branches of GO. The column header line refers to the color of the ROC curve in figure 9.3, the letter indicates which of the three graphs the curve refers to.

and the respective ROC curves are in the figures a),b) and c) 9.3. Some interesting observations can be made regarding the results obtained. First of all, one can notice that when the annotations to be predicted were in CC or MF, the best AUC scores were obtained by including the network of similarities computed from terms in BP; in particular, when predicting CC annotations, the top AUC score was obtained by integrating both the similarity networks obtained from MF and BP. This probably means that the information in BP and in MF are not redundant when trying to predict CC annotations. On the other hand, predicting MF annotations using the composite network from BP and CC or using just the BP derived network did not change a lot on the final AUC score (0.04375 vs 0.04478). This might be seen as the fact that BP provided the most important source of information when predicting MF information, hence adding the similarity network computed from CC did not add any useful information. The stronger relationship between BP and MF with respect to CC can be also seen, to a lesser extent, for the reverse task (i.e. for predicting BP annotations starting from other branches). This somehow confirm the stronger biological relationship existing among MF and BP terms than the relationships among these terms and CC ones. The last point that is worth noting is the presence of better results when one tries to predict annotations using a network computed from GO-BP terms instead of predicting annotations for BP terms. It must be noted that the Biological Process branch of GO is the one that has the largest number of terms. As such,

the BP branch of GO might be considered the most complete one, hence the results obtained in these tests might account for this fact. In conclusion, although the three root GO categories (MF, BP and CC) are structured as independent ontology, their biological relationships (especially between BP and MF terms) may be observed in the way their annotations are distributed among a given corpus of annotated genes.

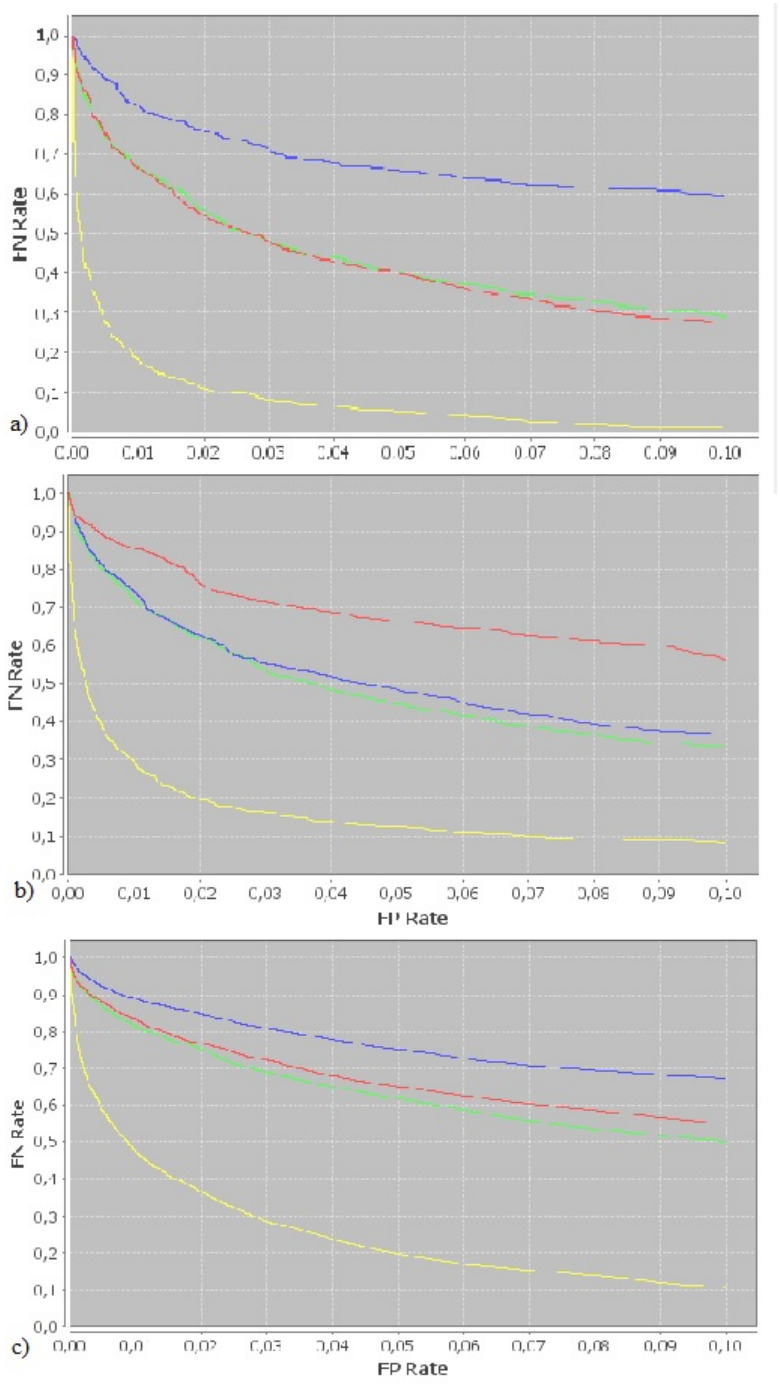


Figure 9.3 – ROC curve for GO branches comparisons.Graph a): Predicted annotations belongs to MF; line blue was obtained predicting using CC data, line green BP and line red using the composition of the two networks. Graph b): Predicted annotations belongs to CC; line red was obtained predicting using MF data, line blue BP and line green using the composition of both. Graph c): Predicted annotations belongs to BP; line blue was obtained predicting using CC data, line red MF and line green using the composition of both. In all the three graphs yellow lines are the prediction obtained by running the prediction algorithm onto the network derived from the same branch of GO terms to be predicted.

9.2 Gene Function Prediction with Heterogeneous Data

This subsection presents the series of tests that were performed to predict GO BP annotations from composite networks obtained by integrating heterogeneous data. The same type of five networks, used in the third section of the previous chapter, were used also in this series of experiments. The aim of this section is to use the GRF prediction algorithm to evaluate the effectiveness of aggregating different data sources when trying to predict gene functional annotations.

The five networks considered were obtained from:

- GO BP annotations;
- GO MF annotations;
- GO CC annotations;
- Pathway annotations;
- Publication reference annotations (associations between genes and Pubmed publication references where the genes appear).

The table 9.5 reports the dimensions of the data in terms of number of annotated genes, number of annotating features and number of annotations.

type	source	#genes	#features	#annotations	depth
Onto	BP	10537	3765	255042	18
Onto	MF	13074	2219	93958	14
Onto	CC	9757	569	136129	14
Flat	Pathway	3543	123	7713	
Flat	Pub Ref	27397	10622	65703	

Table 9.5 – Table of annotations for the Arabidopsis thaliana organism

All the annotation data and genes were retrieved from the GPDW data warehouse. The set of genes selected for these experiments was composed by the protein coding genes of the model organism Arabidopsis thaliana. The number of genes retrieved from the GPDW was 27937, however this set was filtered to retain just the genes that were also annotated to terms of the GO BP ontology. The number of those genes was 10537. All the five networks were constructed using a sparsity threshold of 50, whereas the thresholds used for annotation pruning were set as follows:

- 5-105 for the network constructed from GO BP annotations, which was the target network. The value of 5 allows to exclude annotations that are too specific, that otherwise would be considered in the evaluation of the prediction performance;

- 0-k for the other two networks constructed from GO ontology (MF and CC). k was equal to about one tenth of the number of genes annotated by a given ontology (i.e. 9757 for CC and 13074 for MF);

- 2-0 for the network constructed from Pathway. Since the number of annotations available for this category of data is significantly smaller than the other categories (see table 9.5), I decided to not use an upper bound threshold for the pruning process (setting a value to 0 means that there is no such a threshold);

- 2 - 273 for the network constructed from Publication reference. The upper bound threshold was chosen with the same logic used for the networks constructed from MF and CC. It is interesting to note that all the 27937 protein coding genes have at least one annotations to a publication in the publication_reference table of the GPDW.

The edge weights of the networks constructed from the three GO ontology were computed using the Resnik similarity metric, whereas for the Pathway and Publication reference cases, the metric that was used was the PCC correlation. Since the PCC metric is not based on a DAG structure, too specific annotations could negatively bias the experiment, hence I decided to filter out the terms which annotated just one gene (i.e. minimum threshold equals to 2).

At first, all the networks, except for the one constructed from BP, were used to predict the GO BP annotations. Table 9.6 and figure 9.4 shows the results. The AUC values are the areas under the curves calculated from values of FP going from 0 to 0,1.

type	source	min-max	AUC (0,1)	line
Onto	BP	5-80		
Onto	MF	0-130	0,0535	yellow
Onto	CC	0-97	0,08	green
Flat	Pathway	2-0	0,0715	red
Flat	Pub Ref	3-273	0,0397	blue

Table 9.6 – AUC results for GO annotation predictions when just single networks were used to run the GRF algorithm. The annotations from BP were the one that had to be predicted. The header voice 'line' refers to the graph 9.4

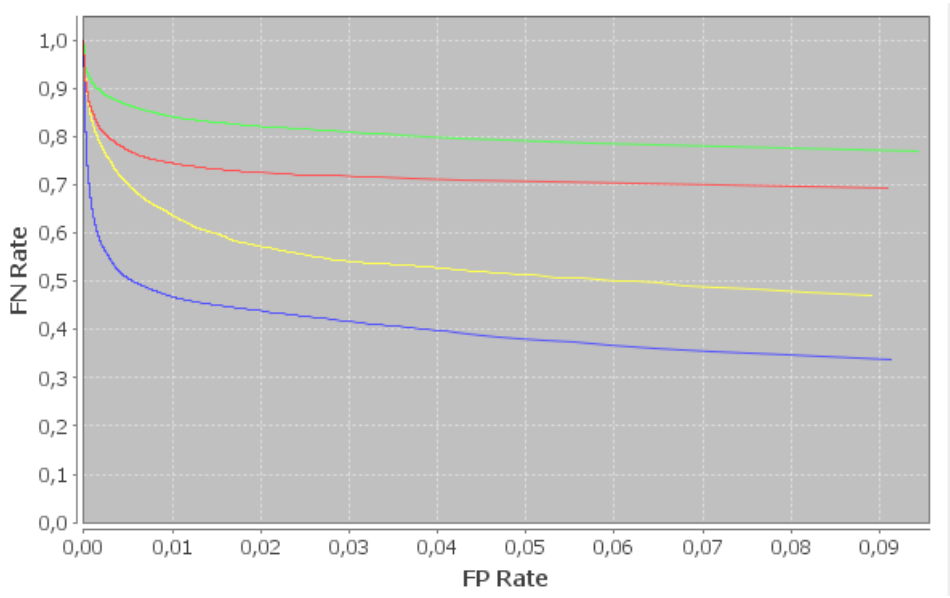


Figure 9.4 – ROC curves for GO BP predictions from single networks. The different curve colors refer to predictions calculated from networks computed from: red-Pathway; blue-Publication reference; yellow-MF; green-CC

The network that had the best performance in predicting BP annotations with the GRF algorithm was the network computed from the associations of genes with scientific publications, followed by the network derived from MF annotations. To evaluate if a composite network, obtained by integrating the single networks computed before, could performe better, I generated a composite network from MF and Pathway, then I added also the information from CC and, at the end, I used all the four input networks calculated before. As it is possible to see in figure 9.5 the level of the ROC curves always shifted down. The addition of information from Pathway and CC (blue and green line respectively) produced small differences with respect to the prediction obtained just with the network constructed from MF (red line), however even these small differences constitute an indication of the fact that little improvements can be obtained also integrating data that perform worse when used alone. The integrated model - composite network - constituted a better source of information for the task of predicting gene terms of GO BP. The results improved far more when the source of information that was observed as to be the best one for predicting the BP terms (i.e. the one computed from Publication reference) was included in the composition process (yellow line). The AUC scores of the predictions on the composite networks are in table 9.7.

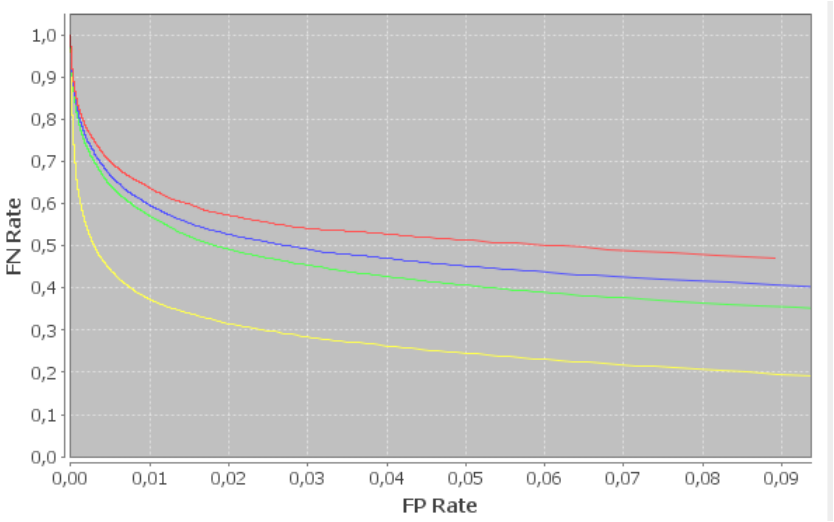


Figure 9.5 – ROC curves for GO BP predictions from composite networks. The different curve colors refer to predictions calculated from networks computed from: red-MF; blue-MF+Pathway; green-MF+Pathway+CC; green-MF+Pathway+CC+Publication ref

source	AUC (0,1)	line
MF	0,0535	red
MF + Path	0,048	blue
MF + Path + CC	0,0438	green
MF + Path + CC + Pub ref	0,0271	yellow

Table 9.7 – AUC results for GO annotation predictions when different networks were combined into unique composite networks. The annotations from BP were the one that had to be predicted. The header voice 'line' refers to the graph 9.5

Chapter 10
CONCLUSIONS

The software system designed and implemented in this thesis provides efficient and effective tools to manage the operations involved in a network construction process from different types of data sets. The software architecture offers also the possibility of integrating heterogeneous data sets into a unique network model by composing network representations of the data sets. Together with the network construction tools, the system provides an implementation of a network analysis technique - based on a semi-supervised learning approach - that can be used to predict unlabeled data belonging to a partially labeled set. The prediction algorithm was used to evaluate the prediction power of a network model obtained by integrating heterogeneous biomedical data sources, setting the prediction task to reconstructing the annotations between genes and their functions.

These results were obtained through:

1. The definition of the workflows executed when performing general network construction processes, aimed at identifying:
   (a) The main categories of data used to build networks;
   (b) The main operations that are performed to process the above cited data.
2. A careful literature exploration to get familiar with issues related to the complex network field and to survey those methodologies that have been proposed to compose heterogeneous networks;
3. The design and implementation of a general and flexible software system which provides the following functions:
   (a) The construction of single networks from the data categories that were identified before (i.e. node associations, flat annotation profiles, ontological annotation profiles); the construction of the networks was decomposed into the main high-level operations detected in the first analysis phase related to the workflows definition;
   (b) Efficient implementations of the high level operations defined in the workflows; multithreading and High-Performance Computing libraries were exploited to enhance the system performances;

(c) The aggregation of heterogeneous networks into a unique network model, named as composite network, based on the coherence the first networks show with respect to a target network; the coherence of the networks with respect to the target was computed by solving a linear regression problem, minimizing the least squares of error;

(d) The prediction of labels for unlabeled data of a given set; the algorithm that was implemented assigns a likelihood of having a certain label to the network nodes. The likelihood is assessed based on the prior evidence of the label and the network structure.

4. The execution of tests - pertaining biomedical use cases - on the system developed; the tests evaluated:

   (a) The performances of the system in terms of requested run time and memory consumption;

   (b) The improvements associated to the prediction of gene functions when using heterogeneous data sources integrated in a unique network model.

Chapter 11
FUTURE WORKS

This thesis may be considered as the first step towards the realization of a complete software architecture for the analysis of complex networks. An application of a network analysis technique has been already implemented in the software system; this has been used to predict gene functions exploiting the structure of networks. As it was briefly introduced in the second chapter of this research work, the analysis and applications of complex networks are powerful tools that should be further investigated. As such, the software developed in this thesis can be used as a basis to build on top of this network analysis tools. Hence, one potential future work could be a vertical enlargement of the architecture, starting from the outputs that the software can already provide. In this terms one may think to design and implement:

- a visualization framework to display network properties and characteristics (such as nodes, links, community organizations etc.). The implementation of effective strategies to provide the user with a nice interface summarizing the network properties would be a plus for the whole software architecture. However, this is not an easy task, due to the dimensions of data that can be potentially visualized (one may think, for example, at the number of nodes of a complex networks, which is often quite large), and a lot of research is being carrying on these topics.

- analysis tools to compute the standard network characteristics discussed in the second chapter of this thesis (i.e. node degree, node degree distribution, clustering coefficient etc.)

- algorithms to assess the centrality of nodes (which in the biomedical context is often referred as the essentiality of a given genes or proteins) or the community structure of the networks. The community structure of a network is often used to predict the functions of genes and proteins.

- algorithms for network alignment, such as the one developed by Phan and colleagues [86]. This is a new promising topic in the literature. Generally network alignment's techniques are applied for cross-species analysis, networks being compared are often of the same data types. The goal that is often pursued

is to find subnetworks that are conserved across several organisms, thus they can reveal a functional module.

The future works suggested above are all extensions to the software architecture of this thesis that work just on the outputs provided by the actual implementation of the software system. However, another direction of growth for the architecture could be the horizontal one. With horizontal direction I refer to all the extensions, improvements and modifications of the system functions already provided, hence the functions for constructing networks. In this respect, potential future works are:

- Development of new functionality as explained throughout the thesis. Since the system was designed to be extendable, the first developments may concern the implementation of new functionality to be included into the system. For instance, a similarity metric that is often used to compute edge weights from annotation is based on the hypergeometric distribution [40]. Or else, there exist other method that can be investigated to compute the similarity of ontological annotation profiles, for example Wang et al. [87] proposed a metric to compute the similarity of GO terms based on the GO structure, weighting in a different way IS_A and PART_OF relationships. Besides implementing new functionality, a study of the effectiveness of different metrics used to compute network weights may be performed.

- Development of an id mapping module to be run after the data acquisition phase and before the network construction. In the tests performed in this thesis, the data were taken from the GPDW data warehouse which already provides an id mapping module service for the data it integrates. However, since the system provides the possibility of using different data retrievers, implementing an id mapping module which permits to maps different names of the same entities under the same id can be nice extension. This module could ease the integration of data taken from different data sources.

- Extensions of the network representation techniques currently implemented in the software system. As it was seen in the chapter 6, an accurate selection of the network representation formats to be used for solving a given task is of fundamental importance to maintain low computational complexity of the algorithms used. As such, a suite of tools that can efficiently convert one network representation into another one should be added to the system architecture.

Bibliography

[1] Sarati R. Metodi e Architetture Software per la Predizione di Annotazioni Genomiche Funzionali e la Stima di Similarità Semantica di Geni e Proteine . Politecnico di Milano; 2009.

[2] Genetic Code [Online]; [updated 2010 Oct 30; cited 2012 Nov 11]. Available from: http://en.wikipedia.org/wiki/Genetic_codel.

[3] DeBakey ME. The national library of medicine: Evolution of a premier information center. JAMA. 1991;266(9):1252–1258. Available from: http://dx.doi.org/10.1001/jama.1991.03470090086037.

[4] Hamosh A, Scott AF, Amberger JS, Bocchini CA, McKusick VA. Online Mendelian Inheritance in Man (OMIM), a knowledgebase of human genes and genetic disorders. Nucl Acids Res. 2005;33(suppl 1):D514–D517. Available from: http://nar.oxfordjournals.org/content/33/suppl_1/D514.abstract.

[5] Matthews L, Gopinath G, Gillespie M, Caudy M, Croft D, de Bono B, et al. Reactome knowledgebase of human biological pathways and processes. Nucl Acids Res. 2009;37(suppl 1):D619–D622. Available from: http://nar.oxfordjournals.org/content/37/suppl_1/D619.abstract.

[6] Degtyarenko K, Hastings J, de Matos P, Ennis M. In: ChEBI: An Open Bioinformatics and Cheminformatics Resource. John Wiley & Sons, Inc.; 2002. Available from: http://dx.doi.org/10.1002/0471250953.bi1409s26.

[7] Courtot M, Juty N, Knupfer C, Waltemath D, Zhukova A, Drager A, et al. Controlled vocabularies and semantics in systems biology. Mol Syst Biol. 2011;7(543). Available from: http://dx.doi.org/10.1038/msb.2011.77.

[8] Ashburner M, Ball CA, Blake JA, Botstein D, Butler H, Cherry JM, et al. Gene Ontology: tool for the unification of biology;25.

[9] Bairoch A, Apweiler R, Wu CH, Barker WC, Boeckmann B, Ferro S, et al. The Universal Protein Resource (UniProt). Nucl Acids Res. 2005;33(suppl 1):D154–D159. Available from: http://nar.oxfordjournals.org/content/33/suppl_1/D154.abstract.

[10] Liu M, Liberzon A, Kong SW, Lai WR, Park PJ, Kohane IS, et al. Network-Based Analysis of Affected Biological Processes in Type 2 Diabetes Models. PLoS Genet. 2007 06;3(6):e96. Available from: http://dx.plos.org/10.1371%2Fjournal.pgen.0030096.

[11] Microarrays: chipping away at the mysteries of science and medicine [Online]. National Center for Biotechnology Information; [updated 2007 July 27;

cited 2012 August 06]. Available from: http://www.ncbi.nlm.nih.gov/About/primer/microarrays.html.

[12] Masseroli M. DNA microarray data analysis [slides of the course Bioinformatica e Biologia Computazionale per la Medicina Molecolare]. Milan, Italy: DEI, Politecnico di Milano; 2010.

[13] Masseroli M, Ceri S, Tettamanti L, Ghisalberti G, Canakoglu A, Campi A. Model-Driven Modular Integration of Genomic and Proteomic Information; 2012. In press 2012.

[14] Atzeni P, Ceri S, Fraternali P, Paraboschi S, Torlone R. Basi di dati. Architetture e linee di evoluzione. 2nd ed. Istruzione scientifica. McGraw-Hill Companies; 2007.

[15] Atzeni P, Ceri S, Paraboschi S, Torlone R. Basi di dati. Modelli e linguaggi di interrogazione. 3rd ed. McGraw-Hill Companies; 2009.

[16] Benson DA, Karsch-Mizrachi I, Lipman DJ, Ostell J, Sayers EW. GenBank. Nucl Acids Res. 2011;39(suppl 1):D32–D37. Available from: http://nar.oxfordjournals.org/content/39/suppl_1/D32.abstract.

[17] Kulikova T, Akhtar R, Aldebert P, Althorpe N, Andersson M, Baldwin A, et al. EMBL Nucleotide Sequence Database in 2006. Nucl Acids Res. 2007;35(suppl 1):D16–D20. Available from: http://nar.oxfordjournals.org/content/35/suppl_1/D16.abstract.

[18] Tateno Y, Imanishi T, Miyazaki S, Fukami-Kobayashi K, Saitou N, Sugawara H, et al. DNA Data Bank of Japan (DDBJ) for genome scale research in life science. Nucl Acids Res. 2002;30(1):27–30. Available from: http://nar.oxfordjournals.org/content/30/1/27.abstract.

[19] PubMed.gov [Online]. Maryland: National Center for Biotechnology Information - National Library of Medicine; [cited 2012 August 06]. Available from: http://www.ncbi.nlm.nih.gov/pubmed/.

[20] Canakoglu A. Genomic and proteomic annotation data file taxonomy [slides of the course ICT for Health Care and Life Sciences]. Milan, Italy: DEI, Politecnico di Milano; 2011.

[21] Hunter S, Jones P, Mitchell A, Apweiler R, Attwood TK, Bateman A, et al. InterPro in 2011: new developments in the family and domain prediction database. Nucl Acids Res. 2012;40(D1):D306–D312. Available from: http://nar.oxfordjournals.org/content/40/D1/D306.abstract.

[22] Hitzler P, Krötzsch M, Rudolph S. Foundations of Semantic Web Technologies. Chapman & Hall/CRC; 2009.

[23] Karp PD. What we do not know about sequence analysis and sequence databases. Bioinformatics. 1998;14(9):753–754. Available from: http://bioinformatics.oxfordjournals.org/content/14/9/753.short.

[24] International Human Genome Sequencing Consortium. Finishing the euchromatic sequence of the human genome. Nature. 1998;431(7011):931–045. Available from: http://dx.doi.org/10.1038/nature03001.

[25] Aittokallio T, Schwikowski B. Graph-based methods for analysing networks in cell biology. Briefings in Bioinformatics. 2006;7(3):243–255. Available from: http://bib.oxfordjournals.org/content/7/3/243.abstract.

[26] Mason O, Verwoerd M. Graph theory and networks in Biology. IET Systems Biology. 2007;1(2):89–119. Available from: http://link.aip.org/link/?SYB/1/89/1.

[27] Rain JC, Selig L, De Reuse H, Battaglia V, Reverdy C, Simon S, et al. The protein-protein interaction map of Helicobacter pylori. Nature. 2001 Jan;409(6817):211–215. Available from: http://dx.doi.org/10.1038/35051615.

[28] Giot L, Bader JS, Brouwer C, Chaudhuri A, Kuang B, Li Y, et al. A Protein Interaction Map of Drosophila melanogaster. Science. 2003;302(5651):1727–1736. Available from: http://www.sciencemag.org/content/302/5651/1727.abstract.

[29] Li S, Armstrong CM, Bertin N, Ge H, Milstein S, Boxem M, et al. A Map of the Interactome Network of the Metazoan C. elegans. Science. 2004;303(5657):540–543. Available from: http://www.sciencemag.org/content/303/5657/540.abstract.

[30] Ewing RM, Chu P, Elisma F, Li H, Taylor P, Climie S, et al. Large-scale mapping of human protein-protein interactions by mass spectrometry. Mol Syst Biol. 2007;3(89). Available from: http://dx.doi.org/10.1038/msb4100134.

[31] Herrgård MJ, Swainston N, Dobson P, Dunn WB, Arga KY, Arvas M, et al. A consensus yeast metabolic network reconstruction obtained from a community approach to systems biology. Nat Biotech. 2008;26(10):1155–1160. Available from: http://dx.doi.org/10.1038/nbt1492.

[32] Salwinski L, Miller CS, Smith AJ, Pettit FK, Bowie JU, Eisenberg D. The Database of Interacting Proteins: 2004 update. Nucleic Acids Research. 2004;32(suppl 1):D449–D451. Available from: http://nar.oxfordjournals.org/content/32/suppl_1/D449.abstract.

[33] Breitkreutz BJ, Stark C, Reguly T, Boucher L, Breitkreutz A, Livstone M, et al. The BioGRID Interaction Database: 2008 update. Nucleic Acids Research. 2008;36(suppl 1):D637–D640. Available from: http://nar.oxfordjournals.org/content/36/suppl_1/D637.abstract.

[34] Szklarczyk D, Franceschini A, Kuhn M, Simonovic M, Roth A, Minguez P, et al. The STRING database in 2011: functional interaction networks of proteins, globally integrated and scored. Nucleic Acids Research. 2011;39(suppl 1):D561–D568. Available from: http://nar.oxfordjournals.org/content/39/suppl_1/D561.abstract.

[35] Tanabe M, Kanehisa M. Using the KEGG Database Resource. Current Protocols in Bioinformatics. 2002;Available from: http://dx.doi.org/10.1002/0471250953.bi0112s38.

[36] Caspi R, Altman T, Dreher K, Fulcher CA, Subhraveti P, Keseler IM, et al. The MetaCyc database of metabolic pathways and enzymes and the BioCyc collection of pathway/genome databases. Nucleic Acids Research. 2012;40(D1):D742–D753. Available from: http://nar.oxfordjournals.org/content/40/D1/D742.abstract.

[37] Gama-Castro S, Salgado H, Peralta-Gil M, Santos-Zavaleta A, Muñiz Rascado L, Solano-Lira H, et al. RegulonDB version 7.0: transcriptional regulation of Escherichia coli K-12 integrated within genetic sensory response units (Gensor Units). Nucleic Acids Research. 2011;39(suppl 1):D98–D105. Available from: http://nar.oxfordjournals.org/content/39/suppl_1/D98.abstract.

[38] Keseler IM, Collado-Vides J, Santos-Zavaleta A, Peralta-Gil M, Gama-Castro S, Muñiz Rascado L, et al. EcoCyc: a comprehensive database of Escherichia coli biology. Nucleic Acids Research. 2011;39(suppl 1):D583–D590. Available from: http://nar.oxfordjournals.org/content/39/suppl_1/D583.abstract.

[39] Mostafavi S, Morris Q. Fast integration of heterogeneous data sources for predicting gene function with limited annotation. Bioinformatics. 2010;26(14):1759–1765. Available from: http://bioinformatics.oxfordjournals.org/content/26/14/1759.abstract.

[40] Insuk Lee EMM Zhihua Li. An Improved, Bias-Reduced Probabilistic Functional Gene Network of Baker's Yeast, Saccharomyces cerevisiae. PLoS ONE. 2007;(10). Available from: http://www.plosone.org/article/info:doi/10.1371/journal.pone.0000988.

[41] Boccaletti S, Latora V, Moreno Y, Chavez M, Hwang DU. Complex networks: Structure and dynamics. Physics Reports. 2006;424:175 – 308. Available from: http://www.sciencedirect.com/science/article/pii/S037015730500462X.

[42] Pastor-Satorras R, Vespignani A. Evolution and Structure of the Internet. Cambridge University Press; 2004. Available from: http://dx.doi.org/10.1017/CBO9780511610905.

[43] Newman MEJ. The structure of scientific collaboration networks. Proceedings of the National Academy of Sciences. 2001;98(2):404–409. Available from: http://www.pnas.org/content/98/2/404.abstract.

[44] Marchiori M, Latora V. Harmony in the small-world. Physica A. 2000 Oct;285:539–546.

[45] Sporns O. Network analysis, complexity, and brain function. Complexity. 2002;8(1):56–60. Available from: http://dx.doi.org/10.1002/cplx.10047.

[46] Piccardi C. Introduction to complex networks [slides]. Milan, Italy: DEI, Politecnico di Milano; 2011.

[47] Barabási AL, Oltvai ZN. Network biology: understanding the cell's functional organization. Nat Rev Genet. 2004;5(2):101–113. Available from: http://dx.doi.org/10.1038/nrg1272.

[48] Albert R. Scale-free networks in cell biology. Journal of Cell Science. 2005;118(21):4947–4957. Available from: http://jcs.biologists.org/content/118/21/4947.abstract.

[49] Newman MEJ. The Structure and Function of Complex Networks. SIAM Review. 2003;45:167–256. Available from: http://arxiv.org/abs/cond-mat/0303516.

[50] Strogatz SH. Exploring complex networks. Nature. 2001;410(6825):268–276. Available from: http://dx.doi.org/10.1038/35065725.

[51] Barthelemy M, Gondran B, Guichard E. Spatial structure of the internet traffic. Physica A. 2003;319(0):633 – 642. Available from: http://www.sciencedirect.com/science/article/pii/S0378437102013821.

[52] Wagner A, Fell DA. The small world inside large metabolic networks. Proceedings of the Royal Society of London Series B: Biological Sciences. 2001;268(1478):1803–1810. Available from: http://rspb.royalsocietypublishing.org/content/268/1478/1803.abstract.

[53] Jeong H, Tombor B, Albert R, Oltvai ZN, Barabási AL. The large-scale organization of metabolic networks. Nature. 2000;407(6804):651–654. Available from: http://dx.doi.org/10.1038/35036627.

[54] Yook SH, Oltvai ZN, Barabási AL. Functional and topological characterization of protein interaction networks. PROTEOMICS. 2004;4(4):928–942. Available from: http://dx.doi.org/10.1002/pmic.200300636.

[55] Barrat A, Barthelemy M, Pastor-Satorras R, Vespignani A. The architecture of complex weighted networks. Proc Natl Acad Sci USA. 2004;101(11):3747–3752. Available from: http://www.pnas.org/content/101/11/3747.abstract.

[56] Ravasz E, Somera AL, Mongru DA, Oltvai ZN, BarabáÂ¡si AL. Hierarchical Organization of Modularity in Metabolic Networks. Science. 2002;297(5586):1551–1555. Available from: http://www.sciencemag.org/content/297/5586/1551.abstract.

[57] Wang XF, Chen G. Complex Networks: Small-World, Scale-Free and Beyond. IEEE Circuits and Systems Magazine. 2003;3(1):6–20.

[58] Watts DJ, Strogatz SH. Collective dynamics of "small-world" networks. Nature. 1998;393(6684):440–442. Available from: http://dx.doi.org/10.1038/30918.

[59] Barabási AL, Albert R. Emergence of Scaling in Random Networks. Science. 1999;286(5439):509–512. Available from: http://www.sciencemag.org/content/286/5439/509.abstract.

[60] Newman M. Networks: An Introduction. OUP Oxford; 2010. Available from: http://books.google.it/books?id=q7HVtpYVfC0C.

[61] Jeong H, Mason SP, Barabási AL, Oltvai ZN. Lethality and centrality in protein networks. Nature. 2001;411(6833):41–42. Available from: http://dx.doi.org/10.1038/35075138.

[62] Yu H, Greenbaum D, Lu HX, Zhu X, Gerstein M. Genomic analysis of essentiality within protein networks. Trends in Genetics. 2004;20(6):227 – 231. Available from: http://www.sciencedirect.com/science/article/pii/S0168952504001015.

[63] Fortunato S. Community detection in graphs. Phys Rep. 2010;486:75–174. Available from: http://adsabs.harvard.edu/abs/2010PhR...486...75F.

[64] Palla G, Derenyi I, Farkas I, Vicsek T. Uncovering the overlapping community structure of complex networks in nature and society. Nature. 2005;435:814–818.

[65] Adamcsek B, Palla G, Farkas IJ, Derényi I, Vicsek T. CFinder: locating cliques and overlapping modules in biological networks. Bioinformatics. 2006 Apr;22(8):1021–1023. Available from: http://dx.doi.org/10.1093/bioinformatics/btl039.

[66] Farkas I, Abel D, Palla G, Vicsek T. Weighted network modules. New Journal of Physics. 2007;9(6):180. Available from: http://stacks.iop.org/1367-2630/9/i=6/a=180.

[67] Resnik P. Semantic Similarity in a Taxonomy: An Information-Based Measure and its Application to Problems of Ambiguity in Natural Language. CoRR. 2011;abs/1105.5444.

[68] Lee I, Date SV, Adai AT, Marcotte EM. A Probabilistic Functional Network of Yeast Genes. Science. 2004;306(5701):1555–1558. Available from: http://www.sciencemag.org/content/306/5701/1555.abstract.

[69] Zhu X, Ghahramani Z, Lafferty J. Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions. In: IN ICML; 2003. p. 912–919.

[70] Dongarra J. Sparse Matrix Storage Formats. In: Z Bai JDAR J Demmel, van der Vorst H, editors. Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide. Philadelphia: SIAM; 2000. .

[71] Basic Linear Algebra Subprograms Technical (BLAST) Forum Standard. University of Tennessee; 2001. Available from: http://www.netlib.org/blas/blast-forum/blas-report.pdf.

[72] xianyi.github.com/OpenBLAS/ll [Online]. Lab of Parallel Software and Computational Science, ISCASl; [cited 2013 March 22]. Available from: http://xianyi.github.com/OpenBLAS/l.

[73] Whaley RC, Petitet A. Minimizing development and maintenance costs in supporting persistently optimized BLAS. Software: Practice and Experience. 2005 February;35(2):101–121. http://www.cs.utsa.edu/~whaley/papers/spercw04.ps.

[74] developer.amd.com/tools/cpu-development/amd-core-math-library-acml [Online]. AMD; [cited 2013 March 22]. Available from: http://developer.amd.com/tools/cpu-development/amd-core-math-library-acml/.

[75] software.intel.com/en-us/intel-mk [Online]. Intel; [cited 2013 March 22]. Available from: http://software.intel.com/en-us/intel-mkl.

[76] Guennebaud G, Jacob B, et al.. Eigen v3; 2010. http://eigen.tuxfamily.org.

[77] cise.ufl.edu/research/sparse/SuiteSparse/ [Online]. Tim Davis, Professor, CISE Department,University of Florida; [cited 2013 March 23]. Available from: http://www.cise.ufl.edu/research/sparse/SuiteSparse/.

[78] Li XS, Demmel JW, Gilbert JR, iL Grigori, Shao M, Yamazaki I. SuperLU Users' Guide. Lawrence Berkeley National Laboratory; 1999. LBNL-44289.

[79] Paige CC, Saunders MA. LSQR: An Algorithm for Sparse Linear Equations and Sparse Least Squares. ACM Trans Math Software. 1982;p. 43–71.

[80] OpenMP Application Program Interface . OpenMP Architecture Review Board; 2011.

[81] Cadini F, Zio E, Petrescu CA. Critical Information Infrastructure Security. Berlin, Heidelberg: Springer-Verlag; 2009. p. 155–167. Available from: http://dx.doi.org/10.1007/978-3-642-03552-4_14.

[82] Li Y, Patra JC. Genome-wide inferring gene-phenotype relationship by walking on the heterogeneous network. Bioinformatics. 2010 May;26(9):1219–1224. Available from: http://dx.doi.org/10.1093/bioinformatics/btq108.

[83] Hoche S, Flach P, Hardcastle D. A Fast Method for Property Prediction in Graph-Structured Data from Positive and Unlabelled Examples. In: Ghallab M, Spyropoulos CD, Fakotakis N, Avouris NM, editors. ECAI 2008 - 18th European Conference on Artificial Intelligence, Patras, Greece, July 21-25, 2008, Proceedings. vol. 178 of Frontiers in Artificial Intelligence and Applications. IOS Press; 2008. p. 162–166.

[84] Tong H, Faloutsos C, Pan JY. Fast Random Walk with Restart and Its Applications. In: Proceedings of the Sixth International Conference on Data Mining. ICDM '06. Washington, DC, USA: IEEE Computer Society; 2006. p. 613–622. Available from: http://dx.doi.org/10.1109/ICDM.2006.70.

[85] Gamma E. Design Patterns: Elements of Reusable Object-oriented Software. Addison-Wesley Professional Computing Series. Pearson Education; 2004. Available from: http://books.google.it/books?id=12S_MKvhDl4C.

[86] Phan HTT, Sternberg MJE. PINALOG: a novel approach to align protein interaction networks - implications for complex detection and function prediction. Bioinformatics. 2012;Available from: http://bioinformatics.oxfordjournals.org/content/early/2012/03/13/bioinformatics.bts119.abstract.

[87] Wang JZ, Du Z, Payattakool R, Yu PS, Chen CF. A new method to measure the semantic similarity of GO terms. Bioinformatics. 2007;23(10):1274–1281. Available from: http://bioinformatics.oxfordjournals.org/content/23/10/1274.abstract.

[88] Liang S. The Java Native Interface: Programmer's Guide and Specification. Addison-Wesley Java series. Addison-Wesley; 1999. Available from: http://books.google.it/books?id=MH5QAAAAMAAJ.

Appendix A

SOFTWARE ARCHITECTURE

This appendix aims at documenting the software architecture of the solution developed. Its organization resembles the software architecture decomposition, which is divided into two main subsystems: Java subsystem and native code subsystem. These two main components communicate with each other by means of the JNI interface[88]

A.1 Java Subsystem

The Java subsystem has the major goal of describing the actions and operations that will be performed in order to construct the desired network. The entry point of the Java subsystem is the CNWJavaController class. This class is the central point of the Java component and it is responsible for intercepting a user request and assembling a new one which is compliant to the format expected by the C++ subsystem. The class diagram in figure shows the relations between classes and packages which stay in the main package of the application (cnw package).

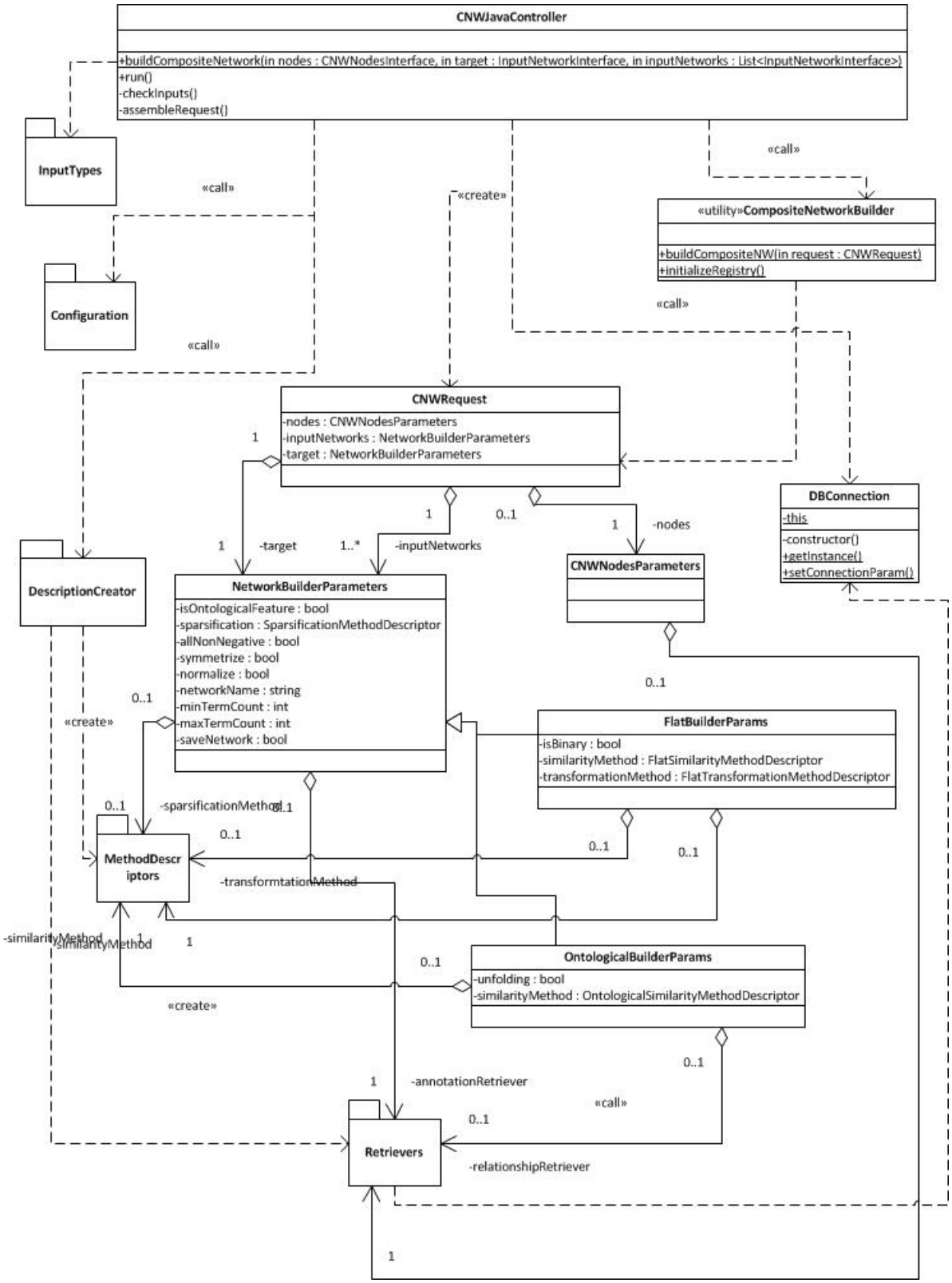


Figure A.1 – Class diagram of the package cnw.

As one may see from the figure A.1 the CNWJavaController class depends on the package inputTypes.

A.1.1 Package InputTypes

This package contains all the interfaces that must be used to create objects that are passed into the buildCompositeNewtork method of the class CNWJavaController. This method starts the process for assembling the request. The figure A.2 displays the class diagram of the package inputTypes.

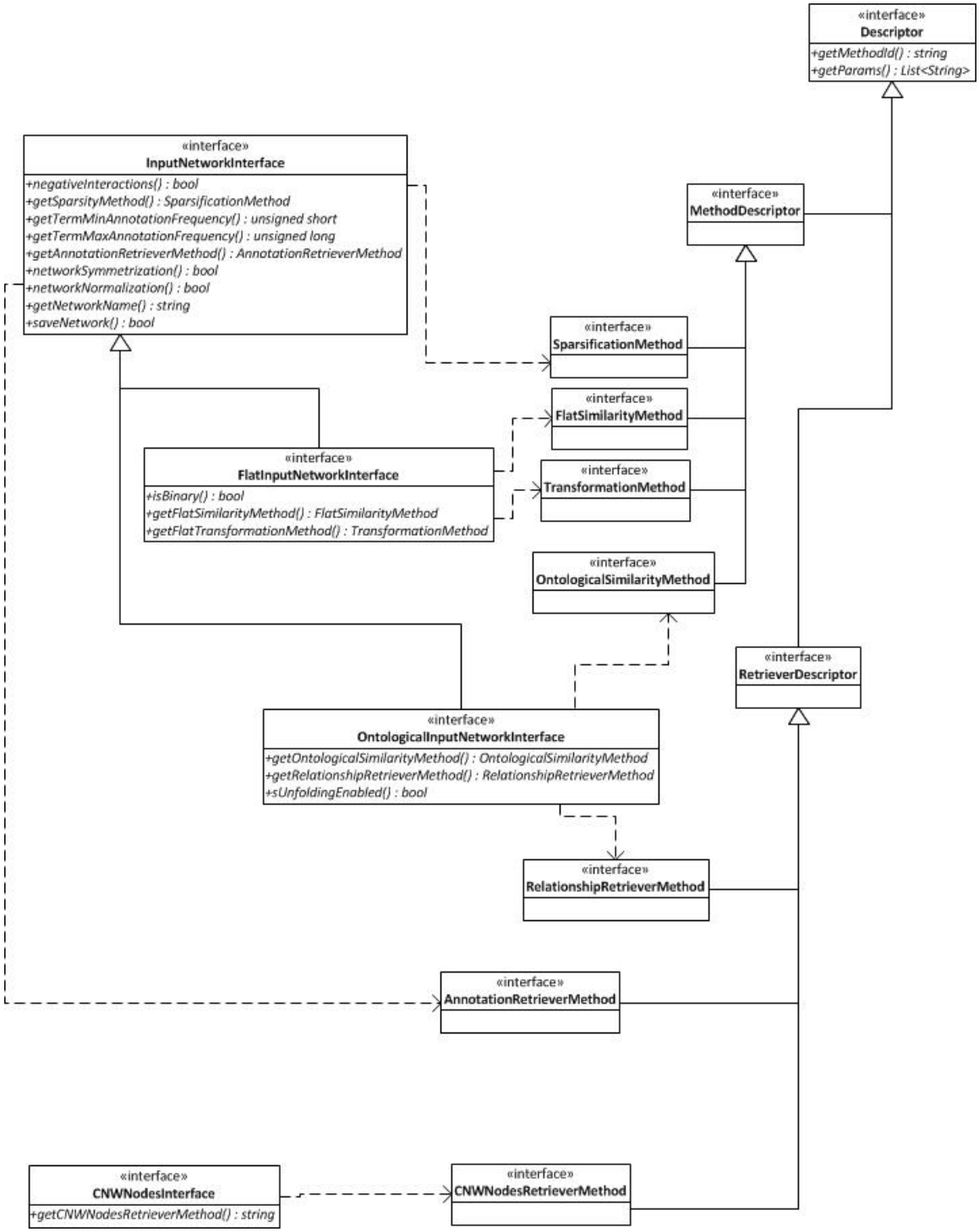


Figure A.2 – Class diagram of the package inputTypes.

In this package, the InputNetworkInterface interface is used to describe the tasks that the user wants to be performed in order to construct the network, besides the

tasks the user also specifies which data retriever has to be used for collecting the data the network is constructed from. The FlatInputNetworkInterface and OntologicalInputNetworkInterface extend the InputNetworkInterface and add tasks and retrievers descriptions which apply only to the network derived from flat annotation and ontological annotation respectively. Another thing that is worthy to note is the Descriptor interface; This is the base interface used to describe all tasks that can be performed to build a network. A description of a method or a data retriever must provide a unique id of the method/retriever to be used and a list of string parameters that the method/retriever takes as input. Note that the xml input file discussed in the chapter of Materials and methods (figure 7.9) resembles the hierarchical organization of the interface in this package. This is because the data in the xml file are parsed and then converted into objects implementing the interface in this package. Every approach can be used to interact with the user (GUI, command line, xml file etc. etc.) as long as there is a translation of the inputs provided by the user into the interfaces exposed by this package.

The CNWJavaController calls the classes inside the Configuration package for knowing which methods and retrievers are available in the system.

A.1.2 Package Configuration

This package contains all the classes needed to parse the xml configuration file and to store its information (figure A.3).

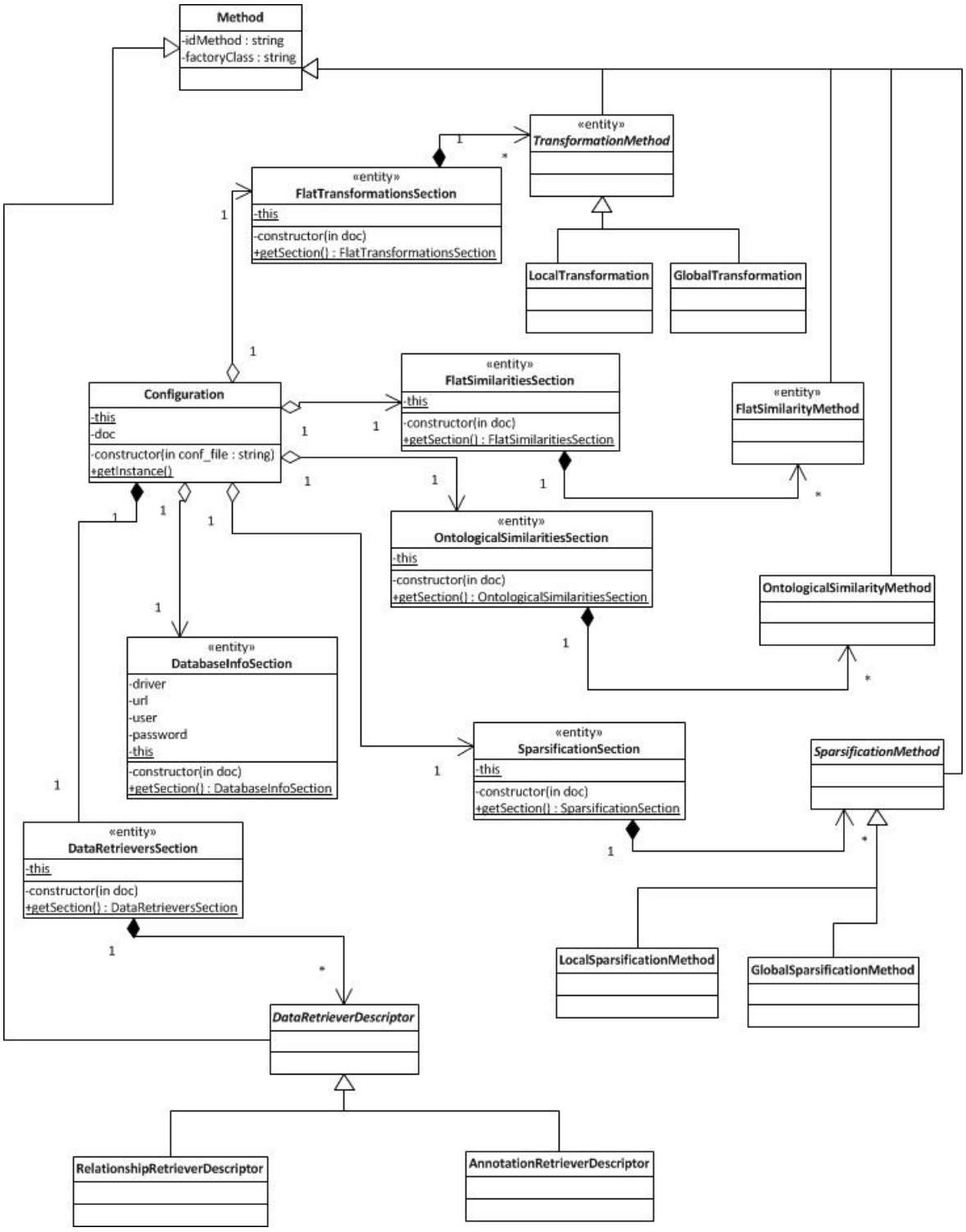


Figure A.3 – Class diagram of the package Configuration.

The xml configuration file is divided into sections, each one dedicated to the description of the alternative methods available to solve a specific task(e.g. compute a flat similarity using a specific metric). All the methods and retrievers in the xml configuration file are denoted with their unique id. In addition to the id, every method/retriever also specifies its factory class which is in charge of instantiate a description of the method (this mechanism is discussed in detail in the chapter Materials and methods). A description of a method/retriever gives information on how

the particular task should be performed (e.g. the sparsification method with adaptive threshold is likely to ask to the user the threshold to be used).

Given the methods and retrievers that the user has specified for constructing the desired network, the CNWJavaController uses the registry contained in the DescriptionCreator package to get the objects which describes those methods and retrievers requested.

A.1.3 Package DescriptionCreator

The package DescriptionCreator defines the registry DescriptionCreatorRegistry (figure A.4) which is used to map the methods and retrievers id with their factory classed. The factories are retrieved from the configuration file asking to the classes within the Configuration package. Once the java path of the factories are known the respective objects are instantiated through Java reflection (further details can be found in the chapter Material and methods). The CNWJavaController asks to the DescriptionCreatorRegistry for getting the factory class associated with a particular method/retriever id.

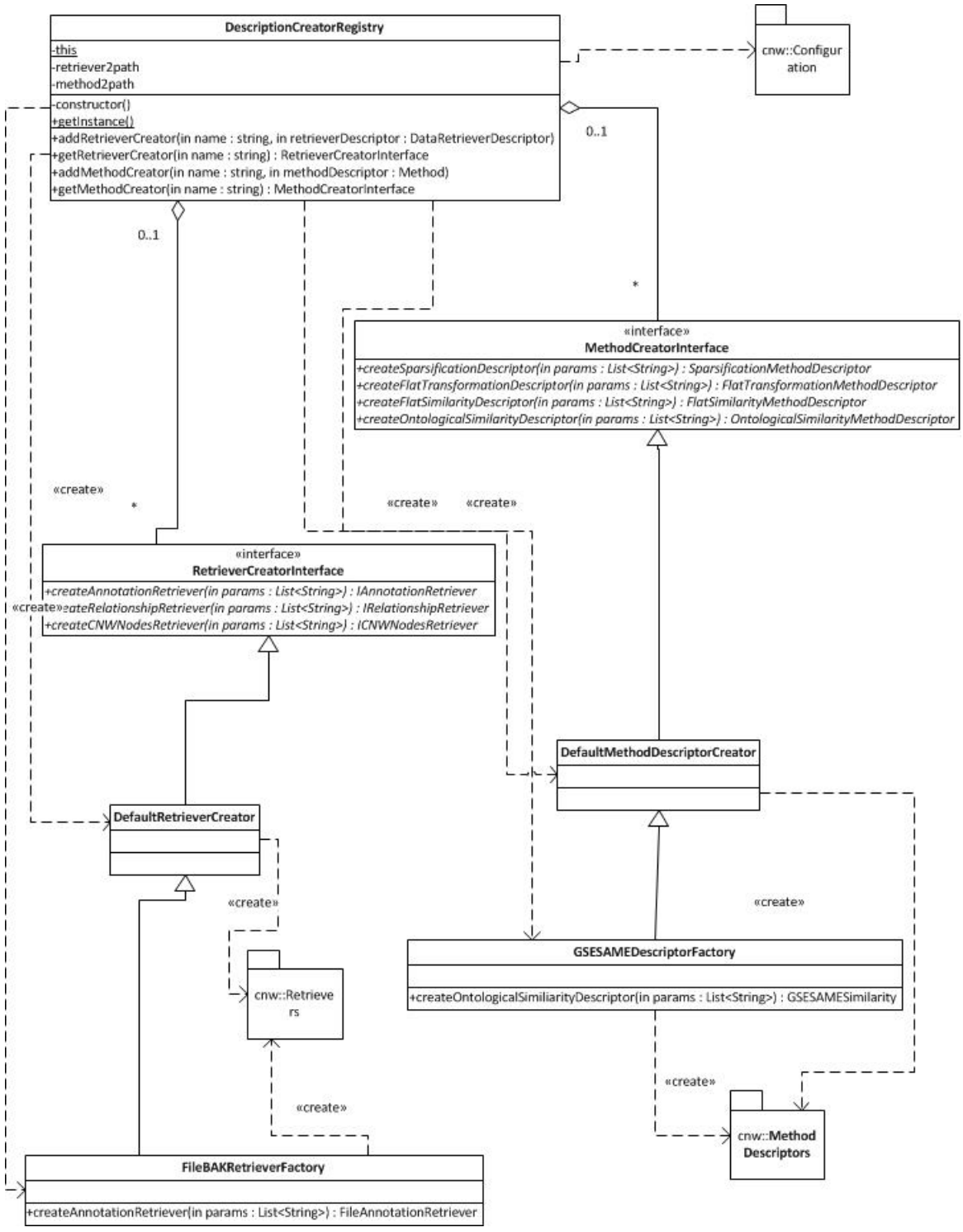


Figure A.4 – Class diagram of the package DescriptionCreator

All the factories that are added into the system must be compliant with the interfaces presented in this package (MethodCreatorInterface and RetrieverCreatorInterface).

Once the CNWJavaController has obtained the factory classes associated to the methods and retrievers requested by the user, it can call the functions that the factories provide to instantiate the method descriptions and the retriever implementations, respectively. Note that the objects instantiated from the id of the methods

specified by the user are just descriptions about how the particular method should be performed (i.e. what parameters should be used to execute the task solved by the method). The actual implementation and execution of the method requested will be performed at C++ side. On the other hand, objects instantiated from the id of the requested data retriever are not description of the retriever to be used, conversely they already implement the code needed to retrieve data. The method descriptions and data retrievers that will be included into the system must implement the interfaces that are present into the MethodDescriptors and Retrievers packages respectively.

A.1.4 Package MethodDescriptors

This package contains the interfaces that must be implemented by the methods that are included into the software system (figure A.5). The methods that are compliant with these interfaces can be used to execute the tasks needed to build a network.

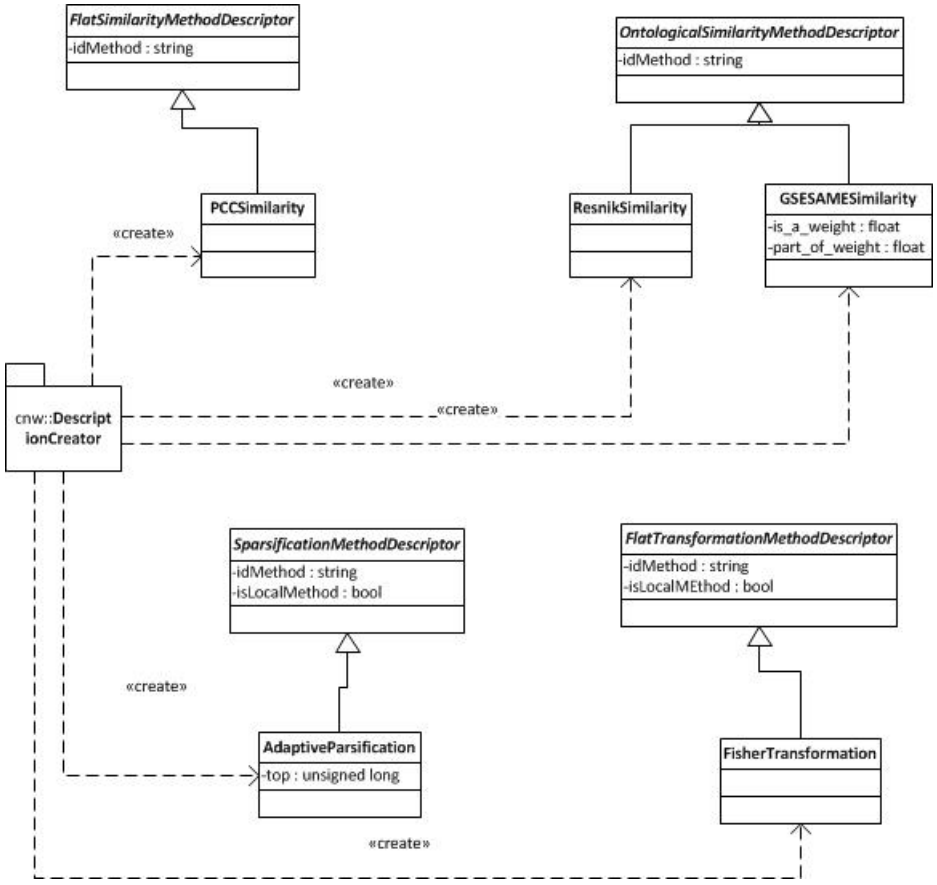


Figure A.5 – Class diagram of the package methodDescriptor

A.1.5 Package Retriever

This package contains the interface that must be implemented by the data retriever attached to the system (figure A.6).

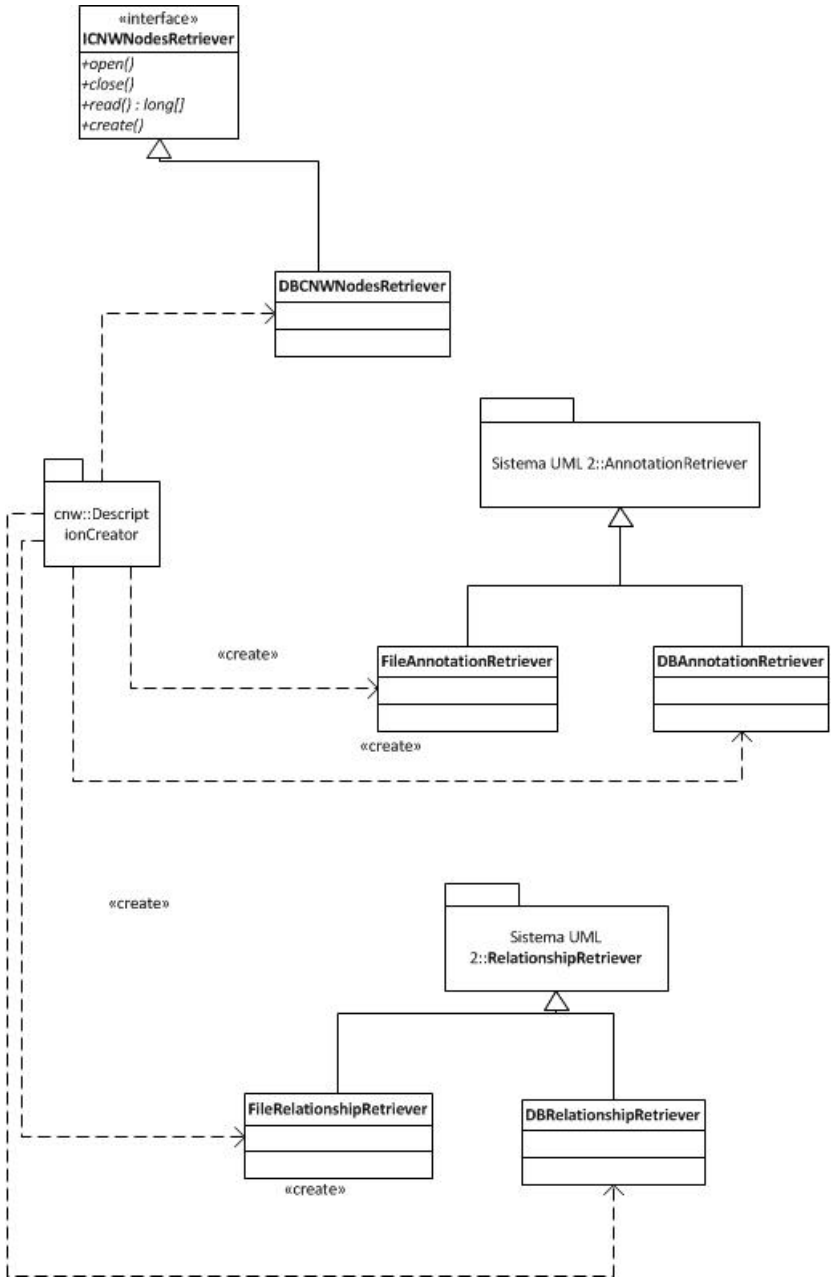


Figure A.6 – Class diagram of the retriever package.

The final step taken by the CNWJavaController is to create a CNWRequest object (figure A.1) and populate it with the method descriptions and data retrievers that have been collected by the controller. The request is then forwarded to the native code side of the software architecture where the network building process takes place.

A.2 C++ Subsystem

The native code subsystem is implemented in C++ language. It is the computational core of the software architecture; Here all the methods selected by the user through the Java subsystem are executed. The subsystem main classes are shown in figure A.7

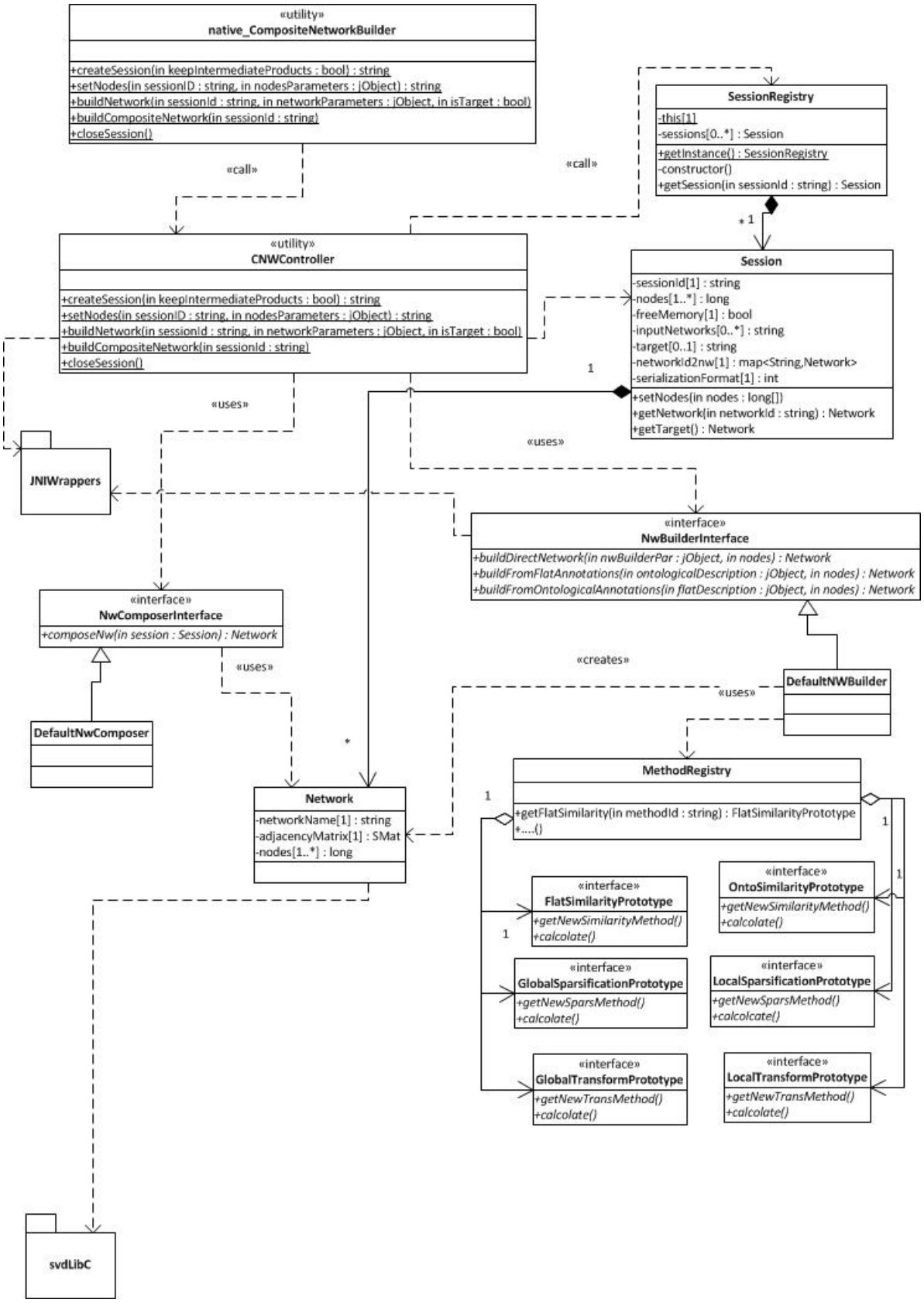


Figure A.7 – Class diagram of the main classes of the native code subsystem

; All the methods of the C++ subsystem that are callable from outside the component are in the “native_CompositeNetworkBuilder” module. The Java subsystem

use these methods to satisfy a user request. The most laborious user request is to build a composite network (see figure 7.16 in Materials and methods chapter). In this case the methods on the native code subsystem that are called are, in the order:

1. createSession()
2. setNodes() [optional]
3. buildNetwork() [target]
4. buildNetwork() [input networks, called n>1 times]
5. buildCompositeNetwork()
6. closeSession()

All user requests are initiated with the creation of a session associated to the user, the session will contain all the intermediate results that are produced by a call to the native code subsystem and are used by successive calls. For example, given the operations listed above, the calls to operations 2,3 and 4 will generate the nodes of the network, the network to be used as target and all the input networks. Then a call to the "buildCompositeNetwork()" method starts the execution of the algorithm implemented for composing networks, the networks that will be composed are retrieved from the session.

Another thing that is worth noting of the figure A.7 is the class "MethodRegistry", this class is a registry of methods that can be used to solve different tasks. The tasks solvable are characterized by the interfaces included in the "MethodRegistry" class; The object performing those tasks implement a specific way of solving a particular task and are instantiated based on the user request. Their instantiation is performed following the mechanism specified by the Prototype pattern [85], which allows to decouple the usage of an object from its mechanism of instantiation.

The last thing about the design of the native code subsystem is the definition of interfaces which describe the requirements of classes that have the duties of building a network or composing networks. Having defined these interfaces it becomes easy to change the architecture by providing other network builder and/or network composer which differ from the default ones.