ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADUADO EN INGENIERÍA INFORMÁTICA

Flujo de Trabajo para el Análisis Exhaustivo de Metagenomas

Computational Workflow for the Fine-Grained Analysis of Metagenomic Samples

Realizado por
ESTEBAN PÉREZ WOHLFEIL
Tutorizado por
OSWALDO TRELLES SALAZAR
Departamento
DEPARTAMENTO DE ARQUITECTURA DE COMPUTADORES

UNIVERSIDAD DE MÁLAGA

MÁLAGA, Junio de 2016

Fecha defensa:
El Secretario del Tribunal

RESUMEN

El desarrollo de nuevas tecnologías de adquisición de datos ha propiciado una enorme disponibilidad de información en casi todos los campos existentes de la investigación científica, permitiendo a la vez una especialización que resulta en desarrollos software particulares. Con motivo de facilitar al usuario final la obtención de resultados a partir de sus datos, un nuevo paradigma de computación ha surgido con fuerza: los flujos de trabajo automáticos para procesar la información, que han conseguido imponerse gracias al soporte que proporcionan para ensamblar un sistema de procesamiento completo y robusto. La bioinformática es un claro ejemplo donde muchas instituciones ofrecen servicios específicos de procesamiento que, en general, necesitan combinarse para obtener un resultado global. Los 'gestores de flujos de trabajo' como Galaxy [1], Swift [2] o Taverna [3] se utilizan para el análisis de datos (entre otros) obtenidos por las nuevas tecnologías de secuenciación del ADN, como Next Generation Sequencing [4], las cuales producen ingentes cantidades de datos en el campos de la genómica, y en particular, metagenómica. La metagenómica estudia las especies presentes en una muestra no cultivada, directamente recolectada del entorno, y los estudios de interés tratan de observar variaciones en la composición de las muestras con objeto de identificar diferencias significativas que correlacionen con características (fenotipo) de los individuos a los que pertenecen las muestras; lo que incluye el análisis funcional de las especies presentes en un metagenoma para comprender las consecuencias derivadas de éstas. Analizar genomas completos ya resulta una tarea importante computacionalmente, por lo que analizar metagenomas en los que no solo está presente el genoma de una especie sino de las varias que conviven en la muestra, resulta una tarea hercúlea. Por ello, el análisis metagenómico requiere algoritmos eficientes capaces de procesar estos datos de forma efectiva y eficiente, en tiempo razonable. Algunas de las dificultades que deben salvarse son (1) el proceso de comparación de muestras contra bases de datos patrón, (2) la asignación (*mapping*) de lecturas (*reads*) a genomas mediante estimadores de parecido, (3) los datos procesados suelen ser pesados y necesitan formas de acceso funcionales, (4) la particularidad de cada muestra requiere programas específicos y nuevos para su análisis; (5) la representación visual de resultados n-dimensionales para la comprensión y (6) los procesos de verificación de calidad y certidumbre de cada etapa. Para ello presentamos un flujo de trabajo completo pero adaptable, dividido en módulos acoplables y reutilizables mediante estructuras de datos definidas, lo que además permite fácil extensión y customización para satisfacer la demanda de nuevos experimentos.

Palabras clave: flujo de trabajo; análisis de metagenomas; ciencia de big data; asignación de lecturas; genómica comparativa

ABSTRACT

The development of new data acquisition systems has promoted an enormous quantity of available data in nearly all scientific fields, thus enabling a more specific-oriented research that is resulting in concrete software developments. In the aim of making it simpler for the final user to obtain processed results from raw data, a new computational paradigm is arising: the use of automated workflows for data processing. These have managed to take the lead because of their potential to build complete and reliable processing systems. In particular, the field of bioinformatics is a clear example where lots of worldwide institutions are offering specific services that need to be combined to produce valuable results. Automated workflow managers such as Galaxy, Swift or Taverna are constantly being used for data analysis, especially since the development of new sequencing technologies such as Next Generation Sequencing that are capable of producing vasts amounts of genomic and metagenomic data.
Metagenomics aims to study the collection of species present at their original environment. Within the metagenomics field, different types of studies can be carried out; among them, comparative studies of the changes in the composition of samples in the aim of identifying significant differences that correlate phenotype and the individuals, which includes functional analysis of the present species to determine the derived consequences of their presence. While genomics still offers computational difficulties, the science of metagenomics proposes a multiple analysis of unknown species, and therefore requires efficient algorithms capable of processing the enormous amounts of data in reasonable time. Some of the difficulties to be faced are (1) the sequence comparison process with a reference database, (2) the process of mapping reads to species (3) the large size of processed results requires functional storage to allow posterior use in an efficient way, (4) the specificity of each sample demands new processing methods, (5) representation and visualization of n-dimensional aspects and (6) quality control and verification methods to ensure the correctness of the process.

Therefore we present a complete workflow divided into software modules connected by open and reusable datafiles, which, in conjunction, enables future extension and customization to satisfy the demand of new experiments.

Keywords: workflow; metagenome analysis; big data science; reads mapping; comparative genomics

TABLE OF CONTENTS

CHAPTER 1.
INTRODUCTION

1.1. Motivation

The use of Internet in the scientific community has promoted an enormous quantity of available data in almost any existing field, enabling a more-specifically-oriented research. Such circumstances, however, are resulting in an increasing number of software projects being developed for particular scientific and processing tasks that once finished are abandoned since it becomes extremely difficult for the average user to run the particular tools on his/her own and therefore gives up rapidly. As a side effect, many of these developed tools are being included as Web Services (in advance WS), facilitating remote and universal access and removing the somewhat tedious step of downloading and installing tools on local machines. However, these increasing number of offered WS are triggering a scenario where processing tools exist for nearly everything, but it is becoming harder and harder to reach them, and has gotten to a point where the term "Web Services Fragmentation" is gaining popularity. In addition, such WS dispersion is causing work to be repeated over and over by different research teams, for searching for an already existing WS can be compared to looking for a needle in a haystack, and thus scientific teams are demanding for register sites to facilitate the so-called WS discovery. Furthermore, independent WS can be combined to produce new data-processing nodes that if not available would have need completely new software developments. This "combination" of tools and services are called Workflows, defined as "*the orchestrated and repeatable pattern of business activity enabled by the systematic organization of resources into processes that transform materials, provide services or process information*"[1]. Workflows have been used since 1921 by The Railway Engineer journal (Lawrence Saunders; S. R. Blundstone, 1921) but the term was not used in the same sense as today since Frederick Taylor and Henry Gantt (known for his popular Gantt charts). Automated workflow managers such as Galaxy, Taverna and MyExperiment [5] (among others) were born to offer a definition language to combine tools, along with the execution components needed to completely run the previously defined workflows. The wide use of these automated workflow managers mainly motivated the development of a series of independent tools and services, that put together would compose a full pipeline which, in addition, could be built and registered and used as a service via automated workflow managers.

Additional reasons that motivated the development of this workflow is the exponential growth of available metagenomic ("beyond genomes") data due to the improvements in the

[1] https://www.ftb.ca.gov/aboutFTB/Projects/ITSP/BPM_Glossary.pdf

low cost of sequencing technologies --such as 454-pyrosequencing[2], Illumina[3] or Ion Torrent[4] -- and the demand of worldwide laboratories requiring new methods to process their data and obtain custom results to address public health issues or to increase productivity in farms, among others. In addition, a vast amount of microbial diversity has been missed as a result of traditional microbiology requiring cultivation methods that are not able to recreate the natural environment of uncultured samples. Recent sequencing methods, known as Next Generation Sequencing have provided a new and broad scope of data obtention (i.e. retrieve uncultured samples directly from swamps or airplane windshields) from all-around-the-world sources at a feasible price and in reasonable time.

Along with sequencing methods, it also became necessary to store all the upcoming data and design fast access procedures to allow efficient retrieving of genetic material. Online databases such as GenBank [6] or the KEGG [7] database are fulfilling these needs and worldwide researchers are constantly uploading their work, producing tremendous quantities of genomic data, resulting in terms being coined such as the "post-genomics" era.

As a result of the availability of data, traditional algorithmic methods have got stuck, creating a processing bottleneck, as contrary to data bottlenecks in the early 90's and 00's. Therefore, a new path opens for the design of algorithms capable of treating Next Generation Sequencing data with ease, in order to answer all the questions posed by researchers. In this line, the developed workflow pretends to fulfil the need for a modular, flexible and exhaustive system capable of processing metagenomic samples in a fine-grained level.

In addition, this project has been developed under the group "Bioinformatics and Information Technologies Laboratory" BitLab[5], part of the Departamento de Arquitectura de Computadores, Universidad de Málaga. Workflows management systems are one of the main lines of research in this group and metagenomics analysis perfectly fits as application domain to benchmark the developments. Additionally, this is a collaborative work between several members in the group that have been working over the years in the field of bioinformatics and advanced computing. The work presented here is part of the software suite produced, along with others, not only in the group but with external collaborations, national and international projects, such as Mr. Symbiomath[6], ELIXIR[7], Science without borders[8], etc.

[2] http://my454.com/products/technology.asp
[3] http://www.illumina.com/techniques/sequencing.html
[4] https://www.thermofisher.com/es/es/home/life-science/sequencing/next-generation-sequencing/ion-torrent-next-generation-sequencing-technology.html#
[5] http://bitlab-es.com/bitlab/
[6] http://www.mrsymbiomath.eu/
[7] https://www.elixir-europe.org/
[8] http://www.cienciasemfronteiras.gov.br/web/csf-eng/faq

1.2. Project objectives

Listed below are the main goals that the proposed method intends to satisfy:

- Independent modules interconnected by datafiles:
    - Current state-of-the-art methods have been developed as pipelines that use both existing tools and datafile formats, and therefore its modules are exchangeable and can be replaced with upcoming newer versions and/or new processing tools.
- Efficient low-level programming for fast processing of data:
    - Although computational power rises up every two years according to Moore's law[9], the amount and size of data, along with more dynamic experiments which require lots of CPU time, still calls for efficient algorithms that handle I/O operations (especially on big data, where it is needed to access disk very frequently) and memory on a low-level, which pays out in terms of processing times and usage of resources.
- Data structure definitions that allow for third party software usage:
    - Traditional bioinformatics is a field where one project or software does not fit all necessities across worldwide researchers, but instead it demands lots of applications that must be welded together up to produce a large diversity of results.
- Visualization techniques to facilitate the understanding of the biological implications.
    - As every Big Data Science, comprehension of results is hardly done by observing numerical results and thus it is needed to provide with high-level, abstract visualization techniques that enable the user to understand the interactions of the data and the results.
- Open workflow to allow on-demand extension of modules.
    - Enclosed monolithic applications require lots of coding times in comparison to the small time that they will be put to use, and, besides, bug fixing and updates become a terrible difficulty to overcome that in many occasions causes the software to be abandoned sooner than expected.
- Customization of parameters and to offer flexible results:
    - Some state-of-the-art methods do not provide with flexible parameters and force users to run their experiments with default settings that not always suit the best results. Therefore, it is of interest to provide a wide range of customizable parameters that enables the researcher to obtain the result the way he or she really wants them.
- Fine-graining and low-level support:

[9] http://www.mooreslaw.org/

- All metagenomic analyzers focus on roughly *big* results, in the sense that they completely discard results that have small evidence (e.g. genomes that have few reads mapping to them) and treat these as statistically insignificant. However, in this project we focus on finding evidence to support those results that have insignificant value in terms of abundance.
- Novel tools: We will propose new tools that aim to provide strong evidence within those organisms that have low representation in the sample, and trying to meanwhile address the question “How much evidence is needed for a organism to be accepted as present?”

CHAPTER 2.
STATE OF THE ART

2.1. Introduction

In this chapter we will discuss the most commonly used tools (both online and local) to analyze metagenomic samples. However, before we start introducing concepts, software packages and typically-recurrent procedures, we should briefly address *what* is analyzing a metagenome. By *analyzing a metagenome* we mostly refer to all the process required to eventually determine the composition of such metagenome, i.e. determine which genomes are present in a sample. Nonetheless, it is not only about which genomes are there, but also about *in what measure* and *what are they doing*. We wish to obtain the expression levels of particular genomes in a metagenome and their functional profile, and this is what metagenomic analysis software mostly aim to. However, some important procedures are to be performed prior to any analysis, though they might not be strictly considered part of the analysis process. These comprise, in particular, the filtering and trimming of the samples. Filtering refers to the removal of redundant, low complexity regions, and more specifically, the removal of eukaryotic-DNA [8] origin (of humans or animals, mostly) since these have larger genomes and will waste analysis time and obscure the species that populate the sample. Trimming removes low quality bases, often introduced because of *base-calling* [9], which is the process of assigning bases to the chemical results obtained during sequencing.

The field of metagenomic analysis can be operated in many different ways, depending on the needs of the experiment. For example, in typical case-control analyses, it is of interest to use comparative metagenomics (i.e. observe variations between metagenomes) and functional analysis (i.e. what is every species doing in the sample and how). In another scenario, e.g. a team of researchers wants to compare how mutations affect a particular bacteria under different conditions, then they would perform what is called a mapping to reference genome, showing mostly where and how is occurring mutation. Furthermore, think of a government that needs to know the broad range of species present in the soil of a particular terrain, then they would perform a microbial diversity analysis, which is faster than mapping to a reference genome since it does not need to compute alignments. Therefore metagenomics provide a wide scope of experiments that are possible to carry out depending on the purpose of the experiment itself. However, the way in which these experiments are performed can vary much more and is subjected to more variables, such as whether proteins or nucleotides are used, if the type of database (in some experiments) is redundant or revised, etc.

In the subsequent sections we will illustrate and talk about the most common

metagenome analysis packages. These have been grouped into two big categories (1) Standalone tools that were developed in the aim of local and private executions, and (2) Online Web-based tools that collected data and enable users to run experiments with only a few clicks.

2.2. Standalone Tools

2.2.1. MEGAN

MEGAN [10] ("MEtaGenome ANalyzer") was born in 2007 as the first standalone metagenome analysis tool. It was initially designed for the use with short reads and focused on studying the composition of single datasets. It performs the mapping based on sequence similarity, and therefore requires a sequence comparison algorithm, which in this case is BLAST [11] search using any of the programs in the suite, such as BLASTX [12] (protein search), MEGABLAST [13] (short sequences) and BLASTN [14] (nucleotides search). Any of these searches are performed comparing the original reads to reference databases, such as GenBank. Then, a taxon ID is assigned to each aligned read based on the NCBI [15] taxonomy. The mapping step is performed using LCA [16] (*Last Common Ancestor*), which maps a read to the last common ancestor of all the fragments reported by BLAST, thus obtaining the read abundances per genome and phylum. MEGAN can work both with single-end reads and paired-end reads.

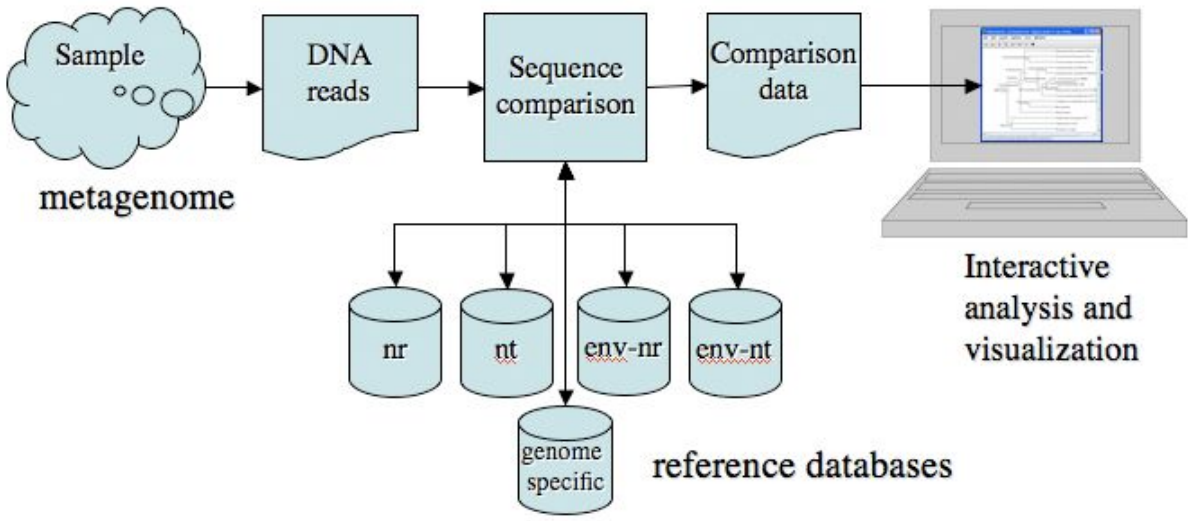

Figure 1: MEGAN's internal pipeline. First, a metagenomic sample is gathered and converted from its chemical nature to raw information (i.e. DNA reads). These are compared against a reference database (e.g. non-redundant database or specific genomes as seen in the figure), producing comparison data that summarizes the homologies found between the original sample and the reference. In the last step, MEGAN offers interactive analysis such as Taxonomical Tree View's or cladograms.

The latest updates of MEGAN (up to MEGAN 6, yet unpublished at the time) have

considerately extended its functionality, including functional analysis, which is performed using the SEED [17] classification of subsystems and functional roles or the KEGG classification of pathways and enzymes. In addition, long processing times due the similarity search performed with the BLAST suite has been tried to cut down by switching to DIAMOND [18], a 20,000 times faster sequence comparison algorithm. One of MEGAN's principal features is their well-known tree-taxa assignation, seen in Figure 2, which enables the user to see reads abundance easily at phylum level, and their easy-to-perform comparative metagenomics between samples, which produces typical bar plots between subsets of the reads abundance.

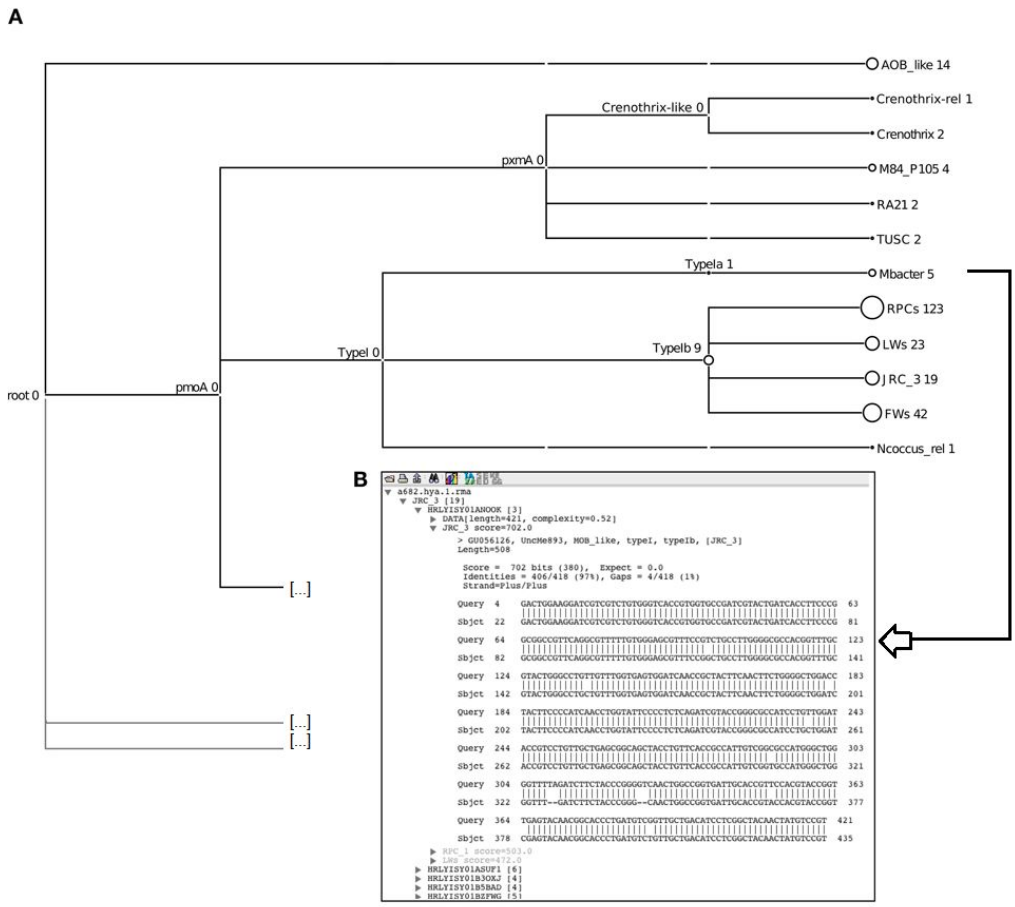

Figure 2: (A) Traditional MEGAN's tree view of reads assigned to taxa. Nodes represent phylum levels (up to the root node being for example the whole Bacteria family), whereas leaves are the particular genomes present in the reference database. (B) Checking the results of the BLAST comparison allows to see the alignments in detail for the mapped read.

MEGAN is implemented in Java[10] and is intended to be used on a local machine. The main drawback of MEGAN is the slow processing times due to its heavy implementation in Java. While many metagenomics analysis packages are developed in Java due to its simple and convenient high-level design, it is not the most appropriate for High Performance

[10] https://www.oracle.com/java/index.html

Computing. In addition, the complete dependency of the BLAST suite to run the sequence comparison steps (up to MEGAN 4 [19]) makes the software monolithic and slow, since there are new methods to compute homology searches in less time and with almost the same results (e.g. KRAKEN [20], CLARK [21] and GECKO [22]). Furthermore, MEGAN does not offer support for low-abundant genomes and only considers as results of interest those that have a large number of mapped reads, hence discarding any taxonomic result that is not directly supported by large mapped reads-abundance. A conceptual drawback of MEGAN is its lack of modular-pipeline design. MEGAN software only allows for fixed processing, i.e. it is not possible to contrast results using different sources, it can not be expanded upon wish and necessity, and more important, MEGAN's individual capabilities can not be combined to produce other results and therefore can not be reused with ease.

2.2.2. MetaPhlAn

MetaPhlAn [23] is a computational tool for profiling the composition of microbial communities from metagenomic shotgun sequencing data. MetaPhlAn can use either BLAST or Bowtie2 [24] to perform a search of the specific marker genes against a reference database.

MetaPhlAn is a fast metagenomic abundance estimation tool which maps reads to a set of selected marker sequences that are unique for each organism in the database. The marker sequences are carefully selected such that a read can only match to one marker and can therefore be assigned to a distinct organism. Together with the small size of the marker sequence database, this makes MetaPhlAn very fast while the accuracy is comparable to other reference based methods. However, since the genomes are reduced to short marker sequences, there is no possibility to detect or even quantify differences between the sequenced organism and the reference. Organisms that contain marker sequences of different reference strains (e.g. by horizontal gene transfer) show up in the results multiple times and it is not possible to detect such cases.

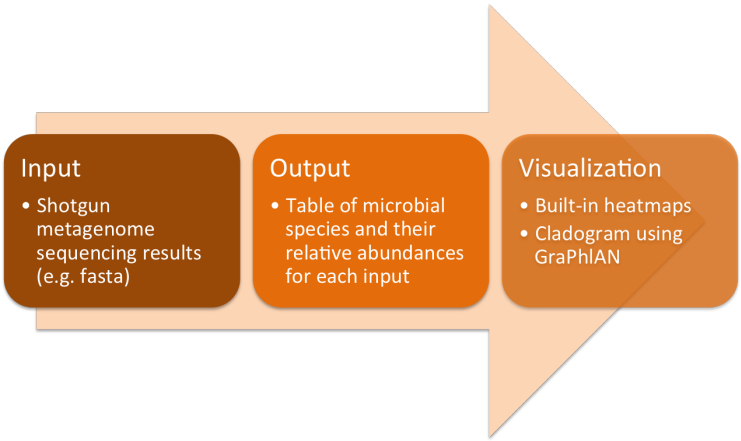

Figure 3: MetaPhlAn's processing pipeline overview. (1) Input stage takes a metagenome produced by shotgun sequencing (e.g. 454-pyrosequencing, Illumina), (2) the output is a table showing the reads abundance per species and (3), the ending visualization stage that enables users to interact and export heatmaps and cladograms, among others.

In contrast to MEGAN, MetaPhlAn is not designed to work with paired-end reads and therefore these type of reads should be concatenated in one single file to work with. MetaPhlAn has been developed under UNIX environments in the Python programming language, and it is mostly oriented to its use on shell, though it also provides a Galaxy workflow instance that can be downloaded and installed.

MetaPhlAn produces a tabular-text file with the reads abundance per species, along with a series of heatmaps and cladograms that help detect significant variation between species and understand the assignation of reads to taxa, respectively.

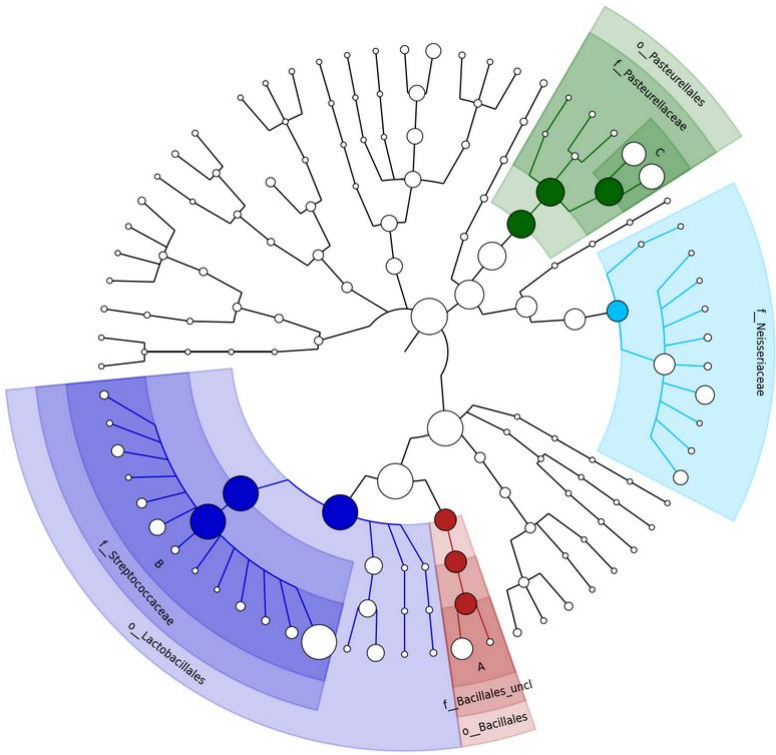

Figure 4: Cladogram produced by MetaPhlAn showing the assignation of reads to genomes by species, phylum and family. Although these type of graphs allow to take a first glance at the taxonomical results, it becomes hard to precisely determine the abundance differences among phylum.

However, since MetaPhlAn only classifies a subset of reads that map to one of its marker genes, its results are only estimations of the species that populate the sample and a large portion of the DNA material is unused, and therefore becomes unused information.

In a field as effervescent as metagenomics, which is steadily expanding itself with new experiments and knowledge extracting methods, it becomes necessary to enable a handy, easy extension of the capabilities of the analysis software, thus avoiding developing functional

packages that will not be used due to new requested experiments. In this line, MetaPhlAn offers a fully functional pipeline design which, in contrast with MEGAN, can be extrapolated to reuse in newer software developments and integrated into larger web-servers that offer metagenomics analysis.

2.2.3. MOCAT

MOCAT [25] is a software pipeline for metagenomic sequence assembly and gene prediction for taxonomic and functional abundance profiling. Since it is a gene prediction tool, it searches for single copies of particular marked genes across different databases. The automated generation and efficient annotation of non-redundant reference catalogs by propagating pre-computed assignments from 18 databases covering various functional categories allows for fast and comprehensive functional characterization of metagenomes.

The second version of MOCAT released supports Illumina single- and paired-end reads in raw FastQ format. MOCAT2 [26] generates taxonomic and functional profiles, as well as assemble reads and predict genes in assembled sequences.

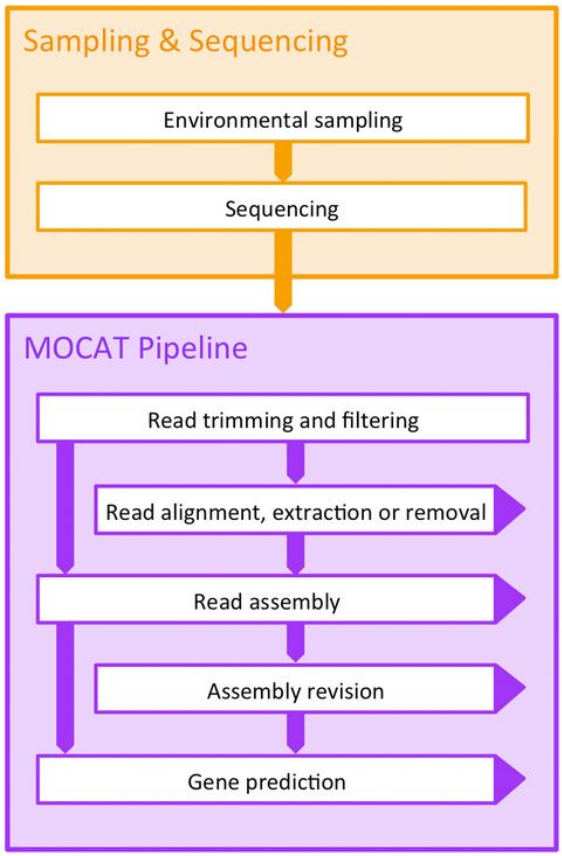

Figure 5: MOCAT's processing pipeline. The top box in orange represents the data-gathering step from the environment, which must be done prior to running the workflow. In light purple, the MOCAT pipeline shows that some steps can be both skipped or ran sequentially, e.g. only the read

trimming, assembly and gene prediction steps are mandatory to produce an estimation of the taxonomical results. On the other hand, read alignment and removal, along with the assembly revision, are optional and should be done for quality improvement if desired.

Sequenced raw reads are quality controlled by removing low quality reads, e.g. reads are aligned to the human genome using SOAP [27], and then these reads are removed since in most cases their presence is due to contamination of the sample. This a-priori step of aligning reads can be used as an estimation of the reads abundances. The remaining reads are considered to be of high quality, and are assembled into contigs which are later used to perform the gene prediction step using Prodigal [28] or MetaGeneMark [29].

MOCAT is implemented in Perl, and since many of the tools used in the pipeline are from third party developers, it is required to manually obtain these licenses, for example in the case of MetaGeneMark. Furthermore, MOCAT lacks a simple interface implementation to help users run their experiments, and is executed via command line, which it makes extremely hard for researchers whose field of knowledge and application is not computer sciences, e.g. biologists and geneticists. In addition, MOCAT is an example of metagenomic analyser package that uses only portions of the original metagenomes (for example by looking for specific highly-conserved genes, i.e. gene calling) and/or against protein databases because supposedly only coding regions have a functional role, and that therefore non-coding regions can be discarded. Furthermore, such procedure removes a high percentage of data to compare and process, thus speeding up the analysis process but in exchange of losing DNA material

2.3. Web-based tools

2.3.1. MG-RAST

The metagenomics RAST server [30] (MG-RAST) is a SEED-based environment that allows users to upload metagenomes for automated analyses. MG-RAST provides the annotation of sequence fragments, their phylogenetic classification, functional classification of samples, and comparison between multiple metagenomes. In addition, it also computes an initial metabolic reconstruction for the metagenome and allows comparison of metabolic reconstructions of metagenomes and genomes. Two approaches are used to perform the taxonomical classification of reads: (1) MG-RAST builds clusters of proteins at a given percentage of identity level using QIIME [31], and then the longest sequence of each cluster is compared using identity as filter in a database search via sBLAT, an implementation of the BLAT[11] algorithm; (2) performing gene prediction by searching for ribosomal RNAs against a non-redundant integration of the SILVA [32], Greengenes [33] and RDP [34] databases.

[11] http://genome.ucsc.edu/cgi-bin/hgBlat

MG-RAST also uses the NCBI taxonomy to perform the taxonomical classification, in similitude to MEGAN. The functional profiles are available through comparison with data sources that provide hierarchical information. Abundance profiles are the main output for displaying information on the datasets. The MG-RAST annotation pipeline usually does not provide a single annotation for each submitted fragment of DNA.

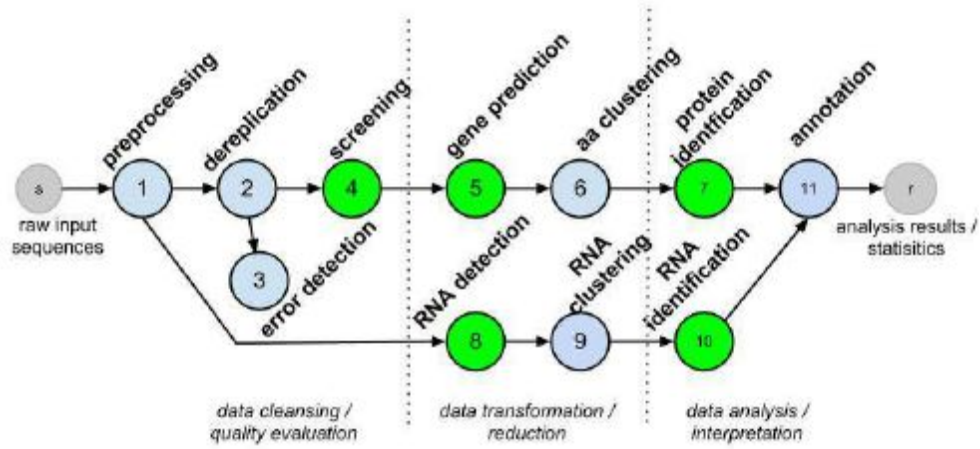

Figure 6: MG-RAST's processing pipeline shows from quality control (1, 2, 3 and 4) to taxonomical classification via gene prediction (5 and 6) or/and RNA detection (8,9). The pipeline can be executed using one or two ways: (1) top layer, using gene prediction and protein identification or (2) using RNA detection. While the first one looks for annotating reads by specific highly-conserved regions (genes) the other looks for annotation via transcriptomes detection.

However, data privacy is one of the concerns of the scientists using this tool: firstly, they are worried about uploading their unpublished and/or confidential data to a public website and secondly, the priority of job analysis submitted to such website are subjected to the confidentiality level of the input data (with lower priority and therefore longer waiting times for private data) and size of the submitted data. For example, for shotgun metagenomes the median-waiting-expected time is 7–10 days, whereas for amplicon datasets, the pipeline usually clears datasets within 24 h, since these are of smaller size in general.

2.3.2. EBI METAGENOMICS

EBI Metagenomics [35] (EMG in advance) has been developed as a free-to-use, large-scale analysis platform for metagenomic sequence data. The resource is capable of processing sequenced metagenomic and metatranscriptomic reads, 16S rRNA amplicon data and user-submitted sequence assemblies. Regardless of the data source (metagenomic, metatranscriptomic, amplicon or assembly), EMG provides a standardized analysis workflow, capable of producing taxonomic diversity and functional annotations. As a result, analyses

can be compared both within and across projects at a broad level, and across different data types (e.g. metagenomic versus metatranscriptomic).

The EMG pipeline is a combination of existing tools that are widely used in quality control, feature prediction and taxonomic and functional prediction, such as TIGRFAMs [36] or BioPython [37]. Rather than developing an entirely new repository for metagenomic data, EMG has partnered with the European Nucleotide Archive [38] to provide a permanent metagenomics data archive and data sharing/publication service. ENA's existing infrastructure and interfaces for data submission ensures that datasets are described with detail and contextual data, and are made available to the scientific community for data mining purposes and for meta-analysis.

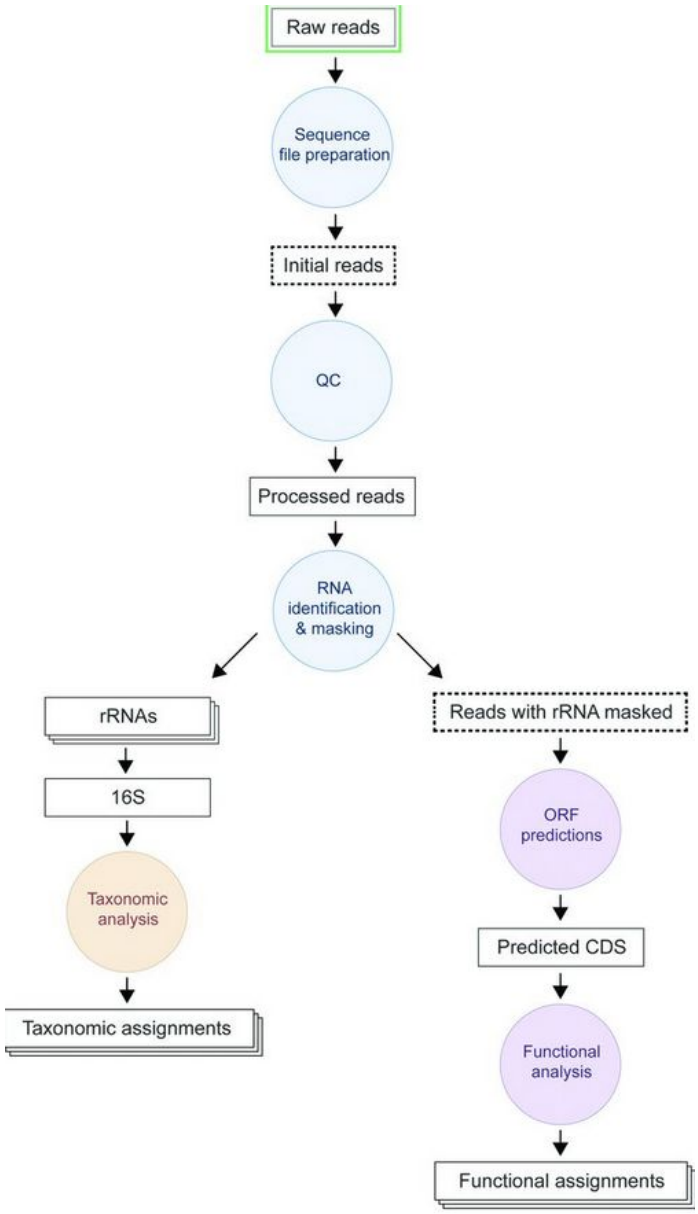

Figure 7: EMG's processing pipeline show two processing ways after the traditional quality control steps. Firstly, the quality control step is performed using Trimmomatic [39]. Two possible process ways are offered by EMG: (1) left side, taxonomic reads-abundance estimation based on 16S rRNAs genes, and (2) functional assignment based on coding regions (CDS).

Once reads (of either metagenomic or amplicon origin) are uploaded, they are queued for analysis on the pipeline. The waiting times depend mostly on size and if the uploaded data is correctly annotated, fulfilling with the standards, as one of the key points in EMG is the possibility of future users to run their own experiments with such data. The analysis pipeline consists of quality control, ribosomal RNA (using rRNAselector [40]) and protein prediction (via FragGeneScan [41]), function prediction (using InterProScan 5 [42]) and taxonomic prediction steps, which uses QIIME. The EMG pipeline is implemented under a Python framework that manages the execution of steps and jobs within queues on a cluster computer.

CHAPTER 3.
ANALYSIS AND DESIGN

In this section we will determine which requisites should be fulfilled by the proposed method. These include, among others, how should inputs be treated, the format of the output files, which times are reasonable for particular tasks, how should the workflow be accessed and executed, etc. Some aspects will be of obliged need due to the nature of the metagenomics analysis itself. We will reflect below the top level requisites and leave low level and detailed issues to be addressed later on the implementation chapter. In addition, specifications will be given to provide a first grasp on the capabilities of the proposed method.

3.1. Analysis of general aspects

1. Platform usage:
    a. Most bioinformatics software (and in general, Data Science related) is designed to be used under UNIX environments. This is due to the powerful native shell, the all-aspects customization, and that UNIX is not only meant to be used as a client operating system (such as Windows or Mac) but also as a developer platform. Therefore, the proposed method will be developed to use under UNIX operating systems, such as Ubuntu[12] or Debian[13].
2. Programming language(s):
    a. Metagenomics processing demands high memory usage (e.g. loading whole genome databases into memory, such as done by KRAKEN) as well as CPU time (e.g. aligning large sequences can be up to $O(n^2)$ for each pair of sequences depending on the algorithm used) and therefore require a low-level programming language that enables the developer to correctly manage how memory is allocated and released, and how CPU cycles are spent. Thus, the C programming language has been chosen, since it is a low-level language almost as fast and effective as writing code directly on machine code.
    b. Plotting, graphs and charts will be produced using the R programming language, since it offers a high-level abstraction, a complete suite of statistical packages to simplify coding tasks and a modular-oriented implementation, in terms of direct shell execution under UNIX systems.
3. Type of metagenomic analysis software:
    a. Across all possible metagenomic analysis categories (e.g. functional annotation, comparative metagenomics, assembly, etc) we will orient the

[12] http://www.ubuntu.com/
[13] https://www.debian.org/

proposed method towards a sequence similarity search, that is, perform a comparison of all reads against a reference database and map the reads based on the properties of the aligned fragments (e.g. similarity and length of the match). As mentioned before, this is an expensive computational process and therefore requires packages to accelerate the pre-processing, comparison, parsing and mapping procedures.

4. Results flexibility:
    a. To enable using different homology search algorithms and in order not to force using a particular one, all results produced by such algorithms will be parsed to a concrete format. See the Implementation chapter for the specific format.
5. Workflow division:
    a. The whole workflow will be divided in two:
        i. A general mapping workflow that will be on charge of mapping reads to taxa.
        ii. A specific workflow for post-processing results at a fine-grained level.

3.2. Pipeline design and specifications

As stated earlier, the software will be developed as a workflow, i.e. a pipeline of single modules that take either raw data or processed data by a previous module and produce the next level of processed data. The following list contains the different modules that will be included. Lower-level definitions and procedure details are given in the Implementation section.

3.2.1. General mapping pipeline

User-inputs of the general mapping pipeline:

1. A custom database of nucleotides in FASTA format, concatenated in a single file.
2. A metagenome in FASTA format. If working with paired-end reads, these shall be concatenated in a single file, thus considering them as independent reads,
3. A cuadrangular scoring matrix for each nucleotide by rows and columns to handle scoring alignments.

Listed below are the modules that will compose the workflow with inputs and outputs. A brief description will be given along with two separated lines, one labeled with *IN* for *Inputs* and *OUT* for *Outputs*.

1. **SeqTrimNext** [43]: A third party tool that will be used to trim reads produced by 454-pyrosequencing.
   IN: The input reads file.
   OUT: The trimmed reads file.
2. **Trimmomatic**: Another third party quality control tool that will be used to process Illumina HiSeq 2000 reads.
   IN: The input reads file.
   OUT: The trimmed reads file.
3. **FormatREF**: A module that will add an identifier to each sequence of the database using special characters to deal with different FASTA headers.
   IN: The input database.
   OUT: The formatted database.
4. **MGRindex**: A tool to produce an index for a FASTA file containing useful information such as starting positions, length and masked residues.
   IN: Reads file.
   OUT: Two binary indexes that contain a structure per read with the read particular information , one in the same order as the input reads file, and the other in sorted order to enable binary search.
5. **SearchSpace**: A tool to calculate the number of nucleotides in a file.
   IN: Formatted input database.
   OUT: A text file containing the number of residues.
6. **FastaFreqs**: A program that will calculate the frequency per nucleotide of the database to support the calculation of Karlin and Lambda parameters.
   IN: The input formatted database.
   OUT: A file containing the frequencies per residue with the sum of the frequencies being 1.
7. **TaxoMaker**: A program that produces the taxonomy file of the database along with other information, such as genomes length and identifier.
   IN: Formatted input database.
   OUT: Tabulated text file containing data per each genome in the database.
8. **GECKO**: An external program to compute homology searches between two FASTA files.
   IN: Input trimmed metagenome, Input formatted database
   OUT: A fragments binary file which contains the alignments of every fragment reported.
9. **ReverseComplement**: A module that produces the reverse complement of a FASTA file.
   IN: Formatted input database.
   OUT: The same input database but with its sequences being complemented and reversed. In addition, sequences will be in reverse order as well.

10. **GeckoToJoiner**: A tool that parses the fragments file of GECKO to the workflow's format.
    IN: Fragments file, non-sorted index for the input trimmed reads, taxonomy file for the formatted database, non-sorted index for the formatted database and non-sorted index for the reverse complement database.
    OUT: Tabular text file containing the results of the homology search parsed to the workflow format.
11. **BlastToJoiner**: A tool that will parse the default results of a BLASTN or MEGABLAST comparison and that will produce the format used in the workflow.
    IN: BLAST suite default results of the comparison of the input trimmed reads and the formatted database.
    OUT: Tabular text file containing the results of the homology search parsed to the workflow format.
12. **KLambdaParam**: A program that will compute the Karlin and Lambda parameters to later on calculate the expected value of a fragment given its properties.
    IN: Nucleotides frequencies, cuadrangular scoring matrix.
    OUT: A text file containing the two parameters, Karlin and Lambda for the formatted database.
13. **Sort-X**: A program to sort the parsed file of fragments produced by GECKO, since its results are given sorted by the formatted database instead of the metagenome.
    IN: Parsed file of fragments produced by GECKO.
    OUT: Sorted parsed file of fragments.
14. **FJoiner**: A tool that will align two sequences given their coordinates by firstly extracting these from the respective FASTA files and then performing an NW alignment between them.
    IN: Parsed comparison file, sorted reads index, the formatted database, the taxonomy file of the database, the cuadrangular scoring matrix and the input trimmed metagenome.
    OUT: The joined/extended file of fragments in the same format as the parsed format used in the workflow.
15. **GMAP**: This tool will be the mapping module of the workflow. It will produce a binary file with a summary of the assignments per read to genome, using a maximum of three mapping options per read. It will also produce the taxonomic reads abundance for the experiment.
    IN: Comparison parsed/joined file, sorted reads index, taxonomy file of the formatted database, Karlin and Lambda file, searchspace file.
    OUT: A binary mapping file containing the summary of the reads assignment using a maximum of three options per read, two plain csv text files containing a table with the taxonomic abundances and a series of statistical indicators, one for the species level and the other one for the strain level.

16. **Full Matrix Distribution**: A tool that will produce a matrix containing the accumulation of fragments by percentage of identity and coverage.
    IN: A parsed/joined comparison file.
    OUT: A MAT matrix of length 100x100.

3.2.2. Specific fine-grained pipeline

User-inputs of the specific fine-grained pipeline:

1. A custom database of nucleotides in FASTA format, concatenated in a single file.
2. A metagenome in FASTA format. If working with paired-end reads, these shall be concatenated in a single file, thus considering them as independent reads,
3. A cuadrangular scoring matrix for each nucleotide by rows and columns to handle scoring alignments.
4. A binary mapping file produced via the execution of the general mapping workflow for the same database and metagenome as (1) and (2).
5. A particular genome in FASTA format for which to aim the specific processing results, which has to be originally contained in the reference database.
6. The taxonomy file of the formatted database produced by the general mapping workflow.
7. The GenBank annotation file for the particular genome for which the experiment is aimed to, and therefore must be the same as in (5).
8. The comparison parsed file produced by either BLAST or GECKO as a result of the execution of the general mapping workflow.

Listed below are the modules that will compose the workflow with inputs and outputs. A brief description will be given along with two separated lines, one labeled with *IN* for *Inputs* and *OUT* for *Outputs*.

1. **Quality Mapper**: A program that will calculate differences in reads abundance, length, percentage of identity and coverage among the three maximum different options.
    IN: The binary mapping file for the experiment, a taxonomy file for the formatted database.
    OUT: A tabular text-based file containing the values addressed per genome in the formatted database.
2. **Genome Cutter**: A simple program that will cut a particular genome out of a reference database to enable comparisons of such genome against the database.
    IN: The formatted database, the taxonomy file of the formatted database.
    OUT: A copy of the database without the targeted genome.

3. **Specific Region Finder**: A tool that will parse the binary fragments file of the GECKO comparison without the target genome in the reference database and produce the regions that are not shared by any other genomes in the reference database.
   IN: Binary fragments file for the comparison of GECKO without the target genome, the non-sorted index for the database without the target genome.
   OUT: A text file containing one specific region per line.
4. **Specific Reads**: Similarly to Specific Regions, a program that will extract the reads that are mapped to the specific regions (if any). It will produce a text file containing one read per line with the mapping coordinates.
   IN: The specific regions for the comparison without the target genome, the mapping binary file for the experiment, the non-sorted index for the reads input file, the target genome ID according to the taxonomy file of the input formatted database.
   OUT: A text file containing the reads identifier and their mapping coordinates to the specific regions.
5. **Gene Parser**: A parser for annotation files retrieved from the NCBI GenBank database, which will produce a parsed text file similar to the one used in the workflow to provide simple functional annotation results.
   IN: A GenBank's annotation file.
   OUT: A parsed GenBank annotation file.
6. **Coding Abundances**: A program that will produce functional abundances of reads that have been mapped to annotated regions, showing these at both genome and read level.
   IN: The binary mapping file of the experiment, a parsed GenBank annotation file, the ID of the target genome for which to produce the data according to the Taxonomy file of the formatted database.
   OUT: The annotational abundance for the target genome as a text tabular file, a file containing the coordinates of the mapped reads to the annotated regions.
7. **Genome Profiler**: A program that will extract the number of mapped nucleotides to each position along a target genome.
   IN: The binary mapping file for the experiment, the taxonomy file for the formatted database, the target genome ID according to the taxonomy file of the input formatted database.
   OUT: A csv file containing the number of accumulated nucleotides per position in the genome, with as many lines as length of the genome.
8. **Specific Genome Matrix Distribution**: A tool that will produce a matrix containing the type of mapped reads for a particular genome.
   IN: The binary mapping file for the experiment, the parsed GenBank annotation file, the target genome ID according to the taxonomy file of the input formatted database.
   OUT: The specific matrix distribution for the target genome.

3.3. Specifications of the workflow manager

To make clearer how can users interact and execute their own experiments, it is of interest to provide with a simple interface that avoids the trouble of sequentially calling separated modules with their respectives inputs and outputs. We decided to use the Galaxy workflow manager, which enables for customization, workflow-building from independent modules, runtime parametrization, datasets history management, job completion notification by email, etc. Thus implementing the proposed method under Galaxy would benefit both users and further developers. However, the pipeline will still be available as a number of C modules that can be run via the command line. See below the specifications for the Galaxy workflow manager.

1. The access to the Galaxy instance will be restricted.
    a. Users will have to register and ask the admin of the instance for permission.
    b. Once registered, users will have to log in with their username/email and password.
2. There will be a main entry for the workflow in the Galaxy instance under the name "METAGECKO" that will comprise all modules of the workflow to use them independently.
3. All of the used formats in the workflow will be registered in the Galaxy instance as to limit the possible selections in drop-down menus. Two types will be considered:
    a. Binary types:
        i. Frag Files.
        ii. Mapping binary files.
        iii. Sorted index files.
        iv. Non-sorted index files.
    b. Text-based types:
        i. Fasta.
        ii. MAT (matrix tabular file).
        iii. Parsed format.
        iv. Joined format.
        v. Genbank format for annotation files.
        vi. Genbank parsed format for parsed GenBank files.
        vii. Qly files for the output of the Quality Mapper.
        viii. Freqs for nucleotide frequency files.
        ix. KLamba format (tabular text file) for Karlin and Lambda parameters storage.
        x. Profile files for genome profiles.
        xi. Searchspace files for nucleotide residues parameter storage.
        xii. Taxonomy file for the taxonomy of the database.

4. Each module will be added via a configuration .xml file. The following information will be mandatory for each module:
    a. Tool ID.
    b. Tool name.
    c. Tool description.
    d. The input tag will include, for each input:
        i. Input name.
        ii. Input label.
        iii. Input help.
        iv. Restricted format to minimize the selection drop-down menu.
    e. The output tag will include, for each output:
        i. Output name.
        ii. Output format.
        iii. Output label.
5. Three non editable workflows will be available by default for all users:
    a. “METAGECKO” for the general mapping workflow.
        i. Inputs:
            1. A reference database in FASTA format.
            2. A metagenome in FASTA format.
            3. A scoring matrix to handle nucleotide scoring values.
        ii. Outputs:
            1. A plot of the raw distribution of fragments reported by the comparison software.
            2. A plot of the joined distribution of fragments reported by the comparison software.
            3. A plot of the distribution of fragments by percentage of coverage and identity.
    b. “METAGECKO PAIRS”.
        i. Two general mapping workflows combined to contrast samples.
    c. “METAGECKO SPECIFIC”.
        i. For the fine-grained specific workflow.
6. Each user will have small fixed starting size share in the Galaxy instance.
    a. Default: 2 GB.
7. Users will have privileges to import tools from published Galaxy repositories.
8. Users will have privileges to upload data and manage histories.
9. Users will have privileges to create their own workflows using combinations of existing tools.

CHAPTER 4.
IMPLEMENTATION AND RESULTS

4.1. Introduction

The developed software is divided in two sub-workflows. Firstly, the general-mapping subsystem, which goes from initially parsing the results of a sequence comparison software to the assignment of reads to taxa. The second subsystem comprises of a more specific set of tools, designed to work at a particular-one-genome level using annotation files to find regions of interest. Moreover, the second subsystem is an extension of the first one, since it uses the mapping files (among others) to produce results.

Nonetheless, it is noteworthy to mention that the pre-processing step is included in the workflow, however it is comprised of third party tools that have been developed for that specific purpose. Developing a pre-processing module for the different type of sequencing platforms is a whole different field in metagenomics, for which a broad scope of tools address each scenario. In particular, the tools included for the pre-processing step are SeqTrimNext for the reads sequenced by 454-pyrosequencing, and Trimmomatic for the reads sequenced using the more recent technology Illumina HiSeq 2000.

Both subsystems include data processing steps and visualization tools. All data processing modules have been developed using the C programming language. Metagenomics is a field that demands high performance computing, and therefore the C language was chosen as a very appropriate one, for its low level programming enables to manage memory resources and CPU to a more precise scale in comparison with other higher level languages such as Java, which was initially considered to be part of this project, however, the fact that it runs on a virtual machine and that memory is self managed were the major drawbacks. Since the project was developed to work under UNIX operating systems, some shell scriptings were used at the datafiles exchange between modules, i.e. sorting lists or replacing expressions.

The visualization modules were programmed using the R language, which mainly focuses on statistical computing and graphics. Along with its simple use under UNIX systems, it turned out to be very appropriate for the project.

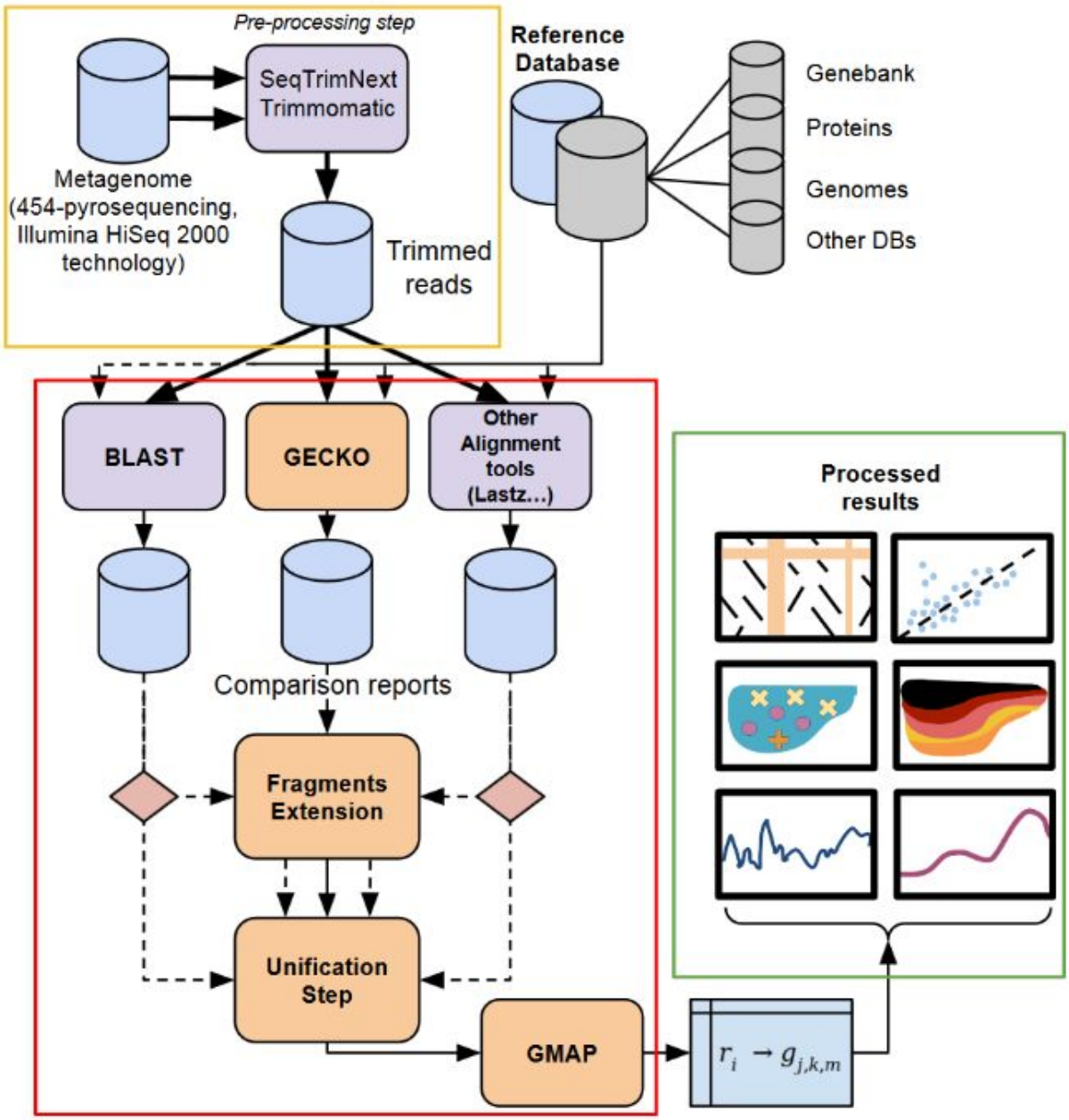

Figure 8: Workflow diagram divided in two subsystems and a prior quality control layer. The red enclosed area is the general mapping workflow, whereas the green enclosed area is the specific-genome-level workflow. The top-side, beige-enclosed area is the pre-processing step which involves quality control of metagenomes.

Although developed as a pipeline of C modules to be called sequentially using UNIX' shell, it seemed that the average user would find trouble in recurrently executing scripts one after another, and that such doing could potentially introduce errors in the pipeline processing. Furthermore, even at testing stages, the developer myself, would commit errors due to the large number of modules to be called in order to analyse single metagenomes. Such reasons promoted the use of a workflow manager, and for that goal, it was decided to be the Galaxy workflow manager, for it not only allowed the management of jobs and queues within the

workflow, but also enabled to run the software both in local machines and online instances. See the section 4.4.0. for details on the Galaxy implementation, which is already installed with the workflow itself in the provided virtual machine.

In order to fit the pipeline representation in A4-sheets and therefore allow printing the document, the general workflow has been cut in two pieces from left (start of the workflow) to right (end of the workflow). See Figure 9 and Figure 10.

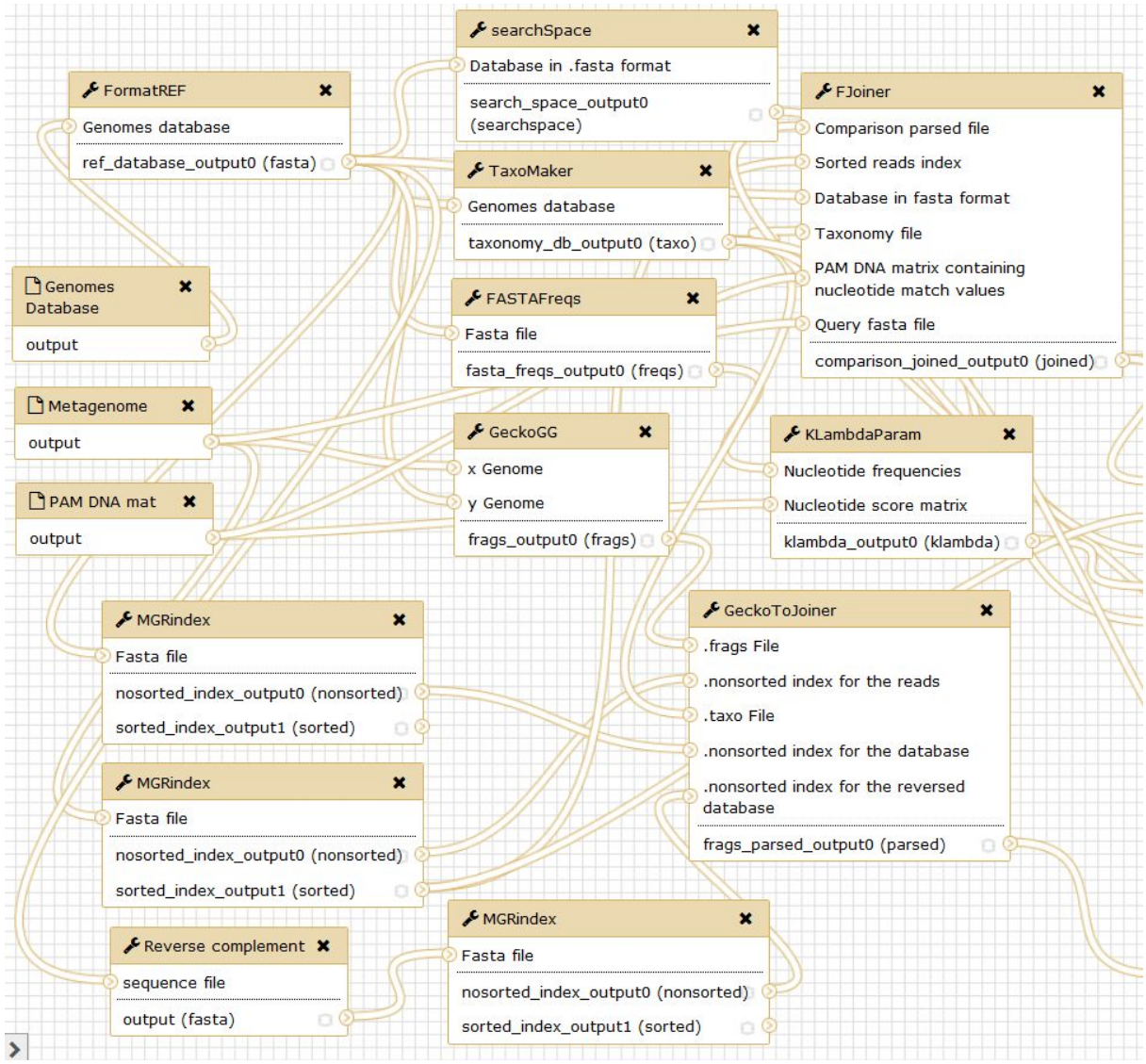

Figure 9: Left side (inputs and processing start) of the general workflow. The workflow canvas shown is the illustration of the connection between modules inputs and outputs. It becomes clearer that manually calling each module could produce potential human-introduced bugs.

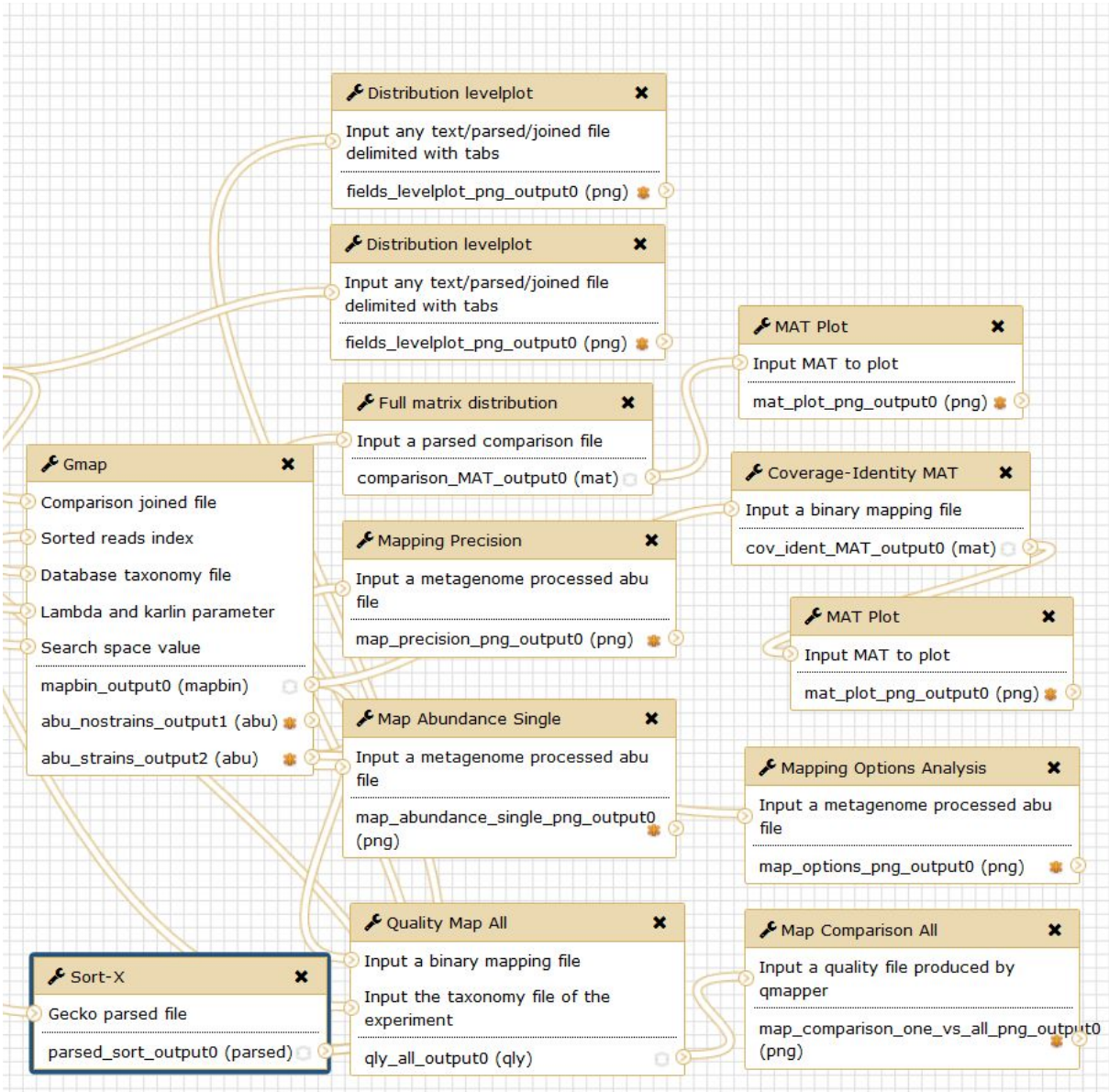

Figure 10: Right side (processing end and visualization outputs) of the general workflow. Star-marked-tool outputs are produced results that will be conserved upon finishing the execution.

On the other hand, the specific workflow (See Figure 11) does fit in a single A4 sheet. Whereas the general mapping workflow provides the tools needed to perform a basic taxonomic classification of metagenomic samples from its very beginning (i.e. pre-processing and homology search), the specific workflow is aimed at a posterior-fine-grained step. It is run with the results from the general workflow and two additional files, namely the genome for which to perform the specific execution in FASTA format and the annotation file from the GenBank database.

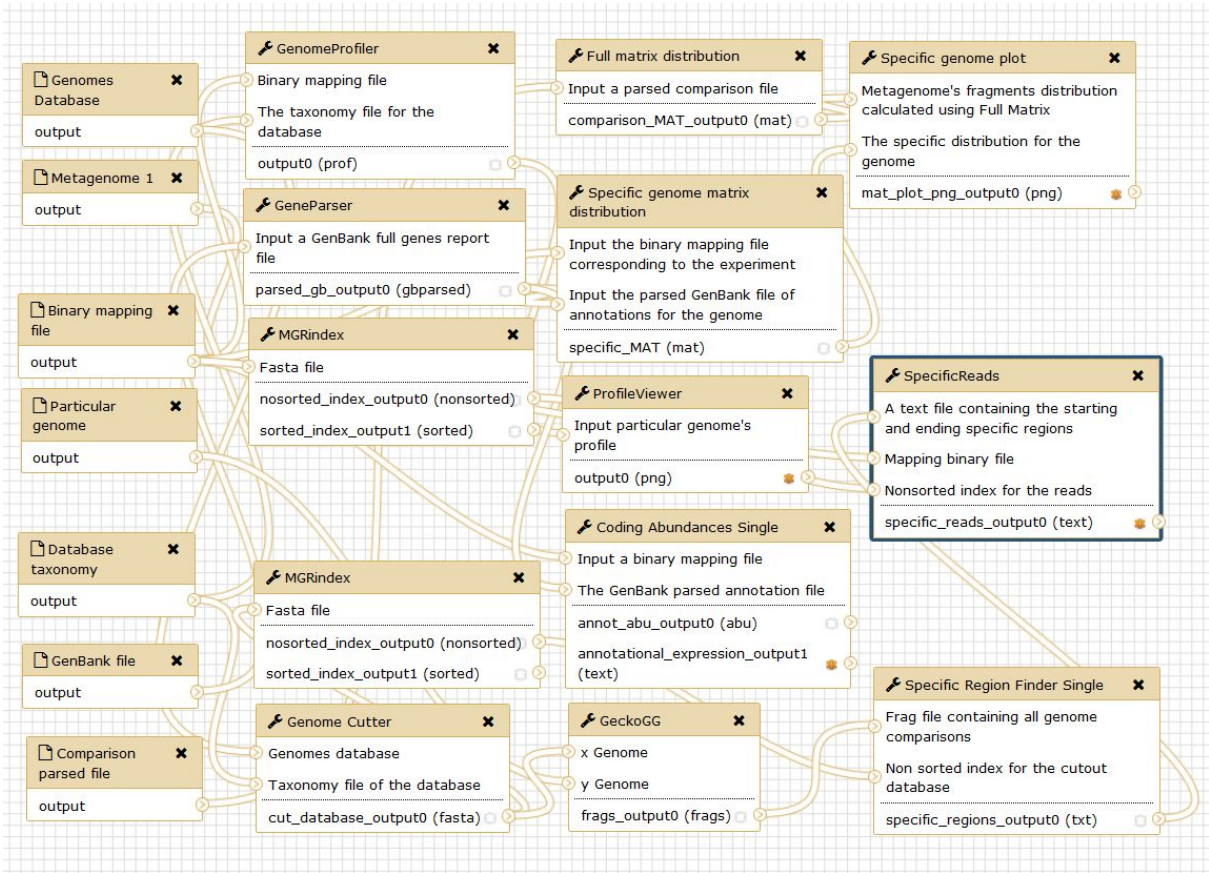

Figure 11: Full overview of the specific workflow, which produces results at a genome level using the output from the general mapping process. The left-layer is the data input modules, the center-layer comprise the processing tools, whereas the last layer are mainly output results and visualization plots.

4.2. Implementation and results

In this section we will briefly discuss the modules that compose the proposed method, pointing out which issues were found and how they were addressed. In addition, the running time and complexity in terms of the input will be provided. We will differentiate between the general processing tools that in some aspect are responsible for the mapping, and the post-processing fine-grained level tools that use the mapping files to produce further results.

4.2.1 Tools for the mapping step

4.2.1.1. FormatREF

This small program was included to internally assign an ID to each genome in the database throughout the analysis process. The reason behind this addition is the disagreement

on the identifiers used in FASTA files to uniquely identify genomes. Although databases such as GenBank provide with GI id's and accession numbers, it is not possible to know if the user which has made a custom selection of species has (by chance or will) modified these numbers or if he is using an old set of genomes with outdated references. Therefore it seemed appropriate to include a simple number to uniquely identify a genome.

The program itself produces a copy of the database adding a number to each genome, which depends on the order of the genomes in the file. The FASTA header starts with a ">" symbol and is followed by an identifier and optional descriptions, up to a break line. The number is added between the opening ">" symbol and the rest of the line, e.g:

Prior to FormatREF	After FormatREF
`>NC_004307.2 Bifidobacterium longum NCC2705 chromosome`	`> |k| NC_004307.2 Bifidobacterium longum NCC2705 chromosome`

Table 1: Changes applied by FormatREF are the inclusion of a simple identifier that will avoid dealing with different types of FASTA headers.

The complexity of FormatREF is linear in the size of the input (the database) and therefore is $O(n)$, for it sweeps the database through copying characters and writing them to another file.

4.2.1.2. MGRindex

Metagenomes tend to be large FASTA or FASTQ files, of normally hundreds and hundreds of megabytes (if not more). Along the execution of the workflow, it will be needed to load reads and/or properties about them, e.g. the length of a particular read to compute the percentage of identity, coverage, etc. Furthermore, linearly scanning a metagenome file every time a particular read or field is needed would be slow, redundant and absolutely intractable.

To fulfill this requirement, two indexes are created: one that is not sorted and serves to hold properties; and a sorted one to allow logarithmic time access using binary search. The stored properties by these indexes are shown in the table below:

Field	Description
`id`	`The identifier that comes after the ">" FASTA header starting symbol.`
`rNumber`	`Read number counting from 0.`

rLen	Length of the read in nucleotides.
rLmasked	Length of masked nucleotides.
nonACGT	Number of letters that are not nucleotides e.g. the "N" is used to represent an unknown nucleotide.
pos	Position of the read in the metagenome file.
Lac	Accumulated length of all reads that appear prior in the file.

Table 2: Data that will be stored in both indexes.This information will be particularly useful when translating the coordinates of fragments of the sequence comparison software GECKO, which yields such coordinates as an accumulated sum of the length of all previous sequences.

The fields are stored as a binary structure. In the case of the sorted index, a QuickSort [41] algorithm is used to sort in $O(n\ logn)$ time. Therefore, when a read identifier is found, a binary search is performed to obtain the binary structure that belongs to it. The MGRindex program takes one by one all reads and firstly stores them (unsorted binary file), which takes $O(n)$ being n the number of reads. Afterwards, such file is sorted with QuickSort and stored again in a different file. Therefore, the complexity is the composition of the two complexities independently, and make it up to $O(n + n\ logn)$ which is still linear.

4.2.1.3. BlastToJoiner, GeckoToJoiner and ReverseComplement

For both sequence comparison software (GECKO and BLAST), the results will need to be parsed to a format that is usable by the workflow. This is done using GeckoToJoiner and BlastToJoiner, respectively of the software used. The two are quick linear parsers that store all relevant information about the reported fragments, such as the matching length, the number of identities or the inserted gaps. Alignments, however, are not stored since they are no longer necessary to perform the mapping; only their properties are needed to decide which are the best candidates for every read.

The format consists in a tab based text file composed of a header with one or more 12-tuples.

$$t_{n,k}^{12} = (k,\ score, identities,\ length,\ similarity,\ igaps,\ egaps,\ strand,\ rStart,\ rEnd,\ gStart,\ gEnd)$$

Where each field corresponds to:

Field	Description
Fragment number	For every read, the position in the list of reported fragments

`Score`	Reported score for the fragment
`Identities`	Number of identities
`Matching length`	Length of the match
`% Similarity`	The number of identities divided by the matching length
`iGaps`	Number of opening gaps
`eGaps`	Number of extension gaps
`Strand`	Strand of the fragment represented with ++ or +-
`Start in read`	Starting position of the match in the read coordinates
`End in read`	Ending position of the match in the read coordinates
`Start in genome`	Starting position of the match in the genome coordinates
`End in genome`	Ending position of the match in the genome coordinates

Table 3: Fields of the parsing format that is used in the workflow. In order to use any sequence comparison software, it will be needed to provide a parser to the format shown above.

Multiple fragments in different genomes can be reported for the same read. Therefore, one to many fragments will be associated to a header, one for each unique combination of *read* and *genome,* along with *length*:

`>Read ID`	`>Genome access number and name`	`Genome length`

Table 4: Header information that represents a tuple read-genome. Each tuple is a unique combination of a read and a genome and can only appear once.

An R script is used to plot the accumulated distribution of fragments by percentage of identity and coverage in log10 scale. See Figure 12.

In addition, in the case of using GECKO as the sequence comparison software, and therefore GeckoToJoiner as parser, it becomes necessary to produce the sorted and unsorted indexes for the reverse complement of the database, since the coordinates of fragments located in the reverse complementary sequence given by GECKO need to be transformed to a forward database system. To create such indexes, first we will need to create the reverse complement of the database and in reverse order. The program ReverseComplement does this by firstly sweeping the database file and storing the starting positions of each sequence (say N) in a vector. Then the file pointer is positioned on each position starting in the N^{th} and down to the 1^{st}, loading the sequence into memory and writing it in reverse order while complementing it at the same time. Thus if the database is of size n, a first sweep is needed,

plus another sweep for every sequence to load into memory and a last one to write and complement, hence making the complexity to be $O(n+n+n)$ which is $O(3n)$.

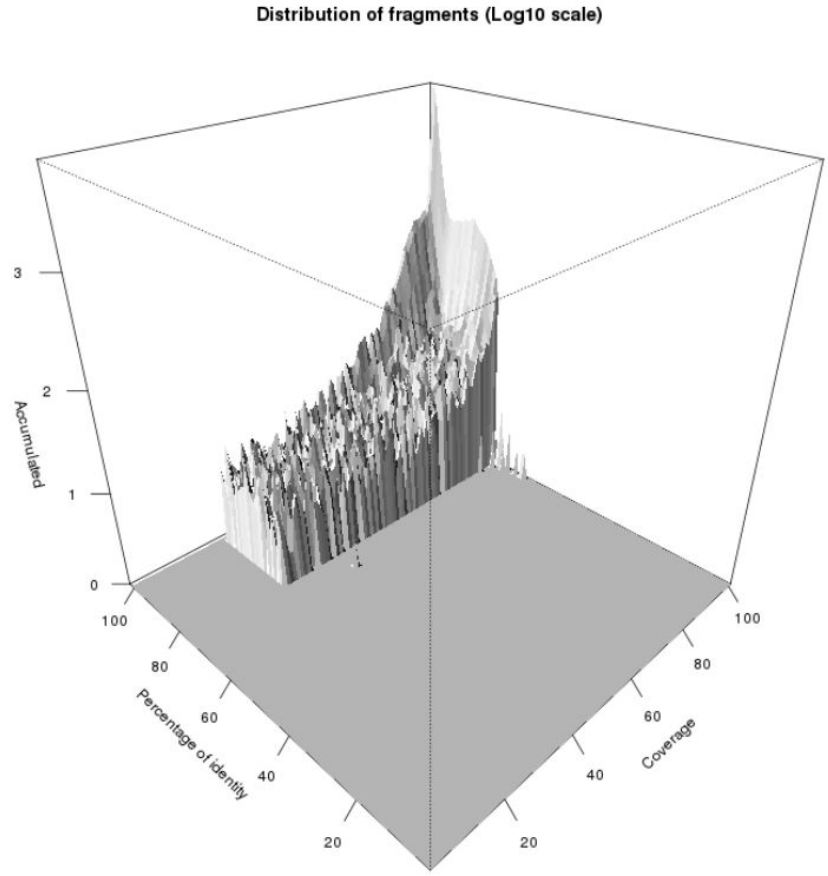

Figure 12: This 3D plot shows the accumulated number of fragments (Z-axis) in logarithmic scale by percentage of identity (Y-axis) and percentage of coverage (X-axis). Spiked (x,y) points represent regions that have a higher number of reported fragments.

Both GeckoToJoiner and BlastToJoiner run in linear time depending on the size of the input. These programs are prepared to run extremely fast since the input files can be of size up to several hundred gigabytes. The complexity is $O(n\ logn)$ for both, being n the number of fragments present in the comparison file. The use of the indexes discussed in section 4.2.1 enable the binary search to retrieve read properties in logarithmic time.

4.2.1.4. TaxoMaker

The workflow uses a taxonomy file that maps id's to genomes and stores the names and length of the genomes in the database. These are used for visual reasons, i.e. show labeled axis for genomes in plots or files, which is almost a must-have when dealing with large databases. The length is used to calculate the expected value of each fragments in a later step in the workflow. These taxonomy files are also used to incorporate custom relationships

between the genomes, as well as creating phylums of similar genomes, e.g. different strains of Mycoplasma Hyopneumoniae could be grouped together. See below an example of taxonomy file:

Specie Level 1	Specie Level 2	Access code	Specie Name
1	0	ref|NC_021831.1|	Mycoplasma hyopneumoniae J
1	1	ref|NC_017509.1|	Mycoplasma hyopneumoniae 168
2	0	ref|NC_015144.1|	Weeksella virosa DSM 16922
3	0	ref|NZ_DS264342.1|	Ruminococcus obeum ATCC 29174 Scfld0253
3	1	ref|NZ_DS264341.1|	Ruminococcus obeum ATCC 29174 Scfld0254
3	2	ref|NZ_DS264340.1|	Ruminococcus obeum ATCC 29174 Scfld0255

Table 5: Example of taxonomical description file where two species are bounded together as substrains using the same number for Specie Level 1, a second specie has no substrain attached and a third specie composed of three contigs.

These files can be build with any text editor or spreadsheet software. However, it can become very tedious to build these files when working with large databases where several genomes are present, and therefore the tool TaxoMaker produces these type of fields automatically. It runs in linear time since the procedure followed is to parse the database file and add new entries to the taxonomy file for each genome found.

4.2.1.5. FastaFreqs, SearchSpace and KLambda

These tools are treated as once, since the three are executed together and are completely dependant. The goal of these tools is to extract a series of parameters needed to later compute the expected probability of reporting a particular fragment given the length and properties of both the query metagenome and the database. FastaFreqs produces a file containing the frequencies per residue, and should be used on the database, along with searchSpace, which computes the total number of residues in the database. In particular, the searchSpace program is no more than a combination of well-known unix scripts:

```
grep -v ">" database.fasta | wc
```

Which produces the total number of residues when computing the difference between the third column (number of characters) and the first column (number of newlines), thus removing \n characters.

The KLambda tool produces the K and Lambda parameter as studied by Karlin and Atschul [45] given a scoring matrix of nucleotides and the frequencies per residue. These parameters are then used to compute the expected probability of a fragment in the mapping module to choose the best fragment.

4.1.2.6. Fjoiner

Metagenomes are uncultured samples, and the bacteria present in such environmental communities suffer from high evolutionary pressure due to all interactions between species, causing mostly small mutations, insertions, deletions, translocations, etc. Such is the main reason behind the development of the tool Fjoiner, which tries to extend fragments by performing a Needleman-Wunsch alignment between those that belong to the same read and genome. Aligned fragments that surpass identity and coverage thresholds are kept and stored. Therefore Fjoiner produces longer fragments, facilitating the decision step of the best candidate for every read. Genomes present in the database are loaded into RAM to accelerate the process, since sequence retrieval from files becomes slower as larger as these are. Then the comparison file is read sequentially loading all fragments belonging to a tuple *Read-Genome*. Every pair of candidate fragments are tried to be aligned using a custom two-rows Needleman-Wunsch algorithm. Table x shows the pseudo-algorithm:

1. Load genomes into memory
2. Quicksort genomes by accession number
3. Load reads index
4. For every read in the comparison file
    a. Sort all k fragments belonging to the read by read coordinates
    b. For all tuples of consecutive fragments $(f_n f_{n+1})$ while $n-1 > k$
        i. If $(f_n f_{n+1})$ distance in both genome and read coordinates satisfy the threshold
            1. Load the complete read using binary search
            2. Perform Needleman-Wunsch scoring matrix between $(f_n.r_1 f_{n+1}.r_2)$ and $(f_n.g_1 f_{n+1}.g_2)$
            3. If the obtained alignment satisfies minimum coverage and similarity thresholds:

```
                    a. Join fragments
            ii.   Else n = n + 1
        c. Store in output file
```

Figure 13: Pseudo-code for the algorithm that extends fragments. Fragments that belong to a tuple read-genome are loaded, their genome-regions are accessed, and a NW algorithm is performed between adjacent fragments.

As can be seen in the algorithm, fragments are sorted by their read coordinates as key using the Quicksort algorithm. Instead of evaluating all possible combination of pairs of fragments, which leads to $(n)(n-1)$ comparisons, only consecutive fragments have to be evaluated, and therefore the number of evaluated fragments is $\frac{1}{2}(n-1)\,n$ in the worst case. As seen in Figure 14, the accumulation of fragments in short length regions is drastically dissipated up to longer regions. The number of identities is also affected, and is generally reduced since gaps are inserted.

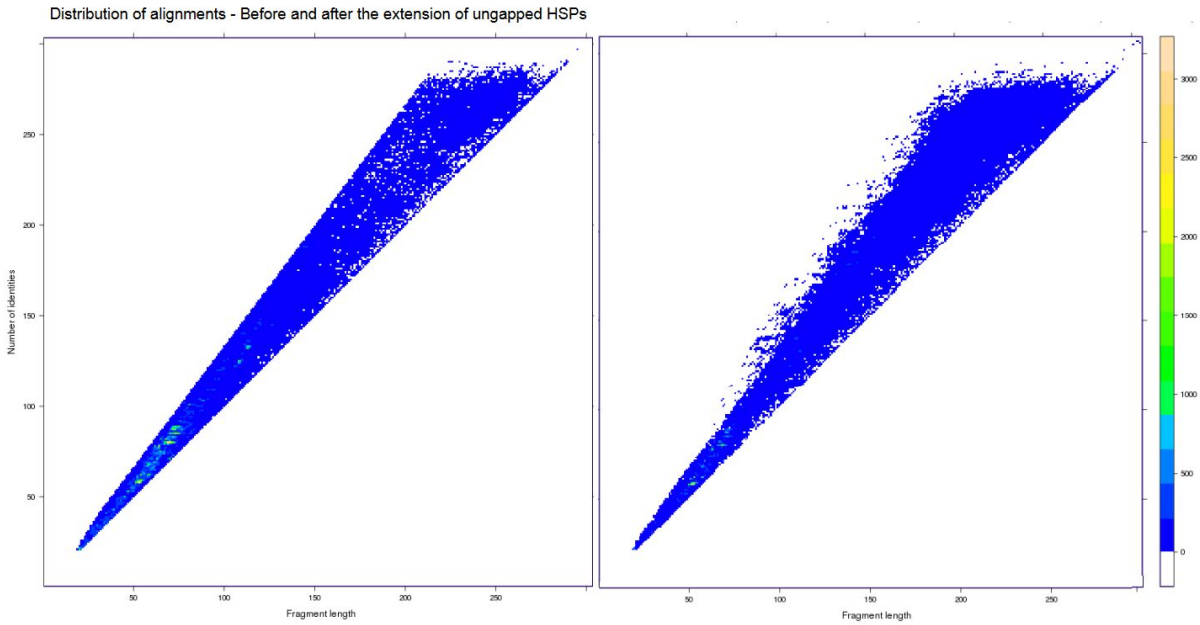

Figure 14: Distribution of alignments by length and number of identities before and after the extension using only 16S rRNA gene sequences. The x-axis shows fragment length in nucleotides, whereas the y-axis shows the number of identities. The color of each point is the accumulation of fragments for a particular point.

The custom Needleman-Wunsch algorithm takes $O(n^2)$ being n the length of the input sequences. This is repeated for k fragments, and therefore the total complexity is $O(kn^2)$.

4.2.1.7. Gmap

So far, the workflow has parsed comparison files to reported fragments, extracted statistical parameters, assigned id's and created indexes to enable fast retrieval, and improved the length and quality of the fragments. However, these individual steps aim to conclude in one thing: to facilitate assigning reads to taxa, that is, deciding from which genome does each read come from. The Gmap tool takes parsed or extended comparison files and chooses the best three genome candidates for each read. This is called a *three-level* mapping, and it allows not only to observe reads abundances in a regular way, but also to verify the certainty of the mapping at a glance and evaluate how large are the existing differences that make one genome a better choice than others.

The process takes a list of parsed and/or extended fragments and performs a series of steps, namely a (1) filtering step, a (2) selection of the best 3 fragments with lowest expected value and (3) deciding and storing the read. The algorithm below illustrates the mapping process:

1. Filtering step: A filtering step allows the researcher to consider only a subset of reported fragments, enabling a levelling-up mapping method. If a fragment does not reach pre-filtering thresholds, it will be discarded. Such filtering allows a two phases pre-filtering:
   a. Coverage threshold phase: The length of the match divided by the length of the read.
   b. Identity threshold phase: The number of identities in the match divided by the length of the match.
2. Repeat this step for *3*-option mapping and, if fragments are still active:
   a. Select the fragment with the smallest expected value and only if it is lower than the maximum allowed expected value. This fragment is included in the mapping file as first, second or third candidate depending on the number of options chosen and the genome is inactivated for the next option iteration.
   b. If no more fragments are still active or none of them exceeds the thresholds, the read is decided with either no mapping option of 3 mapping options.

Figure 15: Pseudo-code for the mapping module. The algorithm is repeated for each tuple of read-genome found in the input parsed file.

The expected value is used to differentiate between fragments. This value is the probability of the fragment to be random itself given the properties of the match and the length of the genome it is aligned with. This is a similar approach as BLAST does[14]. Firstly, a raw score is calculated as:

$$RS = I * Mr + (L - (Gi + Ge) - I) * Mp + Gi * Pi + Ge * Pe$$

Where RS stands for "Raw Score", I for the total number of identities in the fragment, Mr for the match score, L for the total length of the fragment in base pairs, Gi for the total number of open gaps in the fragment, Ge for the total number of extension gaps in the fragment, Mp for the mismatch penalty, Pi for the penalty of an open gap and Pe for the penalty of an extension gap. Then, a bit score is calculated from the rawscore using Karlin and Lambda parameters, which is the normalized score:

$$S' = \frac{\lambda S - ln\, K}{ln\, 2}$$

Where S' is the bit score, S the rawscore, K is the Karlin parameter and λ is the Lambda parameter. Then the expected value is calculated:

$$E = Kmn\, e^{-\lambda S}$$

Where m is the length of the reported fragment and n is the length of the target genome divided by the total number of residues in the database.

After deciding the reads, these are stored as a binary file. Instead of a plain-text association file that maps reads to genome id's, the stored structure is composed of many fields that can be used to later on to perform further experiments.

```
1.  // Read-Genome mapping structure
2.  struct mapRG {
3.     int   rNumber;      // Read number in the original metagenome file
4.     int   gNumber1;     // Genome number level 1 - Specie level
5.     int   gNumber2;     // Genome number level 2 - Subspecie level
6.     int   gPosList;     // Position in the taxonomy file list
7.     double score;       // Obtained score (if any)
8.     int   igaps;        // Number of opening gaps
9.     int   egaps;        // Number of extension gaps
10.    int   lengthGapped; // Length in nucleotides if considering different
    alignments
11.    float expected;     // Expected value (if any)
```

[14] http://www.ncbi.nlm.nih.gov/blast/tutorial/

```
    int   nIdent;       // Number of identities
    int   matchLen;     // Matching length of the alignment
    int   pIdent;       // Percentage of identity
    char  strand[MAXLID];   // Strand [++]||[+-]
    int   rStart, rEnd; // Read matching coordinates
    int   gStart, gEnd; // Genome matching coordinates
    int   option;       // Mapping options
    int   coverage;     // Coverage of the match for the read
};
```

Figure 16. Struct that summarizes a mapping between a read and three different genomes (one struct per option). The output binary file will contain several of these structures, and can be accessed after performing the mapping to run particular experiments.

GMAP produces also two tabular text files with an overview of the mapping results, showing the number of reads mapped to each option, the number of close matches defined as how many reads had a second best option almost as good as the first one, and others. An R script is used to plot the reads-abundances and the differences between options. Figure 17 shows the reads-abundance per genome in the database for a comparison of two samples. Abundance data are primarily used to determine the species that are present in a metagenomic sample and can be exploited in comparative studies on the over- or under-abundance of species in different samples.

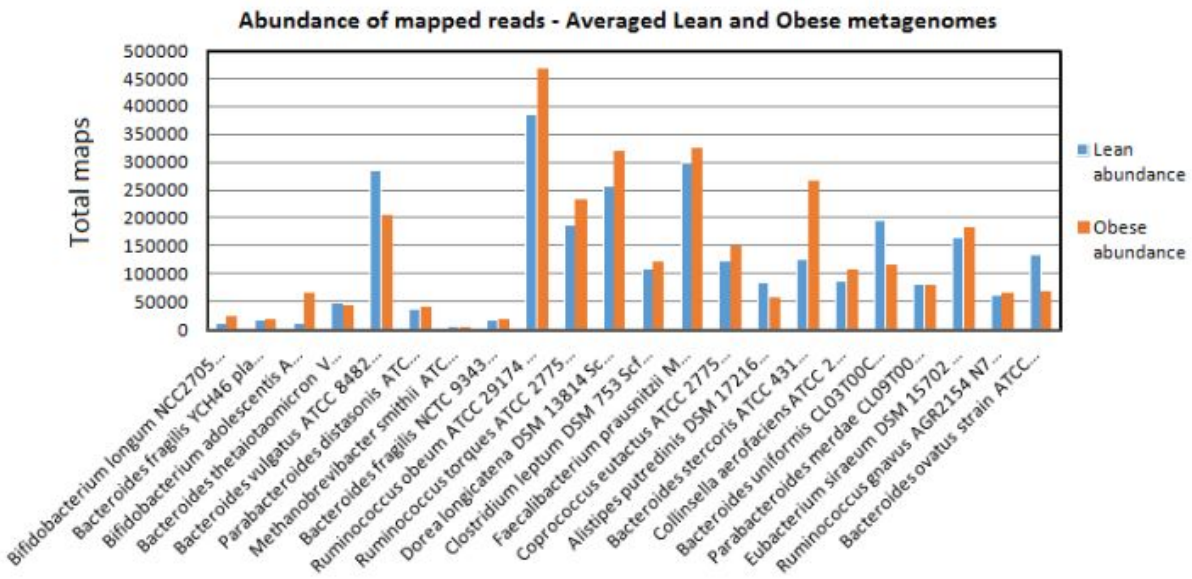

Figure 17: Mapping abundance plot. The X-axis are the genomes by species. The Y-axis are the taxonomic abundance of assigned reads. This plot allows to easily identify the presence and quantity of species in a metagenomic sample.

Figure 18 shows the differences between the first, second and third candidate, which helps determining the mapping certainty. A long separation between the mapping options provides stronger evidence supporting the validation of the mapping procedure. When comparing

3-mapping options, the detection of peaks in second or third options means that a particular species is repeatedly the second or third candidate. These peaks suggest that strong similarities exist between a specific pair of species and careful examination is required since the accuracy of mapping is not certain.

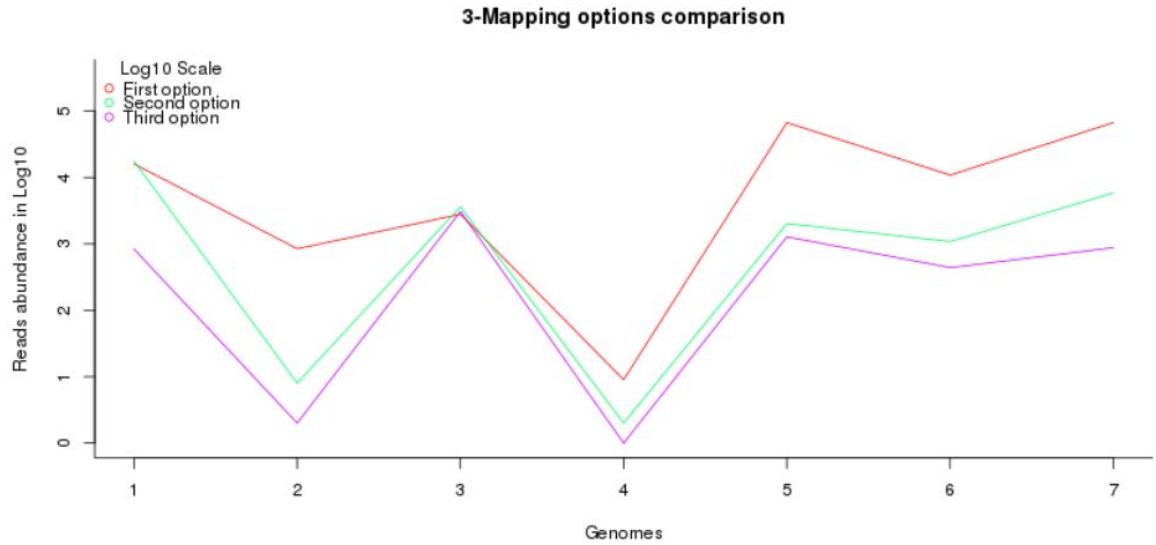

Figure 18: Three-options mapping abundance plot. The X-axis are the genomes by species and the Y-axis are the taxonomic abundance of assigned reads. Each colored line represents either the best candidate (red), the second best candidate (green) and the third best candidate (purple). An ideal scenario would have large separations between the red (first option) and the green (second option).

Figure 19 displays the number of reads assigned to each species and, in relation to each assignation, the times the second option was *almost* as good as the first (namely, "shared" reads). The fact that the red and green lines of two species are close to each other suggests that mapping is not accurate and careful examination is required.

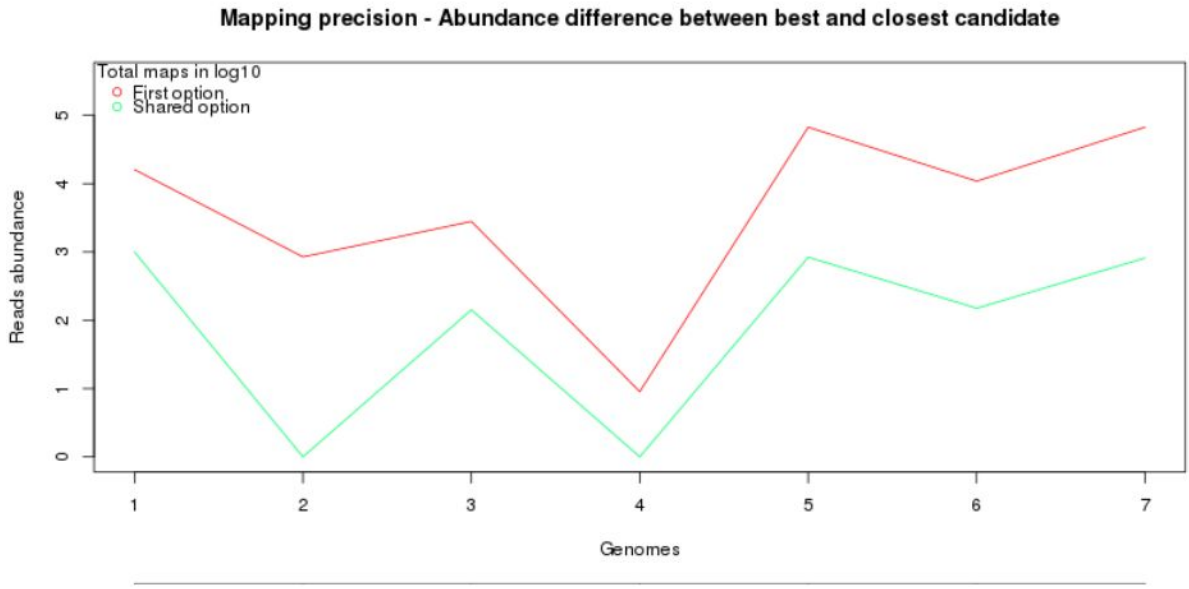

Figure 19: Mapping precision plot. X-axis are the genomes by species and the Y-axis are the taxonomic abundance of assigned reads. The red line is the abundance for the best candidate, whereas the green one is the abundance of the second best candidates.

The GMAP tool is linear on the size of the input since it reads batches of fragments associated to a read and once these are loaded, it computes the best three fragments, and therefore it takes $O(3n \log n)$ in the worst case.

4.2.2. Tools for the fine-grained level and further processing

4.2.2.1. Quality Mapping

One of the first post-mapping results is the quality mapping comparison. This tool takes as input the mapping file and outputs a matrix of n rows and 4 columns per option, where the rows are genomes and the columns are, for each option --3 therefore-- reads abundance, average percentage of identity, average percentage of coverage and average length of matches. Another 4 columns are included with the differences between these previous values between the options (computed difference).

In addition, the 3-mapping approach allows to assess the mapping certainty at both reads and species level; at read level by comparing the identity and coverage indicators of particular genomes against the rest (See Figure 20), and at species level by comparing the abundance levels of the different options for the particular genomes. For example, for all reads mapped over a given genome, information about the identity and coverage level of the second and third mapping option would provide information about the quality or certainty of the first option.

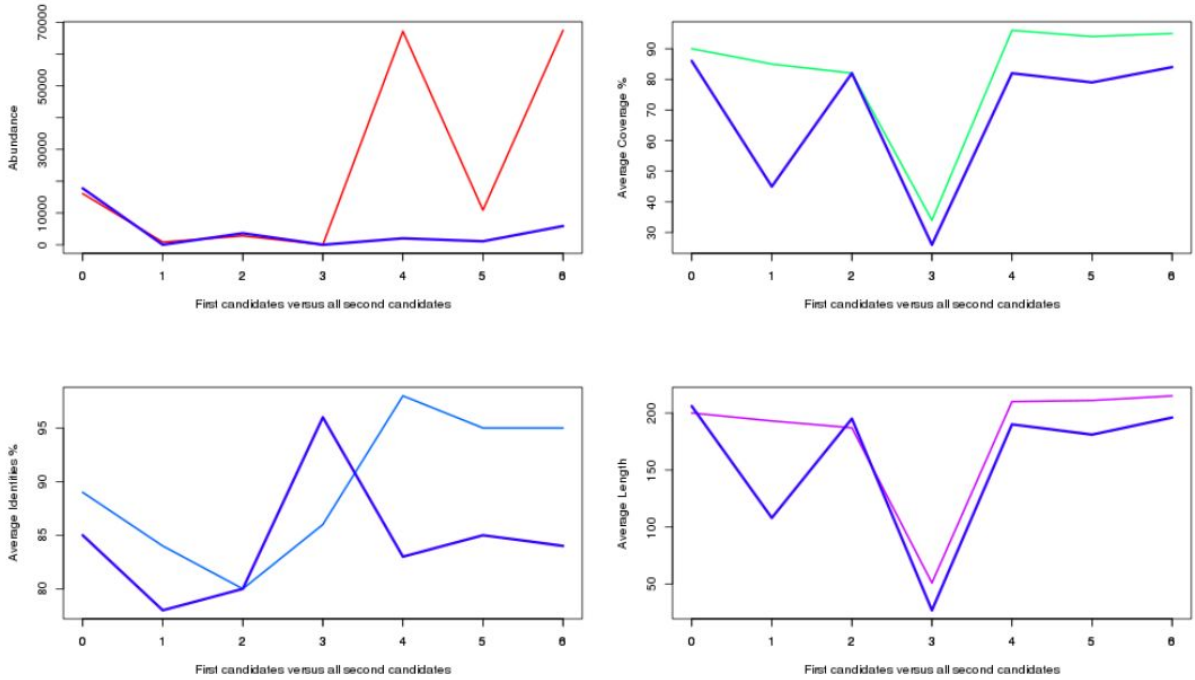

Figure 20: Options comparison plot. The X-axis are the genomes by species and the Y-axis is reads abundance (top left), average percentage of coverage (top right), average percentage of identity (bottom left) and the average length in nucleotides (bottom right).

Figure 20 shows an options-comparison plot between the first best candidate (changing color in the plot) for all genomes and the second candidate (blue line in all plots). For example, the top-left plot shows that only the genomes with ID 4 and 6 have significant distance in terms of mapping abundance respect to their second options, thus ensuring a better mapping. It can be observed in the rest of the plots that their average coverage (top-right), average identities (bottom-left) and average length (bottom-right) is superior in all cases to that of their second best options.

The QualityMapper tool is linear on the size of the input, since its working principle is sweeping the whole binary mapping file adding up the values that conform the different statistic indicators shown.

4.2.2.2. GeneParser

Although this tool does not use the mapping file, it seemed more convenient to put it under the post-mapping tools, since it is of no use without these in further steps. GeneParser was developed to parse GenBank's annotation files to keep only the needed fields, since these

type of files often tend to be large, and for databases of considerable size it can become a lot of unused information, along with the difficulties of computing human-readable files that have not been translated to simple tabular formats. In the line of GeckoToJoiner and BlastToJoiner, GeneParser produces a tabular-text file with as many rows as annotations contained in the file retrieved from GenBank. The following table shows an example of what fields are kept and their meaning.

Gene start	Gene end	Strand	(reserved)	Locus tag	Product
687	3158	f	-	MHO_RS00005	membrane protein insertase YidC

Table 6: Extracted fields and parsed format for an annotation file. The gene start and gene end field refer to the starting and ending coordinates of the gene in the genome. The strand shows an 'f' for forward, and 'r' for reverse. A reserved field is left unused for future implementations. The locus tag field shows the tag given to the gene, whereas the product field refers to the description of the gene and its functional profile.

Table 6 shows an example of parsed annotation file containing one gene. More fields could be added in case of desire for particular and more advanced experiments, such as functional annotation. In particular, annotation files are used to obtain coding and non-coding reads-abundance and to perform other experiments such as differential expression or annotated searches.

4.2.2.3. Coding Abundances

The presence of species in a sample can be measured using the reads abundance. However, this measures might not be enough to ensure the composition of a metagenome, and therefore more experiments are required to be carried out. The tool CodingAbundances produces a list of abundances of annotated and unannotated reads, along with a file per each genome that includes the id's of the reads assigned that fall under annotated regions. Nonetheless, the concept of annotated read is a bit fuzzy: Reads are small DNA sequences, and a single read might have been part of a gene, might be the complete gene or a larger sequence that includes one or more genes, and at last, it might have been the so-called junk DNA. To address this type of situation, mapped reads are divided in the following categories:

(1) Matched reads: Those that are not annotated, not even partially.
(2) Semi-annotated reads: Those that are partially annotated on one side or another.
(3) Fully-annotated reads: Those that are completely inside a gene region.
(4) Extra-covered reads: Those that include more than a single gene region, and may contain more.

The procedure to identify the type of reads follows the algorithm depicted below:

```
1. Sequentially open the parsed annotation files.
2. If the parsed file of annotations contains any annotation:
      a. Load annotations into a fixed-length buffer in memory.
      b. Sweep the binary mapping file and extract the reads that
         map to the genome whose annotation file belongs to.
      c. Sequentially sweep the annotations contained in the
         fixed buffer for every read.
            i. If a hit is found, mark the read and write it to
               the output.
3. If there are more annotation files:
      a. Go to step 1.
4. Else stop.
```

Figure 21: Pseudo-code for the Coding Abundances program. The used approach runs in cubic time, which could be slightly reduced to quadratic time. RAM-based approached would efficiently reduce computation time, however they would be unsuccessful for large databases with many annotations.

It might be noticed by the reader that the running complexity could be reduced. In fact, the binary mapping file is swept through for every genome in the database, and the buffer containing the annotation files is linearly traversed for every read mapped to the genome. Therefore, given g genomes, n reads and a maximum of k annotations per genome, the running complexity is around $O(g * 3n * k)$, which is cubic. However, the number of genomes g is usually very small compared to the number of reads n and same for the number of annotations k. Therefore $g \ll k \ll n$ in the average case and thus the algorithm is tractable. A further modification to improve running times would be performing a binary search over the annotations contained in the buffer, thus reducing it to $O(g * 3n * log\ k)$.

The differential expression experiment is conducted on a particular genome by only considering the reads mapped to annotated regions of the genome and comparing abundance between different samples (See Figure 22) in the same way as RNA-seq transcriptome expression analysis is performed.

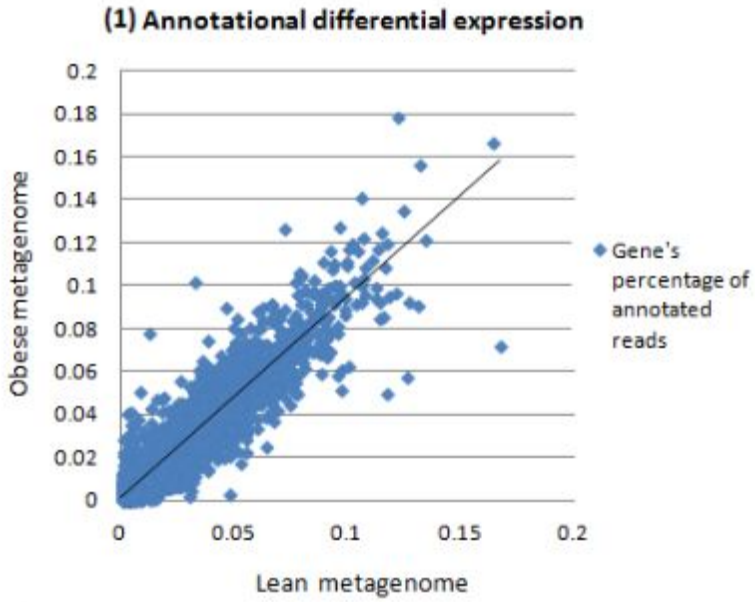

Figure 22: DNA-seq differential expression plot. Each point represents an annotated region for a particular genome. In the x-axis and y-axis, the percentage of reads that are mapped to each annotated region divided by the total mapped reads.

4.2.2.4. Annotational mapping

In the same line as the tool discussed above, the Specific Distribution of Matches tool aims to offer a comparison between the quality of the mapped reads, the distribution of fragments reported for the metagenome (the first is a subset of the second) and the type of matches that said particular genome has. Therefore this tool requires (1) the binary mapping file of the experiment, (2) the reported distribution of fragments by the sequence comparison software (See Figure 23) and (3) the annotation file for the target genome.

The tool produces a matrix of the type of matches of size $3k \times 100$ where k is the number of rows that represent length, and 100 the number of columns, that represent a percentage and normally the percentage of identity is used. For example, a matrix of size 90×100 implies that there are no reads of length larger than 30 bp. Since every three rows represent one base pair, row 30 will be a basic match of length 10 bp, row 31 will be a semi-annotated match of length 10 bp and row 32 will be a fully-annotated read of length 10 bp. The next three rows follow the same scheme but for 11 bp. Therefore the type of match depending on the row is extracted by performing the modulus operation and comparing the rest to either 0, 1 or 2.

Figure 23 shows an example of annotational mapping. The main key of this plot is to show the type of the matched reads for a genome (if these are annotated or not) and to compare their properties (identities and length) with the full distribution of fragments.

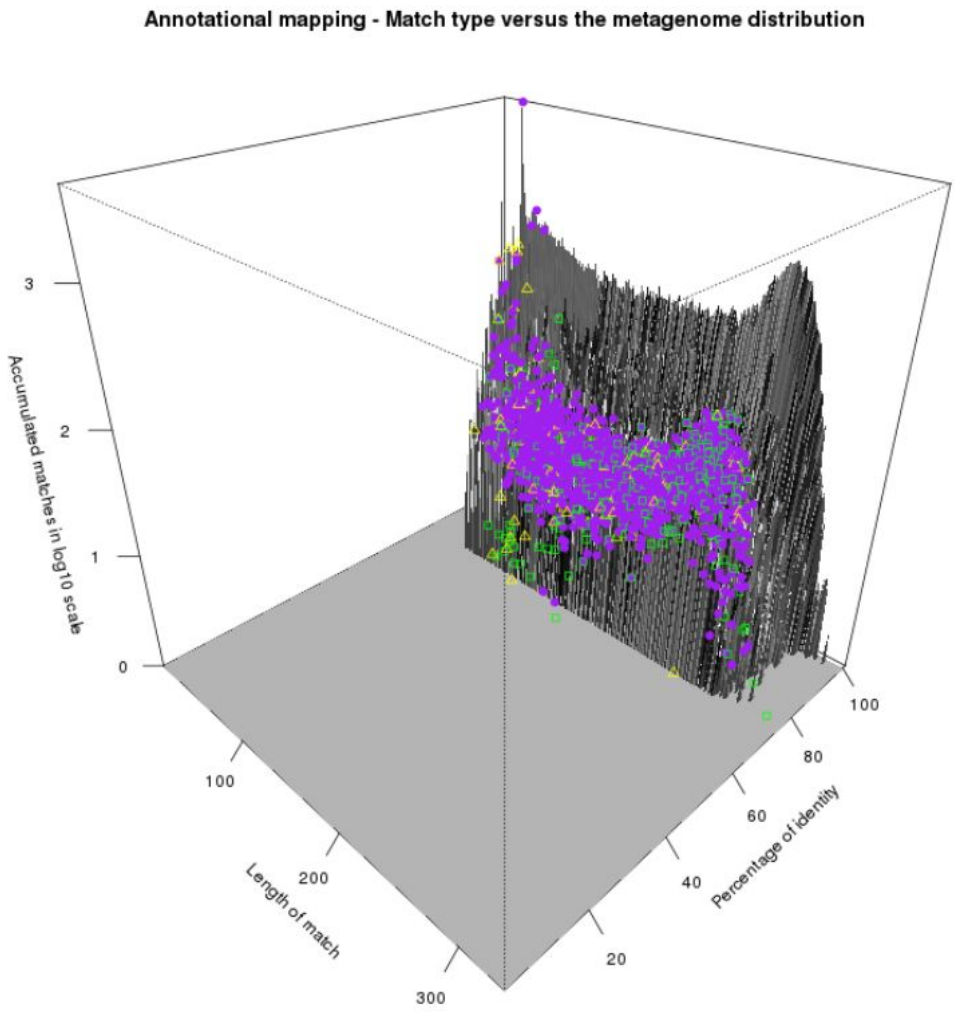

Figure 23: Annotational mapping plot. The X-axis is the percentage of identity of the matches. The Y-axis is the length of the matches. The Z-axis is the accumulation of fragments in log10 scale. The specific type of matches are plotted in purple, green and yellow to illustrate the different type of matches.

The following remarks should be taken into account to understand the plot:

1. In the background, the 3-D grey distribution is the distribution of fragments by length and percentage of identity.
2. By colors, and plotted onto the grey distribution, the type of match that was mapped to the genome. The type of match can be either unannotated (green), partially annotated (yellow) and fully annotated (purple).

4.2.2.5. Genome Profile of Accumulated Reads

An important term in genomics and also in the metagenomics field is the coverage. We have spoken about coverage in reference of a measure that is the percentage of a matching fragment divided by the total length of a read. However, this term is known as an indicator of how many reads or DNA sequences populate or *cover* a genome. In our case, it is of interest to check how many reads map to each position of a genome, as to define a measure of continuity along a genome. The higher concentration of reads along the whole genome, the better.

The binary mapping file is used to find all reads mapped to a particular genome, and the region of the genome where these reads are mapped to are stored and added up. In the end, a profile of the amount of accumulated reads along the genome is obtained and plotted using an R script. See Figure 24. However, bacterial genomes are usually of length 1 Mbp and up to several millions, and therefore it becomes difficult to plot millions and millions of points along a straight line in an image. For this matter, the R script uses a smoothing window of parametrized size (say a 1000 bp) that computes the average number of mapped reads per 1000 bp, and thus a genome of 1 Mbp would only require $\frac{10^6}{10^3} = 10^3$ pixels. Bigger windows can be used to fit larger genomes. Hence the complexity is $O(n+k)$ since it is needed to sweep the binary mapping file once and the length of the genome to perform the smoothing window average.

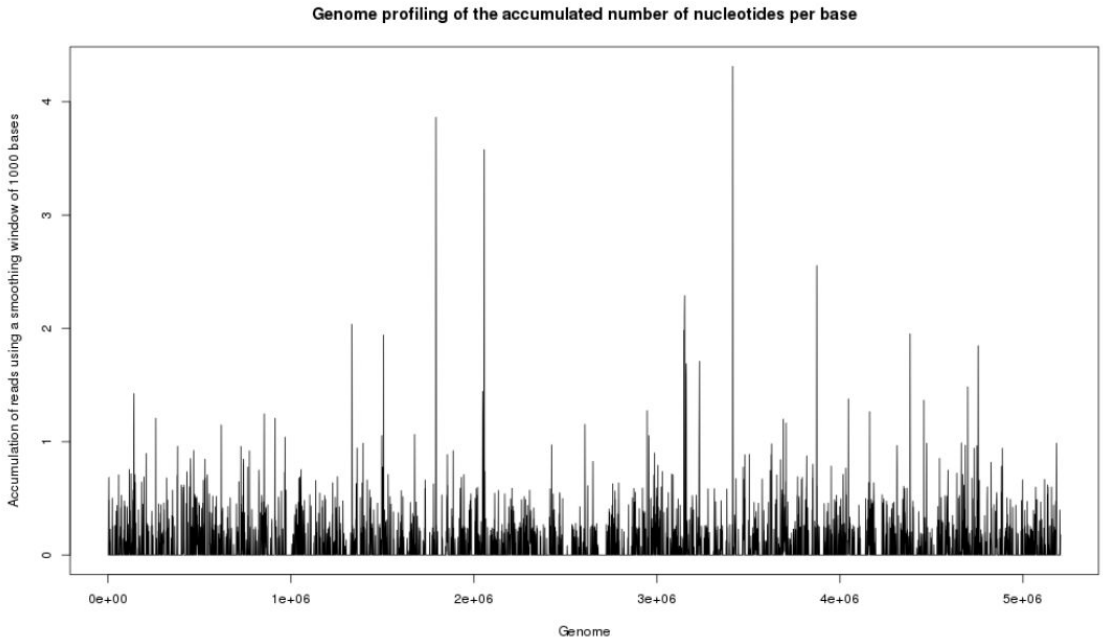

Figure 24: The genome profile of accumulated reads shows how many nucleotides are mapped to each part of the genome. The plot is smoothed using a parametrized sliding window of size 1000 base pairs, which means each position is the average of nucleotides of the surrounding 1000 base pairs.

4.2.2.6. Genome Cutter, Specific Regions and Specific Reads

We have proposed the question “How many reads mapped to a genome are needed to accept or reject the presence of a species in a sample?” earlier in the Analysis and Design section. Current state-of-the-art metagenomic analysis software do not offer any procedure to address this question. The developed workflow offers a set of tools capable of extracting those reads that are mapped to regions that only belong to one genome, i.e. supporting with strong evidence that a species is present even in situations where its reads abundance is low.

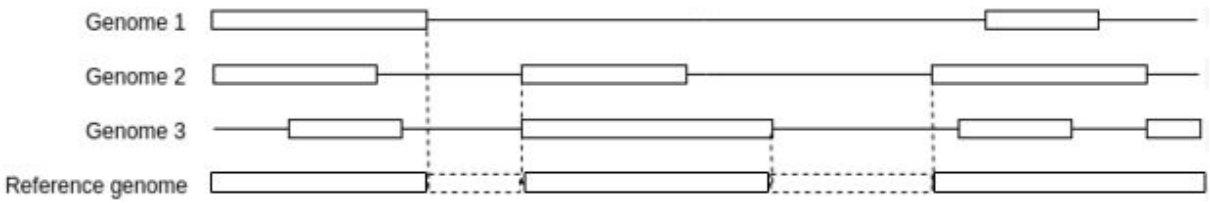

Figure 25: Example of specific regions detected in a database with $N = 4$. Each horizontal line represents a genome. They do not need to be necessarily of the same length, they are showed in the picture in order to facilitate its understanding. Each rectangle (but the dotted ones) represents a fragment shared between the genome of the line where it is present and the reference genome. The dotted rectangles represent the specific regions of the reference genome, where clearly there are no fragments in any of the genomes.

If we take a look at Figure 25 we will see that this type of problem needs genome vs. genome sequence comparisons. For that matter, the first step of this tool is running GECKO to perform a comparison between the target genome and the rest of the database, thus leading to $N - 1$ comparisons for a database of size N. The output of the comparison is taken and fed to the Specific Regions tool, which will extract the regions that are free-of maps. The output is produced as a list of coordinates that represent specific regions to the target genome. Then, the Specific Reads tool uses the list of regions and the binary mapping file of the experiment to compute which reads are mapped to the specific regions.

GECKO is a third party software and therefore its complexity will not be analyzed. In order to run the comparison with GECKO, we will need first to remove the target genome from the reference database, since if it is included it will produce a full alignment and fragments at every position in the genome depending on the k-mer size. To avoid this, the Genome Cutter program removes a particular target genome from the reference database. Its complexity is straight linear since it only requires to copy and write sequences from one file to another, skipping the targeted one.

The Specific Regions tool loads an array of bytes of length equal to that of the genome, where each byte represents either mapped region or free region. This could be improved using $\frac{1}{4}$ of the memory needed by implementing it at a bit-level, however, even the human genome could be fitted into RAM using the byte procedure (~3 GB). Finally, a sweep-through is performed on the array, filtering those regions that are smaller than a given length (e.g. 30 base pairs). Thus the running time is $O(2n)$ being n the number of fragments produced by the GECKO comparison. The Specific Reads tool runs on the list of regions and the binary mapping file. First, the regions are loaded into ram and then for every mapped read to the genome the array is swept-through checking if a read is partially or completely inside, similarly to the tool Specific Distribution of Matches. Therefore it's running time is dependent on the size of the binary mapping file and the number of regions. However, the number of specific regions is inversely proportional to the length of the database, and the number of mapped reads is directly proportional to the size of the database (among others), therefore it the number of specific regions is high, the number of mapped reads is smaller, making it in average of lesser complexity than quadratic.

4.2.2.7. Coverage-Identity Matrix

To provide another measure of quality relatively to the whole mapping distribution, for example, to show properties of all mapped reads at once, the Coverage-identity matrix program was developed. It produces a $n \times m \times k$ tabular-text-file matrix where n is the percentage of identity (from 0 to 100), m is the percentage of coverage (from 0 to 100 as well) and k is the accumulation of mapped reads. Such matrix is read by an R-script plots a 3D distribution of the matrix (see Figure 26) applying logarithmic scale to the k matrix since the accumulation of mapped reads can be extremely high in dispersed points and rather low on others.

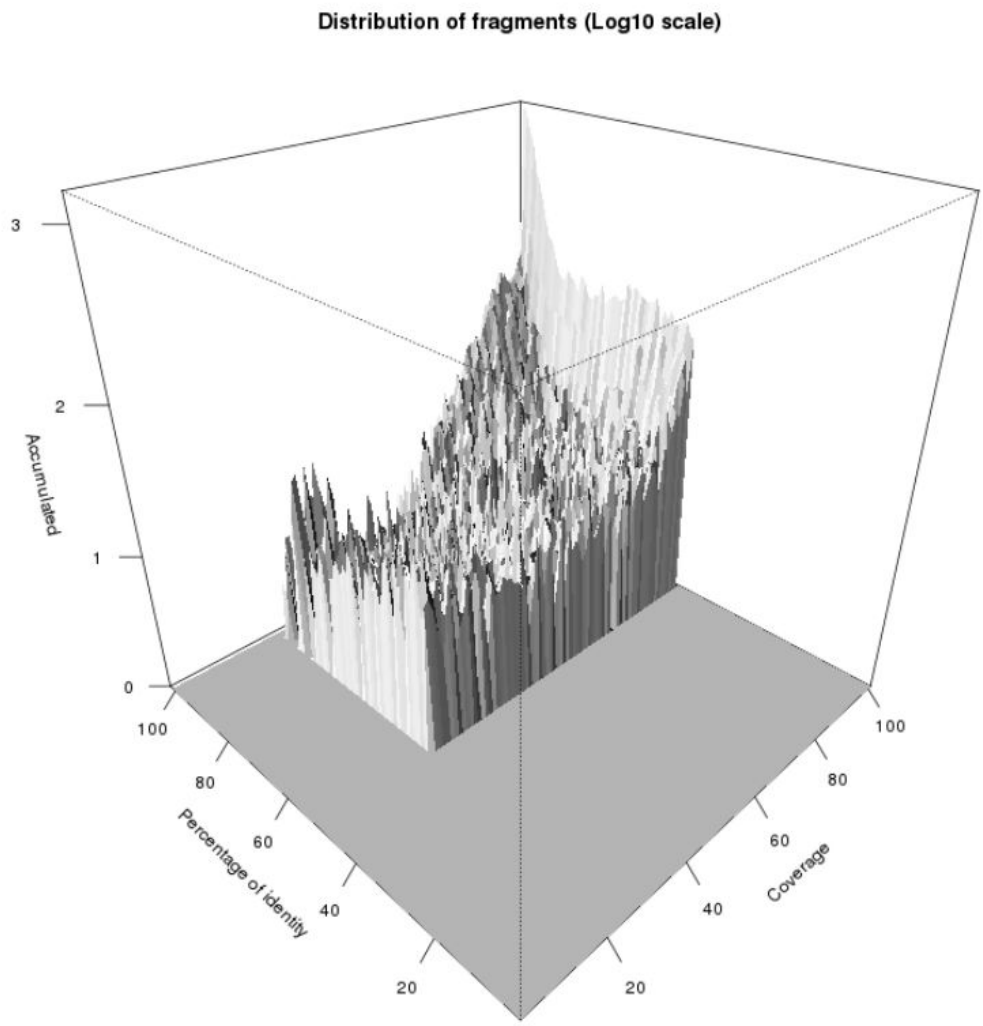

Figure 26: 3D Plot of the distribution matrix produced by Coverage-Identity Matrix. On the x-axis, the percentage of identity of the mapped reads. On the y-axis, the percentage of coverage of the mapped reads. On the z-axis, the accumulation of mapped reads for a given point of percentage of identity and coverage in logarithmic scale. Notice that the distribution is segmented at 60% identity and 30% coverage.

Figure 26 shows that the most accumulation occurs at the point 100% identity and 100% coverage, which implies the mapping module mapped most reads with good properties. However, it can also be seen that a considerable amount of mapped reads had very low coverage, hence only a small part of the read was mapped. More accumulation towards the back corner means a better mapping.

The complexity of Coverage-Identity Matrix is linear on the size of the binary mapping file, for it sweeps such file and counts the number of mapped reads in a two-dimensional vector. Once it finishes sweeping, the vector is written to disk as a 100×100 matrix, which is constant and therefore not accounted for the complexity.

4.2.3. Galaxy implementation

As mentioned in other sections of this manuscript, the whole workflow is available under a Galaxy workflow manager implementation, which makes it simple to run experiments without having to face the difficulties that arise from shell-calls that have many parameters and that have to be run in particular order. Installing a Galaxy instance in a local machine might be tedious in some cases and, furthermore, if we only wish to try out we would need to remove all installed plugins after checking the software out. Therefore we have provided with an Ubuntu virtual machine with all binaries installed along with the Galaxy instance to avoid bothering the final user with installing tutorials and derived issues. It is recommend that the virtual machine is run at least with 4GB of RAM, since metagenomics processing requires still a considerable amount of memory.

In addition, a Galaxy Guided Exercise is included in the documentation which shows how to run the virtual machine, start the Galaxy Instance and also how to run different types of exercises.

4.2.4. Galaxy's tool definition language

Galaxy uses the XML [46] language for tool definition. This is, each developed binary that compose the workflow must have an XML file which uniquely determines the inputs of the binary executable (along with information such as the format of the inputs, the labels and helps that will be displayed, etc.), the outputs (similarly to the inputs) and the system call to the binary executable. See details at the Galaxy Tool Config[15] tutorial.

The XML files must be first created for each tool and then added to the tool registry (namely to the `tool_conf.xml.sample` file) under a section that includes the name of the workflow and the paths to all .xml tools.

```
<tool id="mgReadsIndex2" name="MGRindex">
  <description>Creates indexes for the fast access of fasta
files</description>
  <inputs>
      <param name="input_fasta_file" type="data" format="fasta"
label="Fasta file" help="Create an index of a fasta file"/>
  </inputs>
  <command>/home/galaxy/galaxy/tools/metagecko/binaries/mgReadsIndex2
$input_fasta_file $input_fasta_file ; mv ${input_fasta_file}.nonsorted
$nosorted_index_output0 ; mv ${input_fasta_file}.sorted
```

[15] https://wiki.galaxyproject.org/Admin/Tools/ToolConfigSyntax

```
$sorted_index_output1 </command>
  <outputs>
      <data name="nosorted_index_output0" format="nonsorted" label="Non
sorted binary index of ${input_fasta_file.name}"/>
      <data name="sorted_index_output1" format="sorted" label="Sorted
binary index of ${input_fasta_file.name}"/>
  </outputs>
</tool>
```

Table 7: Example of an .xml file that uniquely identifies a tool. In this case, the .xml belongs to the tool MGReadsIndex, which has one input defined (the FASTA file), and two outputs (the sorted and non-sorted indexes) which must be moved to the output variables manually using the `mv` command in UNIX, since one is created relatively to the other and not with a given direct name.

However, it is noteworthy to say that formats must be registered too. For example, in Table 7, the tool MGReadsIndex uses an output format named "sorted" and "nonsorted". These are customly defined formats that must be registered in the `datatypes_conf.xml.sample` file in the `config` folder. Table 8 shows an example of registering the format "sorted".

```
<datatype extension="sorted" type="galaxy.datatypes.binary:Sorted"
mimetype="application/octet-stream" display_in_upload="true" description="A
sorted binary index file.  You must manually select this 'File Format' when
uploading the file." />
```

Table 8: Example of registering a custom format. Along with the extension, the class from which the data type extends must be specified. In this case, it is the `galaxy.datatypes.binary:Sorted`.

Nonetheless, as seen in Table 8, the format "sorted" extends the class `galaxy.datatypes.binary:Sorted`. and therefore the binary data type must be registered as well. In cases of using text files, it is possible to only specify that the formats (e.g. a tabulated text file) is a subclass that extends `galaxy.datatypes.data:Text`, and therefore it does not need more configuration. On the other hand, for the binary data type, we will need to modify one more file, namely the `binary.py` in the `lib` directory. This file will store the definitions for the binary files, how these should be treated and in case we wish to, that we provide a *sniffer* (a small routine to detect the type of binary file automatically).

```
class Sorted( Binary ):
      """Class describing a sorted index binary file"""
      file_ext = "sorted"

      def set_peek( self, dataset, is_multi_byte=False ):
      if not dataset.dataset.purged:
            dataset.peek  = "Sorted binary index file"
            dataset.blurb = data.nice_size( dataset.get_size() )
```

```
      else:
            dataset.peek = 'file does not exist'
            dataset.blurb = 'file purged from disk'

      def display_peek( self, dataset ):
      try:
            return dataset.peek
      except:
            return "Sorted binary index file (%s)" % ( data.nice_size(
dataset.get_size() ) )

Binary.register_unsniffable_binary_ext("sorted")
```

Table 9: Example of registering a custom binary data type. Notice that the data type is being registered as *unsniffable*, in the sense that it should not be tried to automatically determine, for there is no *sniffer* provided.

After the procedure depicted above is completed for each format, data type and tool, the workflow is well-defined and the Galaxy instance is ready to be used. Only one step left, a quick service restart to make new changes work.

CHAPTER 5.
USE CASES VALIDATION

Despite the presented advantages and the additional processing results offered by the developed method, it is mandatory to compare results in terms of reads abundance with other metagenomic processing packages, i.e. the mapping module should provide with more or less the same results. For this reason, we will be illustrating two comparisons with the standalone tool MEGAN which will serve as test subject, for it is a widely known software package and its results are accepted with unanimity. In addition, it uses a sequence similarity search method (by running the BLAST suite) and therefore can be directly compared to the proposed workflow.

5.1. Lean and obese samples

5.1.1. Dataset availability

Two artificial metagenomes were constructed using six metagenomes that belonged to people with a lean condition and another six metagenomes that belonged to people with obese condition. The artificial metagenomes were built by concatenating (i.e. pooling) every metagenome of each condition in a single metagenome, therefore resulting in two larger metagenomes containing one those of the people with lean condition and the other those with obese condition. The dataset of single metagenomes used is available at the EBI repository[16] and can be downloaded and used. The list of patient metagenomes used for the lean condition were TS1, TS2, TS4, TS5, TS7 and TS8. In the other hand, the list of patients for the obese condition were TS6, TS19, TS20, TS21, TS49 and TS50. These were all randomly selected from all available samples.

The samples were originally collected from fecal microbial communities of adult female monozygotic and dizygotic twin pairs sequenced by 454 GS FLX pyrosequencing and their length ranged from 150 bp to 250 bp after performing filtering and trimming with Replicates [47] and SeqTrimNext. The total number of reads contained in the lean pooled metagenome comprised 2,724,867. In the obese pooled metagenome the number of reads was 2,972,697.

The reference database was built using 22 of the most representative bacteria found in human intestines, and was comprised of mostly bacteroidetes, firmicutes and actinobacteria,

[16] https://www.ebi.ac.uk/metagenomics/projects/SRP000319

and accounted for a total of 325 DNA sequences due to scaffolds, contigs and different strains.

5.1.2. Comparison with MEGAN

In order to prove that the results of the proposed workflow are consistent with those of other metagenomic analysis software suites (in terms of abundance in the taxonomic classification), the following test was performed using results from BLASTN based on metagenomic samples from faecal microbial communities. Both, the workflow (MG workflow) and MEGAN were executed using the same input from BLASTN and ran with default parameters.

On comparison of the lean metagenome based on MEGAN, the abundance plot (See Figure 27) shows similar results to ours. Reads abundance were grouped by strains and phylum to show concordance. However, whereas the mapping procedure of a metagenome using MEGAN can last nearly half an hour, the proposed workflow took 5 minutes and 16.901 seconds to analyze the obese metagenome and 4 minutes 53.988 seconds for the lean one when BLAST the comparison had been done with BLAST, from which CPU-system time was 15 and 17 seconds, respectively. With GECKO, the duration of the process was further reduced, taking about only one minute for the lean sample and three minutes and a half for the obese metagenome. The significant speed of the mapping module when using GECKO compared to BLAST is the number of reported fragments and the amount that can be extended (thus reducing the fragment count). Runtime executions were measured using a standard Intel i5 machine with 4GB of RAM.

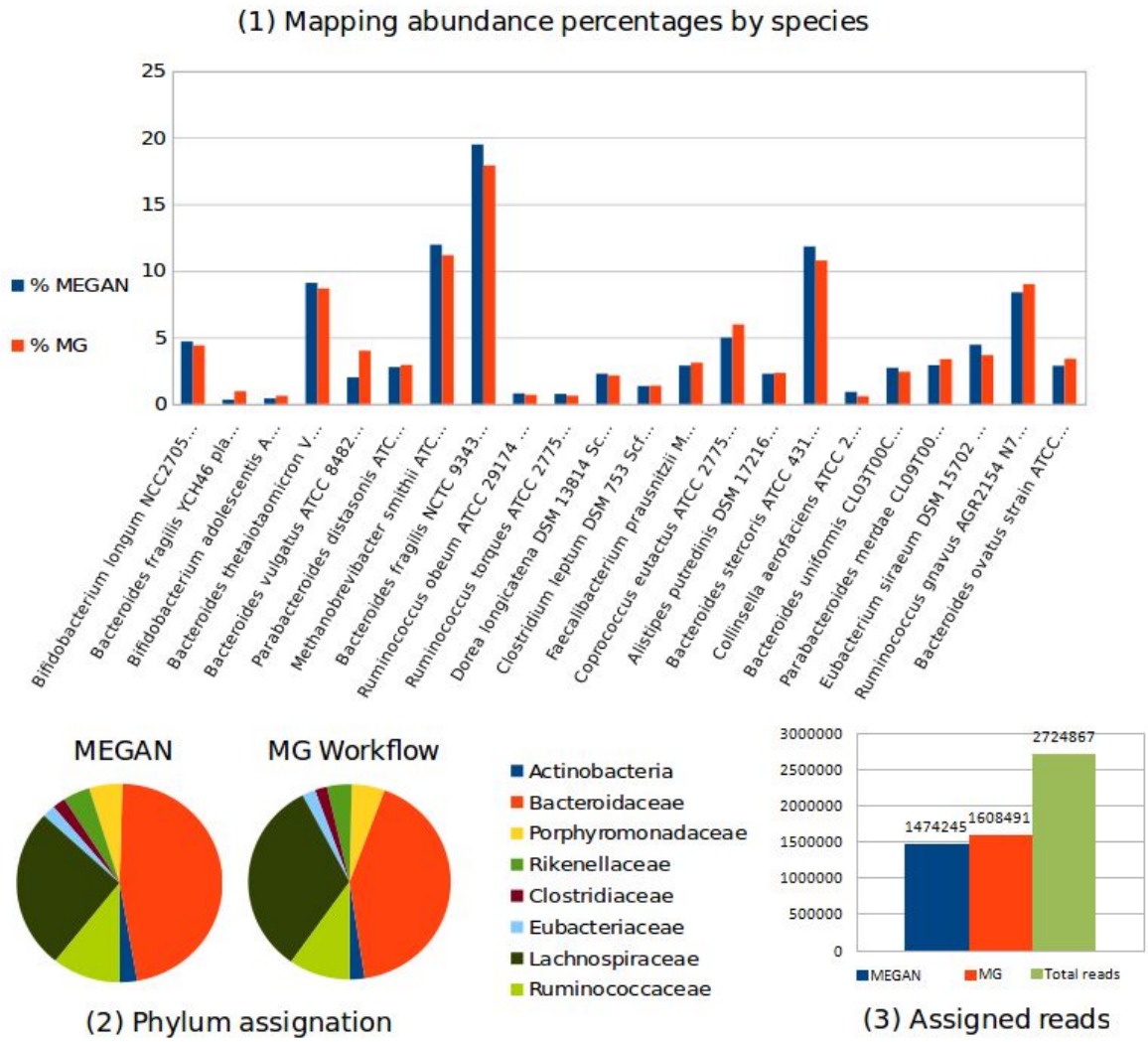

Figure 27: Comparative analysis for the lean metagenome shows similar mapping abundances. Top: Abundance plot by phylum in percentages. Left-side bottom: Abundance chart by Family. Right-side bottom: Total reads assigned by each method and total number of reads in the metagenome.

5.1.3. Statistical validation

A contrast on the differences of means was performed assuming both samples belonged to a normal distribution, in order to determine whether or not the proposition that both results came from the same distribution could be accepted or not. For this purpose, the reads abundance per genome were considered as paired data points in the contrasted hypothesis.

A priori hypothesis on the contrast of the equality of standard deviations:

$$H_0 : \sigma_{MEGAN} = \sigma_{MG}$$
$$H_1 : \sigma_{MEGAN} \neq \sigma_{MG}$$

Which yielded a p-value of 0.69, which implies the supposition that standard deviations are unknown but yet equal at any regular α .

Contrasted hypothesis on the difference of means:

$$H_0 : \mu_{MEGAN} - \mu_{MG} = 0$$
$$H_1 : \mu_{MEGAN} - \mu_{MG} \neq 0$$

The resulting p-values was of 0.99 and therefore the null hypothesis was accepted at any α, concluding that the difference of means of the paired read-abundance values was nearly zero, and thus there exist no significant difference between the results.

5.2. Sick and healthy swine samples

5.1.1. Dataset availability

For the second comparison, we used two collections of metagenome samples from the respiratory tract of healthy and diseased *Sus scrofa* taken from the swine slaughterhouse. The samples had been sequenced using the 454-pyrosequencing technology. The diseased samples contains 678,500 reads whereas the healthy sample contains 581,383 reads. The analysis is based on the understanding that the pigs are naturally infected, and therefore the point of observing significant variation in particular bacteria between the diseased and healthy sample suggest that such species are related to the condition of the swines. Unfortunately, the dataset used in this comparison has not been made public at the time of writing the manuscript and remains yet private.

The raw samples' read average length was 1,200 bp before trimming and filtering. Filtering was performed using Replicates and trimming was performed using LUCY [48]. The average length of the reads dropped from 1,200 bp to 500 bp in average, which is expected for the type of reads after removing adapters, low-complexity regions, etc. The reference database was built using 50 custom genomes which are known to be related to the respiratory tract of swines. The included bacteria was comprised of mostly actinomycetales, bacteroidetes, firmicutes, proteobacteria and mycoplasmataceae, and accounted for a total of 283 DNA sequences due to scaffolds, contigs and different strains.

5.2.2. Comparison with MEGAN

As stated earlier, in order to prove that the results of the proposed workflow are consistent with those of other metagenomic analysis software suites (in terms of abundance in the taxonomic classification), the following test was performed using results from BLASTN based on metagenomic samples from the respiratory tract of swines. Both, the workflow (MG workflow) and MEGAN were executed using the same input from BLASTN and ran with default parameters.

On comparison of the sick and healthy metagenomes (in advance, M01 and M02, respectively) based on MEGAN, the abundance plot (See Figure 28) shows similar results and tendence to ours. Reads abundance were grouped by strains to show concordance. However, whereas the mapping procedure of a metagenome using MEGAN can last nearly half an hour, the proposed workflow took 1 minute and 22.8 seconds to analyze the diseased metagenome and 1 minute 15.178 seconds for the healthy one when BLAST the comparison had been done with BLAST. Runtime executions were measured using a standard Intel i5 machine with 4GB of RAM.

Although some small dissimilitude appear to exist between the reads abundance results, it is important to notice that the plot is showing reads abundance in logarithmic scale, and therefore amplifying the variations of small and low-abundant genomes. This also suggests that, as expected, the developed workflow can report more evidence in terms of low-abundant genomes in part due to the fragments extension step and the adequate combination of filters.

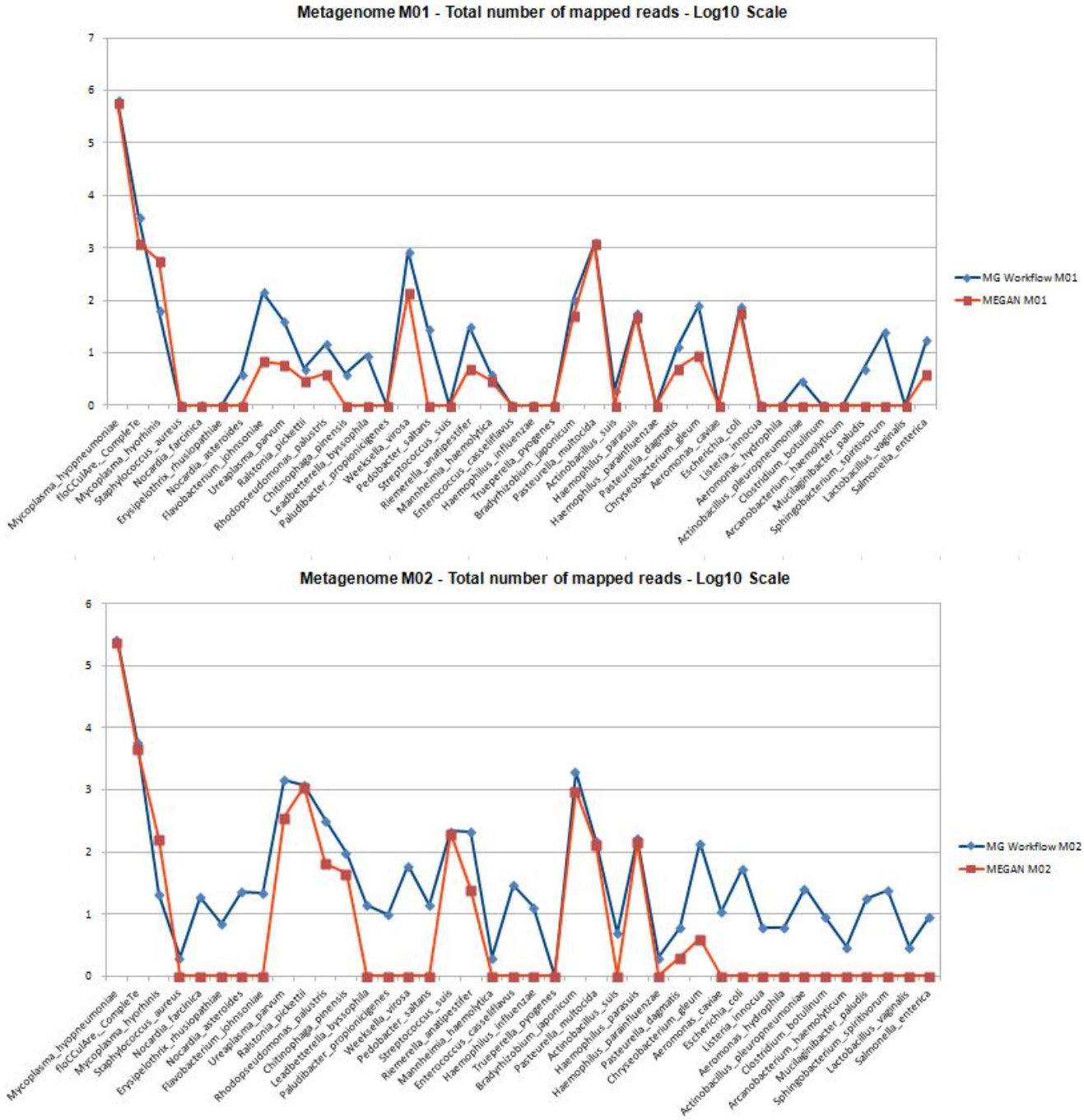

Figure 28: Reads abundance comparison between MEGAN and the proposed method per strains. Top plot shows the reads abundance for the diseased metagenome (M01) in logarithmic scale. Bottom plot shows the reads abundance for the healthy metagenome (M02) in logarithm scale.

5.2.3. Statistical validation

A contrast on the differences of means was performed assuming both samples belonged to a normal distribution, in order to determine whether or not the proposition that

both results came from the same distribution could be accepted or not. For this purpose, the reads abundance per genome were considered as paired data points in the contrasted hypothesis.

- A priori hypothesis on the contrast of the equality of standard deviations for the metagenome M01:

$$H_0 : \sigma_{MEGAN} = \sigma_{MG}$$
$$H_1 : \sigma_{MEGAN} \neq \sigma_{MG}$$

Which yielded a p-value of 0.51 in the diseased metagenome, which implies the supposition that standard deviations are unknown but yet equal at any regular α .

- A priori hypothesis on the contrast of the equality of standard deviations for the metagenome M02:

$$H_0 : \sigma_{MEGAN} = \sigma_{MG}$$
$$H_1 : \sigma_{MEGAN} \neq \sigma_{MG}$$

A p-value of 0.58 was obtained in the healthy metagenome, which implies the supposition that standard deviations are unknown but yet equal at any regular α .

- Contrasted hypothesis on the difference of means for the metagenome M01:

$$H_0 : \mu_{MEGAN} - \mu_{MG} = 0$$
$$H_1 : \mu_{MEGAN} - \mu_{MG} \neq 0$$

The resulting p-value of the M01 metagenome was of 0.29 and therefore the null hypothesis was accepted at any standard α, concluding that the difference of means of the paired read-abundance values was nearly zero, and thus there exist no significant difference between the taxonomic results.

- Contrasted hypothesis on the difference of means for the metagenome M02:

$$H_0 : \mu_{MEGAN} - \mu_{MG} = 0$$
$$H_1 : \mu_{MEGAN} - \mu_{MG} \neq 0$$

The resulting p-value of the M02 metagenome was of 0.23 and therefore the null hypothesis was accepted at any standard α, concluding that the difference of means of the paired read-abundance values was nearly zero, and thus there exist no significant difference between the taxonomic results.

CHAPTER 6.
CONCLUSIONS

6.1. Conclusions of the developed workflow

The proposed workflow is an example of simple low-coupled modules that when put together compose a complete processing method capable of producing fast and reliable results in open environment with easy-to-handle formats. In the computational sense, our intention was to provide an alternative platform for metagenomics analysis, which, in essence, could be easily adapted and expanded to satisfy the growing demand of experiments that researchers are proposing in this novel field, while maintaining a low-level programming strategy that kept computation speed as one of the priorities.

Furthermore, metagenomics is an effervescent field and there are still a number of questions that need to be addressed before a stable version of a definitive data analysis software becomes available. Currently, metagenomic analysis tools generally represent a closed environment and offer few configuration options and limited extension possibilities. In this sense, our aim was to develop a software framework to which other modules could be added. An additional motivation to develop this software was the need for software sensitive enough to detect the presence of low-abundance species. Our intent was to provide data in standard and editable formats that facilitate further analysis with external software. The proposed workflow software offers several notable advantages over the software currently available in the market. Firstly, the use of GECKO enables this software to compute similarity searches in the samples against a collection of genomes in a reasonable time. Providing different mapping alternatives helps set up a sort of quality measures of the mapping process based on abundance differences across mapping alternatives.

The proposed software is designed to provide evidence of the presence of low-abundance species by finding particular specific regions of genomes with mapped reads. These mapped reads provide strong evidence of the species present in samples. The methods developed for assessing and evaluating the quality of mapping also improve accuracy and reliability in terms of the identification of the species present in a sample. However, from our perspective, the most important contribution of this workflow software is that it offers the possibility of incorporating new modules to extend the analysis workflow by showing datafile specifications, which enables fine-grained metagenomic data analysis via a simple Galaxy interface which runs experiments at the cost of a few clicks.

In addition, the proposed workflow depicted on this manuscript, has been submitted to the highly-ranked journal BMC Genomics[17] as a research article, and is under second-round inspection at the time of writing this conclusions.

6.1.1. Conclusiones del trabajo desarrollado

El método desarrollado es un ejemplo de simples módulos cuyo poco acoplamiento permite la composición en un sistema completo de procesamiento capaz de producir resultados rápida y fiablemente en un entorno abierto y con formatos de datos sencillos de manejar. En términos computacionales, nuestra intención fue la de proveer una plataforma alternativa para el análisis de metagenomas, la cual, en esencia, pudiera ser fácilmente adaptada y expandida para satisfacer la gran demanda de experimentos que los investigadores sugieren en este campo tan novedoso, a la par que manteniendo un enfoque de programación a bajo nivel cuya prioridad principal es el reducido coste computacional.

El campo de la metagenómica está en alza y aún siguen existiendo gran cantidad de preguntas que debe ser respondidas antes de que una versión estándar de paquete de análisis de metagenómica esté disponible. Actualmente, las herramientas de procesamiento de metagenomas están generalmente desarrolladas bajo un entorno cerrado, ofreciendo por ello pocas posibilidades de configuración y flexibilidad, a la vez que escasa ampliación. En este sentido, hemos pretendido desarrollar un marco de trabajo al que se le pudieran añadir nuevos módulos sin dificultad. Además de ello, otra motivación ha sido la necesidad actual de software más fino, capaz de detectar y tratar especies cuya abundancia fuera baja. Nuestra intención fue, además, proveer resultados de manera estándar y abierta para facilitar un análisis posterior con herramientas externas. El método propuesto ofrece ciertas ventajas sobre el software actualmente existente. Primariamente, el uso del paquete GECKO permite computar comparación de secuencias entre muestras metagenómicas y genomas patrón en tiempo razonable. La flexibilidad en las alternativas de la fase de *mapping* nos permite extrapolar medidas de calidad del proceso basándonos en las diferencias de abundancias.

El paquete desarrollado fue diseñado con objeto de proveer evidencias sobre la presencia de especies con poca abundancia buscando zonas particulares de genomas con lecturas asignadas, pues éstas dan soporte más fuerte sobre la presencia de dicho genomas. Los métodos propuestos para evaluar la calidad de la asignación taxonómica promueven mayor fiabilidad en la identificación de especies. Sin embargo, desde nuestro punto de vista, la contribución más importante es la posibilidad de incorporar nuevos módulos para extender el flujo de trabajo, dadas las especificaciones de los formatos de salida, permitiendo a su vez una experiencia rica de procesamiento de datos con sólo unos pocos clicks a través del gestor de flujos de trabajo Galaxy.

[17] https://bmcgenomics.biomedcentral.com/

Por último, el flujo de trabajo desarrollado descrito en este documento ha sido presentado a la revista de impacto internacional BMC Genomics como un artículo de investigación, y se encuentra actualmente, a la fecha de escritura de este manuscrito, bajo la segunda ronda de revisión.

6.2. Further extensions and improvements

Although we tried to squeeze the limited time that is dedicated to the End of Degree Project to fit all the modules and results we wanted in the proposed workflow, it was unavoidable to leave some things out of the panorama. Nevertheless, it is our intention, for both the student and the tutor, to keep working on this platform, developing processing tools that answer key-questions in metagenomics, facilitating the still-difficult processing steps of a whole experiment, reducing computational requirements in both time and space, providing with novelty tools to assert new processing standards, etc.

More specifically, there are a few things that we definitely wished to implement, but sadly had not time for it. The following proposals fall under such category:

- A phylum taxonomic tree view that enables users to add up reads abundance on different species levels.
- A complete set of different mapping alternatives, such as including the LCA algorithm.
- A pre-selection system to significantly reduce the size of the reference database to speed up computation time.
- A more completed functional profile of the species contained in the samples, e.g. setting up metabolic pathways.

6.3. Acknowledgements

I wish to express my sincere gratitude to my advisor, Prof. Oswaldo Trelles, for the invaluable opportunity that he gave me by offering me a place in his research team and in the Bioinformatics world, and for having faith in my abilities during the development of this work. I also wish to thank all of my colleagues at the Bitlab team, and specially to Óscar for all the support and provided guidance.

Last but not the least, I wish to thank my family for all what they have done for me through the years, and to Annette, for without her day-to-day support none of this would have been possible.

CHAPTER 7.
BIBLIOGRAPHY

1. Enis Afgan, Dannon Baker, Marius van den Beek, Daniel Blankenberg, Dave Bouvier, Martin Čech, John Chilton, Dave Clements, Nate Coraor, Carl Eberhard, Björn Grüning, Aysam Guerler, Jennifer Hillman-Jackson, Greg Von Kuster, Eric Rasche, Nicola Soranzo, Nitesh Turaga, James Taylor, Anton Nekrutenko, and Jeremy Goecks. The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2016 update. *Nucleic Acids Research* (2016) doi: 10.1093/nar/gkw343.
2. Wilde, Michael, et al. "Swift: A language for distributed parallel scripting." *Parallel Computing* 37.9 (2011): 633-652.
3. Wolstencroft, Katherine, et al. "The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud." *Nucleic acids research* (2013): gkt328.
4. Mardis, Elaine R. "The impact of next-generation sequencing technology on genetics." *Trends in genetics* 24.3 (2008): 133-141.
5. De Roure, D., Goble, C. and Stevens, R. (2009): The Design and Realisation of the myExperiment Virtual Research Environment for Social Sharing of Workflows. *Future Generation Computer Systems* 25, pp. 561-567.
6. Benson, Dennis A., et al. "GenBank." *Nucleic acids research* 41.D1 (2013): D36-D42.
7. Kanehisa, Minoru, and Susumu Goto. "KEGG: kyoto encyclopedia of genes and genomes." *Nucleic acids research* 28.1 (2000): 27-30.
8. Mende, Daniel R.; Alison S. Waller; Shinichi Sunagawa; Aino I. Järvelin; Michelle M. Chan; Manimozhiyan Arumugam; Jeroen Raes; Peer Bork (2012-02-23). "Assessment of Metagenomic Assembly Using Simulated Next Generation Sequencing Data". *PLoS ONE*.
9. Richterich P. (1998): Estimation of errors in "raw" DNA sequences: a validation study. *Genome Res*. 8(3):251–259.
10. Huson, Daniel H., and Nico Weber. "Microbial community analysis using MEGAN." *Methods in enzymology* 531 (2012): 465-485.
11. Altschul, S.F., Gish, W., Miller, W., Myers, E.W. & Lipman, D.J. (1990) "Basic local alignment search tool." J. *Mol. Biol*. 215:403-410.
12. Gish, W. & States, D.J. (1993) "Identification of protein coding regions by database similarity search." *Nature Genet*. 3:266-272.
13. Morgulis A., Coulouris G., Raytselis Y., Madden T.L., Agarwala R., & Schäffer A.A. (2008) "Database indexing for production MegaBLAST searches." Bioinformatics 15:1757-1764.

14. Altschul, S.F., Madden, T.L., Schäffer, A.A., Zhang, J., Zhang, Z., Miller, W. & Lipman, D.J. (1997) "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs." Nucleic Acids Res. 25:3389-3402.
15. NCBI, Resource Coordinators. "Database resources of the National Center for Biotechnology Information." *Nucleic acids research* 41.Database issue (2013): D8.
16. Bender, Michael A., and Martin Farach-Colton. "The LCA problem revisited." *Latin American Symposium on Theoretical Informatics*. Springer Berlin Heidelberg, 2000.
17. Overbeek et al., "SEED database" *Nucleic Acids Res* 33(17), 2005.
18. Buchfink et al, Fast and sensitive protein alignment using DIAMOND, *Nature Methods*, 2015, 12:59-60.
19. Huson et al, Integrative analysis of environmental sequences using MEGAN4, *Genome Res*, 2011, 21:1552-1560.
20. Wood, Derrick E., and Steven L. Salzberg. "Kraken: ultrafast metagenomic sequence classification using exact alignments." *Genome biology* 15.3 (2014): 1.
21. Ounit, Rachid, et al. "CLARK: fast and accurate classification of metagenomic and genomic sequences using discriminative k-mers." *BMC genomics* 16.1 (2015): 1.
22. Torreno, Oscar, and Oswaldo Trelles. "Breaking the computational barriers of pairwise genome comparison." *BMC bioinformatics* 16.1 (2015): 1.
23. Segata, Nicola, et al. "Metagenomic microbial community profiling using unique clade-specific marker genes." *Nature methods* 9.8 (2012): 811-814.
24. Langmead, Ben, and Steven L. Salzberg. "Fast gapped-read alignment with Bowtie 2." *Nature methods* 9.4 (2012): 357-359.
25. Kultima, Jens Roat, et al. "MOCAT: a metagenomics assembly and gene prediction toolkit." *PloS One* 7.10 (2012): e47656.
26. Kultima, Jens Roat, et al. "MOCAT2: a metagenomic assembly, annotation and profiling framework." *Bioinformatics* (2016): btw183.
27. Li et al. (2008) SOAP: short oligonucleotide alignment program" . *Bioinformatics*, 24 no.5,713–714, doi:10.1093/bioinformatics/btn025
28. Hyatt, Doug, et al. "Prodigal: prokaryotic gene recognition and translation initiation site identification." *BMC Bioinformatics* 11.1 (2010): 1.
29. Zhu W, Lomsadze A, Borodovsky M (2010) Ab initio gene identification in metagenomic sequences. *Nucleic Acids Research* 38: 1–15 doi:10.1093/nar/gkq275. doi: 10.1093/nar/gkq275.
30. Meyer, Folker, et al. "The metagenomics RAST server–a public resource for the automatic phylogenetic and functional analysis of metagenomes." *BMC Bioinformatics* 9.1 (2008): 1.
31. Caporaso, J. Gregory, et al. "QIIME allows analysis of high-throughput community sequencing data." *Nature methods* 7.5 (2010): 335-336.
32. Quast, Christian, et al. "The SILVA ribosomal RNA gene database project: improved data processing and web-based tools." *Nucleic Acids Research* 41.D1 (2013): D590-D596.

33. DeSantis, Todd Z., et al. "Greengenes, a chimera-checked 16S rRNA gene database and workbench compatible with ARB." *Applied and Environmental Microbiology* 72.7 (2006): 5069-5072.
34. Cole, James R., et al. "The Ribosomal Database Project: improved alignments and new tools for rRNA analysis." *Nucleic Acids Research* 37.suppl 1 (2009): D141-D145.
35. Hunter, Sarah, et al. "EBI metagenomics—a new resource for the analysis and archiving of metagenomic data." *Nucleic Acids Research* 42.D1 (2014): D600-D606.
36. Haft, Daniel H., Jeremy D. Selengut, and Owen White. "The TIGRFAMs database of protein families." *Nucleic Acids Research* 31.1 (2003): 371-373.
37. Cock, Peter JA, et al. "Biopython: freely available Python tools for computational molecular biology and bioinformatics." *Bioinformatics* 25.11 (2009): 1422-1423.
38. Leinonen, Rasko, et al. "The European nucleotide archive." *Nucleic Acids Research* (2010): gkq967.
39. Bolger, Anthony M., Marc Lohse, and Bjoern Usadel. "Trimmomatic: a flexible trimmer for Illumina sequence data." *Bioinformatics* (2014): btu170.
40. Lee, Jae-Hak, Hana Yi, and Jongsik Chun. "rRNASelector: a computer program for selecting ribosomal RNA encoding sequences from metagenomic and metatranscriptomic shotgun libraries." *The Journal of Microbiology* 49.4 (2011): 689-691.
41. Rho, Mina, Haixu Tang, and Yuzhen Ye. "FragGeneScan: predicting genes in short and error-prone reads." *Nucleic Acids Research* 38.20 (2010): e191-e191.
42. Jones, Philip, et al. "InterProScan 5: genome-scale protein function classification." *Bioinformatics* 30.9 (2014): 1236-1240.
43. Falgueras, Juan, et al. "SeqTrim: a high-throughput pipeline for pre-processing any type of sequence read." *BMC Bioinformatics* 11.1 (2010): 1.
44. Hoare, Charles AR. "Quicksort." *The Computer Journal* 5.1 (1962): 10-16.
45. Karlin, S. & Altschul, S.F. (1990) "Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes." *Proc. Natl. Acad. Sci.* USA 87:2264-2268.
46. Bray, Tim, et al. "Extensible markup language (XML)." *World Wide Web Consortium Recommendation REC-xml-19980210. http://www.w3.org/TR/1998/REC-xml-19980210* 16 (1998): 16.
47. Gomez-Alvarez, V., Teal, T. K., and Schmidt, T. M. (2009). Systematic artifacts in metagenomes from complex microbial communities. *The ISME journal*, 3(11), 1314-1317.
48. Chou, Hui-Hsien, and Michael H. Holmes. "DNA sequence quality trimming and vector removal." *Bioinformatics* 17.12 (2001): 1093-1104.