University of Padova

Department of Information Engineering

Bioengineering

A multi-tier architecture for Graphite Web: a tool for pathway analysis of high-throughput expression data

Marco Matta

Advisor

Barbara Di Camillo

Coadvisor

Gabriele Sales - Chiara Romualdi

Academic Year 2016/17

Abstract

The thesis is the final result of a fellowship work developed in collaboration with the Department of Biology of University of Padova.
It consists on the implementation of a web software, that will be released by the computational biology team led by Professor Chiara Romualdi.
The project is focused on Graphite Web, an already-existing public web server solution for analysis and visualization of biological pathways using high-throughput gene expression data of both microarray and RNA-seq experiments and the goal of this work is to improve its implementation in terms of performance and usability.

The analysis of the gene expression is useful in investigating some diseases, for instance different types of tumor.

Graphite Web is a software that provides different types of analysis which gives to researchers the chance of testing performances of different algorithms on data.
At the moment this software is merely an application that receives the requests of analysis and returns the corresponding results.
For each analysis it performs expensive operations, wasting a lot of time and RAM.
The work tries to solve this problems focusing on three aspects.
First of all, when a client executes twice the same analysis, the actual implementation does not use some shared-results from previous operations, but it analyzes all the data again.
In order to avoid this redundancy of time and resources we implemented a new design for Graphite Web that manages the persistently of data exploiting a database.
An other improvement lies in the fact that in the new version each transfer of data happens in streaming so we save RAM.
Finally we implemented a system that manages the concurrency and parallelism in order to can execute more operations at the same time.

We extended the Graphite Web software's architecture, developing a multi-tier architecture and implementing it in Python.

The back-end of the resulting application consists in a HTTP server (based on the gunicorn library and gevent) which has the aim of managing multiple independent requests concurrently.
Furthermore the new version of Graphite Web release can track and save every operation carried out by the user. This feature is going to be helpful to researchers to save time during the analysis of huge amounts of data such as high-throughput gene expression.

In the first chapters of this thesis we performed the analysis of a case study on ovarian cancer dataset of RNA-seq using current version of Graphite Web in order to discover its limits, then we explained the implementation of the new version.

Abstract

La tesi è il risultato del lavoro svolto durante una borsa di studio presso il Dipartimento di Biologia dell'Università degli studi di Padova dove si è implementato un software web con il gruppo di biologia computazionale guidato dalla Prof.ssa Chiara Romualdi.

L'obbiettivo del progetto è la re-implementazione di Graphite Web, un web server che mette a disposizione una serie di analisi statistiche su dati di espressione genica (microarray e RNA-seq).
Al momento il software consiste semplicemente in un'applicazione che riceve delle richieste di analisi, le esegue e ne restituisce il risultato. Per ogni analisi vengono eseguite delle operazioni computazionalmente pesanti, che impiegano tempo e consumano molta RAM.

Durante il progetto si è concentrato lo sviluppo sui seguenti tre aspetti.
Prima di tutto sulla ridondanza delle operazioni: se un utente esegue due volte la stessa analisi tutte le operazioni di controllo ed analisi già eseguite non vengono sfruttare ma ripetute ogni volta.
Per risolvere questo problema, nella nuova versione viene sfruttato un database che permette di tenere traccia di ogni task eseguito.
Un altro miglioramento consiste nel gestire il trasferimento dei dati utilizzando lo streaming con lo scopo di risparmiare RAM.
Infine si vuole sfruttare il calcolo concorrente e parallelo per poter eseguire più operazioni alla volta così da risparmiare tempo durante l'esecuzione dei task più pensati.
La nuova versione di Graphite Web è organizzata in un'architettura multi-strato implementata principalmente in Python. Il software consiste in un server HTTP basato su gunicorn e gevent.
La combinazione di questi miglioramenti rende l'utilizzo del software più snello, facendo risparmiare tempo ai ricercatori durante l'analisi di dati high-throughput di espressione genica, che sono dati di grandi dimensioni.

Nei primi capitoli della tesi si eseguirà un'analisi con Graphite Web di una caso studio su un dataset, di tipo RNA-seq, di tumore all'ovaio, poi verranno analizzati i limiti del software.
Successivamente spiegheremo l'implementazione della nuova versione.

Contents

Chapter 1

Introduction

In this chapter we give to the reader the essential notions of biology to understand the importance of analysis on expression data speaking about DNA, RNA and proteins.

Then we explain the Gene Set Analysis that has the aim of identifying the different groups of functionally related genes with possibly moderate, but coordinated, expression changes across different biological conditions.

Finally, we introduce two Bioconductor packages developed by the computational biology team of University of Padova that allows to perform Gene Set Analysis exploiting pathways information: `graphite` and `clipper`.

1.1 DNA, RNA and proteins

Every single somatic cell, about 10^{14} in human body, contains a copy of the DNA (deoxyribonucleic acid). In the eukaryotic cells the DNA is stored in the nucleus and it contains all the biological information essential for life. According to the model of Watson and Crick of the year 1962, its structure consists in two helical chains formed by the nucleotides. The two chains are related by nitrogenous bases, known as nucleobases.

In every cell there are four different DNA nucleotides, each defined by a specific nucleobase: adenine (A), thymine (T), guanine (G), and cytosine (C), they bind together in characteristic way as shown in figure 1.1.

A genome is the complete set of genetic information, stored in long molecules of DNA called chromosomes, 46 in human cells, useful to code proteins necessary to life. The human genome contains $3.2 \cdot 10^9$ base pairs so it has a storage capacity of $2 \cdot 3.2 \cdot 10^9$ bit, approximately 700 Mb.

Small sections of DNA, called genes, code for an operative biological product: functional RNA (fRNA) or RNA that contains assembly information for proteins that are molecules required by the organism for essential life functions [4].

RNA (ribonucleic acid) is a single-stranded nucleic acid like the DNA with a different base: uracil (U) instead of thymine, which is formed by short sequences and so it degrades more easily than DNA.

There are various types of RNA that play different roles working as transfer RNA (tRNA), messenger RNA (mRNA) and others which control gene expression.

Less than 2 % of Homo Sapiens' DNA is coding (exons) while 98 % is non-coding

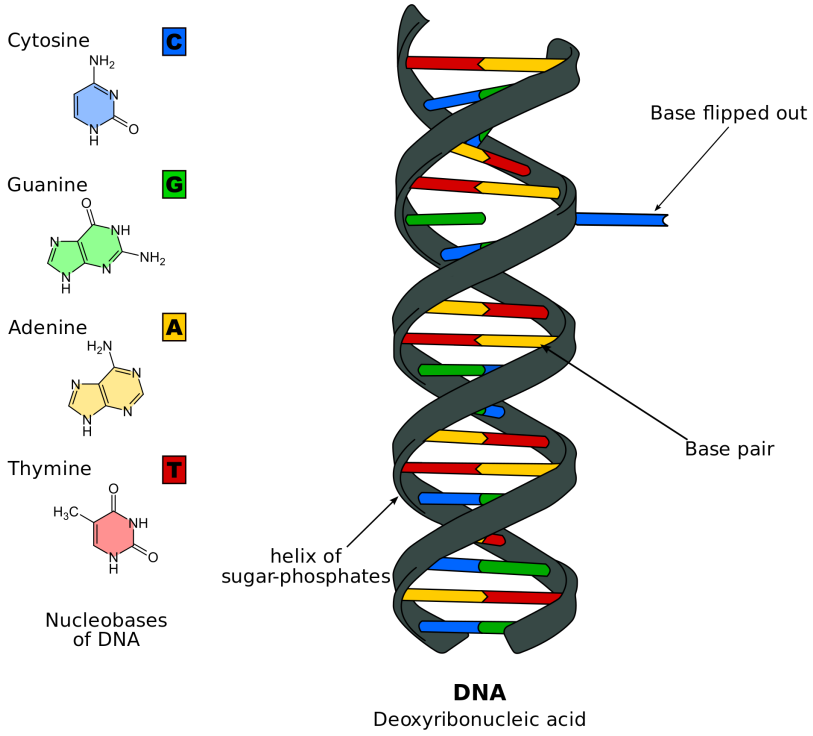
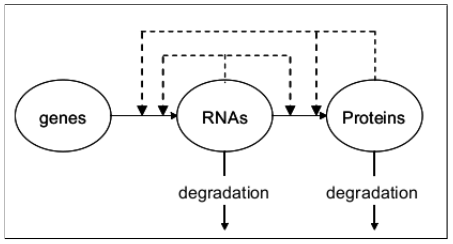


Figure 1.1: Structure of DNA: nuclobases bound together (according to base pairing rules (A with T, and C with G) with hydrogen bonds to build double-stranded DNA (dsDNA).

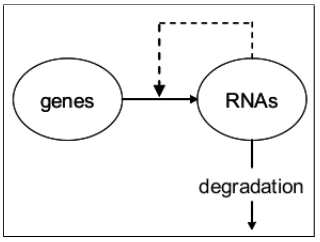
(introns and intragenic) and at the moment we have classified about 24000 over 30000 supposed genes.
The mRNA is subjected to splicing in which introns are removed by some enzymes and proteins are involved in this process.
The alternative splicing allows the synthesis of different proteins from the same transcripts. At the moment we have verified about 48000 over 150000 supposed transcripts.
During the normal activity of every organism, genes are continuously copied, transcripted and translated, in specific way that depends on the cellular kind, on the phase of the cellular life and on the external stimuli.
The gene expression is the passage of a gene, first at the mRNA in the cytoplasm (transcription process) and after that to the proteins (translation process).
The proteins, after their assembly, undergo some post-translational changes in order to define their properties.
The diversified gene expression allows a series of biological processes, such as cellular differentiation, development, cellular activity in space and in time, answer and the adaptation to the external stimuli.

The proteins and the functional RNA contain the information which regulate the transcription, thanks to their interaction aptitudes with the different molecules, especially the DNA, which are inside the cell.
The key to understand the cell differentiation through time space is the protein

expression (see figure 1.2a).
For instance we know proteins called enhancer that influence the gene expression in the phase of transcription.
Because of missing of the protein expression's data, we approximate it with the one of the RNA, as shown in figure 1.2b.



(a) Model of the interactions between regulatory molecules.



(b) Simplified model: the RNA approximates the protein expression.

Figure 1.2: System models that describe the regulation of the gene expression.

In this way we can study gene regulatory networks using the high-throughput technologies, like RNA-seq where the measure is the gene expression.

1.2 Gene sets analysis

The goal of Gene Sets Analysis (GSA) is to identify the different groups of functionally related genes with possibly moderate, but coordinated, expression changes across different biological conditions.
In general, the a priori definition of gene sets is obtained from Gene Ontology or from biological pathways.

The Gene Ontology (GO) is an ontology to describe the functional annotation, that is information about the function of genes and gene products, as knowledge of gene and protein roles in cells, in biological processes [1].
The GO defines three types of controlled vocabularies: cellular component (CO), molecular function (MF) and biological process (BP).
There are different kinds of database that contribute to GO, such as Berkeley DrosophilaGenome Project (BDGP), dictyBase (Dictyostelium discoideum), Reactome.
The Gene Ontology is structured as a directed acyclic graph (DAG), and each term has defined relationships to one or more other terms in the same domain, and sometimes to other domains [13].
Gene products, represented by identification code, are associated with the GO terms that describe their properties; in this way each GO term may contain a list of genes.

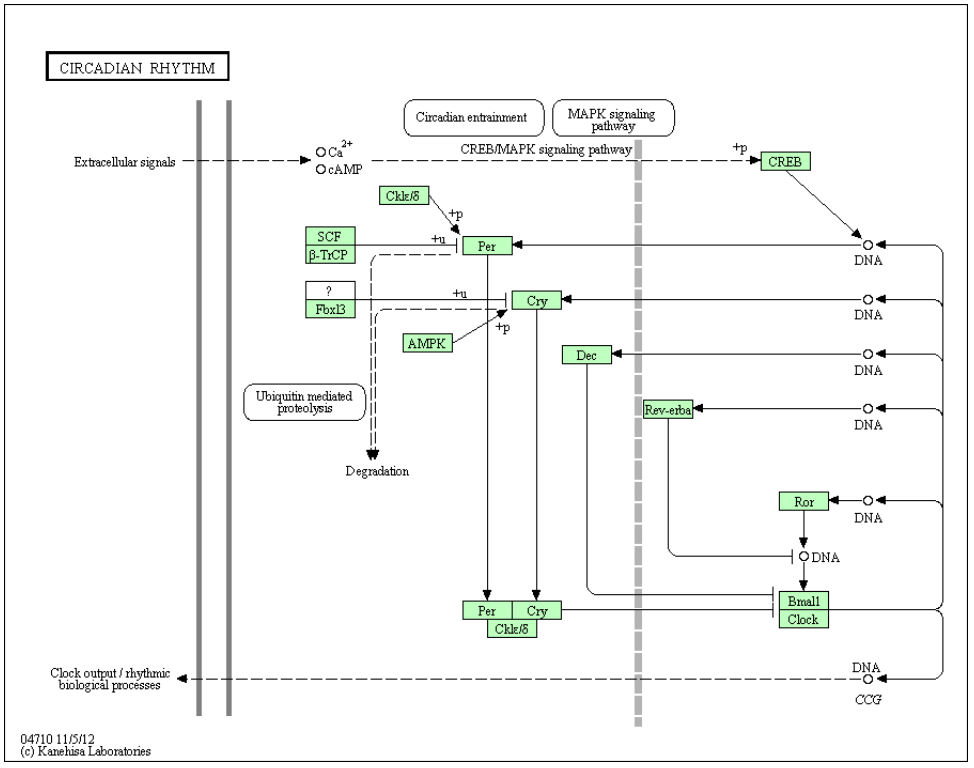


Figure 1.3: Circadian rhythm pathway is a network whole the relative GO:0007623 is a merely list of 1991 genes and gene products annotated.

Instead pathways are models delineated within the entire cellular biochemical network that help us to describe and to understand the specific biological processes.

A pathway can be defined as a set of interactions between physical or genetic cell components, often describing a cause-and-effect or time-dependent process, that explains observable biological phenomena [2] where genes and their relations are nodes an edges.

The main difference between them is that a GO term contains a list of genes that has no explicit connections among them (apart from being involved in the same function) while genes in the same pathway are structured in a network with some explicit biological interactions [5], for example see figure 1.3 that shown circadian rhythm pathway.

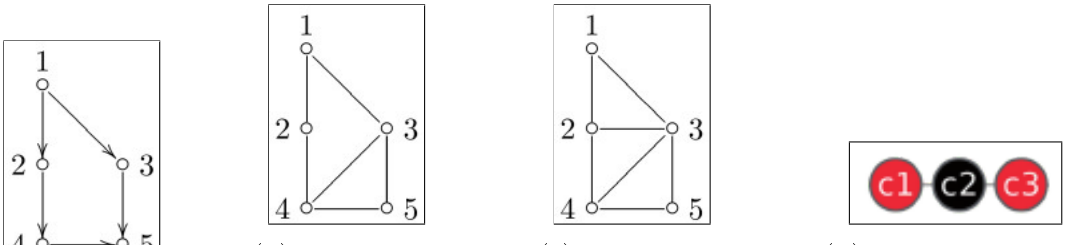
1.3 Two Bioconductor packages: `graphite` and `clipper`

GSA methods are based on two fundamentally different null hypotheses, these are called *competitive* and *self-contained*.
Suppose have to two groups of samples with two given phenotypes, the first type hypothesizes the same level of association of a gene set with the given phenotype as the complement of the gene set.
The main drawbacks of competitive methods are (i) the assumption that genes are independent; and (ii) the use of a cut-off threshold for the selection of differentially expressed genes (DEGs).
The second type only considers the genes within a gene set and hypothesizes that there is no gene in the gene set associated with the phenotype, relaxes the assumption of independence among genes belonging to the same gene sets and does not require arbitrary cut-offs.
Self-contained methods are divided into *non-topological* when applied to biological pathways, most of them use merely the list of genes belonging to a pathway, and *topological* that exploit the topological information.

To realize topology-based GSA (TGSA), we need to convert pathways into gene networks inside which each node is a measurable variable that is gene expression.
In order to solve this problem we use `graphite`, a Bioconductor package that takes pathway information from four distinct database (Biocarta; KEGG; NCI/Nature Pathway Interaction Database; Reactome) and converts them using specific rules (see [10] for more details).

A pathway is a graph that models a biological process in which nodes are gene products like protein complexes, gene family members and chemical compounds while edges are their interactions.
With the purpose of carry out the TGSA analysis, a pathway has to be converted into a gene/protein network using `graphite`.

Since we expect that only some portions of pathways are altered, it is important to find out the signal paths within a pathway mostly involved in a biological problem.
In this context, `clipper`, a Bioconductor package, is a two-step empirical approach



(a) DAG (b) moralization (c) triangulation (d) consecutive cliques

Figure 1.4: Example of a DAG (a), the corresponding moral graph (b), and one possible triangulation of the DAG (c). Finally the corresponding cliques (d) formed by relative nodes: c1 (1-2-3), c2 (2-3-4) and c3 (3-4-5)

.

that, after selecting significant pathways, identifies within those pathways the signal

paths having the greatest association with a specific phenotype.
In particular, `clipper` needs to convert the graph. Such conversion might require some or all of the following steps: moralization (figure 1.4b), triangulation (figure 1.4c), clique identification (figure 1.4d) and junction tree construction (figure 1.5), more details are in [5].

Clique identification finds out the cliques of the triangulated graph, i.e. the complete sub-graphs having all their vertices joined by an edge; junction tree construction builds a new hyper-tree having cliques as nodes and satisfying specific properties.

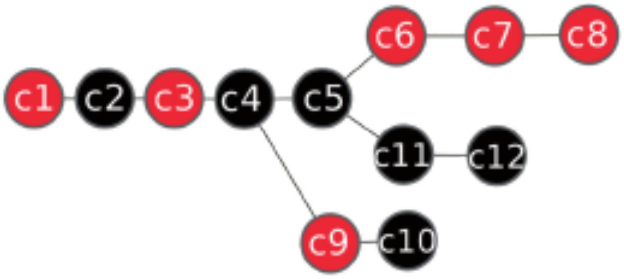


Figure 1.5: Example of a junction tree. The red cliques are significant. In this case the relevant path is only c1-c2-c3-c4-c9-c-10 in according that we accept only one black clique, not significant, between two red ones.

The first step of `clipper` consists on testing the whole pathway, modelling data, cases and controls in the two experimental conditions with two graphical Gaussian models with the same indirect graph G:

$$M_1(G) = \{Y \sim N_P(\mu_1, \Sigma_1), K_1 = \Sigma_1^{-1} \in S^+(G)\} \tag{1.1}$$
$$M_2(G) = \{Y \sim N_P(\mu_2, \Sigma_2), K_2 = \Sigma_2^{-1} \in S^+(G)\} \tag{1.2}$$

where Y is the multivariate distribution of expression, P is the number of genes (vertices of the graph), K_1 and K_2 are the concentration matrices (inverse of the covariance matrices) of the two models and $S^+(G)$ is the set of symmetric positive definite matrices with null elements corresponding to the missing edges of G [5].
The strength of the link between genes is tested with $H_0 : K_1 = K_2$ while the differential expression of pathway is tested with $H_0 : \mu_1 = \mu_2$.

In the second step `clipper` identifies the relevant signal paths (list of consecutive significant cliques) only for pathways detected in the previous phase.
For each pathway and the corresponding moralized graph, the approach is based on three main steps:

1. construction of the junction tree;
2. test on cliques in mean or variance;
3. identification of paths as list of consecutive cliques;
4. computation a path score proportional to the significance of the cliques (see figure 1.5).

The path with the maximum score is then selected. Finally the `clipper` results consist on a number of relevant signal paths as you will see in the next chapter.

Chapter 2

Case Study

The purpose of this chapter is to show how to perform a typical statistical analysis using Grapithe Web and how it allows the pathways visualization.

Then we speak about the limits of the current implementation of Graphite Web.

2.1 Ovarian Cancer

The ovaries are two organs which are part of the female reproductive system.
They are situated in the pelvis, one on each side of the uterus: the hollow, pear-shaped organ where a fetus grows.
Each ovary is about the size and shape of an almond. These ovaries make eggs and female hormones, chemicals that control the way certain cells or organs work in the body.
The ovarian cancer is the kind of cancer that can infect an ovary.[6]

In 2012, ovarian cancer occurred in 239,000 women and resulted in 152,000 deaths worldwide. This makes it, among women, the seventh-most common cancer and the eighth-most common cause of death for cancer.[7]

Ovarian cancer is subdivided into stages using the FIGO (International Federation of Gynecologists and Obstetrics) system. The stages are shown in table 2.1.

I	Cancer is completely limited to the ovary.
II	Pelvic extension of the tumor or primary peritoneal tumor, involves, one or both ovaries.
III	Cancer found outside the pelvis or in the, retroperitoneal lymph, nodes, involves one or both ovaries.
IV	Distant, metastasis (i.e. outside, of the peritoneum).

Table 2.1: FIGO stages of ovarian cancer.

The symptoms of this cancer are usually absent in the first stage, this is the reason why it is diagnosed late, quite often after it has reached the second stage.
Because of it the study of RNA expression can give useful information about the progress of the cancer study.

In this thesis we test the first two stages of the cancer compared to the third one, with the purpose of analyze if occur some statistical differences between them.

2.2 Dataset pre-processing

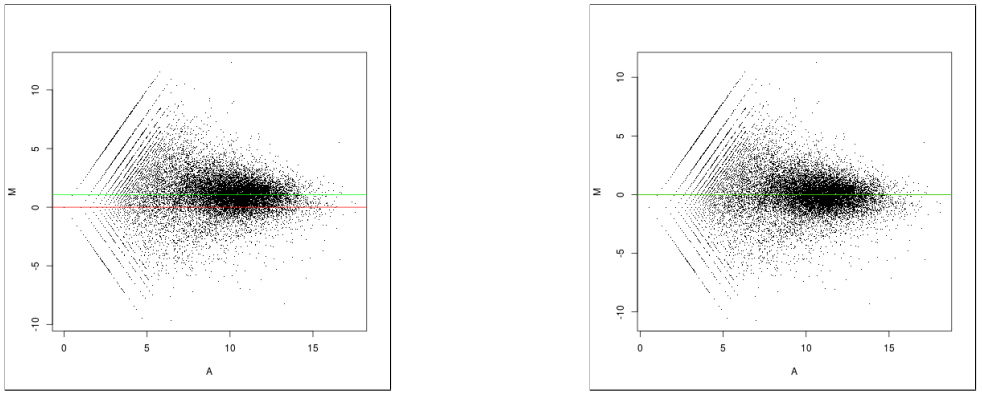
We examined RNA-seq expression data belonging to patients affected by ovarian cancer, the data are provided by The Cancer Genome Atlas (TCGA) project.
RNA-seq is the high throughput sequencing of cDNA, which derives from a process of a reverse transcription, using NGS technologies.
RNA-seq executes the sequencing of RNA molecules and profiles the expression of a single gene by counting the number of times that its transcripts have been sequenced.
The summarized RNA-Seq data is widely named as count data (e.g. table 2.2).

	sample 1	sample 2	—	—	sample N
gene 1	5	3	—	—	8
gene 2	17	23	—	—	42
gene 3	10	13	—	—	27
—	—	—	—	—	—
—	—	—	—	—	—
gene P	1507	1225	—	—	1455

Table 2.2: Count data contains genes in rows and samples in columns, thus the matrix contains the counts of transcript expression for every sample.

From TCGA-OC dataset we test patients in stages I and II vs stage III after normalization and filtering using a custom R script.

The count data is affected by some bias thus it is necessary to effectuate a preprocess it before executing the statistical analysis. Count data are often affected by crucial systematic bias that have to be removed before any subsequent analysis.



(a) MvA plot before normalization (b) MvA plot after normalization

Figure 2.1: MvA plot for a single patient where A is the mean of the log-intensities and M is the difference of the log-intensities, between two samples.
The red line corresponds with zero while green one with the mean. In fact after normalization they overlap as we wanted.

Normalization is the transformation of the data that should reduce this bias and it can be done within and between experiments to make counts comparable. Normalization assumes the following hypotheses:

1. the largest fraction of the samples is not differentially expressed;
2. symmetry between over and under expressed genes;
3. differential expression does not depend on the mean.

We normalize the dataset using the Trimmed Mean of M-values (TMM). In figure 2.1 we show MvA plot before and after normalization, each patient data is scaling on a reference patient.
Finally it is useful to filter the genes that are expressed at a very low level expression in each subject according to a threshold.

2.3 Statistical Analysis with Graphite Web

Graphite Web is a web tool for pathway analyses and network visualization for gene expression data [11]. It provides different statistical analyses and it is based on `graphite`.
Here we perform a `clipper` analysis as an example of a typical use of Graphite Web. Graphite Web pipeline is divided in the following steps: Graphite Web requires as

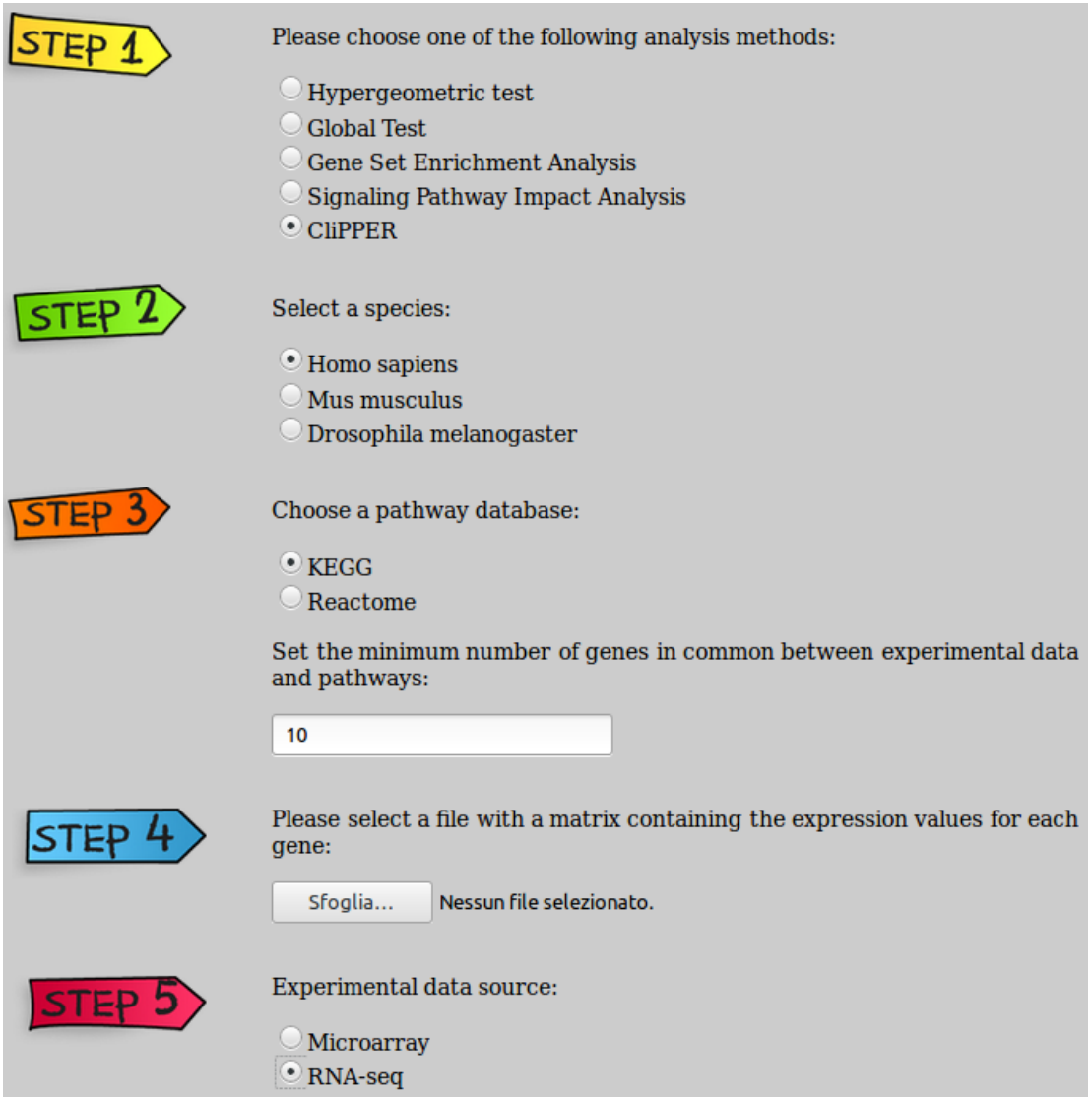


Figure 2.2: Graphite Web user experience.

input file a tab-delimited text files where the first row should contain sample names (the sample name represents the sample class) and the first column the gene IDs.
In this case we group I and II stages as class A and III stage as class B such as in table 2.3. We perform a `clipper` analysis for Homo Sapiens using the KEGG database.
The result consist on the list of selected pathways, shown in table 2.4, ordered by mean and variance test.

EntrezID	A	A	A	A	B	B	B	B
100133144	94	90	27	79	103	74	66	49
100134869	184	173	45	146	172	120	52	36
10357	111	229	99	25	91	32	106	256
645851	47	44	54	24	36	40	24	20

Table 2.3: An extract of count data input. The full matrix has 22 samples for population A and 240 samples for population B with 16618 genes (EntrezID) to test.

serial	Pathway	p value Mean	p value Var
1	Long-term depression	0	0
2	Melanogenesis	0	0
3	Axon guidance	0.01	0
4	Gap junction	0.01	0
5	Hepatitis C	0	0.01

Table 2.4: First 5 selected pathways.

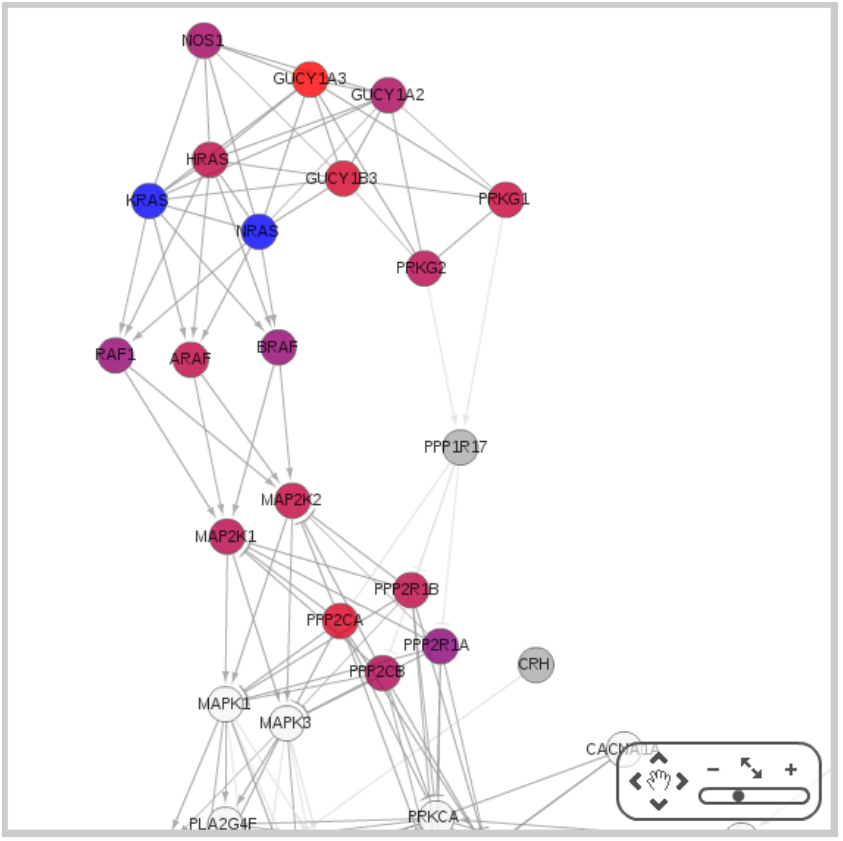


Figure 2.3: A portion of long-term depression pathway in which the most relevant paths are coloured.

For each of these pathways you can see in detail the relevant paths.

For example in the case of long-term depression pathway there are two relevant paths, in figure 2.3 one of these is coloured according to its score.

In this case we uploaded a RNA-Seq data of 19,4 MB contains 16618 rows and 262 columns (not very large) and exploiting a connection with 70 MBps in upload (very fast), the software takes 33 s to upload the file and to perform the analysis. Now if we try to upload the same file, the system executes again the same operations wasting time.

2.4 The aim of this thesis: a solution for limits of Graphite Web

Today, thanks to high-throughput technologies such as NGS sequencers, we can generate hundreds of gigabases of DNA and RNA sequencing data in a few days of work. Anyway, these raw results require further analysis before to be usable by researchers, and the manipulation of such big-data is time-consuming and even the simple transmission of them through the web requires a huge amount of time and resources.

Graphite Web is a web server able to provide different types of analysis which gives to researchers the chance to test different algorithms performances on data. The actual version of the software is a simple application that can run some statistical analyses on expression data.
In the case of a client requests twice the same analysis, the software executes again all the operations that have been already performed.
These operations are computationally expensive and we want to avoid this situation. In this project we work in the following limits for the actual version of Graphite Web:

- since a typical dataset can reach up 100 Mb, the state-of-the-art solution would be to stream data, while the current version just keeps to allocate more RAM for them;
- at the moment no database is used to store the results. Instead, when a database is implemented we could store both the knowledge of biology and the performed operations;
- the application works only with a single process right now. With this system, all the requests are execute one on one, generating a long queue of pending operations before to effectively run them.

In order to solve these problems, we have designed a new architecture of Graphite Web and its implementation consists in a database to store data, a business logic that works using streaming and a server that uses concurrent and parallel computing. In the next chapter we describe how we implemented these solutions.

Chapter 3

A multi-tier architecture for Graphite Web

In this chapter we describe the requirements analysis that explains how the software should run.

Next we see the multi-tier architecture of Graphite Web that is formed by the following parts:

- storage tier: it is responsible for the data persistence mechanisms (file system and database);
- business logic tier: this part controls the application's functionality;
- presentation tier: this is the topmost level of the application and it communicates with other tiers in order to display information.

This structure allows to make each part of application independent from others.

3.1 Requirements analysis

The goals of the system is to manage the knowledge of biology delivered by different pathway database and to allow some statistical analyses through the web.
In particular, the system has to satisfy the following requirements:

1. has to manage the updating of the knowledge of biology;
2. has to permit statistical analysis only between entities with the same version;
3. has to manage large datasets (e.g. 100 MB) without loading them all in memory but using streaming;
4. has to track any executed operations;
5. has to work asynchronously in case of expensive operations;
6. has to implement a mechanism of garbage collection.

The entities are divided into three main categories:

1. biological data: knowledge bases, nodes, interactions and pathways;
2. user data: microarray or RNA-Seq dataset;
3. analysis data: the result of different kinds of analysis;

The system does not aim to manage the biological relations between entities because they are treated by biological database.
In fact, the software keeps track of the version of every entity, in this way the analysis can be performed only between the entities with the same version.
Lastly, the system should be easy to maintain with the possibility to extend its features.

3.2 Storage tier

In the design of the software every entity is uniquely represented by its content in terms of bytes, which we will called "blob", and by some metadata which describe it.
The database is a collection of structured data, which have a different form and which are logically related to each other: it is possible to obtain information from these data.
Its contents is definable as a table of records, and every record has some different fields. In general, a record can contain several fields describing different types of content.
We could save the blobs inside a database field but if we consider that a typical expression matrix (dataset) can reach up 100 MB, we prefer to store it inside the file system.
In this way we have more control of blob storage mechanism independently of a specific database.

3.2.1 File system

A typical blob size of nodes, pathways and interactions can be around 20 MB while dataset is about 100 MB.

To ensure both efficiency and simplicity in the storage of blobs we have decided to exploit the file system. In order to manage stored blobs we use two tables: *BLOBS* and *BLOB_HINTS*, as shown in figure 3.1.

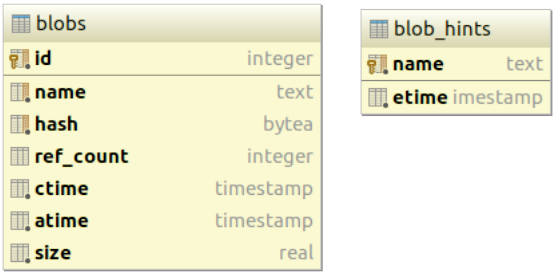


Figure 3.1: The tables relative to the phase of storing blobs: *BLOBS* and *BLOB_HINTS*.

The table *BLOBS* helps to describe a blob with metadata, in which:

- *id* is the primary key;
- *name* is the filename;
- *hash* is the hash code that depends on the content of the blob;
- *ref_count* is the number of other entities that are linked to this blob;
- *ctime* is the creation time;

- *atime* is the last access time;
- *size* is the size of blob in bytes.

The table *BLOB_HINTS* represents the hint to save a file with a certain filename and it is useful for the garbage collection how we explain in the next section.

Writing a blob

The phase of writing of blob consists on the following steps:

1. we assign a *name*, generated as a random UUID (universally unique identifier) to the new blob, e.g. 'c9acb2e6-e7d0-4af3-ba77-ee50d7499183';
2. we save this *name* in *BLOB_HINTS* with the *ctime* as the current time;
3. inside a default folder */blobs* we create a first subfolder named with the first char ('c') of *name* and inside this an other subfolder named with the second and third chars ('9a');
4. the file is written in streaming in this computed path ('/blobs/c/9a/c9acb2e6-e7d0-4af3-ba77-ee50d7499183');
5. when the previous operation has been completed we compute a file hash code using SHA512 algorithm;
6. we save *name* ('c9acb2e6-e7d0-4af3-ba77-ee50d7499183') and *hash* in the table *BLOBS*.

In this way, thanks to the support of these two tables, we prevent concurrency problems that may happen when two processes try to save simultaneously to the same file.
In fact the hash code, differently to the filename, depends on the content of blob and this involves that in the worst case we save two identical files in different paths for the same blob. In the first time we insert a record both in *BLOB_HINTS* and *BLOBS* while in the second times only in *BLOB_HINTS*. Then the garbage collector will delete the duplicated files as explained in the next section.

Deleting blob

The deletion of unused files is a job for the garbage collector that periodically controls if some files are obsoletes.
This process exploits the tables *BLOBS* and *BLOB_HINTS* and consists in the following steps:

1. we delete all records from *BLOB_HINTS* where *etime* (expiration_time) $<$ current time that corresponds to the case of duplicated files explained previous;
2. we unlink these duplicated files;

3. we delete records in *BLOBS* where *ref_ count* = 0 and *atime* is passed from a certain period of time;

4. we unlink these obsolete files.

In this easy way we manage also the deleting mechanism of blobs.

3.2.2 Database

In this section we explain the conceptual schema of the database.
A knowledge_base represents the version of biology information that we use for an analysis on data expression in a specific moment.

The following fields are common for more tables: *ref_ count* is the number of references that use the relative instance, *ctime*: is the creation time, *atime* is the last access time.
The last two are useful to the garbage collection mechanism how explained in the previous section.

We model the biological data with the following entities: knowledge_base (KB), node, pathway and interaction. The corresponding tables are *KNOWLEDGES_ BASES*, *NODES*, *PATHWAYS* and *INTERACTIONS*.
A KB can contain many nodes, pathways and interactions and this involves a one to many relation among them.
The allowed states of a KB are:

- open: the phase of filling in with nodes, pathways and interactions;

- active: only one KB can be active at a time, we can execute a statistical analysis only on active KB, in this state KB are not modifiable;

- closed: is an old version of KB that will be erase from garbage collector we will satisfy the condition to delete an entity that depends on *ref_ count* and *atime* as already explained.

Nodes are characterized by species (human or other), biotype (gene or mirna) and its blob.
Pathways and interactions are characterized by species (human or other) and their blobs.
The relations one-to-many from nodes, pathways and interactions to KBs and blobs are implemented as shown in figure 3.2.

The user data corresponds to microarray or RNA-Seq data. These are uploaded by users in order to carry out statistical analyses but, before of this, the user data have to be validated.

For this reason the insertion of expression data happens in two phases, using two different tables:

1. validation: the relative blob has to be validated in terms of format and meaning of data, for this reason it is saved in table *VALIDATIONS* and it stays here until is not validated;

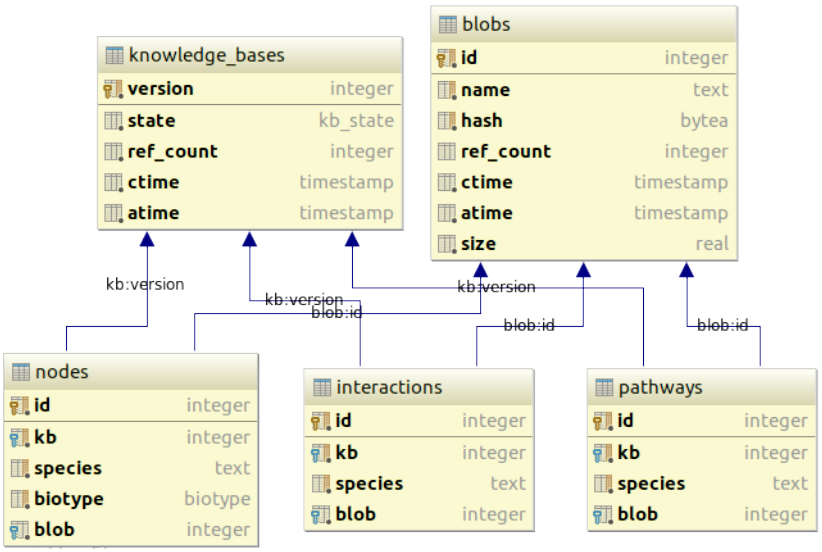


Figure 3.2: Tables that describe the biological data.

2. dataset: when a validated blob (validation) becomes a dataset, it is moved from *VALIDATIONS* to *DATASETS*; the validation process of a dataset can pass or fail.

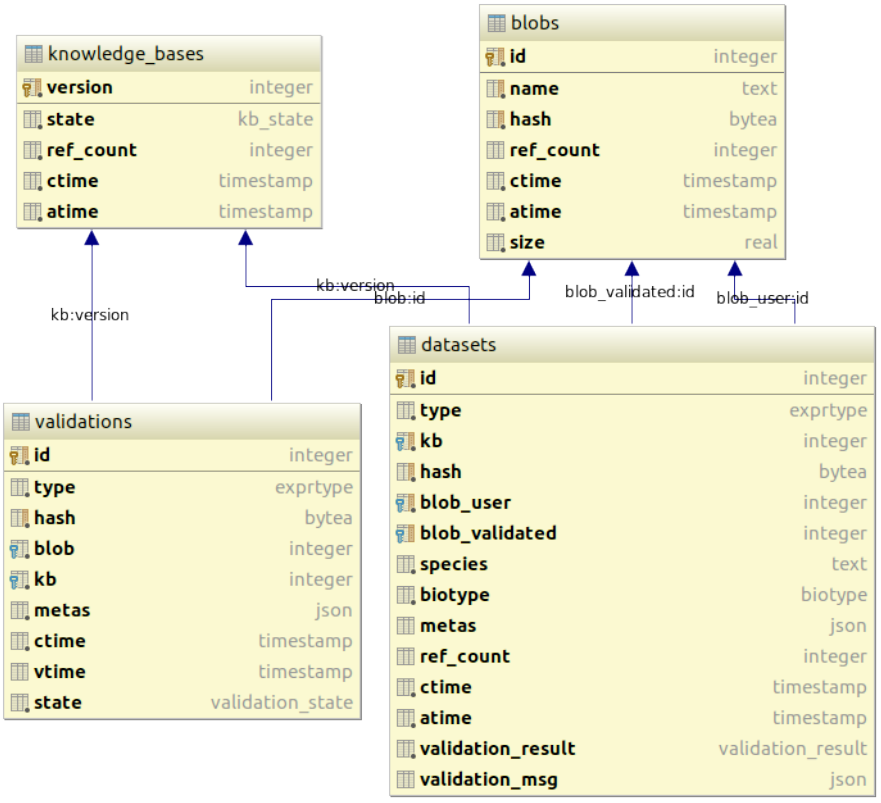


Figure 3.3: Tables that describe the user data.

VALIDATIONS and DATASETS have the same *hash*, *vtime* is the validation time that serves to notify when a process takes too long to validate a dataset; *type* is microarray or RNA-Seq.
An analysis can be in state "to do" or "in progress". The field *metas* is a json that contains parameters about the specific dataset.
DATASETS has more fields, which are: *blob_ validated* that is a modification of the original blob to execute in better way the analysis, *validation_ result* and *validation_ msg* that describe the result of a validation process; figure 3.3 shown these tables.

The analysis data are represented in similar way to user data how you can see in figure 3.4.

In the business logic we explain how we exploit these tables to carry out validation and analysis in an efficient way.

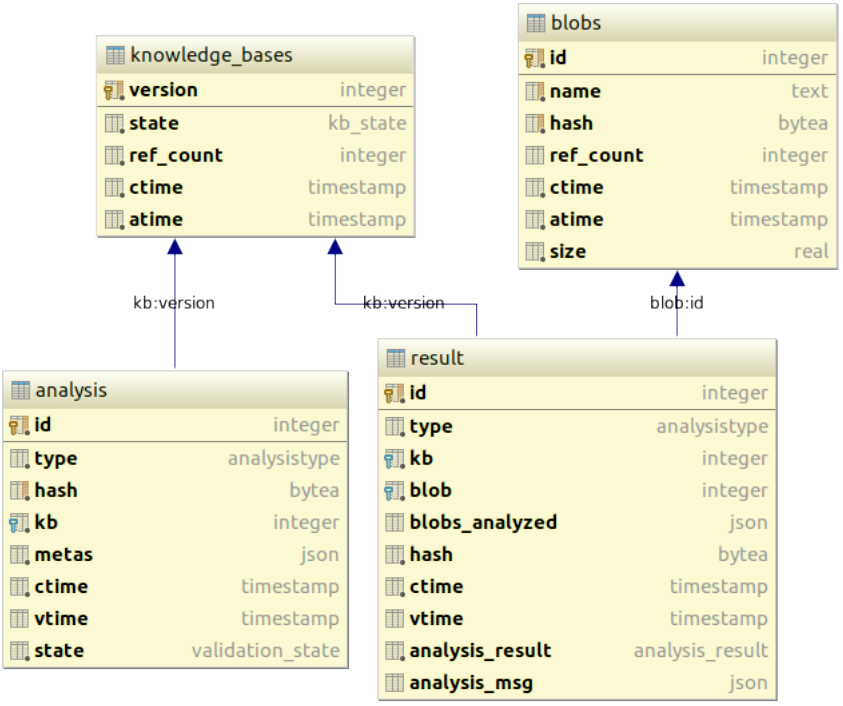


Figure 3.4: Tables that describe the analysis data.

PostgreSql (`https://www.postgresql.org/`)is an open source object-relational database management system (ORDBMS), for implement the storage tier we use PostgreSql 9.5.

We model dataset, and also analysis, with two tables. With this approach we insert redundancy in the conceptual schema and the tables are not normalized.
But in this way we can manage the concurrency in easy way avoid one or more fields set with NULL.
We have to think to concurrency because in the system more processes could run in the same time and in the worst case they could try to execute the same operation.
To allowed the mechanism of validation and analysis we exploit the NOTIFY com-

mand of PostgreSql.
It sends a notification event together with an optional "payload" string to each client application that has previously executed LISTEN channel for the specified channel name in the current database.
In our case there are two channel: "validation" and "analysis" so that different process can communicate between them.

3.3 Business logic tier

The business logic is developed in Python. The application allowed to manage the version of knowledge bases and its components. It also provides many types of analysis on the current version of the knowledge base.
The project is divided in two essential modules:

- storage: can access to database and provides an interface to model to execute queries;
- model: defines logic rules between different entities.

3.3.1 Storage module

The storage module has the responsibility to give to the model some methods to access the database so that the model can carry out the essential operations for running the application.
These methods can be divided in the following groups:

- saving blob operations:

```
stream_blob(blob_id) -> stream
add_blob(stream) -> blob
```

- insertion of biology entities:

```
insert_kb(kb_version) -> None
insert_nodes(kb, species, biotype, blob) -> bool
insert_pathways(kb, species, blob) -> bool
insert_interactions(kb, species, blob) -> bool
```

- registration of data expressions or analyses:

```
register_expr_data(kb, species, biotype, blob,
   expression_type) -> validation_info
register_anlysis(metas, analysis_type) ->
   analysis_info
```

- request if there are validations or analyses that are available to execute:

```
get_dataset_to_validate() -> Optional[bytes]
get_analysis_dependencies() -> Optional[
   analysis_dependencies]
```

- updating validations or analyses executed:

```
update_dataset(dataset_hash, blob, metadata,
   validation_result, validation_msg) -> None
update_analysis(analysis_hash, blob, metas,
   analysis_result, analysis_msg) -> None
```

- some useful queries:

```
highest_open_kb() -> Optional[kb_version]
highest_kb() -> Optional[kb_version]
has_nodes(kb, species, biotype) -> bool
has_nodes_by_kb_and_species(kb, species) -> bool
has_nodes_by_kb_and_biotype(kb, biotype) -> bool
has_validation(validation_hash) -> bool
has_analysis(analysis_hash) -> bool
```

We take care to store blobs using streaming because we want to save resources for each operation.

When the model adds a new entity, such as a pathway, it has to add the relative blob and only then it can insert the pathway in the database.

All queries, in PostgreSQL, are implemented as atomic transitions using Psycopgs, a Python module to communicate with PostgreSQL.

3.3.2 Model module

This module contains the logic rules to each application operation, these are implemented in specific functions and are divided for type of entity:

- knowledge_bases: the state of a new KB is open, in this state we can insert nodes, pathways and interactions but we can't perform any analyses.
  If for each species saved in database there is almost a node and a pathway of genes, then the KB can pass to the active state in which we can execute analyses.
  When we want to update the biology information we create and populate a new KB and at the moment of its activation the previous KB will be closed.

- nodes: for a given species, a node can be a gene or mirna and its biological information is stored in a blob that may change for different KB versions;

- pathways and interactions: for a given species, its biological information is stored in a blob that may change for different KB versions;

- datasets: it can be microarray or RNA-Seq; it is relative to a specific species and before we save a dataset we must check if nodes and interactions of its species are stored in the database.
  After the storage of the corresponding blob, we insert a record in *VALIDATIONS* with a hash as the primary key that depends on the dataset metadata and KB.
  When an external process validates the dataset, this is moved in *DATASETS* populating it with the same metadata and with the result of the validation process (passed or failed).

- analyses: for one or more datasets we can perform analyses.
  After the check if there is at least one gene dataset, we insert a record in *ANALYSIS* with a hash as the primary key that depend on the specific analysis

and KB.
When an external process executes the analysis, this is moved in RESULTS populating it with the same metadata and with the relative blob that contains the result of analysis.

The model module can perform all checks thanks to the interface exposed by storage module.
A sub-module common has the goal of provides common utilities to other module, such as enum or recurrent pieces of code.
In figure 3.5 we show the structure of the Python module of model.

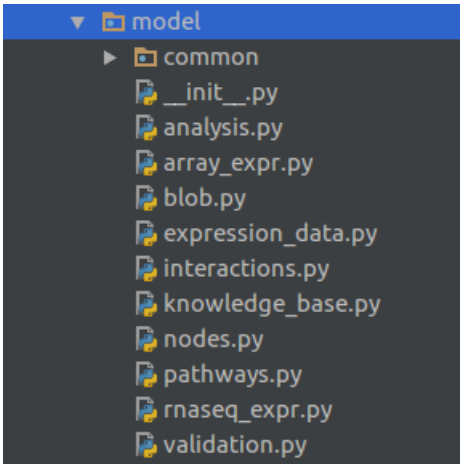


Figure 3.5: The Python package of model that contains a module for each entity to manipulate.

3.4 Presentation tier

3.4.1 Server module

We implemented the server according to Web Server Gateway Interface (WSGI) defines in a document called PEP-3333 [3].
A Web Browser communicates with a Web Server using Hypertext Transfer Protocol (HTTP) which functions as a request-response protocol. In this process of communication, we send a request to the Web server and expect a response in return [8].
We use the following HTTP request methods:

- GET: this fetches information from the given server using given URI (Uniform Resource Identifier);
- PUT: this creates or overwrite all the current rappresentation of the target resources, when we intend to create a new URL;
- POST: this can submit data to the server that we wish to process.

A Uniform Resource Locator (URL), commonly informally termed a web address is a reference to a web resource that specifies its location on a computer network and a mechanism for retrieving it.
In order to mapping URL pattern of the application we use the Python package Wheezy Routing.
We implemented these GET handler:

- *http://host/kb*: returns information about all KBs saved in the database;
- *http://host/kb/lastOpen*: returns number of highest open KB;
- *http://host/kb/{version}*: returns information about KB for the passed version;
- *http://host/validation*: returns a validation to carry out; if there are none then it waits some seconds to try again, this because someone could register a new dataset meantime;
- *http://host/dataset/{dataset_hash}*: returns validation information about dataset hash passed (microarray or RNA-Seq), previous registered; if there are none then it waits some seconds to try again because the validation process may be in progress;
- *http://host/analysis*: returns an analysis to carry out; if there are none then it waits some seconds to try again because someone could require a new analysis;
- *http://host/analysis/{analysis_hash}*: returns analysis information about analysis corresponding to the hash passed; if there are none then wait some seconds to try again because the analysis process may be in progress;

We implemented these PUT handler:

- *http://host/kb/{version}*: inserts a new KB with passed version;
- *http://host/kb/{version}/activate*: actives the corresponding KB;
- *http://host/kb/{version}/{species}/{biotype}*: inserts a new node for species and biotype in a specific KB;
- *http://host/kb/{version}/{species}/interactions*: inserts a new interaction for species in a specific KB;
- *http://host/kb/{version}/{species}/pathways*: inserts a new pathway for species in a specific KB;
- *http://host/RNA-Seq/{species}/{biotype}*: registers a RNA-Seq dataset that will be validated if it is in according of the correct format;
- *http://host/microarray/{species}/{biotype}*: registers a microarray dataset that will be validated if it is in according of the correct format;
- *http://host/analysis/{analysis_type}*: register an analysis to execute.

We implemented these POST handler:

- *http://host/dataset/{dataset_hash}*: validates a dataset, it receives the metadata;
- *http://host/result/{analysis_hash}*: updates the result of an analysis, it receives the metadata;

These routes can be used by a client to execute some operations.

The Graphite Web server is built on the WSGI implementation delivered by Guinicorn that is a Python WSGI HTTP Server for UNIX.
We implemented a standalone application that is built on the Gunicorn framework (`http://gunicorn.org/`). We run it using as asynchronous worker gevent (`http://www.gevent.org/`) that is a Python networking library that uses greenlet (michrothreads with no implicit scheduling) to provide a high-level synchronous API .

3.4.2 Web client

We have two types of client: system administrator and user (typically a research).
The role of system administrator is to manage and update the knowledge of biology (through insertion of nodes, pathways and interactions) that will be used to execute statistical analyses from user.
In order to deliver these two different functionalities we need of two html web sites:

- Dashboard site: only for system administrator operations such as insertion of KBs, nodes, pathways and interactions;
- Graphite Web site: user interface that leads a researcher to load its expression data and to require a particular type of analysis.
  Then its must show the result of analysis.

We implemented these using react.js that is a framework, delivered from Facebook, that implements the Model-View-Control pattern.
This section is not the focus of this thesis because is developed by another student.

Chapter 4

Software development

In this chapter we explain the technical requirements that drove the implementation and the application development life cycle that we followed during this project.

4.1 Technical requirements specifications

We developed the software in according to the the following technical requirements:

- the programming language to adopt is Python 3.5;
- we have to adopt the test driven development (TDD) method;
- we have to manage the version control;
- we have to set up a continuous integration framework.

The first requirement is because the Python language is largely used by biologists so in the future someone, in this area, could extend the Graphite Web functionalities. Moreover Python is a very flexible and permits to implement many kinds of operations at different levels, e.g a software web or numerical analyses.

The other requirements help us to work in team with minimum effort and maximum efficiency.

4.1.1 Python programming language

Python (`https://www.python.org/`) is widely used in computer science and in particular by computational biologists.
Python is an high-level Object-oriented programming language that support also other paradigms. It was created by Guido van Rossum in 1991.

Python is a dynamically typed language: the check of variable types is performed only at run-time.
The benefit of this approach lies in the fact that typing phase is fluent and rapid but, in on other hand, it is easy to make a mistake when to software structure is complicated.
To avoid this problem we use mypy (`http://mypy-lang.org/`) that is an experimental optional static type checker for Python that aims to combine the benefits of

dynamic and static typing.
Python supports the optional annotation of types: both in variables and function or method parameters (both input and output).
If we annotate our code, we can check the coherence of these annotations using mypy.

Python language generates code documentation exploits docstring. A docstring is a string literal specified in source code that is used, like a comment, to document a specific segment of code.
During all phases of development we wrote the docstring for each essential method. Then, in order to auto generate API documentation for modules we used pdoc (`https://github.com/BurntSushi/pdoc`), a Python tool, to creates the documentation of Graphite Web source code in javadoc style.

We decided to adopt Pycharm (`https://www.jetbrains.com/pycharm/`)as Python IDE (Integrated development environment) delivered by Jet Brains to be more productive. Pycharm provides these useful functionalities:

- smart completion of code;
- code inspections;
- on-the-fly error highlighting and quick-fixes;
- automate code refactorings as rename and delete, extract method, introduce variable, and others;
- debugging, testing and profiling;
- connection to various types of database;
- version control integration;
- an interactive Python console;
- a Linux shell.

The debug functionality of Pycharm is essential to be productive, it allows to watch the content of variables during the execution of the application or the tests.
The Python Standard Library contains many built-in packages to manage various aspects of an application such as zip, json, stream, audio and many others. In few specific cases we used external Python package, they are the following:

- psycopg2: to connect to the PostgreSql database;
- wheezy.routing: to manage the url patterns;
- gevent: to provide concurrent computing with micro-thread that optimizes the use of resources;
- gunicorn: to implement the server, it is a pre-fork worker model and provides parallel computing;
- zipstream: to generate file zip in streaming.

4.1.2 Test driven development

The basic concept of test driven development is to write a failing test, before writing any code. The tests should drive the development by failing in a way that allows to write a piece of code.
Writing our test first forces us to think about the problem we are try to solve [9]. This process is a cycle consisting in:

1. to write a failing simple test;
2. to make the test pass;
3. to refactor: here we could also add new functionalities not yet tested;

and then again at the first step because we could need to add more tests.
In order to test the model and the storage, we structured the test module in two main sections:

- unit test in the folder *test/unit/* in which we test the model and the storage independently;
- integration test in the folder *test/integration/* in which we test the integration of the model with the storage.

The unit test verifies individual unit of code, typically as method, in isolation to see if given certain conditions its responds in the excepted way.
This because if any of the smaller part of the application do not function has expected, the application as a whole can fail completely.
We run tests with PyTest (`http://doc.pytest.org/en/latest/`) as alternative test runner to the built-in one. It offers many options as targeting specific test to execute, it support the debugging and it can check the coverage of test.

With the purpose of isolate a part of application during the test we exploit mocks. Python provides a built-in mock library to help developers in this work.
A mock is a dummy implementation of a piece of code that simulates the behaviour of the real application returning what we expected.
For example we want to test the model independently by type of storage and so we use mocks.

4.1.3 Version control

Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.

We version the source code with Git (`https://git-scm.com/`) that is an open source distributed version control system.
This means that the server has a backup of original files and when clients fetch a repository they make a local copy of files. Then they work on these files and push their changes on the server.
Git makes easy the teamwork by tracking each change of code. Each save is called commit and it is uniquely identified by a hash.

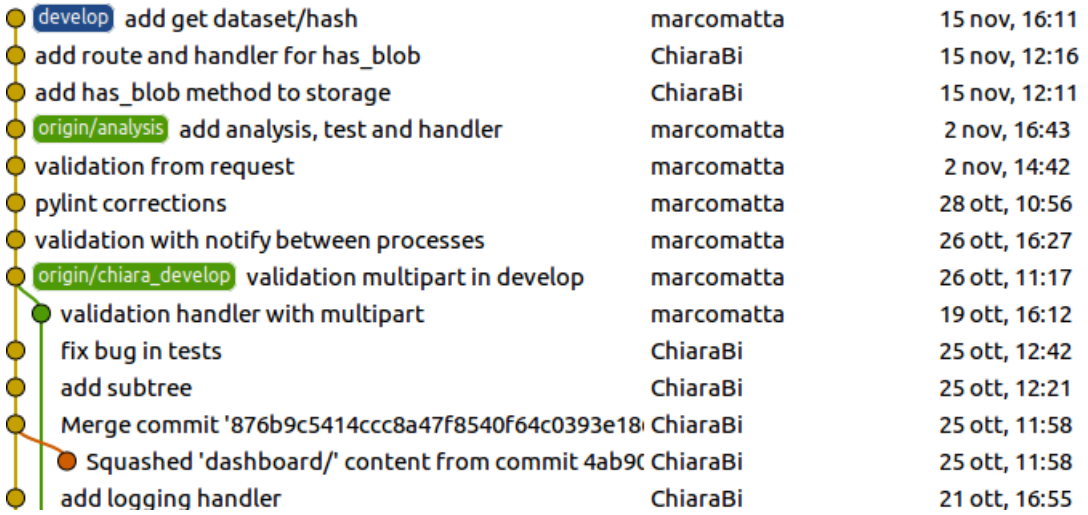


Figure 4.1: Git workflow of Graphite Web development.

Git runs in a similar way to a timeline: starting from a point, a commit, we can create alternative time lines called branch.
In general the main develop branch is called master and each developer can, starting from here, work in a personal branch. This branch, when the work is finished, can be merged in the master.
In Graphite Web the main branch is called develop and when a programmer wants to implement new features works in his branch; figure 4.1 shows the typical Graphite Web workflow.
This approach is largely adopted when it is necessary to work for a project in many developers at a time.

4.1.4 Continuous integration

When a system is under versioning and the team work following the TDD is a good practice to set up a system integration software as Jenkins (`https://jenkins.io/`) that provides these main functionalities:

- automatic build and history of the performed builds;
- automatic test and history of the result of tests;

Jenkins is a Web server written in Java with an user experience that periodically, or when a developer pushes updates, run a build that consists in testing the entire software with exiting tests and eventually in running some scripts.

4.2 The application development life cycle

The phases of development follow, in general, the division of entities.
Firstly we implemented the following steps for the the biological data, then for the user data end finally for the analysis data.
For each entity the steps consists in writing:

- the unit test;
- the corresponding model;
- the storage test;
- the storage;
- the integration test of model with storage.

The next step is implement the server. We prepared the server module and client in Python.
The client executes various requests to the server using the Python request module and we run this like a test.
This simple test consists in the following operations executed via HTTP request:

1. we run the server in localhost;
2. we populate the database with various KB version and create one that has nodes, pathways and interactions. We activate it so that we can perform analyses;
3. we try the activate command verifying that it works as we expected in various cases;
4. we upload a dataset and validate it;
5. we request an analysis and carry out it.

In this way we verify all implemented operations also through the server, they successfully run.

Chapter 5

Conclusion

5.1 Analysis of code

The PEP-8 is the official document that defines the style guide for Python code, e.g. the naming convention or the max line-code's length [12].

Pylint (`https://www.pylint.org/`) is a Python tool that creates a report showing some of the areas in code that do not meet the PEP-8 defined standard.
It helps the coding checking:

- line-code's length;
- if variable names are well-formed according to your coding standard;
- if imported modules are used.

It performs the error detection, e.g. checking if declared interfaces are truly implemented and if modules are imported and much more. Moreover it detects duplicated code.

Here we show some significant tables generated by Pylint:

type	number	perc
code	2826	66.78
docstring	286	6.76
comment	117	4.18
empty	943	22.28

Table 5.1: The project contains 2826 line of source codes.

type	number
convention	446
refactor	51
warning	120
error	6

Table 5.2: This table shows the category of check: convention no respected, need of refactor, warning and error.

module	error	warning	refactor	convenction
model.knowledgebase	0.00	2.50	1.96	2.02
model.blob	0.00	0.00	0.00	0.22
model.nodes	0.00	4.17	1.96	1.35
model.interactions	0.00	4.17	1.96	1.35
model.pathways	0.00	4.17	1.96	0.90
model.microarrayexpr	0.00	4.17	0.00	0.45
model.rnaseqexpr	0.00	4.17	0.00	0.45
model.validation	0.00	3.33	0.00	1.35
model.analysis	0.00	5.00	0.00	2.24
storage.persistent	0.00	0.83	9.80	2.69
storage.uniquefile	0.00	0.83	1.96	1.57
storage.stream	0.00	0.00	3.92	2.02
server.server	0.00	0.00	0.00	0.24
server.handlers.knowledgebases	0.00	5.00	0.00	2.24
server.handlers.nodes	0.00	0.83	0.00	0.45
server.handlers.interactions	0.00	0.00	0.00	0.45
server.handlers.pathways	0.00	0.00	0.00	0.45
server.handlers.userdata	0.00	2.50	0.00	1.12
server.handlers.validation	33.33	5.83	0.00	1.12
server.handlers.analysis	33.33	3.33	0.00	1.79

Table 5.3: This table shows the percentage of error/warning by module. We show only the most significant modules.

The global evaluation gives to Graphite Web source code a rate of 7.19/10, it is a good score but we can improve the quality of code following the information given by Pylint. During the development we have implemented test for each essential

module	statements	miss	perc cover
model.knowledgebase.py	65	4	94
model.blob.py	7	0	100
model.nodes.py	43	1	98
model.interactions.py	39	1	97
model.pathways.py	39	1	97
model.microarrayexpr.py	21	21	0
model.rnaseqexpr.py	17	0	100
model.validation.py	29	1	97
model.analysis.py	42	7	83
storage.persistent.py	155	10	94
storage.uniquefile.py	45	5	89
storage.stream.py	40	12	70

Table 5.4: This table shows the percentage of coverage for the most significant modules.

functionality.

Coverage (`https://coverage.readthedocs.io/en/coverage-4.2/`) is a Python tool that provides coverage information for the full program. In table 5.4 we show the coverage of some modules.

In the global analysis we have 3049 statements and we covered 73%, it is a positive result.

5.2 Concluding remarks

Graphite Web is a Web server that provides different types of analysis and which gives to researchers the chance of testing performances of different algorithms on data.
For example it provides simple algorithms as fisher test or more complex as `clipper` that after selecting significant pathways, identifies within those pathways the signal paths having the greatest association with a specific phenotype.

The goal of this project was to improve the implementation of Graphite Web to solving its limits.
The new version of Graphite Web server was built on the WSGI implementation delivered by guinicorn that is a Python WSGI HTTP Server.
In particular, we implemented in Python the following solutions:

- each transmitted data from and to server is sent as a stream to avoid to waste memory resources;
- the file system is used to save data and a PostgreSQL database manages the corresponding metadata. The database stores both the knowledge of biology and all performed operations;
- we implemented a gunicorn server that uses gevent as worker. With this solution we make the system able to use concurrent and parallel computing in order to can perform more analyses at the same time.

We structured the project as multi-tier architecture formed by this tier:

- storage that consists in the integration of file system and database;
- business logic that defines rules of the application, it is divided in two module: the storage (part of application that communicates with the database) and the model (part of application the provides methods to manage the entities);
- presentation that consists in the programmatic presentation delivered by server module and in the presentation of data that we can implement as a web site.

At the moment we are still working on the new version of Graphite Web to improve the quality of the implementation and to integrate all the statistical analyses provided by the old version.

List of Figures

List of Tables

Bibliography

[1] The Gene Ontology Consortium, Michael Ashburner, Catherine A Ball, Judith A Blake, David Botstein, Heather Butler, J Michael Cherry, Allan P Davis, Kara Dolinski, Selina S Dwight, Janan T Eppig, Midori A Harris, David P Hill, Laurie Issel-Tarver, Andrew Kasarskis, Suzanna Lewis, John C Matese, Joel E Richardson, Martin Ringwald, Gerald M Rubin, and Gavin Sherlock. Gene ontology: tool for the unification of biology. *Nature genetics*, 25(1):25–29, May 2000.

[2] Emek Demir, Michael P Cary, Suzanne Paley, Ken Fukuda, Christian Lemer, Imre Vastrik, Guanming Wu, Peter D'Eustachio, Carl Schaefer, Joanne Luciano, Frank Schacherer, Irma Martinez-Flores, Zhenjun Hu, Veronica Jimenez-Jacinto, Geeta Joshi-Tope, Kumaran Kandasamy, Alejandra C Lopez-Fuentes, Huaiyu Mi, Elgar Pichler, Igor Rodchenkov, Andrea Splendiani, Sasha Tkachev, Jeremy Zucker, Gopal Gopinath, Harsha Rajasimha, Ranjani Ramakrishnan, Imran Shah, Mustafa Syed, Nadia Anwar, Ozgun Babur, Michael Blinov, Erik Brauner, Dan Corwin, Sylva Donaldson, Frank Gibbons, Robert Goldberg, Peter Hornbeck, Augustin Luna, Peter Murray-Rust, Eric Neumann, Oliver Ruebenacker, Matthias Samwald, Martijn van Iersel, Sarala Wimalaratne, Keith Allen, Burk Braun, Michelle Whirl-Carrillo, Kei-Hoi Cheung, Kam Dahlquist, Andrew Finney, Marc Gillespie, Elizabeth Glass, Li Gong, Robin Haw, Michael Honig, Olivier Hubaut, David Kane, Shiva Krupa, Martina Kutmon, Julie Leonard, Debbie Marks, David Merberg, Victoria Petri, Alex Pico, Dean Ravenscroft, Liya Ren, Nigam Shah, Margot Sunshine, Rebecca Tang, Ryan Whaley, Stan Letovksy, Kenneth H Buetow, Andrey Rzhetsky, Vincent Schachter, Bruno S Sobral, Ugur Dogrusoz, Shannon McWeeney, Mirit Aladjem, Ewan Birney, Julio Collado-Vides, Susumu Goto, Michael Hucka, Nicolas Le Novere, Natalia Maltsev, Akhilesh Pandey, Paul Thomas, Edgar Wingender, Peter D Karp, Chris Sander, and Gary D Bader. The biopax community standard for pathway data sharing. *Nat Biotech*, 28(9):935–942, September 2010.

[3] P.J. Eby. *PEP 3333 – Python Web Server Gateway Interface*, 2010.

[4] Carrie Drovdlic Maggie Koopman Heidi Chial, Ph.D. *Essentials of Genetics*. Cambridge, MA: NPG Education, 2009.

[5] Paolo Martini, Gabriele Sales, M Sofia Massa, Monica Chiogna, and Chiara Romualdi. Along signal paths: an empirical gene set approach exploiting pathway topology. *Nucleic Acids Research*, 41(1):e19–e19, August 2012.

[6] NCI. Ovarian cancer prevention (pdq®), 12 2013.

[7] World Health Organization. *World Cancer Report*. ISBN 9283204298. World Health Organization, 2014. Chapter 5.12.

[8] Bala Subrahmanyam Varanasi Rakesh Vidya Chandra. *Python Requests Essentials*. PACKT, 2015.

[9] David Sale. *Testing Python*. WILEY, 2014.

[10] Gabriele Sales, Enrica Calura, Duccio Cavalieri, and Chiara Romualdi. graphite - a bioconductor package to convert pathway topology to gene network. *BMC Bioinformatics*, 13(1):1–12, 2012.

[11] Gabriele Sales, Enrica Calura, Paolo Martini, and Chiara Romualdi. Graphite web: web tool for gene set analysis exploiting pathway topology. *Nucleic Acids Research*, 41(W1):W89–W97, July 2013.

[12] Guido van Rossum. *PEP 8 – Style Guide for Python Code*, 2013.

[13] Wikipedia. Gene ontology - wikipedia, the free encyclopedia, 2016. [Online; accessed 19-October-2016].