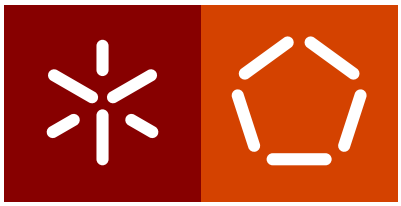
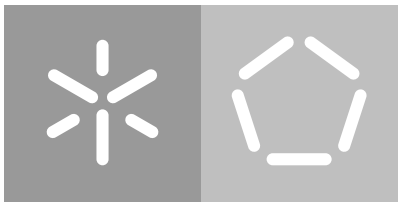


Universidade do Minho
Escola de Engenharia
Departamento de Engenharia Biológica

Cláudia Isabela Sampaio Ganâncio

Implementation of new tools and approaches for the reconstruction of genome-scale metabolic models

July 2020



Universidade do Minho
Escola de Engenharia
Departamento de Engenharia Biológica

Cláudia Isabela Sampaio Ganâncio

Implementation of new tools and approaches for the reconstruction of genome-scale metabolic models

Master dissertation
Master Degree in Bioinformatics

Dissertation supervised by
Oscar Manuel Lima Dias

July 2020

Despacho RT - 31 /2019 - Anexo 3

DIREITOS DE AUTOR E CONDIÇÕES DE UTILIZAÇÃO DO TRABALHO POR TERCEIROS

Este é um trabalho académico que pode ser utilizado por terceiros desde que respeitadas as regras e boas práticas internacionalmente aceites, no que concerne aos direitos de autor e direitos conexos.

Assim, o presente trabalho pode ser utilizado nos termos previstos na licença abaixo indicada.

Caso o utilizador necessite de permissão para poder fazer um uso do trabalho em condições não previstas no licenciamento indicado, deverá contactar o autor, através do RepositóriUM da Universidade do Minho.

Licença concedida aos utilizadores deste trabalho

Atribuição
CC BY

https://creativecommons.org/licenses/by/4.0/

Cláudia Isabela Sampaio Ganâncio

(Cláudia Isabela Sampaio Ganâncio)

Universidade do Minho, 17/07/2020

Despacho RT - 31 /2019 - Anexo 4

Declaração a incluir na Tese de Doutoramento (ou equivalente) ou no trabalho de Mestrado

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

Cláudia Isabela Sampaio Ganâncio

(Cláudia Isabela Sampaio Ganâncio)

University of Minho, 17/07/2020

AGRADECIMENTOS

Há compromissos que quando assumidos têm que ser terminados, dando o melhor que conseguimos, o nosso maior empenho e esforço. Termina aqui um capítulo muito importante da minha vida, a realização de um sonho. Termina com algo que me permitiu sair da minha zona de conforto em todos os níveis, desafiar-me e sobretudo, ganhar conhecimento em duas áreas distintas pelas quais consegui ganhar um gosto especial. Não foi fácil, mas com a presença de um grupo de pessoas que nos apoia incondicionalmente tudo é possível.

Gostaria de fazer um agradecimento especial ao professor Oscar Dias, meu orientador durante este percurso, e ao Davide Lagoa, membro do grupo Biosystems do centro de Engenharia Biológica da Universidade do Minho, que apesar do seu nome não se encontrar na primeira página deste trabalho, também foi uma ajuda fundamental para mim. Obrigada aos dois por toda a atenção, disponibilidade, paciência e partilha de conhecimento.

Aos meus amigos de mestrado, o meu profundo obrigada pela experiência incrível que me proporcionaram na Universidade do Minho, nestes dois anos. Aos meus amigos de sempre, obrigada por todas as conversas, motivação e por estarem sempre lá quando eu preciso. Sem vocês tudo tinha sido mais difícil.

À minha família, obrigada por serem a melhor do mundo. Sou uma sortuda! Guardo um obrigada especial aos meus pais, a quem dedico este trabalho, por terem abdicado de tanto para me proporcionar a melhor experiência da minha vida, aquela que nunca tiveram oportunidade de vivenciar. Obrigada à minha irmã por me chatear sempre que pode, sei que é sempre com muito amor. Ao Nuno, obrigada por seres meu namorado e melhor amigo, obrigada pela paciência, compreensão e por nunca me deixares desistir mesmo quando tudo parece impossível de se concretizar.

Terão sempre um lugar especial na minha vida. Obrigada a todos!

ABSTRACT

The reconstruction of high-quality genome-scale metabolic (*GSM*) models can have a relevant role in the investigation and study of an organism, since these mathematical models can be used to phenotypically manipulate an organism and predict its response, *in silico*, under different environmental conditions or genetic modifications. Several bioinformatics tools and software have been developed since then to facilitate and accelerate the reconstruction of these models by automating some steps that compose the traditional reconstruction process.

"Metabolic Models Reconstruction Using Genome-Scale Information" (*merlin*) is a free, user-friendly, Java^{TM} application that automates the main stages of the reconstruction of a *GSM* model for any microorganism. Although it has already been used successfully in several works, many plugins are still being developed to improve its resources and make it more accessible to any user. In this work, the new tools integrated in *merlin* will be described in detail, as well as the improvement of other features present on the platform. The general improvements performed and the implementation of the new tools, improve the overall user experience during the process of reconstructing *GSM* models in *merlin*.

The main feature implemented in this work is the incorporation of the *BiGG Integration Tool* (*BIT*) in *merlin*. This plugin allows the collection of metabolic data that integrates the models present in the BiGG Models database and its association with the genome of the organism in study, by homology, creating, if possible, the boolean rule for each BiGG reaction in the model under construction. All the computation required to execute *merlin*'s *BIT* takes place remotely, to accelerate the process. Within a few minutes, the results are returned by the server and imported into the user's workspace. Running the tool outside the user's machine also brings advantages in terms of information storage, since the BiGG data structure that supports the entire tool is available remotely. The implementation of this tool provides an alternative to obtaining metabolic information from the KEGG database, the only option available in *merlin* so far. To test the implemented tool, several draft genome-scale metabolic networks were generated and analyzed.

Keywords: *merlin*, *GSM* models, BiGG Models, *BiGG Integration Tool*

RESUMO

A reconstrução de modelos metabólicos à escala genómica (*MEG*) de alta qualidade, pode desempenhar um papel relevante na investigação e estudo de um organismo, uma vez que estes modelos matemáticos podem ser utilizados para manipular fenotipicamente um organismo e prever a sua resposta, *in silico*, sob diferentes condições ambientais ou modificações genéticas. Várias ferramentas bioinformáticas e software têm sido desenvolvidos desde então para facilitar e acelerar a reconstrução desses modelos por automatização de algumas etapas que constituem o processo de reconstrução tradicional.

O "Metabolic Models Reconstruction Using Genome-Scale Information" (*merlin*) é uma aplicação Java^{TM} gratuita, e fácil de utilizar, que automatiza as principais etapas de reconstrução de um modelo *MEG* para qualquer microrganismo. Apesar de já ter sido utilizado com sucesso em vários trabalhos, muitos plugins ainda estão a ser desenvolvidas para aprimorar os seus recursos e torná-lo mais acessível a qualquer utilizador. Neste trabalho, serão descritas em detalhe as novas ferramentas integradas no *merlin*, bem como a melhoria de outras funcionalidades presentes na plataforma. As melhorias gerais realizadas e a implementação das novas ferramentas permitem melhorar a experiência global do utilizador durante o processo de reconstrução de modelos *MEG* no *merlin*.

O principal recurso implementado neste trabalho é a integração da *BiGG Integration Tool* (*BIT*) no *merlin*. Este plugin permite a recolha dos dados metabólicos que integram os modelos presentes na base de dados BiGG Models e a sua associação ao genoma do organismo em estudo, por homologia, criando, se possível, a boolean rule para cada reação BiGG presente no modelo sob construção. Todo o processamento exigido para executar a *BIT* do *merlin* ocorre remotamente, para acelerar o processo. Em poucos minutos, os resultados são devolvidos pelo servidor e importados para o ambiente de trabalho do utilizador. A execução da ferramenta fora da máquina do utilizador traz também vantagens ao nível do armazenamento da informação, já que a estrutura de dados BiGG que sustenta toda a ferramenta está disponível remotamente. A implementação desta ferramenta fornece uma alternativa à obtenção de informação metabólica a partir da base de dados KEGG, única opção disponibilizada pelo *merlin* até ao momento. Para testar a ferramenta implementada, várias redes metabólicas à escala genómica rascunho foram geradas e analisadas.

Palavras-chave: *merlin*, modelos *MEG*, BiGG Models, *BiGG Integration Tool*

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

AIBENCH Artificial Intelligent workbench.

API Application Programming Interface.

BIGG Biochemical Genetic and Genomic knowledgebase.

BIOPAX Biological Pathway Exchange.

BIT BiGG Integration Tool.

BLAST Basic Local Alignment Search Tool.

BRENDA Braunschweig Enzyme Database.

COBRA Constraint-Based Reconstruction and Analysis.

COGS Clusters of Orthologous Groups.

CORECO Comparative ReConstruction.

CPU Central Process Unit.

DNA Deoxyribonucleic Acid.

E-VALUE Expected value.

E. COLI *Escherichia coli*.

EC Enzyme Commission.

EGGNOG Evolutionary Genealogy of Genes: Non-supervised Orthologous Groups.

FAME Flux Analysis and Modeling Environment.

FBA Flux Balance Analysis.

FVA Flux Variability Analysis.

GC4S GUI Components for Swing.

GO Gene Ontology.

GPR Gene-Protein-Reaction.

GSM Genome-Scale Metabolic.

GUI Graphical User Interface.

HMMER Hidden Markov Models.

HTTP Hypertext Transfer Protocol.

JAR Java ARchive.

JDBC Java Database Connectivity.

JPA Java Persistence API.

JSON JavaScript Object Notation.

KEGG Kyoto Encyclopedia of Genes and Genomes.

KO KEGG Orthology.

MATLAB Matrix Laboratory.

ME Metabolic Enginnering.

MEMOSYS Metabolic Model research and development System.

MERLIN Metabolic Models Reconstruction Using Genome-Scale Information.

MIRIAM Minimal Information Required In the Annotation of Models.

MOMA Minimization of Metabolic Adjustment.

MRNA Messenger RNA.

MS Mass Spectrometry.

MVC Model-View-Controller.

NCBI National Center for Biotechnology Information.

NMR Nuclear Magnetic Resonance.

ORF Open Reading Frames.

ORM Object/Relational Mapping.

OS Operating System.

PFBA Parsimonious Flux Balance Analysis.

PGDB Pathway/Genome Database.

RAM Random Access Memory.

RAST Rapid Annotation using Subsytems Technology.

RAVEN Reconstruction, Analysis and Visualization of Metabolic Networks.

RDBMS Relational Database Management Systems.

REFSEQ Reference Sequences.

REGEX Regular Expression.

RNA Ribonucleic Acid.

RNA-SEQ RNA Sequencing.

ROOM Regulatory On/Off Minimization.

RRNA Ribosomal RNA.

SB Systems Biology.

SBML Systems Biology Markup Language.

SMART Simple Modular Architecture Research Tool.

SVG Scalable Vector Graphics.

SVM Support Vector Machine.

TC Transporter Commission.

TRANSYT Transport Systems Tracker.

TRIAGE Transport Proteins Annotation and Reactions Generation.

TRNA Transfer RNA.

UNIPROT Universal Protein Resource.

VMS Virtual Machines.

1

INTRODUCTION

1.1 CONTEXT AND MOTIVATION

Systems Biology (SB) is an interdisciplinary field that provides a novel approach, aimed at studying not only of the components that make up the biological system individually, but also the interactions between them, by using mathematical and computational modelling. Models provided by SB approaches allow to study complex systems as a whole and have been used to understand how the interactions between organism's components give rise to the function and behaviour of that system (Nielsen and Jewett (2008); Palsson (2006)). One application of SB is Metabolic Engineering (ME) that allowed the manipulation of the organisms as a whole in order to obtain desirable phenotypes (Vemuri and Aristidou (2005); Stephanopoulos et al. (1998)). In the past few years, advances in SB and ME areas have enabled the reconstructions of genome-scale metabolic (*GSM*) models. A *GSM* model is a mathematical representation that intends to describe the entire metabolism of an organism using genomic information (Terzer et al. (2009); Zhang and Hua (2016)). The reconstruction process of a *GSM* model involves reconstructing a metabolic network - *GSM* network, which is obtained from the annotated sequence of the genome and information collected from various biological databases and literature (Dias et al. (2015)). The functional annotation of the target-organism genome, the identification of the associated reactions, the identification of the location of these reactions and the determination of an equation that defines the biomass composition as well as the associated energy requirements are some of the steps involved in the reconstruction process of the *GSM* network. The *GSM* network can then be integrated into a mathematical model and used to predict, in silico, the phenotype of microorganisms under different environmental and genetic conditions (Dias et al. (2015, 2018); Thiele and Palsson (2010)). Thus, the reconstruction of a *GSM* model is not an easy task, requiring collecting information of several *omics*'s fields and several computational tools and laboratory methods, thus being a slow and time-consuming process (Dias et al. (2018)). Several software, which support various stages, have been developed to accelerate the reconstruction process.

The Metabolic Models Reconstruction Using Genome-Scale Information (*merlin*) is a free user-friendly Java^{TM} application that automates the main stages of the reconstruction of *GSM* models, including the functional genomic annotation of the genome for any organism (Dias et al. (2015)). Although this software is currently one of the most user-friendly to perform the reconstruction process and has been used with success in several works, it is still being improved as various tools are being developed and several new features included (Dias et al. (2018)). Hence, in this work, existing tools will be improved and new tools implemented, to extend its capabilities and to improve its features, making it more accessible to non-bioinformaticians.

1.2 OBJECTIVES OF THE THESIS

The objective of this work is to improve *merlin*'s performance by implementing new user requested features to expand its functionalities, making it increasingly more natural to use by information technologies laymen. One of the primary purposes of this work is to allow the import of metabolic data from the BiGG Models database during the assembly phase of the *GSM* model.

In detail, this work consists in performing the following approaches:

- Miscellaneous improvements on *merlin* platform, such as:

  Creation of a plugin to manage the configuration files;
  Implementation of a plugin to allow creating a backup and restore of a workspace.
  Configure e-mail setter;
  Logger configuration;
  ...

- Creation of a tool capable of retrieving BiGG metabolic data.

  Access BiGG Models to collect all metabolic data;
  Association of BiGG data to the query genome by homology;
  Development of a web-server to manage user's submissions;
  Integration of the tool as a *merlin* plugin.

1.3 STRUCTURE OF THE DOCUMENT

This document is organized as follows:

Chapter 2: State of the Art

Chapter 2 contains an overview of the main points of the thesis. Here, a brief description of Systems Biology and Metabolic Engineering evolution, as well as the tasks and methodologies associated to the reconstruction of *GSM* models are presented. At the end of the chapter, available *GSM* model reconstruction computational tools and existing biological databases are introduced. Likewise, application frameworks and a software library used to develop computational tools are also described.

Chapter 3: Methods and Tools

In the third chapter, all the resources used to implement overall improvements and new features in *merlin* platform are presented. Moreover, the methodology applied for the development of the *BiGG Integration Tool* is described in this chapter.

Chapter 4: Software Development

This chapter contains a detailed description of all the work performed in *merlin*, new tools and improvements implemented. Simultaneously, the outcome from each implemented task is also presented.

Chapter 5: Results and Discussion

In chapter 5, an analysis of the results obtained from the *BiGG Integration Tool* validation as well as some case studies carried out to test the new tool integrated in *merlin* are presented.

Chapter 6: Conclusion

The last chapter summarizes the conclusions of the developed work, and the limitations to be overcome in future work.

2

STATE OF THE ART

2.1 SYSTEMS BIOLOGY

Since the dawn of the genomic era, in which genome sequences became available, understanding and characterizing all the components that compose a living organism has been, for many years, the focus of biology (Palsson (2006); Kitano (2001)). However, the identification of the components in the organism's system only provides an individual characterization of the components. Most living organisms are very complex systems, comprising a set of several simple elements that interact between them to attain a particular behaviour (Kitano (2002)). Thus, understanding how all these components function as a whole system is fundamental to know the structure and dynamics of a system (Kitano (2001)). Systems Biology (SB), an emergent science that combines several areas, namely biology, computational modelling and mathematics was the answer to this challenge. Unlike components biology, that only provides a catalogue of individual components of an organism, SB uses mathematical models to examine how biological components interact and form networks, and how the properties of the network are associated to the cell functions, to understand and predict the biological system behaviour under specific conditions (Nielsen and Jewett (2008); Palsson (2006)). The enhancement of high-throughput biological technology contributed to the expansion of SB as a new field of study, providing a detailed understanding of the biological components of complex systems on a large scale. In addition to a study focused on the systems' components, SB allowed biologists to understand the networks of biochemical reactions that result from the chemical interactions that link these components and support various cellular functions (Palsson (2006)). Thus, SB's main objective is to allow a quantitative description of the main characteristics of whole biological systems, through mathematical models that can be used in the simulation of biological behaviour under different conditions from those already characterized, or for the discovery of knowledge that complements hidden information of large amounts of experimental data (Nielsen and Jewett (2008); Kitano (2001)). Models provided by SB approaches have proved to be useful in several fields, such as medical problems and industrial biotechnology as these approaches allow to identify genetic targets able to improve the production of metabolites of interest, and to increase the robustness of

biological processes, which reduce the amount of resources and time to market (Otero and Nielsen (2010)).

2.1.1 *Omics data*

SB is an area focused on both the study of the interactions that occur between the components that make up the biological system and the components themselves. That information is integrated into network models to facilitate comprehension, and sophisticated computational analyses are performed to generate predictive hypotheses of the behavior of living systems (Pray et al. (2011)). As such, SB uses advanced computational methods to integrate large amounts of data, with different levels of complexity, the so-called *omics*, such as genomics, transcriptomics, proteomics, and metabolomics (Gligorijević and Pržulj (2015)). *Omics* techniques measure characteristics of large families of various types of molecules that make up the cells of an organism to explore relationships, namely genes, proteins, and metabolites, providing a systems-level understanding of the cell (Gligorijević and Pržulj (2015); Petranovic and Vemuri (2009)). A brief description of the main *omics*' fields is presented next.

Genomics

The most mature of the different *omics* fields is genomics, an interdisciplinary field of SB dedicated to mapping, sequencing, and analyzing organisms' genomes, to understand its structure and function. Therefore, it is common to divide genome analysis into structural genomics, which aims to identify and characterize each gene in the genome, comparative genomics, that compares genetic traits of different organisms using computational tools, and functional genomics, in which genome-wide functional modules are identified (Bhatnagar et al. (2008)).

Transcriptomics

Transcriptomics can be defined as the field dedicated to the study of the transcriptome. The transcriptome is the complete set of Ribonucleic acid (RNA) transcripts and the template for protein synthesis in the translation process (Horgan and Kenny (2011)). Determining a transcriptome involves using technologies, such as Deoxyribonucleic acid (DNA) microarrays and RNA sequencing (RNA-Seq), that allows, among many other things, the measurement of messenger RNA (mRNA) molecules expression levels of every Open Reading Frames (ORF) in the genome under a given condition (Petranovic and Vemuri (2009); Kitano (2001)).

Proteomics

Proteomics is the scientific area that deals with the study of the proteome, which is the set of all proteins and variant proteins expressed by the genome present in an organism, to understand the functional relevance of proteins (Horgan and Kenny (2011)). Proteome analyses provides a better understanding of the physiological state of an organism, which may be useful in the development of therapeutic and diagnostic techniques or even in the identification of potential biomarkers (Cristea et al. (2004); Mischak et al. (2007)). Among the analysis techniques used in proteomics are two-dimensional gel electrophoresis (2-DE) and mass spectrometry (MS) (Cristea et al. (2004)).

Metabolomics

Metabolomics focuses on the study of global metabolite profiles, known as the metabolome, in a cell, tissue, or organism, under a given set of conditions (Horgan and Kenny (2011)). Metabolite profiling is essential in SB, since the availability of metabolites determines the connectivity of the networks. The metabolite profile is typically determined by MS and nuclear magnetic resonance (NMR) (Bhatnagar et al. (2008)).

2.2 METABOLIC ENGINEERING

Metabolic engineering (ME) can be defined as the reconstruction, redirection, and manipulation of cellular metabolism by the introduction of genetic modifications that leads to desirable phenotypes (Stephanopoulos et al. (1998); Vemuri and Aristidou (2005)). Before the genomic era, these modifications were performed by traditional random mutagenesis, followed by screening to design a phenotype of interest, or later, performing modifications in genes directly associated with the product of interest. However, both strategies are time and labor-intensive and often failed the objective (Kim et al. (2008); Dai and Nielsen (2015)). ME is different from other cellular engineering approaches because it focuses on the complete networks of biochemical reactions, pathway fluxes and its control. The idea of a metabolic network is fundamental as it demonstrates that ME considers the whole set instead of isolated reactions (Stephanopoulos (1999); Nielsen and Arnold (2005)). ME aims at optimizing metabolic processes to improve, for instance, the production of pharmaceuticals, industrial compounds, or other desirable products. These phenotypes are achieved by manipulating the biochemical pathways fluxes, that determine the cell physiology (Dai and Nielsen (2015); Stephanopoulos et al. (1998)). As shown in Maia et al. (2016), ME uses strategies such as:

1. Gene deletions - elimination of genes to suppress a specific metabolic function;
2. Heterologous insertions - addition of genes or pathways to increase the organism's metabolic ability;

3. Gene over/under-expression - which has a considerable importance in the ME community and can be useful to avoid the deletion of critical genes;
4. Cofactor specificity swaps - modulate the cofactor binding specificities to cope with the scarcity of some necessary cofactors;
5. Manipulation of environmental conditions.

Traditional ME method involve manipulating the metabolic pathways near the desired product (Stephanopoulos et al. (1998)).

Besides biotechnology, ME also has relevant applications in other fields, such as medicine, allowing the analysis of the metabolism of whole organs and tissues, as well as the identification of targets for disease control (Stephanopoulos (1999)). ME will be able to expand to new areas of application, considering the incorporation of new computational and analysis tools (Nielsen and Arnold (2005)).

2.3 Genome-scale metabolic models

The first reconstructions of metabolic models were performed two decades ago, mostly from available literature, in the absence of genome-scale information (Pfau et al. (2016)). In 1999, the publication of the first genome-scale metabolic (*GSM*) reconstruction for *Haemophilus influenza*, the first free-living organism to have its entire genome sequenced, initiated a new era for SB (Edwards and Palsson (1999); Fleischmann et al. (1995)). Afterward, the improvement of sequencing technologies and the development of bioinformatics tools led to an exponential increase in the amount of information available in biological databases (Pfau et al. (2016)). The availability of large amounts of data regarding metabolism and the existence of several online databases as a source of whole-genome sequences and well-studied biochemical reactions, allows reconstructing *GSM* networks of several organisms, even for the less characterized in the literature. *GSM* networks are mathematical representations of metabolic networks, in which metabolites are linked by biological reactions associated with enzymes that are encoded in the genome of the target organism (Terzer et al. (2009); Zhang and Hua (2016)). These networks allow performing topological analysis of the organisms' metabolism. Besides metabolic network data, a *GSM* model includes detailed information regarding the biomass composition and energetic requirements. Thus, reconstructing a *GSM* model is a laborious process, involving collecting data from several *omics*' fields and the use of computational tools and laboratory methods (Dias et al. (2015)). These models can be used to simulate *in silico*, using software such as *OptFlux* (Rocha et al. (2010)), the microorganisms' behaviour under different genetics and environmental conditions and has been used both in industrial biotechnology and in human health research (Zhang and Hua (2016)).

The procedure to obtain a high-quality *GSM* reconstruction has been described in a detailed protocol, involving more than 100 steps, that can be simplified in four major stages: genome annotation, assembling the genome-wide metabolic network, conversion of the network to a mathematical model (stoichiometric model) and metabolic model validation (Dias et al. (2018); Thiele and Palsson (2010)).

The reconstruction of a *GSM* model begins with the thorough functional annotation of the target organism's genome (Dias et al. (2015)). If the organism's genome was previously annotated, it can be obtained by accessing public repositories of genomic data in which a manual curation may have been performed. Examples of these biological repositories are the National Center for Biotechnology Information (NCBI) (Sayers et al. (2020)), Kyoto Encyclopedia of Genes and Genomes (KEGG) (Kanehisa and Goto (2000)) or the SEED (Overbeek et al. (2005)), which maintains genome annotations provided by the Rapid Annotation using Subsystems Technology (RAST) (Aziz et al. (2008)), an automated tool that performs high-quality annotations. Alternatively, the annotation can be performed with specialized tools. In this phase, it is essential to collect relevant information about each gene fundamental for the reconstruction, namely the gene and product names, the Enzyme Commission (EC) numbers (Webb et al. (1992)) and the Transporter Classification (TC) numbers (Saier (2000)). Subunits of protein complexes should also be identified, as an enzyme may be encoded by more than one gene (Dias et al. (2015)). Therefore, it is possible to identify the metabolic genes, i.e., genes encoding enzymes or transport systems. This stage is decisive for the development of a high-quality *GSM* model, since it is assumed that the genome annotation is correct. Nevertheless, notice that these annotations may be outdated or information about the genes may not be assigned. Thus, it is recommended to perform an annotation of the previously annotated genome in the first phase, the so-called re-annotation process (Dias et al. (2015, 2012)).

Once the genome annotation is complete, the next step is to collect metabolic information to form the *GSM* model network. At this stage, all the different metabolic reactions that define the organism are identified and retrieved. Initially, both enzymatic and transport reactions are obtained by searching biochemical reaction databases, such as KEGG, Braunschweig Enzyme Database (BRENDA) (Schomburg et al. (2002)), MetaCyc (Caspi et al. (2018)), MetRxn (Kumar et al. (2012)) and Biochemical Genetic and Genomic knowledgebase (BiGG) (King et al. (2016)), with the EC number, TC number or other identifier assigned during the genome annotation stage. Then, the associations between metabolic genes, proteins and reactions (the gene-protein-reaction (GPR) rules), which are fundamental for a correct prediction of phenotypes, are established in the initial draft network (Dias et al. (2015); Thiele and Palsson (2010)). All reactions classified as spontaneous, as well as other reactions known to exist in a given organism, should be added to complement the metabolic network. After collecting the set of reactions, the next stage is to identify where the reactions take

place. Compartmentalization is important for the development of *GSM* models because it determines in which organelles enzymes operate. In prokaryotes, compartments are limited to the cytosol, periplasmic space, and extracellular space. Whereas in eukaryotes, the reactions can take place in several different compartments (Golgi apparatus, lysosome, mitochondria, endoplasmic reticulum, among several others) (Dias et al. (2015)). This information may be found in the Universal Protein Resource (UniProt) (Apweiler et al. (2004)) or literature; however, since compartmentalization can be hard, several bioinformatics tools such as the ones from the PSort family (Yu et al. (2010); Horton et al. (2007)) and LocTree3 (Nair and Rost (2005)) were developed to facilitate the process. Although very useful, automated processes are still prone to reconstruct incomplete or incorrect models (Francke et al. (2005)). Therefore, at this stage, the compartmentalized *GSM* network is just a draft, with several inaccuracies. Thus, the manual curation and refinement of the draft *GSM* network are requirements to improve the reliability of the network. Manual curation consists of reviewing all reactions that have been added to the *GSM* network by checking organism-specific databases, expert researchers, and available literature. At this stage, problems such as the existence of gaps in a pathway, the proteins and functions identifiers inconsistencies, and the assignment of ambiguous identifiers to reactions should be corrected (Dias et al. (2015)). Also, the characteristic reactions of the organism may not be available in the data sources consulted. It is essential to use the information available in databases such as BRENDA to check the stoichiometry of each reaction, otherwise the internal flux distribution of the *GSM* network can be affected, impairing simulations' results (Dias et al. (2018)). Reactions' directionality and reversibility are other useful features to validate the network because all the reactions in KEGG are set as reversible by default. The directionality and reversibility of reactions can be determined by calculating the Gibbs free energy of formation ($\Delta_f G_{10}$) and reaction ($\Delta_r G_{10}$) or by accessing published curated models of the target-organism or closely related organisms that may be found in databases such as BiGG and Model SEED (Dias et al. (2015); Jankowski et al. (2008); Fleming et al. (2009)).

Before converting the network into a mathematical model (third step of the *GSM* model reconstruction process), it is necessary to add a reaction that defines the composition of the biomass to the reactions set. This reaction represents the rate of growth of the organism, as well as the energy requirements associated with the growth (Thiele and Palsson (2010); Dias et al. (2015)). Applying a steady-state approximation, in which it is assumed that production and degradation rates of all metabolites are equal, allows obtaining a system of linear equations represented by the following equation:

$$S.v = 0 \tag{1}$$

where S is the stoichiometric matrix and v the fluxes vector. Each column of the matrix represents the reactions, and the rows represent the metabolites. The stoichiometric matrix

includes further constraints about the reactions and metabolites. Most constraints are associated with the reversibility and directionality of the reactions and the concentration of some metabolites (Dias et al. (2015); Terzer et al. (2009)). This mathematical representation should be stored in a computational-friendly format, such as the Systems Biology Markup Language (SBML) (Hucka et al. (2003)), the Biological Pathway Exchange (BioPax) (Demir et al. (2010)), the Pathway/Genome Database format (Caspi et al. (2007)) or an Excel file. These formats allow to import the model into different software that use *GSM* models to perform simulations or optimization.

Finally, the *GSM* model should be validated by comparing the simulations of the target organism behaviour against experimental data to evaluate the accuracy of the model (Dias et al. (2018)). If these predictions are not in agreement with the experimental data, the previous steps should be carefully reviewed (Thiele and Palsson (2010)). Otherwise, the *GSM* model can be used to simulate the organism phenotype in different conditions. There are several specialized algorithms in the analysis of metabolic models. One of the most used is the Flux Balance Analysis (FBA) (Varma and Palsson (1994)), a constraint-based method which allows to perform a variety of physiological analysis useful to validate the model. FBA can be used to predict the reactions' fluxes through the metabolic network, the growth rates, and to calculate theoretical yields amongst others (Hamilton and Reed (2014)). Another flux-based analysis technique is the Method of Minimization of Metabolic Adjustment (MOMA) (Segre et al. (2002)). MOMA is based in the same stoichiometric constraints as FBA, but the optimal growth flux for mutants is relaxed for gene knockouts (Klanchui et al. (2012); Edwards et al. (2002)). Moreover, The Regulatory On/Off Minimization (ROOM) (Shlomi et al. (2005)) can also be used for predicting the steady state of metabolic networks with gene knockouts. Another algorithm used in model simulations is the Parsimonious Flux Balance Analysis (pFBA) (Lewis et al. (2010)), an improved approach that arises to overcome a relevant limitation of the FBA. For a unique optimal value of the objective function, FBA provides a single flux distribution; however, a large number of flux distributions that lead to the optimal value may exist, and the pFBA minimizes the sum of the flux values to select a specific one. Finally, the Flux Variability Analysis (FVA) (Lewis et al. (2010)) aims at characterizing the space of possible variation of specific fluxes, given a set of constraints and is used to evaluate the robustness of a flux distribution.

Simulation-specialized tools such as Optflux and Constraint-Based Reconstruction and Analysis (COBRA) (Schellenberger et al. (2011)) use the analysis' algorithms presented before to study the phenotype of microorganisms, under different environmental and genetic condition (Rocha et al. (2010)).

2.4 GSM models' reconstruction tools

The previously described process for reconstructing *GSM* models can take from weeks to more than a year (Dias et al. (2018)). Since this is such a time-consuming process, the development of software that automates most stages of the reconstruction has become a requirement for new ME approaches. Hence, software like *merlin*, Model SEED, RAVEN Toolbox or Pathway tools, explicitly developed to assist in the reconstruction of *GSM* model's, are increasingly available. In this section, the main features of some of these software packages are presented.

2.4.1 *merlin*

Metabolic Models Reconstruction Using Genome-Scale Information (*merlin*) is a user-friendly software, developed at the University of Minho to automate the main stages of the reconstruction of *GSM* models (Dias et al. (2015)). Published in 2015, this application, built on top of the Artificial Intelligent workbench (AIBench) (Glez-Peña et al. (2010)) framework, is fully implemented in Java^{TM}, a popular platform-independent programming language, and performs the main stages of *GSM* model's reconstruction for any organism (Dias et al. (2015)).

Currently, *merlin* has two main models: the first module includes tools that help in the genome functional annotation, curation and the reconstruction of the draft model. The second module is oriented to the *GSM* model assembling, providing several operations that help in the reconstruction of the *GSM* model (Dias et al. (2018)).

The first module of *merlin* consists in three main sections: enzymes annotation, transporters annotation and compartmentalization. In the semi-automatic enzymatic (re-)annotation, *merlin* allows users to use Basic Local Alignment Search Tool (BLAST) (Altschul et al. (1990)), or Hidden Markov Models (HMMER) (Eddy (1998), to perform sequence alignments in the genome by accessing different databases (NCBI and UniProt). *merlin* annotates each gene, considering the frequency and taxonomy of the homologous genes' annotation (Dias et al. (2015)). For the transporter's annotation and their reactions, *merlin* used to offer the Transport Proteins Annotation and Reactions Generation (TRIAGE) tool, a unique tool, developed specifically for this purpose, which automatically performs the annotation of transporters (Dias et al. (2015)). TRIAGE allowed to identify, classify and annotate membrane transporters, and automatically generates the respective transport reactions, which could be directly integrated with *GSM* models (Dias et al. (2016)). Recently, *merlin* stopped using this tool. In its latest version (v4.0.2), *merlin* offers TranSyT (Transport Systems Tracker), a standalone software for the identification of transport systems, developed to overcome some limitations of TRIAGE. Regarding the compartmentation of the model, *merlin* uses the

LocTree3, PSORTb 3.0 (Yu et al. (2010)) or WoLF-PSORT (Horton et al. (2007) algorithms to predict, if possible, the compartments for all genes/proteins (Dias et al. (2018)). *merlin* also includes several tools for the curation of the genome annotation and the draft model, essential in the reconstruction of an accurate model (Dias et al. (2015, 2018)).

The second module of *merlin* offers operations to assemble the metabolic model with all reactions, by combining the information collected in enzymes annotation stage with KEGG's metabolic data (Dias et al. (2018)). The annotation of transport proteins and transport reactions, automatically generated by TranSyT, can also be integrated into the draft model. For GPR associations, *merlin* provides an operation that creates them automatically by accessing information available in the KEGG BRITE to build the network. The e-biomass reaction is also included in the model (Dias et al. (2018)).

All information retrieved during the process is stored in an internal database, shared by the two modules, which can be accessed through *merlin*'s interface (Dias et al. (2018)). Finally, the *GSM* model available in *merlin*'s internal database can be exported to the SBML format, with Minimal Information Required In the Annotation of Models (MIRIAM) annotations (Le Novère et al. (2005)), to be validated and used in other software (Dias et al. (2015)).

2.4.2 *RAST*

The increasing numbers of prokaryotic genome sequences available in several databases led to the development of bioinformatics tools to rapidly annotate genomes. To face this problem, the RAST server (Rapid Annotations using Subsystems Technology), a fully automated service for annotating microbial genomes emerged in 2008 (Aziz et al. (2008)). Built upon the framework provided by the SEED technology, where the annotated genomes are maintained, RAST has become one of the most popular tools to obtain complete, consistent, and curated annotations in a short time (Brettin et al. (2015)). This tool also offers the possibility of identifying protein-encoding, ribosomal RNA (rRNA) and transfer RNA (tRNA) genes, assigning functions to the genes and predicting which subsystems are represented in the genome, using this information to reconstruct the metabolic network (Aziz et al. (2008)).

Users must register to access the framework, where a genome can be uploaded as a set of contigs in FASTA format, and when the annotation is complete users are notified (Aziz et al. (2008); Overbeek et al. (2014)). RAST includes an interface that allows registered users to change the annotation and add or delete genes calls, before retrieving them. The RAST server also includes a genome viewer (SEED viewer) to allow a detailed comparison of the annotated genome with public genomes or other genomes already submitted in RAST environment (Overbeek et al. (2014)).

2.4.3 *Model SEED*

Model SEED is a freely available web-based interface released in 2010, built upon the genome annotation provided by the SEED framework (Overbeek et al. (2005)), that includes several tools to support the reconstruction, exploration, comparison and analysis of genome-scale metabolic models (Henry et al. (2010)). In a first stage, Model SEED uses the RAST server to perform the annotation of the assembled genome sequence, allowing obtain a high-quality genome annotation to generate draft genome-scale metabolic models (Henry et al. (2010)). Before assembling the *GSM* model , users should use the available tools to perform the manual curation, which is fundamental to obtain a final accurate draft model. Then, a metabolic model is built by combining enzyme and transport reactions, that make up an organism's metabolism, the spontaneous reactions based on pathway completeness, the detailed GPR associations, and the biomass reaction to represent cellular growth, which is also automatically generated by the Model SEED tools (Henry et al. (2010)). Moreover, Model SEED supports exporting the *GSM* model in SBML format (Hamilton and Reed (2014)).

Contrary to others *GSM* model's reconstruction software, that use biochemical reaction databases to access the metabolic information to develop the *GSM* model, such as KEGG and BiGG, Model SEED contains its internal database (Hamilton and Reed (2014)). Another important aspect is that Model SEED is focused on prokaryotic organisms (especially bacterial and archaeal reconstructions) and so does not support reaction compartmentalization (Hamilton and Reed (2014)).

2.4.4 *RAVEN Toolbox*

Another software that aims at decreasing the time needed for reconstructing high-quality *GSM* models is the RAVEN Toolbox (Reconstruction, Analysis and Visualization of Metabolic Networks), published in 2013 (Agren et al. (2013)). As its name suggests, it is a complete environment for the reconstruction, analysis, visualization and simulation of *GSM* models within Matrix Laboratory (MATLAB) (Agren et al. (2013)). The origin of the metabolic information used to develop the *GSM* model is the KEGG database (Agren et al. (2013)). This approach allows the user to input already published *GSM* models of closely related organisms and, through protein homology, identify the KEGG Orthology (KO) identifier that better matches each gene. Then, reactions associated to that KO are imported to the reconstruction together with the corresponding gene (Agren et al. (2013); Hamilton and Reed (2014)). In parallel, RAVEN Toolbox uses the KEGG database for automatic identification of reactions that may be missing or incorrect in the model template (Caspeta et al. (2012)). Because RAVEN Toolbox does not validate reaction stoichiometry, users are encouraged

to balance them manually. However, to facilitate the process, RAVEN Toolbox offers an operation to identify unbalanced reactions. RAVEN Toolbox supports both prokaryotic and eukaryotic reconstructions, using WoLF PSORT to predict reaction compartmentalization (Hamilton and Reed (2014)). Regarding GPR associations, RAVEN Toolbox generate lists of genes associated with each reaction allowing users to create them themselves. This toolkit also does not automatically generate the biomass equation, requiring manual user interaction (Hamilton and Reed (2014)). Users' intervention is also required to perform the transporters annotation and to insert the spontaneous reactions, since RAVEN Toolbox does not include automatic tools for this purpose. Finally, this software allows to import and export models in both SBML and Excel formats (Agren et al. (2013)).

2.4.5 *Pathway Tools*

Pathway Tools is a bioinformatic software, published in 2002, that combines several genome, metabolic and regulatory informatics tools to create and manage a type of model-organism database called Pathway/Genome Database (PGDB) (Karp et al. (2002)). A PGDB gathers the evolving knowledge about an organism's genes, proteins, metabolic network and regulatory network. Pathway Tools consists of four main modules:

- PathoLogic: oriented to the creation of the PGDB from the annotated genome of an organism;
- Pathway/Genome Navigator: allows querying, visualization, and analysis of PGDBs;
- Pathway/Genome Editors: let users refine and update the contents of a PGDB;
- Pathway Tools ontology: provides resources for high-quality modelling of biological data within a PGDB (Karp et al. (2002, 2010, 2016)).

Pathway Tools does not perform enzyme annotation, so the upload of an organism's annotated genomic sequence, in the form of a Genbank file, is required, and using data from the manually curated MetaCyc database it is possible to infer about organism metabolic pathway complement (Hamilton and Reed (2014)). An advantage of using a manually curated database is the reliable annotations, so users do not need the time-consuming manual curation. Resembling Model SEED, this software does not support reactions compartmentalization, as it only performs prokaryotic reconstructions and the GPR associations are automatically generated. The reaction representing biomass equation should be inserted manually according to the users' specifications (Hamilton and Reed (2014)). Pathway Tools also provides visual analyzing to ease the network reconstructions evaluation, automatically generating organism-specific metabolic diagrams and also presents capabilities for perform-

ing model evaluation simulations (Karp et al. (2002, 2016)). The model could be exported to SBML, BioPax or PGBD format.

2.4.6 *Other tools*

Besides the tools previously presented in more detail, several other software are available to expedite the reconstruction of *GSM* models, an emerging area in SB. A general overview for the software FAME, SuBliMinal Toolbox, MEMOSys, CoReCo and MicrobesFlux is presented below.

FAME

Flux Analysis and Modeling Environment (FAME) is a browser-based graphical interface tool that offers a set of tasks to create, edit, run, analyze and visualize stoichiometric models (Boele et al. (2012)).

SuBliMinal Toolbox

The SuBliMinal Toolbox offers different modules with several independent tools that facilitate the process of reconstruction of *GSM* models. This toolbox uses data from KEGG and MetaCyc and is able to generate draft reconstructions, determine metabolite protonation state, mass and charge balancing reactions, predict reaction compartmentalization, generate GPR associations, perform transporters annotation and add the biomass equation (Swainston et al. (2011)).

MEMOSys

MEtabolic MOdel research and development System (MEMOSys) is a web-based system that offers an intuitive user interface to assist the management, storage and development of metabolic models (Pabinger et al. (2011).

CoReCo

Comparative ReConstruction (CoReCo) is a software that allows modelling metabolism of several related microbial species in parallel (Pitkänen et al. (2014)).

MicrobesFlux

MicrobesFlux is a semi-automatic web-based platform explicitly developed to draft and reconstruct metabolic models based on microorganism's genome annotation in KEGG database. MicrobesFlux is able to provide multiple tools to assist in the development of the draft model and also offers mathematical approaches such as FBA to perform simulations (Feng et al. (2012)).

2.5 BIOLOGICAL DATABASES

2.5.1 *BiGG Models*

Any living organism requires energy to maintain life (grow, reproduce, maintain its structure). Hence, metabolism is the set of reactions in an organism to transform and use the energy obtained. Understanding the metabolism of an organism is fundamental, as deviations in metabolism may cause several diseases (Li et al. (2012)). Thus, metabolic models were developed to simulate, predict and understand cellular behaviour of microorganisms, subject to different conditions. As shown before, several software tools have been developed over the last few years, to accelerate the reconstruction of high-quality metabolic models, which caused an increased number of available models. The first Biochemically, Genetically and Genomically (BiGG) knowledge base that gathers curated and high-quality reconstructions of genomic and bibliomic data in a single database was published in 2010 (Schellenberger et al. (2010)). BiGG provided two main features: the BiGG browser that allowed obtaining useful information of model contents, such as metabolic reactions, metabolites, genes, proteins and literature citations, and the BiGG exporter that allowed exporting whole reconstructions in the SBML standard format (Schellenberger et al. (2010)). BiGG provided curated pathway maps of several organisms, rendered with Scalable Vector Graphics (SVG) (Ferraiolo et al. (2000); Schellenberger et al. (2010)).

In 2015, the BiGG knowledge base was extended to the redesigned BiGG Models, a platform for integrating, standardizing and sharing *GSM* models, composed of a relational database, a web application programming interface (API) and a website (King et al. (2016)). BiGG Models contains more than 75 high-quality curated models with the gene identifiers linked to 71 NCBI Reference Sequences (RefSeq) genome annotations (Pruitt et al. (2007)), and metabolites linked to many external databases (KEGG, MetaCyc and more). BiGG Models uses a set of unique identifiers (BiGG ID's) from BiGG (2010), though having consistency failures. To address this problem, BiGG Models provides a single source of correct BiGG identifiers, with a simple specification, providing standardized identifiers for metabolites, reactions and genes to be used by other applications (King et al. (2016)). As mentioned above, BiGG Models is composed of a user-friendly website for browsing and searching models as well as the models' content (reactions, metabolites or genes) in the knowledge base. Besides an overview, BiGG Models allows exporting the model in SBML, JavaScript Object Notation (JSON) (Bray (2014)) or MATLAB MAT standard format (King et al. (2016)). Reactions pages provide information such as the stoichiometry of the reaction, reaction bounds and lists with models in which the reaction is present. Finally, the metabolites page shows the molecular formula for the metabolite, and the genes page presents the position of the gene in the chromosome. All pages provide links to external

databases for additional information (King et al. (2016)). The website also provides an advanced search to search for metabolites and reactions by entering an identifier from an external database. Furthermore, interactive pathway maps are included in BiGG Models, powered by Escher (King et al. (2015)), a web-based tool for building, viewing, and sharing visualizations of biological pathways to contextualize data about metabolism. BiGG Models are compatible with the COBRA Toolbox, a MATLAB package containing an ensemble of computational procedures designed to simulate, analyze and predict metabolic capabilities of organisms (Schellenberger et al. (2011); King et al. (2016)). Another important feature of BiGG Models is its web API, which can be accessed from any programming language that supports Hypertext Transfer Protocol (HTTP) (Fielding et al. (1997)) requests. The BiGG Models website retrieves data through the same API; thus, the knowledge base content can be accessed programmatically using the web API. Therefore, BiGG Models can easily be used to build *GSM* models by accessing its data through the web API (King et al. (2016)).

2.5.2 *eggNOG*

Homology refers to the relationship between two distinct species sharing a common ancestry. Thus, two genes are homologous when their DNA sequence derives from a common origin (R McCune and C Schimenti (2012)). When two homologous genes originate from gene copies that diverge to two distinct species, an event known as speciation, the genes are designated orthologs. Hence, two orthologous genes share a common ancestor and have similar functions (Kristensen et al. (2011)). Paralogous genes are homologous originated by a gene duplication event and, unlike orthologous, are genes that hold new functions (Gevers et al. (2004)). The identification of orthologous and paralogous genes is extremely useful for the study of species evolution, as these allow the functional annotation of newly sequenced genes (Kristensen et al. (2011)). As the classification and determination of homologous genes is analytically hard and a computational challenge, several methods based on different approaches were developed (Koonin (2005)). Methods for inferring orthology relationships can be branched in two main groups, namely the graph-based and tree-based methods (Schreiber and Sonnhammer (2013)). Graph-based algorithms allow the analysis of more species at once and produce orthologous groups at multiple taxonomic levels. Tree-based approaches use tree topology to differentiate orthologous from paralogous genes, providing finer resolution but requiring more demanding computational resources (Huerta-Cepas et al. (2016)). The Clusters of Orthologous Groups (COGs) database was the first effort to provide orthologous protein sets from sequenced genomes, but due to requiring manual curation, it is not regularly updated and does not provide phylogenetic resolution (Tatusov et al. (2003); Jensen et al. (2007)). The eggNOG (evolutionary genealogy of genes: Non-supervised Orthologous Groups) database, available as a web interface, was developed in 2007. This

database uses a graph-based unsupervised clustering algorithm to provide a hierarchy of orthologous groups, each with a high-quality functional annotation from the three domains of life: Archaea, Bacteria and Eukaryota (Jensen et al. (2007); Huerta-Cepas et al. (2016)). The automatic functional annotation of the orthologous groups is one of the most important features of the eggNOG database. eggNOG collects data from several sources, including textual annotation available in genome databases, COG functional categories, the annotated Gene Ontology (GO) terms (Consortium (2015)), KEGG pathways (Kanehisa et al. (2014)) and protein domains from Simple Modular Architecture Research Tool (SMART) (Letunic et al. (2015)) and Pfam (Finn et al. (2014)) databases, to infer the functional categories for each orthologous group. A machine learning algorithm, Support Vector Machines (SVM), performs the individual assignment using the previously mentioned data as features (Jensen et al. (2007)). A recent publication of Huerta-Cepas et al. (2017) describes the implementation of the eggNOG-mapper, a novel tool for easily annotating large sets of proteins based on fast orthology mappings. With five published versions, eggNOG has proven to be one of the best public resources for orthology prediction and functional annotations of newly sequenced genes and very useful for ecological, evolutionary or medical-omics analysis (Huerta-Cepas et al. (2016, 2019)).

2.6 Basic Local Alignment Search Tool

The Basic Local Alignment Search Tool, commonly known as BLAST, is a research tool most frequently used to compare a nucleotide or protein sequence (called query sequence) with a sequence database or library (reference sequence) (Altschul et al. (1990)). Thus, BLAST is an efficient algorithm to align a query sequence to a large collection database in order to identify the exact identity of the sequence. Since it is a heuristic algorithm, it uses faster approaches to calculate the optimal alignment between the reference and the query sequences, comparing to the conventional approaches such as Smith-Waterman algorithm and Needleman-Wunsch algorithm, with comparable accuracy. BLAST performs local alignments. Therefore, BLAST works by breaking the query sequence in individual small chunks of short sequences. Each individual sequence is then compared against short sequences found in the database, until an almost identical match is obtained. After all short sequences from the query sequence are compared and extended maximally, the algorithm assembles the best alignment for each query.

The BLAST tool is one of the most appreciated tools in the bioinformatics community, not only because of its good performance, but also because there are many variations of BLAST searches, each with a specific purpose. Table 1 provides a brief overview of the commonly used BLAST variations: BLASTN, BLASTP, BLASTX, TBLASTN, and TBLASTX.

All programs compare protein sequences, except for BLASTN, which compares nucleotide sequences.

Table 1: **Summary of the most commonly used variants of BLAST.**

	Query Sequence	Database Sequence	Alignment Level	Common use
BLASTN	Nucleotide	Nucleotide	Nucleotide	Find identical nucleotide sequences
BLASTP	Protein	Protein	Protein	Find homologous proteins
BLASTX	Translated Nucleotide	Protein	Protein	Analyze new DNA to find genes and seek homologous proteins
TBLASTN	Protein	Translated Nucleotide	Protein	Search for genes in unannotated genome
TBLASTX	Translated Nucleotide	Translated Nucleotide	Protein	Find very distant relationships between nucleotide sequences.

2.7 JAVA APPLICATION FRAMEWORKS

2.7.1 *AIBench*

The Artificial Intelligent workBench (AIBench), released in 2010, is a Java^{TM} application framework that allows to accelerate the production of research applications based on input processing-output cycles (Glez-Peña et al. (2010)). The development of applications using this framework brings benefits to both developers and users, and has proven to be suitable for the development of several scientific software, such as *merlin* or OptFlux. Following the Model-View-Controller (MVC) architectural pattern, which divides an application into three main logical components (model, view and controller), AIBench applications manage three main concepts (operation, datatype and view) (Glez-Peña et al. (2010)). Thus, AIBench facilitates the connection, execution and integration of operations with well-defined object types (López-Fernández et al. (2011)).

This implementation is advantageous because it allows the combination of entirely new software components with existing ones (Glez-Peña et al. (2010)). AIBench is structured in several layers (Glez-Peña et al. (2010)). The Core layer contains two main built-in plugins: Core plugin and Workbench plugin. The Core plugin stores the operations, executes these when requested and saves the results of the operation. The Workbench plugin is responsible for implementing the graphical user interface (GUI), using Java Swing (set of basic elements of a GUI for Java), creates menus with all implemented operations and retrieves user parameters through input dialogues when an operation is performed (López-Fernández et al. (2011)). The Services layer includes the additional services and together with the

Core layer are the basis of all development, constituting the built-in code of the framework (López-Fernández et al. (2011)). The Plugin and Application layers are dependent on the application context and contain operations, datatypes, and views developed by AIBench users. The plugin has a configuration file, called plugin.xml, in which custom operations and views are declared, allowing the connection and dependency between plugins (Glez-Peña et al. (2010)).

2.7.2 *Hibernate*

For several years, the concept of how the access and storage of data in an application's relational database is performed, usually known as persistence, has been a debating point in the Java community (Bauer and King (2005)). With the popularization of Java as a language for the development of large-scale applications, it was soon realized that the use of JDBC (Java Database Connectivity) for Java database connectivity requires much effort from the developer (Bauer and King (2005)).

JDBC is a specific API for the Java language that brings together a set of classes, interfaces and methods that make it possible to query and update data from a relational database using SQL statements. Although JDBC fulfils its objective, other options can be more efficient depending on the type of application to be developed. The use of Hibernate is an alternative to JDBC.

Hibernate is an open-source framework that facilitates data persistence in Java. For this purpose, it brings as a difference an ORM (Object/Relational Mapping) solution that abstracts the programmer from many repetitive tasks, such as the coding of SQL queries (Bauer and King (2005)). As an ORM framework, the main feature of Hibernate is the automatic mapping of Java class objects to tables in the relational database, also offering query and data retrieval capabilities. In addition to its API, Hibernate implements the JPA (Java Persistence API) specification (Bauer and King (2005)). Thus, an application developed with Hibernate will free the programmer from almost all everyday tasks related to data persistence. Another great advantage of using Hibernate is its portability between multiple databases that use this technology (Bauer and King (2005)).

Recently, *merlin* has been entirely factored to integrate Hibernate. Therefore, the goal is for *merlin* to no longer be dependent on only H2 and MySQL databases (current internal databases available for data storage) and to be possible to use other databases without having to update the source code.

2.8 FLASK FRAMEWORK

Flask is a lightweight Python web framework widely used to simplify the development and execution of web applications. It depends only on the external libraries Jinja2 (Ronacher (2008)), responsible for rendering templates, and the Werkzeug, which performs all the web control. Its simple model allows to facilitate the most common tasks involved in the process of developing a web application. Despite being classified as a microframework, Flask supports extensions that allow access to a wide range of libraries, making the development environment highly customizable according to the complexity of the application (Aslam et al. (2015)).

2.9 JAVA SOFTWARE LIBRARIES

2.9.1 *GC4S*

With the increase in the availability of genomic data, the offer of reliable bioinformatics tools and software packages to support the specific needs of biological research is increasingly available. Although these tools and software are usually data-oriented and reliable, most fail an important requirement: not offering a user-friendly graphical interface, making its use difficult by information technologies non-specialists. Thus, an effective and efficient interface of a software, to interact with the user, is a key element for its success (Bolchini et al. (2009)).

Hence, frameworks and libraries for several programming languages were recently created to provide an easier development of new successful bioinformatics platforms with user-oriented features (Perez-Riverol et al. (2014)). For this purpose, the GUI Components for Swing (GC4S), a free, open-source library, which provides GUI components for Java Swing useful for programmers in the development of views for bioinformatics software was put forward in 2018. GC4S is a collection of new or improved versions of already existing components in Java Swing (López-Fernández et al. (2018)). With this library, the user can reuse a set of generic GUI components, useful for the development of GUI for scientific applications. GC4S was implemented in Java 8, and includes a main gc4s module containing the components library and other five dependent modules:

1. the *gc4s-genomebrowser* providing an interactive genome browser;
2. the *gc4s-heatmap* providing an interactive heat map visualization component;
3. the *gc4s-jsparklines-factory* containing classes to ease the creation of JSparkLines renderers (Barsnes et al. (2015));

4. the *gc4s-multiple-sequence-alignment-viewer* which provides a multiple sequence alignments viewer;

5. the *gc4s-statistics-tests-table* module providing a statistical tests table (López-Fernández et al. (2018)).

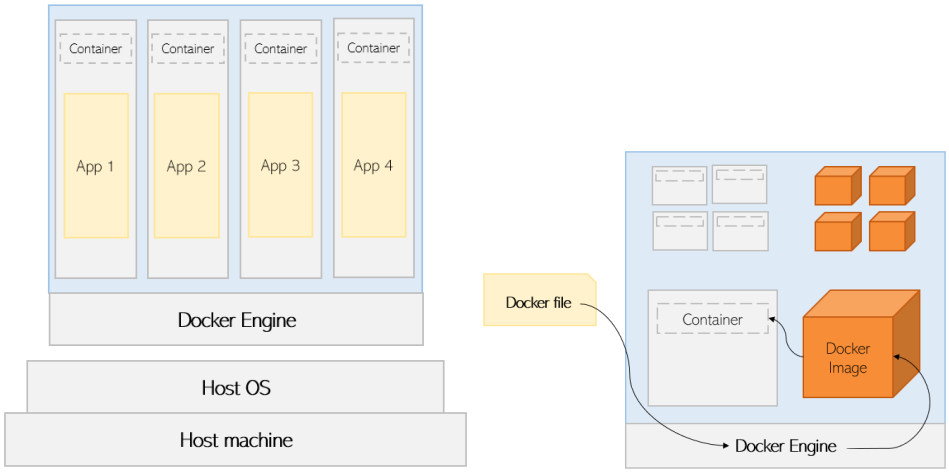
Another important aspect is that GC4S can be used independently or together with AIBench framework to accelerate the software development process.

2.10 DOCKER

Docker is a containerization software designed to facilitate the creation, deploy and execution of applications. Docker performs virtualization at the operating system (OS) level, which allows applications to run in an isolated environment to the host machine (Chung et al. (2016)). This isolation is achieved because Docker applications are developed in containers, which function as if they were a physical server (Figure 1a). In this way, containers are capable of running applications in the same way that they would run on the host machine.

Docker containers are often compared to "lightweight virtual machines". In fact, both technologies offer an isolation environment in which applications can run independently of the host OS, though differing in how they achieve that isolation. In the case of Dockers, the container runs on top of an OS kernel (Turnbull (2014)). The kernel is the part of the OS that mediates the access to the machine's physical resources (such as Central Process Unit (CPU) and Random Access Memory (RAM)). When the container runs, instead of directly accessing the kernel, it accesses through the Docker Engine, the layer responsible for building, executing and storing Docker containers, to access system resources (Chung et al. (2016)). Docker only exposes the kernel and not the host machine's operating system. On the other hand, in virtual machines (VMs) it is the hypervisor that creates and runs the VMs, isolating system resources, as well as entire working environments from the physical server to the virtual environment (Turnbull (2014); Chung et al. (2016)). Thus, the container is the "virtualization" of the application and not of the operating system as a whole (VMs). This lightweight form of encapsulation is what makes Docker special.

Creating a container in Docker involves creating a file called Docker file. It is a .txt file that contains all the instructions that the Docker Engine needs to build the Docker Image, a file which is used to start a container. The container is created based on this image and is ready to run in isolation, alongside other containers that may be inside the Docker (Figure 1b)).



(a) **Docker architecture.** (b) **Process of building a container.**

Figure 1: **Virtualization with Docker.**

2.10.1 *Docker Compose*

Docker is an efficient solution to manage single containers. However, when it comes to complex applications that depend on several tools in a standard workflow, the management of several different containers can become complicated and quite time-consuming (List (2017)). In these circumstances, the use of Docker is no longer efficient. Docker Compose emerged, an extension of Docker designed specifically to handle multiple Docker containers at the same time. With Docker Compose it is possible to manage all containers from a single Docker Compose configuration file, a human-readable and machine-optimized YAML file, which defines how all containers should run. After defining in the .yml file which services are essential for the application as well as the relationship between them, a single command (*docker-compose up*) is enough to create and start the application environment (Turnbull (2014)).

3

METHODS AND TOOLS

3.1 AIBENCH FRAMEWORK: OPERATIONS, DATATYPES AND VIEWS

As mentioned earlier, any AIBench application is a set of operations, datatypes and views, following the MVC architectural pattern. AIBench operations (controller) are units of logic processing data defined through a set of ports (Glez-Peña et al. (2010)). Ports define the points where data can be established as the input or output of the operation, or both. Thus, each port of each operation is associated with a method that: receives data by parameters (if the port is IN), returns data (if the port is OUT) or that receives and returns data (if the port is IN/OUT) (Glez-Peña et al. (2010)). Equally important are the AIBench datatypes (model). A datatype is a Java class created with the aim of supporting problem-specific data structures, without any additional code, used as inputs and outputs of the AIBench operations (Glez-Peña et al. (2010)). Finally, AIBench views are used to display datatypes of the executed operations within the workbench. In that regard, its implementation aims to create a user-friendly interface to present the results of the operation, improving the interactivity of the application (Glez-Peña et al. (2010)). It is easy to understand that there is a relationship between the three aforementioned AIBench components. Figure 2 illustrates this relationship. The AIBench operation (controller) is automatically activated, verifying that a request is occurring on the part of the user. These operations receive as input and produce as output instances of AIBench datatypes (model). In its turn, the operation will, according to its output, render the information in the AIBench view so that the user has access to the complete information.

As the objective of this work is to implement and improve some features of the *merlin* software, which is entirely built on top of the AIBench framework, several operations will be created and, datatypes continually implemented and adapted for the development of tasks. Thus, throughout this work, new views will also be implemented and existing ones improved, always with the purpose of enhancing user experience.

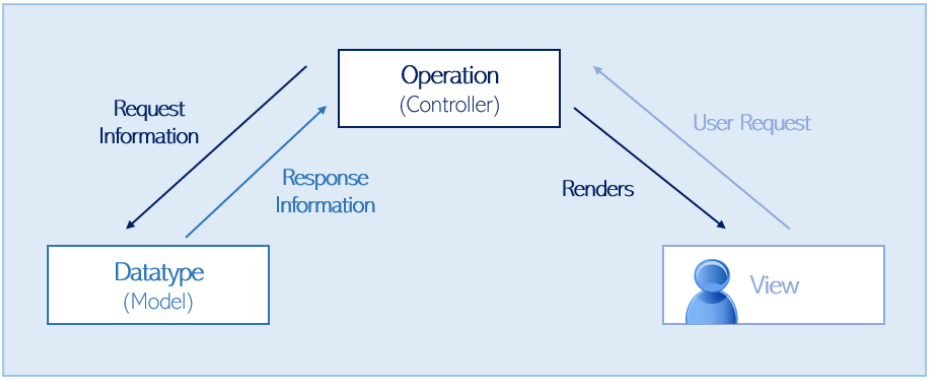


Figure 2: **AIBench application main logical components relation.**

3.2 BLAST + APPLICATIONS

There are several ways to perform a BLAST search. The most commonly used is through NCBI BLAST Website, as it has a simple to use graphical interface, presenting results quickly and intuitively. However, running BLAST outside the NCBI servers can be very useful for users who pretend to use their local database, or perform a personalized BLAST (Madden (2013)). Thus, a way to run BLAST locally is through BLAST + applications, freely available from NCBI. Although the high computational costs, these applications provide a faster and customized BLAST search and are very useful as they offer more features than those offered by NCBI BLAST Website, allowing the user to do a BLAST search that best suits his needs. BLAST + applications can run on a wide range of different operating systems and can also be used with the command line. When using the command line, some input parameters must be specified, namely:

- The BLAST+ variant option that will perform the search;
- The reference database name, which could be a public database or a local FASTA file;
- The query FASTA file name, containing the query sequences that will align against the reference sequences;
- The output file name.

In addition to the main parameters shown above, the user can customize the search by adding more parameters to the BLAST search. It may be useful to view BLAST results in different formats. For this, it is possible, for example, to specify the BLAST output format among several options, such as XML BLAST file, tabular file or pairwise alignment view option (default value). For an easy computational parsing, the XML BLAST output will be used in this work. Thereby, the results of the alignment obtained from BLAST come in an XML file. The structure of the XML BLAST file is shown in Figure 3.

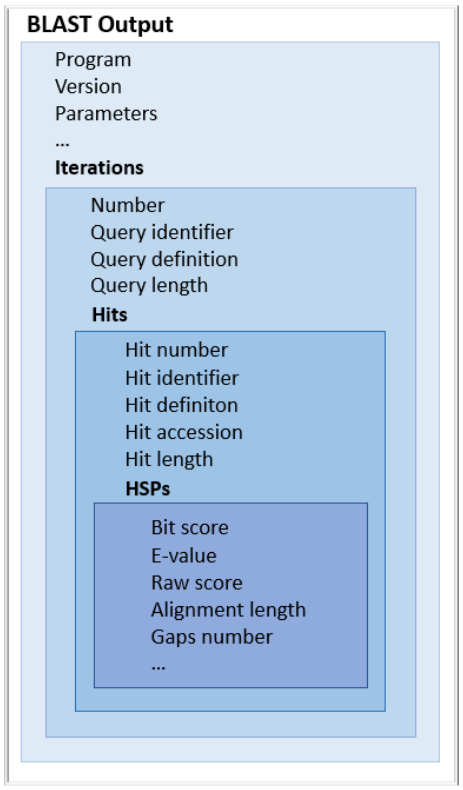


Figure 3: **XML BLAST output file structure.**

Each 'iteration' stores all the query strings for which matches were found in the database. Within each 'iteration', if BLAST finds any match for the query sequence, that data will be stored as 'hit' elements. If there are 'hit' elements represented in the sequence corresponding to the query in the database, 'hsp' elements will be found in XML, which correspond to segments of local alignments that have a high level of similarity. The level of similarity between the strings depends on the settings defined in the BLAST search. For each 'hsp' element, several statistics are displayed. In this work, thresholds will be defined for the expected value (E-value) and the bit score, to filter the BLAST results. The E-value provides an indication of the statistical significance of an 'hsp', corresponding to the number of BLAST hits with a similar score, which is expected to be observed in a random database just by chance. The lower the score, the higher the similarity between the query sequence and the reference database. The bit score is another essential statistical indicator calculated from the raw alignment score, which indicates the quality of an alignment. The higher the bit score value, the better the alignment.

3.3 HTTP COMMUNICATION

Hypertext Transfer Protocol (HTTP) is an application layer protocol responsible for communication and data exchange in a web-based application (Fielding et al. (1997)). It is a request/response protocol and as such, based on the client-server model. The HTTP client is the one performing the request, and the HTTP server is the web-server responsible for returning a response for that request (Figure 4). HTTP is a connectionless and stateless protocol (Fielding et al. (1997)). In a request-response cycle, after a request is made, the connection established between the HTTP client and the web-server is deactivated, re-establishing itself when the response is ready. The response is returned to the customer, and the connection is lost at the end of the transaction. Thus, the HTTP client and the HTTP server know each other only during the request and the current response, without being related to any other HTTP transaction that has been carried out previously. Another important feature is that HTTP is able to return responses on any type of data, as long as both the server and the client are able to read its content.

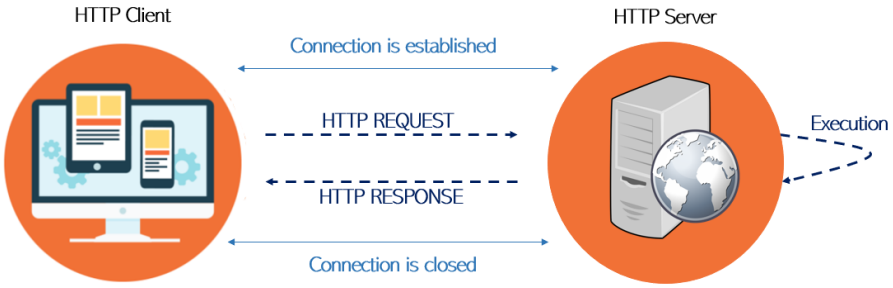


Figure 4: **Representative schema of a client-server communication framework under HTTP.**

Whenever the client sends a request to the server, called an HTTP message, it always specifies a special action or method along with it. Several special actions can be requested by the client and performed by the server. Among the most used methods are:

- GET: The most common HTTP request method. The client making a GET request wants to access a specific resource on the web-server.
- POST: It is used by the client to submit data to a specific resource. It is mostly used when uploading files or submitting forms.
- PUT: It is similar to POST, but it is used by the client to update an already existing resource with the new sent data.
- DELETE: The client making a DELETE request intends to remove a specific resource.

HTTP defines response status codes that the servers send back to the client, which is very useful because from this code it is possible to understand if the HTTP request was completed successfully, or what may have caused the failure when a transaction is not completed. The response status codes are grouped in five classes: informational responses, successful responses, redirects, client errors and server errors (Fielding et al. (1997)).

In this work, the HTTP protocol was used in two different phases of the development of the *BIT*. In an initial phase, HTTP requests were used to access the information of the models available in the BiGG Models database, through a web API. Later, it was also used in the development of a web-server, to manage user submissions.

3.4 BIGG INTEGRATION TOOL ARCHITECTURE

The main feature to be implemented in this work is the *BiGG Integration Tool* (*BIT*). With the integration of this tool in *merlin*, it is expected to retrieve metabolic information from the BiGG Models database during the reconstruction of *GSM* models. The *BIT* development process was planned as follows:

1. Implement a method to automatically retrieve all information available in BiGG Models database, using the BiGG API;
2. Construct a data structure that stores the recovered metabolic information and lists all the information to support the tool;
    - Create a text file for each model present in BiGG Models database with the details of all its components;
    - Create *BIT*'s support files;
    - Manipulate BiGG metabolic information.
3. Develop a strategy for the association of BiGG metabolic information with the genome of the case-study organism;
4. Implement an algorithm to generate the boolean rule for each reaction in the draft model;
5. Create a web-server to manage the submissions to *BIT* and return the results to the user;
6. Create a user-friendly graphical interface to send requests from *merlin*;
7. Import the results returned by *BIT* to the user's workspace.

Following the aforementioned planning, the *BIT* should be composed of four different components: a component responsible for obtaining and processing metabolic data; a web-server that allows managing submissions and obtaining results; a component responsible for loading metabolic data into *merlin*, and a graphical component that allows the user to submit requests from *merlin*. The creation and update of the BiGG data structure, as well as all the processing that occurs during the execution of the *BIT* must be done on a server, inside a Docker component, as this allows to improve the performance of the tool. A simplified schema of *BIT* is shown in Figure 5.

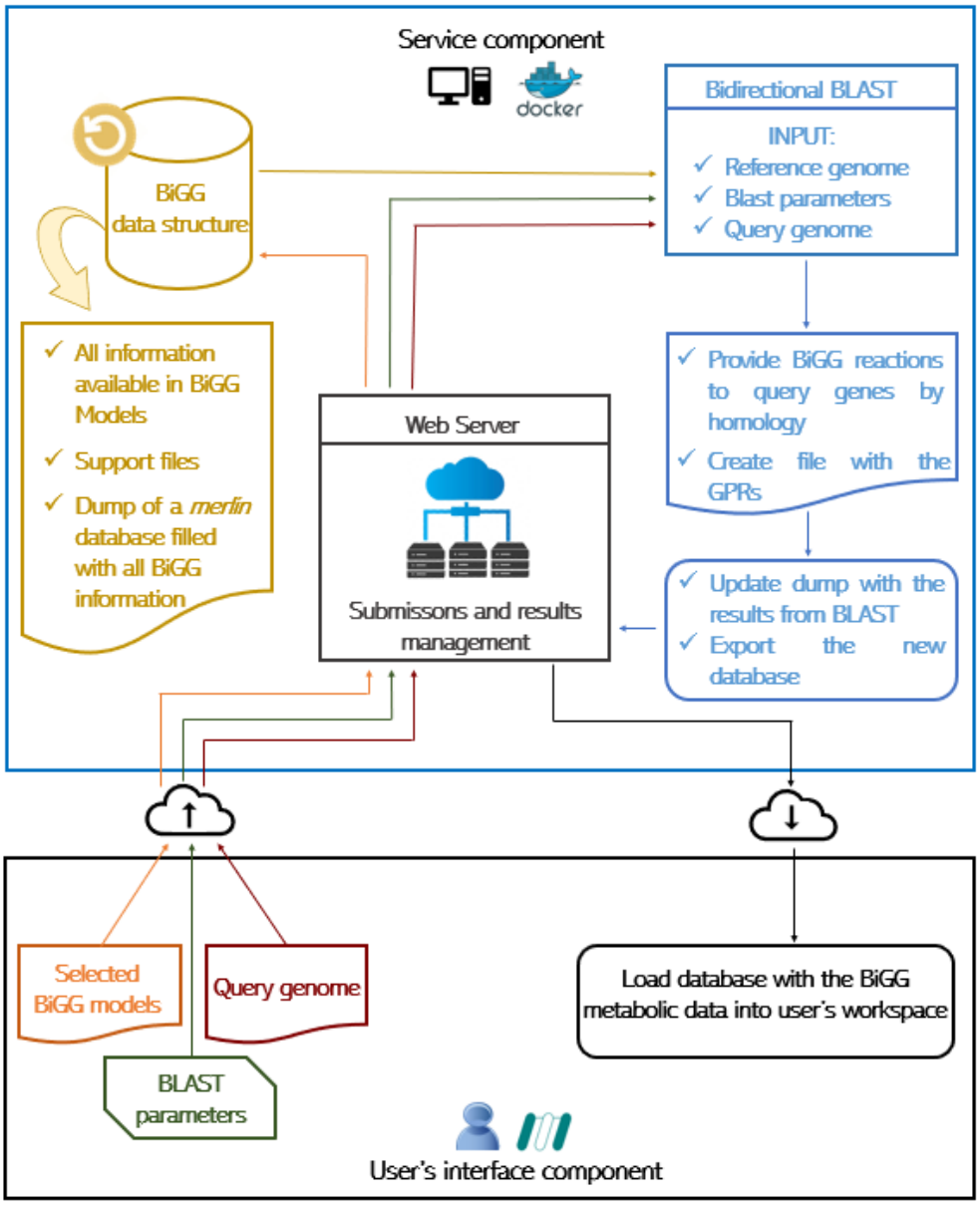


Figure 5: ***BIT*'s architecture.**

3.5 THIRD-PARTY TOOLS AND LICENSES

Table 2 lists all third-party tools used in the development of this work. For each one, the license type, the source and a short description indicating its purpose are presented.

Table 2: **Third-party tools used in the development of *merlin*'s new features.**

Tool	Task	Type of license	Source
Eclipse Photon	Development of the software	Eclipse Public License - v2.0	https://www.eclipse.org/downloads/
AIBench	Implement operations	GNU Lesser General Public License v3.0	https://github.com/sing-group/aibench-project
GC4S v1.2.2	Expedite GUIs implementation	GNU Lesser General Public License v3.0	http://www.sing-group.org/gc4s/
mysqlDump	Dump of a MySQL database	GNU General Public License v3.0	https://dev.mysql.com/downloads/mysql/
BiGG Models	Retrieve BiGG models data	Open Source License	http://bigg.ucsd.edu/
BLAST + v2.9.0	BiGG Integration Tool homology search	Open Source License	https://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/
Flask v1.0.2	Framework used to facilitate the web-server assembly	BSD License (BSD-3-Clause)	https://www.anaconda.com/distribution/
Postman	Make HTTP calls to perform tests during web-server developments	A free version was used	https://www.getpostman.com/downloads/
Docker v19.03	Run BiGG application isolated from the host machine	Apache 2.0 license	https://docs.docker.com/
Ubuntu	Run Docker on the server machine	Open Source License	https://ubuntu.com/download
WinSCP	File transfer between the local machine and the remote server	GNU General Public License v3.0	https://winscp.net/eng/download.php

4

SOFTWARE DEVELOPMENT

Throughout this work, new features were developed, and improvements were implemented in *merlin*, to extend the capabilities of the software and improve the overall user experience. All improvements and new tools were implemented in the Java programming language using the Eclipse Luna IDE. However, the Python language and other external resources were used to facilitate and allow the development of specific tools. All the work developed will be presented in detail on the following pages.

4.1 OVERALL IMPROVEMENTS IN *merlin*

Several improvements were made in *merlin* during this work. In addition to the integration of two new plugins, which facilitate the use of the software, other changes were performed. Although minor, these features requested by the *merlin*'s user community are quite important. The overall improvements implemented in *merlin* will be described below.

4.1.1 *Creation of a plugin to manage the configuration files*

Many of the implemented tools that are made available by *merlin* are based on default values, thresholds or other parameters that define that the algorithm must perform its actions taking into account these settings. The information of the configurations used by a given tool is usually saved in .conf files so that the operation can be customized without having to recompile the program. This way, both the programmer and the user can change the settings using a simple text editor. However, this way of managing the configuration files can be impractical for *merlin* users, especially for computer laypeople. Besides, the manual modification of a configuration file by the users can cause problems that might compromise the execution of the tool, in case the users accidentally change the name of a parameter, or change the default value of a parameter to one that does not make sense in the context of the tool. Therefore, graphic interfaces that allow the users to manipulate configuration files of different software components, according to their preferences were implemented. This tool

is implemented as a *merlin* plugin called *merlin-settings*. The integration of the functionality in a plugin allows to facilitate the development of new features for the platform, since, users will only need to install a new plugin, without having to update the version of *merlin* in its entirety. Additionally, the management of features through plugins also allows to offer the users the possibility to customize *merlin*, and for that, they are able to install only the plugins they want, using the repository manager tool, available on the platform.

With the integration of this plugin, the user will be able to configure, within *merlin* itself, the database connection settings, the e-biomass equation default contents, the gene-protein-reaction rules parameters and, finally, the 'find genes' tool default thresholds.

Database connection settings

Currently, *merlin* users can choose between two different relational database management systems (RDBMS), H2 or MySQL, to store data from *GSM* models. By default, the type of database used for data storage is H2. However, when users want to use a MySQL database, they have to edit the `database_settings.conf` file manually (Figure 6). Hence, the option 'database configuration' has been added to the 'settings' tab of the *merlin* menu bar, which redirects the users to an easy to use graphical interface, in which the values predefined for database connection settings can be changed. This required the implementation of the *DatabaseConfigGUI* class and the AIBench operation *DatabaseConfigSettings*. The association of the GUI with the operation in *merlin*'s menu bar was carried out within the `plugin.xml` file of the *merlin-settings* plugin.

```
----------------------------
| merlin database settings |
----------------------------

dbtype: h2

**mysql database credentials**

username: your_username
password: your_password
host: your_ip_address
port: 3306


**h2 database credentials**

h2_username: root
h2_password: password
```

Figure 6: **Database settings configuration file.**

The graphic component to manage the database settings configuration file is implemented in the *DatabaseConfigGUI* class, based on the InputParametersPanelDialog provided by the

GC4S library. As this demonstration of GC4S is based mainly on textboxes and drop-down lists, just like the features intended for the GUI, the extension of this class allowed to accelerate the development process of this graphical interface. Three classes from the GC4S library were used:

- InputParameter – This class encapsulates three components of an input parameter:
  1. a description label, which allows identifying the parameter;
  2. a label containing a small help message for the user to access in case of having doubts regarding the input that the parameter is waiting to receive
  3. a JComponent that allows to store what will be entered as an entry by the user for the fields database type, username, password, host and port.

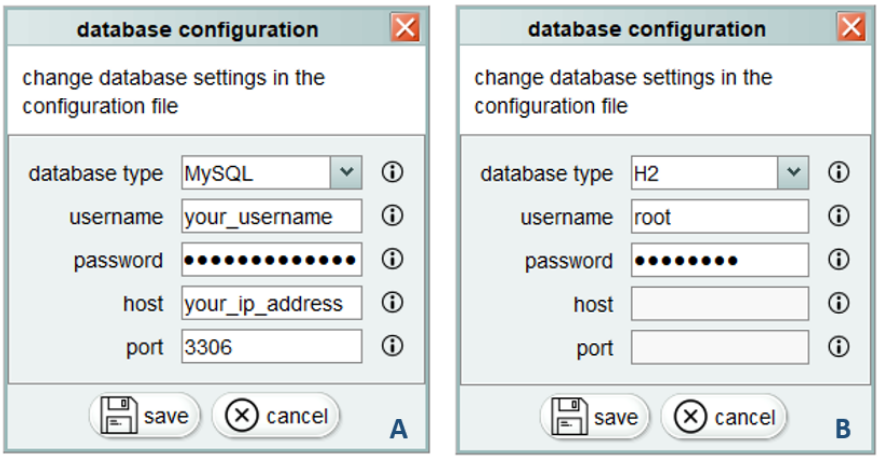
     In this GUI, three different JComponents were used. The username, host and port fields are simple JTextFields. The password field is a JPasswordField, similar to JTextField but the user's input is hidden, for safety purposes. Finally, for the database type, an ExtendedJComboBox was used, since there are specific values the user can choose from (H2 or MySQL). During the implementation of the GUI database configuration, the class InputParameter was instantiated five times, once for each parameter present in the `database_settings.conf` file.

- InputParameterPanel - This class instantiates a panel to display all created InputParameter objects.

- ExtendedJComboBox – As the name suggests, it is an extension of the Java component JComboBox. In addition to creating the combo box, this class adapts its size to the length of the items it contains. This component was used to create a combo box for the 'database type' parameter, containing the H2 and MySQL items, corresponding to the only two values that the user can choose for that parameter.

The combination of the three aforementioned GC4S elements, with other elements from Java Swing allowed to define the graphical interface layout (Figure 7). In addition to the layout, the GUI database configuration aims to display the information present in the configuration file automatically. The *fill()* method is implemented in the *DatabaseConfigGUI* class, responsible for loading the interface with the values corresponding to each parameter. Here, the `database_settings.conf` is read, and the file data are assigned to the JComponent of the respective InputParameter object. Thus, when the user opens the interface, the GUI is automatically filled with the information present in the configuration file. If the configuration file holds MySQL as the database type, all fields in the GUI are filled in (Figure 7a). Otherwise, only the database type, username and password parameters will be loaded, since the host and port parameters are not mandatory in H2 type databases (Figure 7b)).

The *fill()* method is called when there is a change in the value of the combobox database type. Here, an ActionListener that triggers an action when this value is changed, filling the fields accordingly, is used.



(a) **MySQL database type parameters.** (b) **H2 database type parameters.**

Figure 7: **Graphical interface to change the database settings.**

As already mentioned, *merlin* is based on the AIBench framework, so the use of its operations is essential to add custom GUIs, such as the 'Database Configuration' GUI. Thus, a *DatabaseConfigSettings* operation was implemented, consisting of five 'IN' ports corresponding to the database type, username, password, host and port fields. This operation executes the *save()* method, which is responsible for saving the new settings in the `database_settings.conf` file. The operation only performs this action if the user presses the save button; otherwise, the method is not executed and, therefore, the .conf file will remain with the same settings. Validations for users input are also implemented in the *save()* method, preventing from saving an invalid configuration and alerting the users to the error. A new panel displaying a warning message will appear when users:

- Do not fill in all mandatory fields - If the selected database type is MySQL, all fields are mandatory. In the case of H2 databases, only username and password are mandatory fields, the host and port fields appear as blocked text boxes, preventing the user from entering any input.
- Fill the port field with non-numeric characters - The port field only accepts numbers. The regular expression (Regex) "[0-9] +" was used to validate the users input.

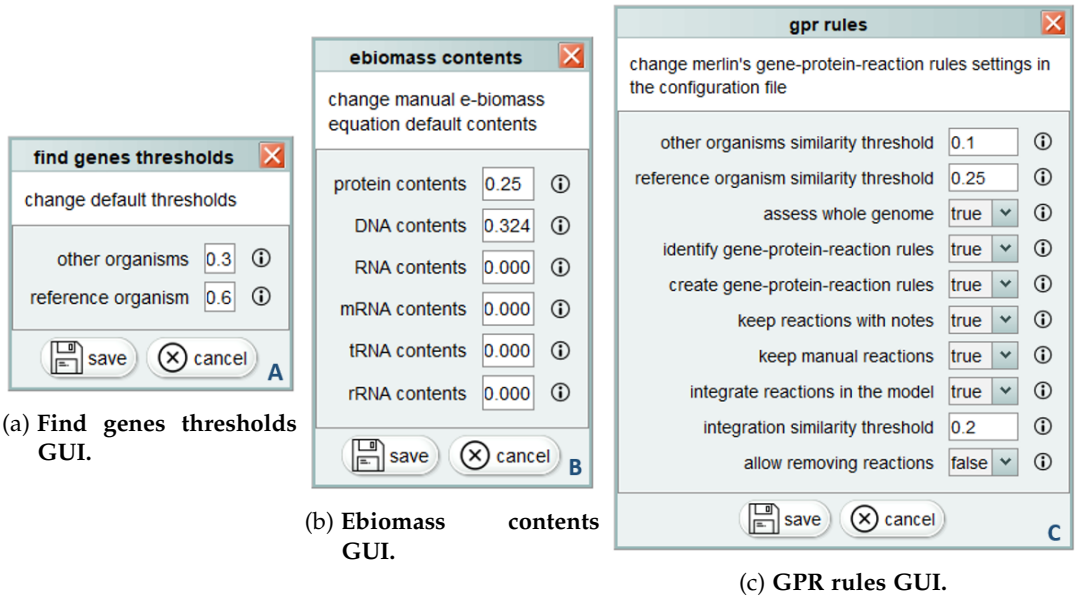
When all defined parameters are valid, *merlin* should be able to establish a connection with the user's database server.

Tools configuration

merlin has tools to read .conf files, applying each configuration during its execution. The 'tools configuration' option has been added to the 'settings' tab of the menu bar to allow users to change the tools' predefined parameters, within *merlin*. In this option, users may be redirected to one of the three implemented interfaces:

- find genes threshold;
- ebiomass contents;
- gpr rules.

The implemented interfaces can be seen in Figure 8.



(a) **Find genes thresholds GUI.**

(b) **Ebiomass contents GUI.**

(c) **GPR rules GUI.**

Figure 8: **Graphical interface to change *merlin* tools configurations.**

The methodology for implementing these interfaces was similar to the one used for the implementation of the database configuration interface:

1. Creation of a class to define the GUI layout from the InputParametersPanelDialog demonstration of the GC4S library;
2. Implementation of a method that allows the GUI to be filled with the current values present in the .conf file;
3. Creation of an AIBench operation to associate the GUI with *merlin*;
4. Implementation of a method that validates and saves the new configurations in the .conf file.

Hence, when any of the 'tools configuration' options are selected, the user is redirected to the respective GUI. The settings in the configuration file are automatically loaded into the view fields, allowing the user to access the current settings. The user is now able to change the fields as desired, as long as the input values are valid.

4.1.2 *Creation of a plugin that backs up and imports a workspace*

In this phase, the *BackupDatabase* operation was implemented to allow users to save a workspace. Backups are essential as these preserve information outside the central system. Therefore, this option offers users a tool that allows backing up a specific workspace, ensuring that in case of data loss during the reconstruction of a model all results obtained, up until the time of the backup, will remain stored elsewhere. However, this operation is only useful if there is a way to recover the stored data to the original location. The process of recovering data from a backup performed is known as a restore. The *RestoreDatabase* operation, which allows importing the information stored in the backup into *merlin*, was implemented to complement the *BackupDatabase* operation.

As already mentioned before, *merlin* has been entirely refactored to integrate Hibernate. Now, Hibernate allows *merlin* to use any SQL database without having to track and change the source code. At the time of this work, Hibernate's integration was still ongoing; thus, the report of the implementation of these operations (backup/restore) is divided into two distinct phases, namely before and after the integration of Hibernate, describing the updates required between phases.

This tool was also integrated as a plugin, the *merlin-exporter*. The 'export workspace' option was added to the 'workspace' tab of the *merlin* menu bar, which redirects to an interface that allows users to perform the *BackupDatabase* operation. Likewise, the 'import workspace' option was added to the 'workspace' tab of the *merlin* menu bar, allowing users to perform the opposite operation *RestoreDatabase*, efficiently. The declaration of the operations and insertion in the *merlin* menu bar was carried out within the `plugin.xml` file of the *merlin-exporter* plugin.

These operations allow users to backup and restore a workspace, which can be used for personal backup, importing their project on another machine or simply sharing it with other *merlin* users.

BackupDatabase AIBench operation

The *BackupDatabase* AIBench operation, created to run the tool that allows the user to back up a workspace, is composed of two 'IN' ports that receive as input: a 'WorkspaceAIB' object, corresponding to the workspace and an object of type 'File' corresponding to the selected directory. The input dialogue was generated automatically by AIBench, and no

custom GUI was created at this stage. The GUI generated by AIBench (Figure 9) allows users to select the workspace to backup, and the directory to store the backup.

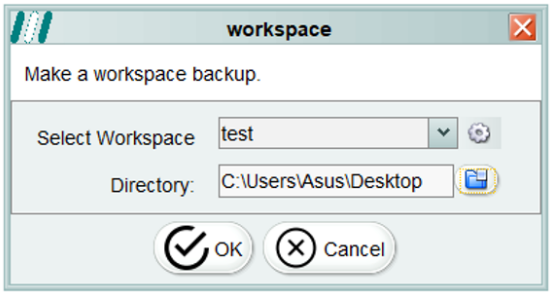


Figure 9: **Implemented GUI to perform a workspace backup.**

- Implementation prior to Hibernate

A simplified schema of the tool's workflow to export a backup of a workspace, before Hibernate's integration into *merlin*, is shown in Figure 10.

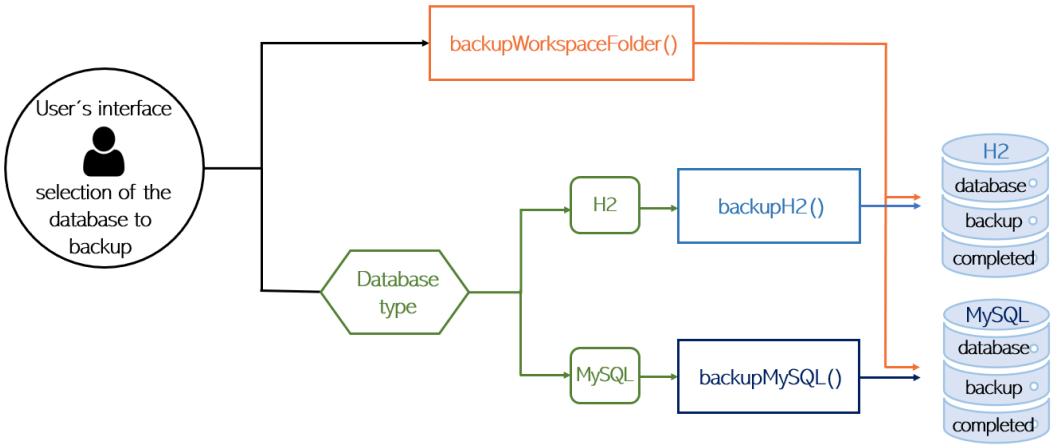


Figure 10: **Schema of the 'Backup workspace' tool before Hibernate.**

When users open a new project, they are required to enter a name for the workspace and the NCBI taxonomy identifier, to identify the organism during the reconstruction process. When creating a project, a folder with the name assigned to the workspace is created automatically in the 'ws' folder. This folder is where all files created during the *GSM* model reconstruction process are stored. Thus, in the first stage of the backup, a method (*BackupWorkspaceFolder()*) that copies the contents of the selected workspace in the 'ws' folder, to a temporary backup folder, was implemented. Also, a dump of the entire database is necessary to complete the backup. H2 type databases store all the information in the database in '.mv.db' files. Thus, when backing up H2 databases, the *backupH2()* method is

executed, copying such file to the backup folder containing the workspace files. Whereas, for MySQL databases the *backupMySQL()* method is executed, dumping the database through the 'mysqldump' command. This utility is provided by the MySQL application which has to be previously installed in the user's machine. The mysqldump results in a '.sql' file, containing CREATE TABLE and INSERT statements that define the structure and contain all information present in the database. This file is stored in the backup folder that contains the workspace files.

The backup folder is saved with the name of the workspace, followed by the complete date (day, month, year and time) on which the backup is performed, thus ensuring that the name will always be unique, and therefore, no backups will be overwritten or another user file replaced. In the end, the backup folder is converted to a '.mer' file.

- Implementation after Hibernate

After the integration of Hibernate in *merlin*, the *BackupDatabase* operation was updated, as the tool's operation has to be guaranteed regardless of the type of database used for data storage. A simplified schema of the tool's pipeline for exporting a backup of a workspace, after integrating Hibernate, is shown in Figure 11.

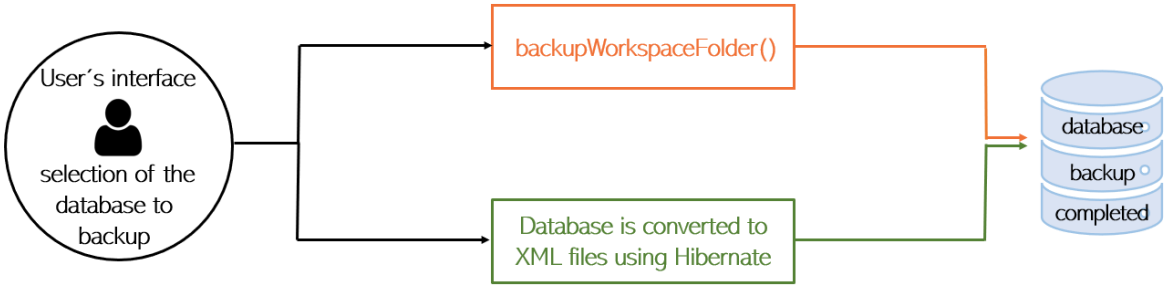


Figure 11: **Schema of the 'Backup workspace' tool after Hibernate.**

The first stage of the process, in which the workspace files are copied to the backup folder, remained unchanged. The main change is in the way the database is delivered. With Hibernate, it is no longer necessary to determine which type of database the user wants to export. The implementation of the operation was simplified, only requiring Hibernate methods that directly convert the database into XML files, corresponding to each of the tables that currently make up a *merlin* database. The XML files are kept in the workspace files backup folder. The backup folder is converted into a '.mer file', automatically labelled according to the workspace name and the time/date of the export operation. These files can be imported through the antiparallel operation *RestoreDatabase*.

RestoreDatabase AIBench operation

The *RestoreDatabase* AIBench operation, created to run the entire tool pipeline that allows the users to import a workspace backup, consists of three 'IN' ports. The graphical interface shown in Figure 12, created automatically by AIBench, allows the users to import the '.mer' file, corresponding to the backup of the workspace they want to restore, through the file picker field. Additionally, users can also force import, which will replace all files, when a workspace with that name already exists, as the imported workspace will inherit its original name by default. When users import a project not checking the checkbox nor filling the text field with a new name, a warning message will pop up, forcing the user to enter a name for the workspace to proceed with the import.

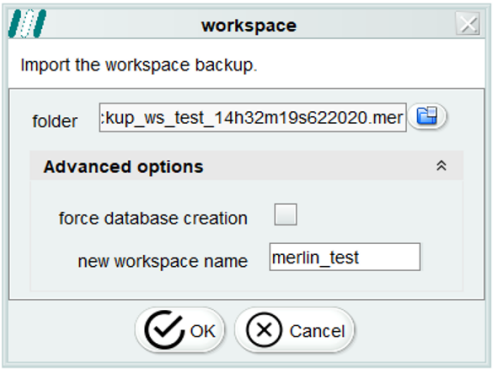


Figure 12: **Implemented GUI to perform a workspace restore.**

- Implementation prior to Hibernate

A simplified schema of the tool's pipeline to import a backup from a workspace, prior to the integration of Hibernate into *merlin*, is shown in Figure 13.

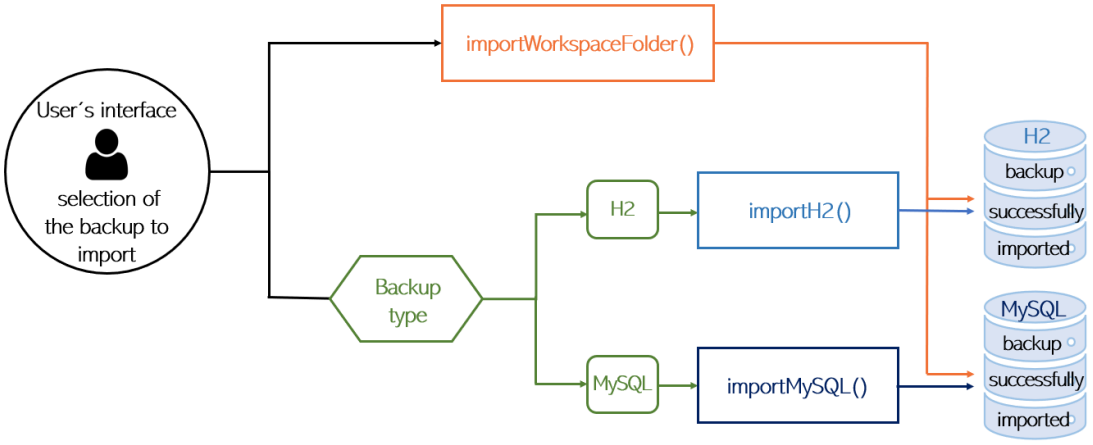


Figure 13: **Schema of the 'Restore workspace' tool before Hibernate.**

The worskpace restore process begins after the imported file is validated. If the file selected by the user does not have the expected extension (.mer) or does not contain the backup files, the user will be warned, and the operation will not be performed. When all necessary conditions are fullfilled, the restore process starts with the execution of the *importWorkspaceFolder()* method. After decompressing the '.mer' file, the backup is pasted into the 'ws' folder. If an existing workspace is used, the files will be replaced. Then, the type of backup is determined. For this purpose, a validator which checks the extensions of the backup files was developed. If a file with the extension .mv.db exists, the backup type is H2; otherwise, the backup is MySQL. In the case of an H2 backup, the *importH2()* method is executed, and the '.mv.db' file is pasted in the destination folder. Otherwise, the *importMySQL()* method in which the '.sql' is restored, is used.

Before starting the restore process, the database connection credentials are validated, verifying that the user does not attempt to restore a backup of a MySQL database as H2 or vice-versa, as this implementation is not prepared for converting one type of database to another.

- Implementation after Hibernate

After the integration of Hibernate in *merlin*, the *RestoreDatabase* operation was updated to complement the changes made to the *BackupDatabase* operation, as the backup contains other types of files. A simplified schema of the tool's pipeline for importing a backup from a workspace after integrating Hibernate into *merlin*, is shown in Figure 14.

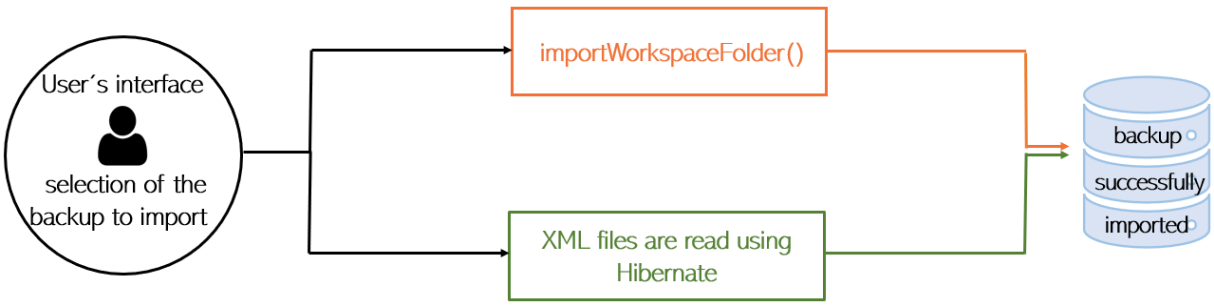


Figure 14: **Schema of the 'Restore workspace' tool after Hibernate.**

The first stage of the restore process remained unchanged, after validating the user input and decompressing the '.mer', all files obtained during the model reconstruction process are pasted in the 'ws' folder. Then, since the output of the dump is now a set of XML files, Hibernate methods read each file and convert them to Java objects, which are then loaded into the database.

This change allows overcoming a significant limitation of the first implementation as users can now backup a type of database and import it as another type.

4.1.3 *Email setter*

The user's email is mandatory to access certain tools such as BLAST, remotely. Thus, several interfaces had email as a mandatory field, requiring the user to enter their email each time. Hence, this field was included in the main 'open workspace' interface, which allows opening an existing workspace or creating a new one, as a mandatory field (Figure 15). When the user creates a workspace, the email is stored on the user's local machine, in a configuration file, and is used to load the email field automatically. A regular expression was used to verify that the email entered follows the format aaa@bbb.ccc, preventing the user from creating the workspace if the email is invalid. Regarding the interfaces that perform operations that require the user's email, the email field was removed, and the value is obtained directly from the configuration file.

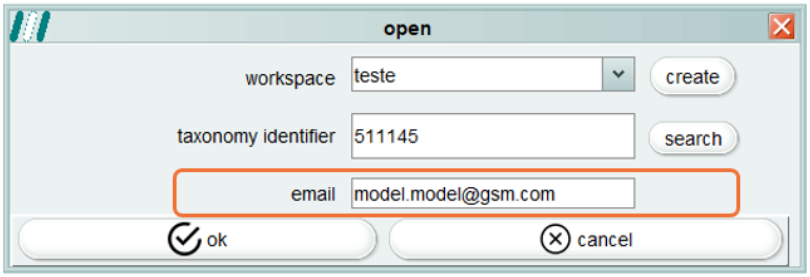


Figure 15: **Addition of the email field to the *merlin*'s open GUI.**

4.1.4 *Confirm cancel with CustomGUI*

All tools provided by *merlin* have cancel buttons that allow users to cancel operations. However, users tend to accidentally click on cancel, immediately interrupting the operation, which can be very annoying, especially when tools take a long time to return results, such as the enzymes annotation tool. Thus, a CustomGUI which presents a panel with a question and options, was used to overcome this problem. In this case, the CustomGUI was used offering the options 'yes' or 'no', as shown in Figure 16. This CustomGUI was implemented in all *merlin* operations that have a progress bar, so a confirmation GUI should appear whenever the cancel is triggered and the operation is only interrupted after the user's approval. This feature was one of the users' top requests.

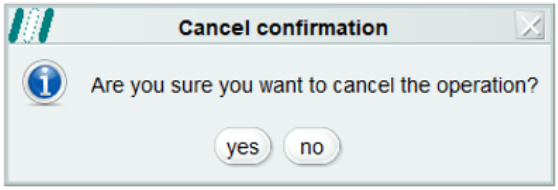


Figure 16: **Cancel confirmation CustomGUI.**

4.1.5 *Search in notes column*

The 'model reactions' and the 'annotation enzymes' views from *merlin* include a 'notes' column, which allows users to write custom notes. Both views have a search toolbar, allowing the user to search by the name of the gene/reaction ID, or search in all columns, if the search toolbar is defined with the 'all' option. However, searching just by the 'notes' column can also be very valuable for the user. Hence, the *merlin*'s 'SearchInTable' class was adapted to allow the 'notes' option to the search toolbar, thus providing search options to the user. Figure 17 shows the search for the word 'atp' in the notes column of the 'annotation enzymes' view, with the blue highlight on the lines in which the word is found.

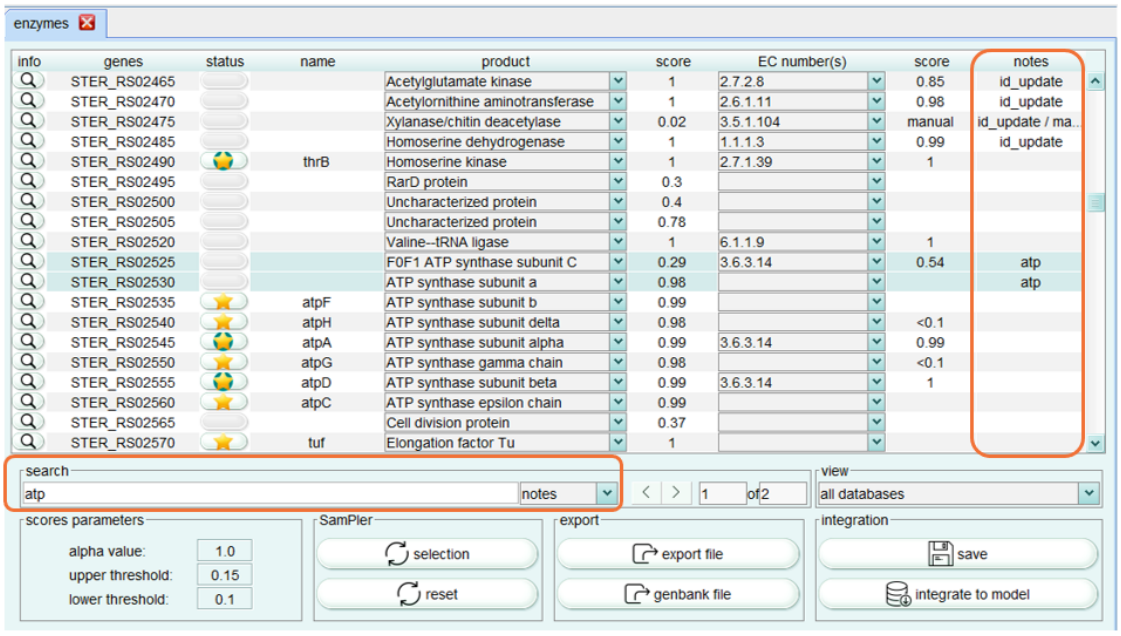
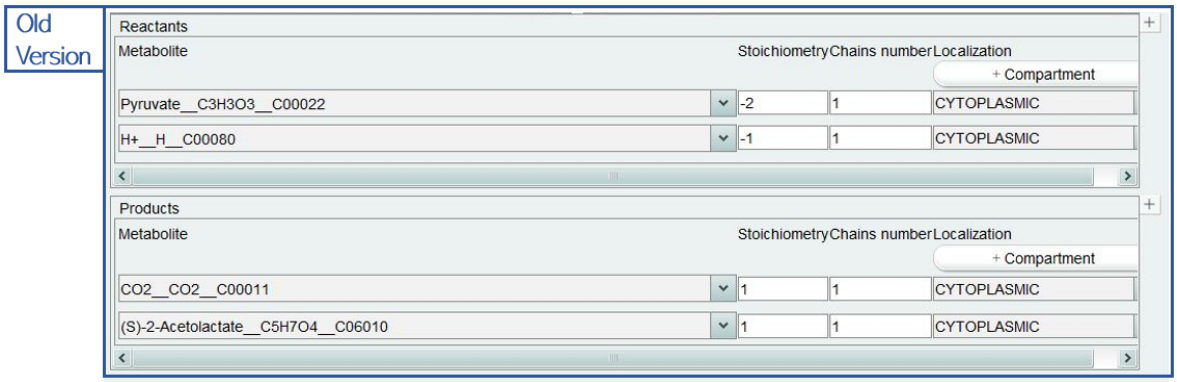


Figure 17: **Addition of the option 'notes' to the search tool bar.**

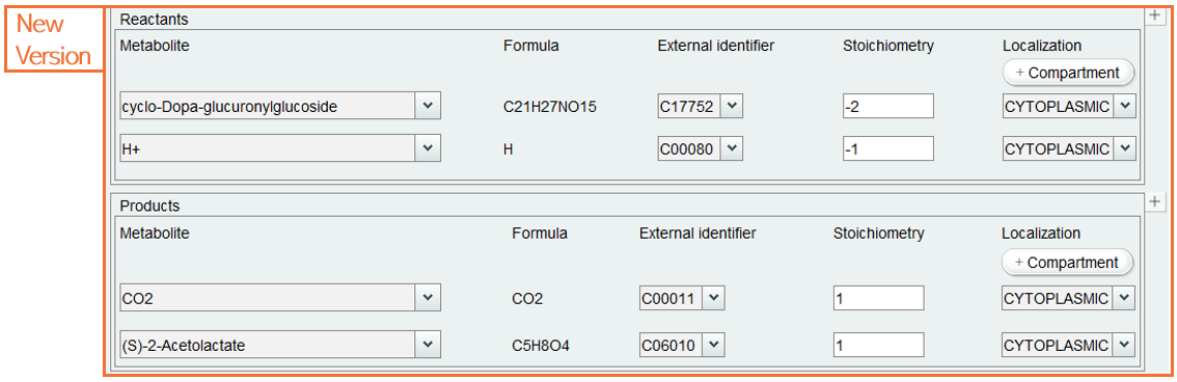
4.1.6 *Insert/Edit Reaction interface improvement*

Other improvements include the 'Insert' and 'Edit' operations of the reactions panel, which can be found in the 'model reactions' view. As the name suggests, these operations allows the users to manually enter a new reaction, or edit an already existing one. When selecting these operations, the users are taken to a view that allows to insert/edit a reaction. The improvement implemented in these interfaces is focused on the reagents and products panel in which the combo box containing the metabolites that form the reaction, had a single string to indicate the name, formula and identifier of the metabolite (Figure 18a). Now, the string was split to create three different columns: one column with the combo box that contains the name of each metabolite; a column to display the formula, as a non-editable text field and, finally, a third column with a combo box containing the external identifiers. The user can search for the metabolite either by name or identifier, automatically updating the value of all other columns. Additionally, the 'Chains number' column was removed.

The resulting layout from the *merlin* 'Insert Reaction' interface is shown in Figure 18b. The same layout can be viewed in the 'Edit Reaction' interface.



(a) **Reactants and products panel before the implementation.**



(b) **Reactants and products panel after the implementation.**

Figure 18: **Reactants and products panel.**

4.1.7 *Logger configuration*

The activity log of any application over time is critical, as it allows the programmer to quickly understand what the application code is doing at each moment of the execution, defining if a particular action was carried out successfully, and more importantly, where and why a failure occurred. This registry is done through a set of logs that informs the developer. It is possible to configure the logger through a configuration file to specify how logs will be displayed. Thus, the `logback.xml` file was configured with the purpose of writing the logs in text files and not just on the console of the integrated development environment. For this, an appender 'FILE' was added to the logger configuration, allowing the recording of logs in text files, which can be configured according to the storing option, selected by the programmer. The logger was configurated to record logs in files with a maximum of 100MB. These files are stored in a zipped folder, up to a maximum of 30 files per day, for the last 30 days, providing a log history to the developers.

4.2 IMPLEMENTATION OF BIGG INTEGRATION TOOL

During this work, the main feature developed and implemented in *merlin* was the *BiGG Integration Tool* (*BIT*). This tool aims at retrieving metabolic information from BiGG Models, a knowledge base of high-quality genome-scale metabolic models. As previously mentioned, obtaining metabolic information is a mandatory step for the reconstruction of a *GSM* model. Until now, *merlin* only offered an operation that allowed obtaining this information from KEGG. Thus, the models built from *merlin* exclusively use data from that database. In this sense, the *BIT* was integrated as a *merlin* plugin, allowing the loading of metabolic BiGG information to be used in the *GSM* model assembly.

4.2.1 *BiGG data structure*

As the objective is the creation of a tool that allows the loading of metabolic BiGG data to *merlin*'s databases, the first stage of the development process is focused on creating a robust data structure, capable of supporting the requirements of the following stages. In this phase, all BiGG metabolic data were collected and processed, and support files were created to assist in the remaining development phases. Additionally, at this stage, a database was loaded with all the information collected from BiGG Models. A dump of this database was created to speed up the next step in the process. Each of these steps is described in detail below.

Obtaining and processing metabolic data

The collection of metabolic data was the first step in the process of creating the data structure. The BiGG Models knowledge base allows to programmatically access the content of all the models through HTTP requests to the BiGG Models API. Java is a programming language that supports HTTP requests. Thus, it was possible to use the API to collect the metabolic data of each BiGG model efficiently, through successive GET requests. It is possible to download a complete model with a single call to the API. However, the answer does not return all the information present in BiGG Models. As the objective is to obtain a data structure that contains all information to support this plugin and, eventually, other tools that use BiGG's metabolic data, thorough requests were performed to the API. For each model, an individual call was made to all reactions, metabolites and genes that compose it, thus obtaining detailed information on each component. Each time a request is performed to the API, a JSON object with the results is returned. JSON objects that return data related to the same model are integrated into a single JSON and stored in text files, for a better organization of the information. Thus, 108 files, corresponding to the 108 models currently available at the BiGG Models database, containing information about the model as well as the details of each reaction, metabolites and genes that constitute it, were created.

To facilitate access and manipulation of data from text files, BiGG containers, Java classes that mirror the structure of the BiGG Models database, were created. Thus, whenever it is necessary to access the text files, these classes are loaded, easing the retrieval of any given parameter. The developed containers, their primary attributes, as well as the relationship between them, are shown in Figure 19. Each BiGG Model has a list of BiGG Genes, BiGG Metabolites and BiGG Reactions and additional parameters to represent some details of the model, such as its BiGG identifier and the name of the organism to which it corresponds. For the most used BiGG Models, Escher Maps are also available for pathway visualization. Besides general details, BiGG Reactions exhibit a list of BiGG Metabolites and a list of Reaction Results that provide details about the reaction, such as the lower bound, upper bound and gene reaction rule. BiGG Genes and BiGG Reactions that have associated pathway visualization, also include an interactive pathway map viewer powered by Escher. Links to external databases such as KEGG, MetaCyc, Reactome and Model SEED are also associated with BiGG Genes, BiGG Metabolites and BiGG Reactions.

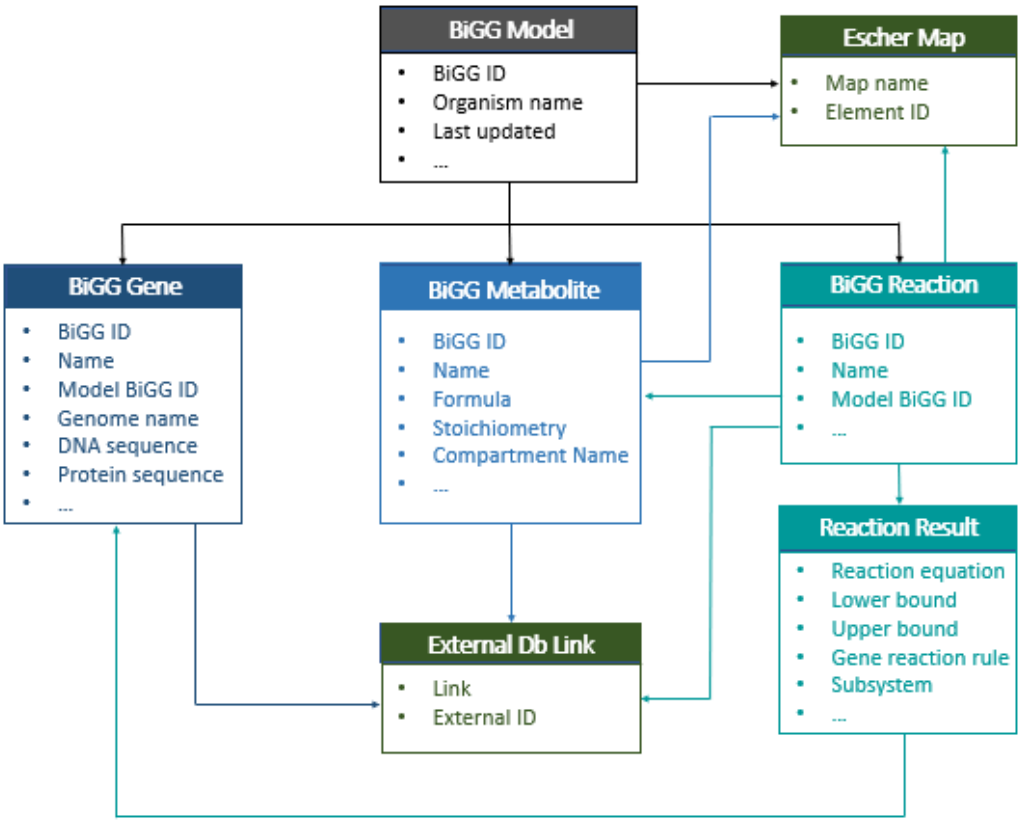


Figure 19: **Collected data structure.**

Creating files to support the BiGG Integration Tool

From the metabolic information collected for each of the BiGG models in the previous phase, support files that list important information to be used in other stages of the development of the tool were created. Thus, five files were created at this stage, namely:

- `SequenceIDsGenesRelation.txt` file

  In BiGG Models, all genes mapped to a genome annotation are referenced by their *locus tag*. Thus, BiGG genes have unique identifiers for a specific genome annotation, which makes models that integrate the same annotation to share BiGG identifiers with each other. For this reason, there are many genes that, despite having the same protein sequences, are assigned to different BiGG identifiers. A unique identifier was generated for each sequence to guarantee the uniqueness of information. The identifiers follow the gXXXXXXXXX standard. For instance, the first sequence receives the identifier g000000001, the second different sequence receives the identifier g000000002 and so forth. Thus, for all text files created in the previous phase, which contain information from BiGG models, its list of genes was retrieved, to identify those that have different identifiers but the same protein sequence. A text file, `SequenceIDsGenesRelation.txt`,

which maps each sequence ID to the BiGG IDs genes that contain the protein sequence represented by that identifier was created, to store this information.

- `ModelsGenesProteinSeq.faa` file

  In the next stage of the *BIT* development process, BLAST is used to associate query genes with BiGG metabolic information. In addition to the FASTA containing the genome, it is also necessary to pass as input a file containing the set of BiGG sequences against which the alignment will be carried out. Thus, the `ModelsGenesProteinSeq.faa` file was created at this stage with all the protein sequences present in BiGG Models. From the previous file, it was possible to create a FASTA file with unique sequences, reducing the total number of sequences and, consequently, the time required to perform all the alignments in the BLAST phase. The file description line contains the sequence identifier to distinguish each sequence.

- `GeneReactionRelation.txt` and `SeqIDReactionRelation.txt` files

  From the BLAST results, it will be possible to determine whether there was a significant level of similarity between query genes and BiGG genes. If the two genes exhibit sequence similarity, it is legitimate to assume that both are associated with similar reactions. Therefore, the list genes in each BiGG model was browsed to obtain the reactions associated with each one. This information was compiled and saved in a text file, `GeneReactionRelation.txt`, which maps each gene to a list of BiGG reactions. In parallel, a second file was created, `SeqIDReactionRelation.txt`, which establishes the direct relationship between the generated sequence identifier and the associated reactions. This information will ease the association between query genes with BiGG reactions at a later stage in the development of the tool.

- `GeneReactionRule.txt file`

  All reactions that have associated genes, include boolean rules that describe the gene-protein-reaction (GPR) relationships. The file `GeneReactionRule.txt`, which maps each BiGG reaction to the boolean rule, was created to assist *BIT* on generating the boolean rules for the reactions in the draft model. Note that reactions shared between models may exhibit different boolean rules. Thus, the logical operator 'OR' was used to separate the boolean rules from the different models.

Creating a database dump filled with BiGG data

A *merlin* database called *biggdata* was generated to complete the BiGG data structure. This database was loaded with the BiGG metabolic data retrieved and was exported to be used in the next step. Thus, the development of this phase can be divided into three distinct parts:

1. Creation of a class, *BiGGDataLoader*, responsible for reading the metabolic data of each BiGG model and filling the *merlin* containers, which describe the structure of the database in an organized way;

2. Creation of a class, *BiGGLoadMetabolicData*, responsible for reading all the filled containers and inserting the metabolic data in the database;

3. Creation of a dump of the database loaded with BiGG metabolic data, for storage.

During this development phase, several methods and classes already implemented in *merlin* were used, facilitating to load information into the database. Each of the steps implemented until the exporting of the *biggdata* database is described in more detail below.

- *BiGGDataLoader* class

The *BiGGDataLoader* class was created to manipulate the BiGG metabolic information, transforming it into information that can be read by the *merlin* software. As previously mentioned, all BiGG information collected from the models available in the database was loaded into BiGG containers, which reflects the way the data is organized in BiGG Models. In the same way, other containers that represent the *merlin* structure allow sending information to the database. Thus, for each BiGG model, this class is used to make all the necessary procedures before entering the data, organizing the information loaded in BiGG containers so that it can be used to fill *merlin* containers.

Of the six main components of a model, only the GeneContainer and PathwaysHierarchyContainer, which correspond to the *merlin* containers that store information about genes and pathways, respectively, were not used. Whereas the GeneContainer was not necessary to fulfil the purpose of the tool, the information available in BiGG was not enough to deploy the PathwaysHierarchyContainer. Hence, four methods were implemented in this class, responsible for reading the BiGG containers of each model, processing and converting them into the *merlin* container, namely: CompartmentContainer, ProteinContainer, MetaboliteContainer and ReactionContainer.

The CompartmentContainer did not require significant processing. Each BiGG Metabolite was scanned to obtain the parameters compartment identifier and compartment name, for the conversion to CompartmentContainer. Each container is added to a single entry list, which is used later to fill the ModelCompartment table in the database.

Regarding enzymes, several BiGG reactions are mapped to EC Numbers in the database external links. Thus, the database links of each BiGG Reaction were reviewed to obtain, whenever available, the EC identifier for each enzyme. Since only the identifier is provided in BiGG Models, the KEGG API was used to fetch the name of each enzyme to fill in the ProteinContainer. When more than one name is used to designate the same EC, the first name on the list was used to fill the protein name field, and the remainder added to the

synonyms list. Finally, the entries were processed to obtain the list of BiGG Reactions that are associated with each EC Number.

Each BiGG Metabolite was scanned to access its external identifier, name, molecular formula, and list of external database links, to fill the MetaboliteContainer. An algorithm was implemented to indentify metabolites that, despite labelled with different identifiers, have the same name and molecular formula, thus representing the same entry. This algorithm is executed for each BiGG Metabolite and synonymous metabolites are added to a list available in the MetaboliteContainer.

Finally, for each BiGG Reaction, the BiGG Reaction Result was obtained, which provides more details for the reaction. The fields external identifier, chemical equation, source and the list of database links were directly filled in ReactionContainer. For the list of enzymes associated with each reaction, the list of database links was sought to obtain, when available, the EC Number. Regarding the metabolites present in the reaction, an algorithm that parses the chemical equation to obtain the list of the reaction's reactants and products was developed. For each metabolite, a MetaboliteContainer is created and added to the ReactionContainer's ReactantsStoichiometry and/or ProductsStoichiometry. Besides the external identifier, the compartment name and stoichiometry are added to such field. As a result of the metabolites redundancy, similar reactions can also be found in BiGG Models. Thus, an algorithm that, for each BiGG Reaction, verifies whether other reactions have the same list of products and reactants, was implemented. Each reaction identified with this approach is added to the names list of the ReactionContainer.

The algorithm implemented in the *BiGGDataLoader* Java class is presented in the pseudo-code displayed in the Algorithm 1. Additionally, the Algorithm 2 presents the method used to fill the ReactionContainers list.

Algorithm 1 Parse info to fill *merlin* containers - parseInfo(arg1, arg2)

```
1: Input: models, database
2: databaseInitialData = retrieveAllData(database) ▷ Get the current data in the database
3: for each model in models do
4:     modelInfo = UpdateBiggFiles.get(model)              ▷ Get the model information
5:
   ▷ Fill merlin containers
6:     data.setCompartments(compartmentsParser(modelInfo.getCompartments()))
7:     data.setMetabolites(metabolitesParser(modelInfo.getMetabolites()))
8:     data.setEnzymes(enzymesParser(modelInfo.getEnzymes()))
9:     data.setReactions(reactionsParser(modelInfo.getReactions()))
10: end for
11:
12: BiggLoadMetabolicData.load(database, data, databaseInitialData);
```

Algorithm 2 Fill reaction container - reactionsParser(arg1)

```
1: Input: biggReactionsList                                   ▷ BiGG Models reactions
2:
3: reactionContainers = new List();
4: for each biggReaction in biggReactionsList do
5:     reaction = new ReactionContainer(biggReaction.Id());
6:
7:     reaction.enzymes = biggReaction.enzymes();
8:     reaction.dbLinks = biggReaction.dbLinks();
9:     reaction.equation = biggReaction.equation();
10:
11:    dir = getDir(reaction.equation());          ▷ Gets the the equation direction
12:
   ▷ Split the equation in reactants and products
13:    metabolites = reaction.equation.split(dir);
14:
   ▷ Split the metabolites by "+" to get a complete list
15:    biggReactants = metabolites[0].split("+");
16:    biggProducts = metabolites[1].split("+");
17:
18:    reactants = new List();
19:    products = new List();
20:
21:    for each reactant in biggReactants do
22:        met = new MetaboliteContainer(reactant);
23:        met.compartment = biggReaction.getCompartment(reactant);
24:        met.compartmentName = biggReaction.getCompName(reactant);
25:        met.stoichiometry = biggReaction.getStoichiometry(reactant);
26:        reactants.add(met);
27:    end for
28:
29:    for each product in biggProducts do
30:        met = new MetaboliteContainer(reactant);
31:        met.compartment = biggReaction.getCompartment(reactant);
32:        met.compartmentName = biggReaction.getCompName(reactant);
33:        met.stoichiometry = biggReaction.getStoichiometry(reactant);
34:        products.add(met);
35:    end for
36:
37:    reaction.reactantsStoichiometry = reactants;
38:    reaction.productsStoichiometry = products;
39:
40:    reactionContainers.add(reaction);
41:
42: end for
43: return reactionContainers;
```

- *BiGGLoadMetabolicData* Class

After filling in the *merlin* containers, the BiGG metabolic data is organized correctly to be loaded in the *biggdata* database. The insertion of the data was performed through Hibernate methods already implemented in *merlin*, which allow filling the tables without SQL statements. The *BiGGLoadMetabolicData* class was implemented, to load the *biggdata*, which reads all the *merlin* containers from each BiGG model.

The same order for filling the containers was maintained when loading data into the database. Consequently, the information stored in the CompartmentContainer was inserted first, followed by the ProteinContainer and MetaboliteContainer.

The insertion of compartments, enzymes and metabolites is performed similarly. After reading each container, the database is analysed using the external identifier to determine if it is present in the database. The analysis either returns an integer value relative to that element's database identifier or inserts the element and returns the new identifier. This operation loads *merlin*'s tables ModelCompartment, ModelProtein and ModelCompound, as well as the ModelDbLinks and ModelAliases.

Reactions must be inserted in last place, given that they depend on information from the aforementioned containers. The reactions processing follows the same procedure previously described for compartments, enzymes and metabolites, determining whether the reaction has already been inserted. Initially, the lists of reactants and products are validated. Then, the stoichiometric coefficient of each metabolite is inserted in the ModelStoichiometry table, establishing the connection between the ModelReaction and ModelCompound tables. Enzymes and reactions are connected through the intermediate table ModelReactionHasModelProtein. Finally, the external cross-reference links and synonyms for each reaction are inserted in the ModelDbLinks and ModelAliases tables, respectively.

The algorithm implemented to load BiGG compartments in the *biggdata* database is presented in the pseudo-code displayed in the Algorithm 3.

Algorithm 3 Fill compartments table - loadCompartments(arg1, arg2, arg3)

```
1: Input: database, data, databaseInitialData
2:
3: metabolicDataLoader = new BiggModelDataLoader(databaseName);
4:
5: for each compartment in data.getCompartments() do
6:     if databaseInitialData not contains compartment then
7:         metabolicDataLoader.loadCompartment(compartment);
8:     end if
9: end for
```

- Exporting *biggdata* database

The last stage of this phase refers to exporting the *biggdata* database, loaded with all BiGG metabolic data. The database dump was performed with Hibernate methods that allow exporting the database in XML files. The database dump was stored in a zipped folder to be used at a later phase of the development.

Configurations / Server component

This component of the tool controls the collecting of online information, its processing and the creation of files that combine the necessary data structure throughout the development process. This structure, and all tasks required to its creation, are available in a Docker component, allocated in a server. Additionally, all support files and the *biggdata* database dump, along with the md5 key, are stored in a FTP server. This location ensures that all files are accessible by the tool, whichever server it is running on. The use of Docker is an advantage as besides providing a more efficient use of the host machine's system resources during the execution of tasks, the application's containerization with all its dependencies, guarantees its operability in any environment. Thus, the JAR (Java ARchive) that compiles all the instructions to create the BiGG data structure is executed inside a docker. Updating this data structure regularly over time is crucial to ensure that the database contains the most recent information. Therefore, a cronjob that automatically runs the docker according to the timing specified in the crontab configuration file was set up. Taking into account the frequency with which BiGG Models is updated, the configuration was implemented for the update of the data structure to occur monthly. Despite this periodic execution, the collection of information from the BiGG API and subsequent creation of the data structure will only be performed if new models have been included in the host database, or if metabolic data of a model already available is updated. Therefore, each time the operation is performed, only updated or added model data is collected, ensuring that unnecessary requests to the API will not be made and, consequently, reducing the task execution time. Updating or adding a text file with information from a BiGG model also requires updating the tool's support files, as well as the *biggdata* database dump. A track record of all generated data is kept with each update to trace possible errors.

4.2.2 *Association of BiGG metabolic data to the query genome*

This component of the tool relies on the interaction with the user, as each task's performance depends on the submitted input. Thus far, a *merlin* database has been created with the metabolic data collected from all models available in BiGG Models. The next phase is to find which BiGG metabolic data are present in the *GSM* model being built for the

case-study organism, to associate each metabolic gene with BiGG reactions. This association is performed by homology, through BLAST similarity searches. BLAST requires as input the genome of the case-study organism, and the genome of each organism in the list of target models. This list may include all models available at the BiGG Models database, or a number of models specified by the user, leading to two different outcomes in this phase:

- Using the `ModelsGenesProteinSeq.faa` file - If the BLAST is to be performed against all BiGG models, the FASTA already created in the data structure can be directly used as a subject genome in the BLAST input.
- Creation of the `ModelsGenesProteinSeqTemp.faa` file - When only a specific set of models is required by the user, it is necessary to create a temporary FASTA file that has only the list of gene sequences present in the indicated models. The temporary FASTA follows the same structure as the previous FASTA; each sequence is associated with an identifier, ensuring that there are no repetitions. Likewise, the `SequenceIDsGenesRelationTemp.txt` file is created simultaneously to store the mapping between the generated sequence identifiers and the BiGG genes identifiers. In the case of temporary files, created upon specific requests, these are deleted as soon as the operation is completed.

When all conditions are fulfilled, the *BIT* performs BLAST similarity searches between input genome sequences (query sequences) and BiGG database sequences (subject sequences). Available Java classes, previously developed for *merlin*, that allow the execution of BLAST from the command-line using the BLAST + software (version 2.10.0), were used to perform this task. This phase is the most time-consuming, as the running time will always depend on the performance of the host machine, the size of the input genome, and the list of BiGG models chosen by the user. The following settings are used for run BLAST, by default:

- scoring matrix = BLOSUM62;
- E-value threshold = $1E^{-10}$;
- bit score threshold = 50.

The query coverage threshold value has been set to 75%, to ensure that the alignment covers most of the length of the query string. All configurations can be customized according to the user's preferences. A bidirectional BLAST is performed to decrease the probability of obtaining false positives and, consequently, increase the accuracy of the results. After aligning the case-study genome sequences against the set of sequences retrieved from BiGG Models, a reverse BLAST is performed, using the FASTA file of the input genome as the subject genome and the FASTA file containing the BiGG sequences as a query genome. After

executing the bidirectional BLAST, the alignment results obtained are processed. If the match obtained in the direct BLAST is not reciprocal in the reverse BLAST, the alignment is discarded, and the genes are not recognized as orthologous. A diagram illustrating the procedure is shown in Figure 20. For instance, the gene A1 is orthologous of genes B1 and B3, because these genes similarity has been verified both in the forward and reverse BLAST searches. The orthology between the gene A2 and gene B4 is not confirmed, because there was no match between the sequences in the reverse BLAST. Furthermore, another post-processing algorithm was implemented to verify the occurrence of perfect matches. Thus, if a query gene corresponds 100% to a subject gene, all the other BLAST hits for that query gene are discarded.

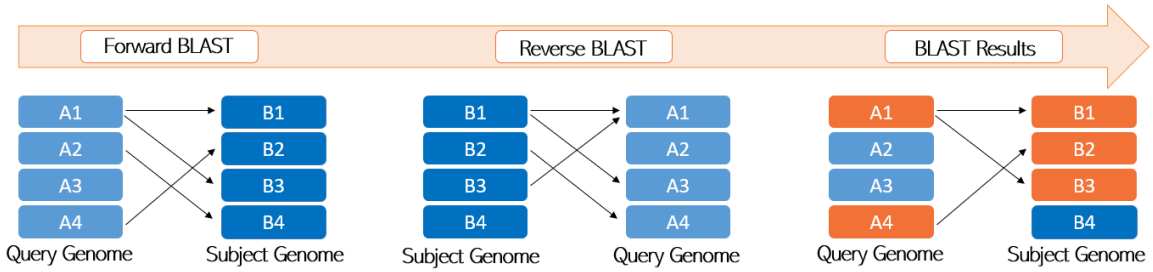


Figure 20: **Bidirectional BLAST representative schema.**

The next step is the assignment of BiGG reactions to the case-study genes, taking into account the similarities with BiGG sequences. The bidirectional BLAST returns a map containing the list of BiGG sequences identified as orthologous. The assignment of reactions to these genes is straight forward using the `SeqIDReactionRelation.txt` file of the data structure. Nevertheless, when BLAST searches are performed against gene sequences of specific models, further processing is required. First, for each sequence identifier associated with each query gene, the corresponding list of BiGG genes is retrieved from the `SequenceIDsGenesRelationTemp.txt` file. After obtaining the association of the case-study genes with BiGG genes, the reactions are retrieved using an algorithm similar to the one used to create the `GeneReactionRelation.txt` file. This process returns a map of the case-study genes to BiGG reactions, creating a draft *GSM* model.

Creating GPR associations

The gene-protein-reaction (GPR) relationships determine the set of metabolic reactions encoded in the genome. In the most simple cases, a reaction is catalyzed by a single enzyme encoded by one gene, or one enzyme can catalyze different reactions. The most complex cases are typically described using boolean rules, involving associations between various genes and proteins, as shown below, where catalytic activity is modelled by:

- 'AND' logic: The protein that catalyzes the reaction is a complex enzyme, consisting of subunits encoded by multiple genes;
- 'OR' logic: A reaction can be catalyzed by equivalent proteins that are encoded by different genes (isozymes);
- 'AND' logic and 'OR' logic.

Besides associating the genome of the case-study organism with BiGG reactions, carried out in the previous step, the *BIT* also creates, for each reaction, a boolean rule. This task involved devising an algorithm that, using the `GeneReactionRule.txt` file, creates the GPRs from the reactions' rules of each BiGG model. Due to the complexity observed in several rules annotated to BiGG reactions, and to facilitate the process of generating the new boolean rule, an algorithm that simplifies rules was implemented. Thus, the boolean expression was deconstructed in a set of 'AND' and 'OR' operations. For instance, the below rule for the BiGG reaction PI45P3K_cho:

(100689251 OR 100689252 OR 100769225) AND (100750761 OR 100771181 OR 100774067)

will result in the deconstructed rule:

100689251 AND 100750761 OR 100689251 AND 100771181 OR 100689251 AND 10077 OR 100 100689252 AND 100771181 OR 100689252 AND 100774067 OR 100769225 AND 100750761 OR 100769225 AND 100771181 OR 100769225 AND 100774067

The algorithm for the boolean rules creation is shown in the pseudo-code displayed in the Algorithm 4. For each reaction 'in model', the BiGG model rule, in its deconstructed version, is used to create the reaction rule in the new *GSM* model. Considering the BLAST results and the rule that the reaction presents, the algorithm behaves as follows:

1. BiGG rule consists only of 'OR' operators: a split is performed using the operator, to obtain a list of BiGG genes that encode the reaction. As it is a catalytic activity modeled by 'OR' logic, the similarity between case-study genes and one of the BiGG genes is enough to build the boolean rule. Thus, for each BiGG gene in the list, the corresponding case-study genes are added to the boolean rule, separated by the 'OR' operator.
2. BiGG rule consists only of 'AND' operators: the same logic described above is performed in this situation. However, similarity between case-study genes and all the genes present in the BiGG rule must exist. If the validation is successful, the case-study genes are added to the boolean rule, separated by the 'AND' operator.
3. BiGG rule formed by a single gene: there is no need to parse the rule, thus progressing directly to the validation phase. It should be noted that if there is more than one

orthologous BiGG gene in the case-study, all genes are added to the boolean rule, separated by the 'OR' operator.

4. BiGG rule consists of 'AND' and 'OR' operators: this is the most complex situation. In these cases, the BiGG rule is splitted by the 'OR' operator, obtaining a list of BiGG genes and 'AND' logic subrules. Then, the process follows the logic described in sections 2. and 3. After all validations are performed, the case-study genes are added to the final boolean rule, separated by the 'OR' operator.

Finally, the mapping of the BiGG reactions to the created boolean rules is stored in the `GPRsResults.txt` file. For the reactions whose BiGG rule validation failed, the boolean rule is discarded. The boolean rules are stored in the text file with the integer identifiers of the case-study genes, to allow the insertion of the GPRs in the user's *merlin* database in the next phase.

Algorithm 4 Create GPRs - createGPRs(arg1)

```
1: Input: biggGPRs
2:
3: finalRules = new Map();
4:
   ▷ Get all the GPRs for the reactions in model
5: reactionsRules = booleanRulesParser(biggGPRs);
6:
7: for each reactionRule in reactionsRules do
8:     reaction = reactionRule.getKey();
9:     rule = reactionRule.getValue();
10:
11:    if rule.contains("OR") then
12:        splittedByOr = rule.split("OR"));
13:        for each subRuleOr in splittedByOr do
14:            if rule.contains("AND") then
15:                splittedByAnd = rule.split("AND"));
16:                queryGenes = validateInBlastResultsAnd(splittedByAnd);
17:                if queryGenes.isNotEmpty()) then
18:                    distributionAnd(queryGenes, reaction, finalRules);
19:                end if
20:            else
21:                queryGenes = validateInBlastResultsOr(subRuleOr);
22:                if queryGenes.isNotEmpty()) then
23:                    distributionOr(queryGenes, reaction, finalRules);
24:                end if
25:            end if
26:        end for
27:    else if rule.contains("AND") then
28:        splittedByAnd = rule.split("AND"));
29:        queryGenes = validateInBlastResultsAnd(splittedByAnd);
30:        if queryGenes.isNotEmpty()) then
31:            distributionAnd(queryGenes, reaction, finalRules);
32:        end if
33:    else
34:        queryGenes = validateInBlastResultsOr(subRuleOr);
35:        if queryGenes.isNotEmpty()) then
36:            distributionOr(queryGenes, reaction, finalRules);
37:        end if
38:    end if
39: end for
40:
41: return finalRules
```

Exporting updatedBiggData database

The last processing stage of the tool involves exporting the *updatedBiggData* database. This database is created, loaded with the user's request results and exported for loading into *merlin*. Initially, the *biggdata* database dump, which contains all BiGG metabolic data, is cloned into the *updatadedBiggData* database. Before obtaining the BiGG Models database dump from the FTP server, the md5 key is downloaded, and compared with the key available at the plugin host server. If the md5 keys values are the same, the database dump has not been updated since the last operation, and can be used directly. On the other hand, a mismatch of the md5 keys will involve downloading the new database from the FTP to the host server, ensuring that the plugin uses the most up-to-date version.

After cloning all metabolic information into the *updatedBiggData* database, the similarity searches will determine which data should be present in the draft *GSM* model. For all BiGG reactions associated with the case-study genes, the Hibernate method *updateModelReaction-InModelByReactionId()* is used to update the ReactionContainer 'inModel' boolean field to 'true'. This feature allows to automatically identify all reactions, metabolites and enzymes included in the draft model. The 'boolean rule' field of all 'inModel' reactions, present in the `GPRsResults.txt` file, is also populated with the corresponding boolean rule, using the Hibernate method *updateBooleanRuleAndNotes()*. Thus, besides all metabolic data, the database stores information on the presence of data in the case study's model.

Finally, the *updatedBiggData* database that comprises the draft model, is exported into XML files. These files store the data according to the *merlin* structure, and will be used to load the users' *merlin* workspace as a result of their submission.

4.2.3 *Creating a web-server to manage the users' requests*

A web-server based on Docker, to manage user submissions, was created. This component of the *BIT* is in charge of receiving the request, executing the tool and returning the results to the user. The web-server engine was developed in Python, using Flask, a framework that provides resources to facilitate the creation of Web applications. Thus, like other Web applications, this component relies on HTTP requests at various stages of the process: submission of a request (POST request); check the status of the application throughout the process (GET request) and, obtaining the results (GET request). The web-server engine operation schema is presented in Figure 21.

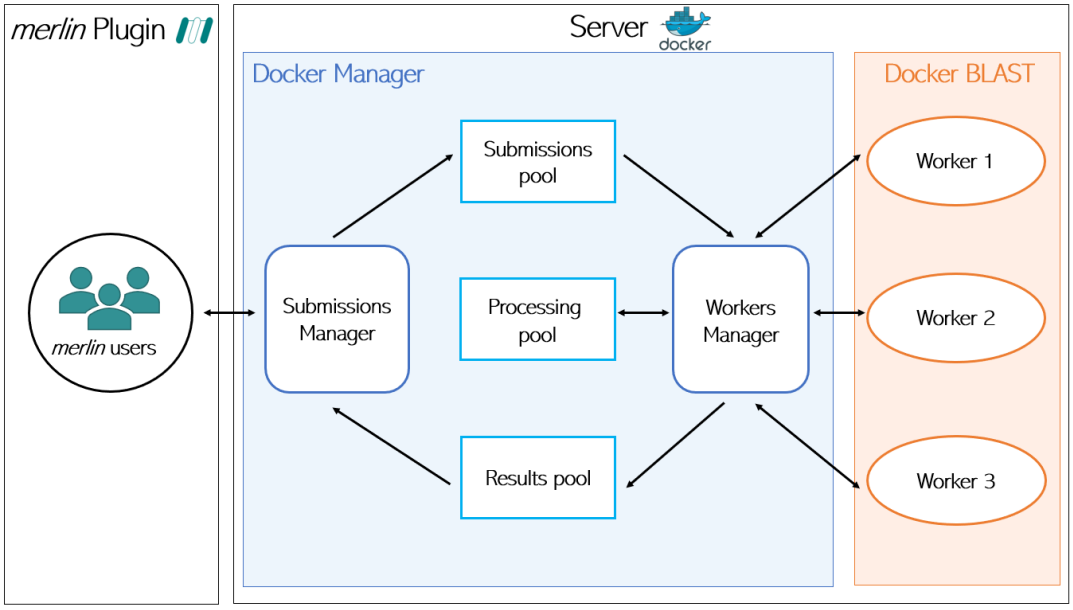


Figure 21: **Web-server workflow.**

The web-server engine is composed of two different docker images, which work independently: Docker Manager and Docker BLAST. The first receives and manages users submissions and the second is in charge of executing BLAST, processing and building the results. The replication of the Docker BLAST into multiple containers, allows the web-server to execute multiple submissions simultaneously. Users can make submissions using HTTP POST requests to the web-server, through the Submissions Manager. The submission is processed, analyzing whether all necessary inputs of the tool have been correctly sent. The submission requires:

- a file with the extension .faa (genome of the case-study organism);
- a text file `idGenes.txt`, necessary for the construction of the boolean rules (mapping of each gene to the identifier in *merlin*'s database);
- a text file `params.txt`, which includes the list of target BiGG models, the parameters E-value, bit score and query coverage, and other *BIT* settings.

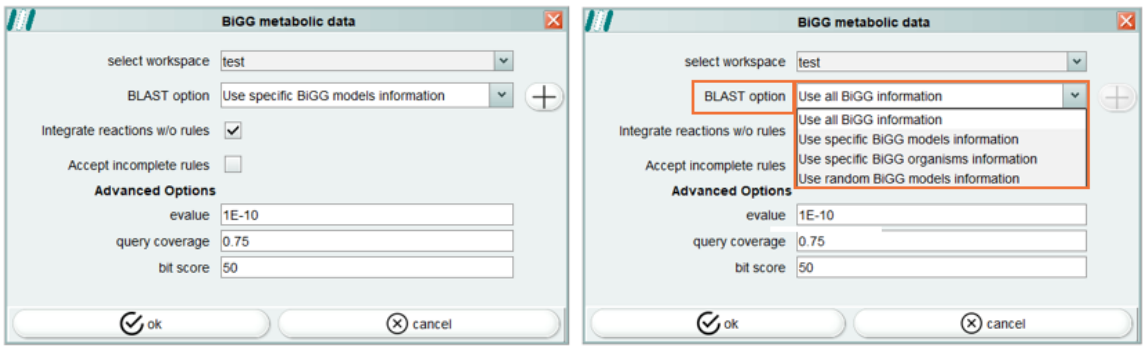
The submission returns an identifier which is stored together with the input files in the Submissions pool folder. When the Workers Manager recognizes the submission as the next submission to be processed, the manager checks the availability of a worker. When available, a Worker is assigned to the next submission in queue; if not, the submission waits in queue until a worker becomes free. Then, the Submissions pool files are moved to the Processing pool, which stores the submissions waiting for results. Right after assigning the submission

to a worker, the Workers Manager uses an HTTP POST request to transfer the input files to a folder inside the worker's container, to start the BLAST process. While the action is performed in the background, the Workers Manager performs HTTP GET requests, every ten seconds, to determine the status of the process. As previously mentioned, the BLAST phase ends with the creation of the file `GPRsResults.txt` and the export of the *updatedBiggData* database. These results are compressed into a .zip file, and a response code, indicating the completion of the process, is returned by the Workers Manager. After downloading the .zip, the Workers Manager deletes the submission from the Processing pool, creating the results folder with the submission identifier in the Results pool. Finally, the results are returned to the user, and the whole process is complete.

Both docker containers are managed through Docker Compose, allowing to define the workflow from a single configuration file. Executing BLAST and all associated processes on a server is much more efficient than running the plugin on a personal computer. Moreover, users do not need to have BLAST + installed on the machine as it will run remotely on the server. All support files required to run BLAST and build the results are obtained from the FTP server.

4.2.4 *Incorporate BiGG Integration Tool as a merlin plugin*

The *merlin-bigg* plugin was developed to create a user-friendly graphical interface that allows sending a request to *BIT*, using *merlin*. This interface allows users to interact with the tool. The *merlin-bigg* plugin adds a new option to the 'model' tab of the menu bar, namely the option 'Load BiGG metabolic data'. This operation loads the GUI that will run the entire process, taking into account the user's settings. The main layout of the GUI is shown in Figure 22a. Users need to select: the workspace; the BiGG models (BLAST option); and the BLAST parameters. As seen in Figure 22b, four BLAST options can be selected:

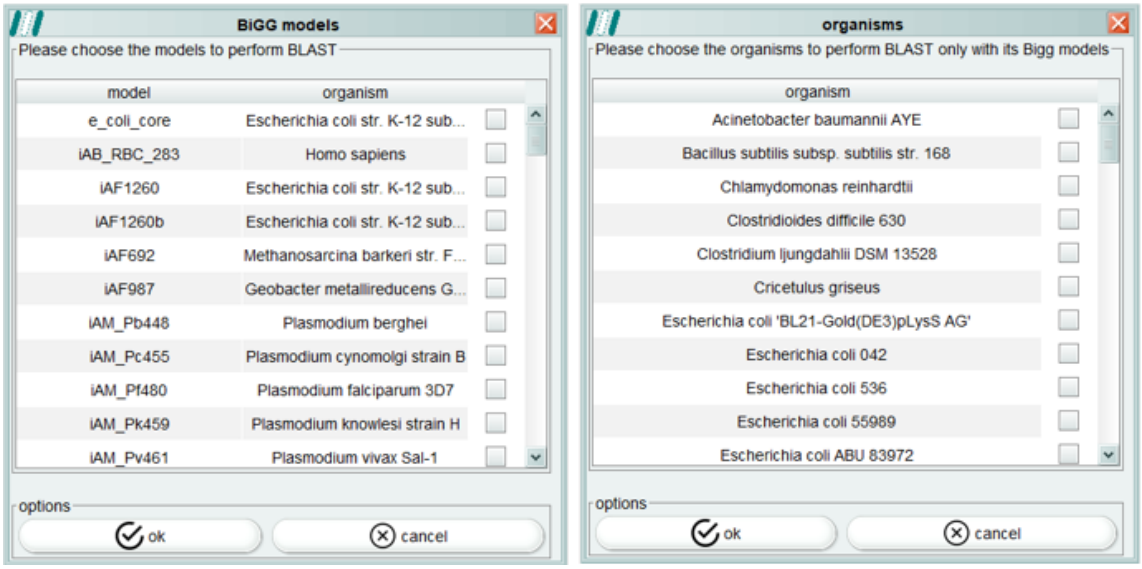


(a) **Implemented GUI to perform requests.** (b) **BLAST options available.**

Figure 22: **'BiGG metabolic data' interface.**

1. Use all BiGG information: BLAST will be performed against all the BiGG Models;
2. Use specific BiGG models information: Users will select a set of BiGG models, and the alignments will be performed against the selected models' genomes. If this option is selected, the '+' button is enabled, redirecting the users to the customized GUI 'SelectBiggModels' shown in Figure 23a, which allows the selection of the models.
3. Use specific BiGG organisms information: Option similar to the one presented in 2. However, instead of selecting models, the users can directly select organisms and, the alignments will be performed against genome sequences of organisms present in BiGG Models. Here, the '+' button is enabled, redirecting the users to the customized GUI 'SelectBiggOrganisms' present in Figure 23b, to proceed with the selection of the organisms.
4. Use random BiGG models information: When selecting this option, a set of BiGG models (between 5 and 20 models) will be randomly selected to proceed with the BLAST, which may be interesting depending on the type of study that the user is trying to carry out.

Despite the different options, cases 2., 3. and 4. require the same workflow.



(a) **Select BiGG models GUI.** (b) **Select organisms GUI.**

Figure 23: **GUIs displayed when the '+' button is clicked.**

Furthermore, users are able to choose if they want to integrate reactions not associated to genes in the selected BiGG models, and whether or not they want to add the reactions for which no boolean rule was generated by *BIT*, in the draft *GSM* network, by selecting the plugin options 'Integrate reactions without rules' or 'Accept incomplete rules', respectively.

After clicking in the 'Ok' button on the main GUI, the AIBench *LoadBiggMetabolicData* operation starts. This operation submits the request to *BIT* and retrieves the results. After analyzing users' options, the file `params.txt` is created, which contains the selected BLAST option, the values set for the BLAST parameters E-value, bit score and query coverage, and the selected plugin options. If the BLAST option mentioned above in cases 2., 3. and 4. is selected, the list of target-models must be present in the file, otherwise, all models available in BiGG Models database will be used. Additionally, the genome of the case-study organism is obtained by accessing the users' workspace folder.

The *BIT*'s input files are sent in the body of the POST request to the web-server, which will start the entire process described above. When the returned response code indicates the completion of the process, a GET request is made, to download the results to the 'bigg' folder in the users' workspace. Finally, Hibernate methods are used to perform a partial load of the *updatedBiggData* database dump into the users' database.

5

RESULTS AND DISCUSSION

5.1 BIGG INTEGRATION TOOL INTERNAL DATA STRUCTURE

BIT's internal data structure contains all metabolic data present in the BiGG Models, operating as a data source and supporting the entire tool. This structure is constantly updated always to provide BiGG's most up-to-date metabolic data. The information is stored in 108 different files, representing each model present in BiGG Models, so far. Each file provides details about the model and information related to its components. From this information, all files that support the tool are created. In detail, BiGG's internal data structure currently consists of:

- 108 text files (one for each BiGG model)

  28547 metabolic reactions with unique BiGG identifiers

  20259 reactions with GPR

  103115 genes with unique BiGG identifiers

  15721 unique metabolites (compartmentalized)

  22 compartments

- `sequenceIdsGenesRelation.txt` file

  41118 sequence identifiers generated for each unique protein sequence

- `sequenceIdsReactionsRelation.txt` file

  41118 sequence identifiers mapped to 20259 reactions

- `geneReactionRelation.txt` file

  103115 genes mapped to the 20259 reactions

- `geneReactionRule.txt` file

  boolean rules of 20259 BiGG reactions

- `modelsGenesProteinSeq.faa` file

  41118 protein sequences (1 for each generated sequence identifier)

- `biggdatabase.zip` file

  76 XML files that mirror the *merlin*'s database structure

5.2 VALIDATING THE BIGG INTEGRATION TOOL

The validation of the results returned by the *BIT* is essential to test the consistency and potential of *merlin*'s most recent plugin. Although the reconstruction and simulation of a curated model is required to assess the *BIT* model created, it is a time-consuming process, involving various steps. Thus, taking into account the objectives of this work, which consists in developing tools to increase *merlin*'s usability and improving the user experience, the *BIT*'s validation is focused mainly on developing a model for an organism present in BiGG Models, based on alignments against itself, and then comparing it to the actual BiGG model. The tool was validated for two different bacterial microorganisms: *Escherichia coli* (*E. coli*) and *Bacillus subtilis* (*B. subtilis*).

Escherichia coli

Two new draft genome-scale metabolic networks were built for *E. coli*, a well-known gram-negative anaerobic bacterium, using *merlin*'s *BIT* plugin to validate the results. The *E. coli* genome files were automatically obtained from the taxonomy identifier 511145 - NCBI's Assembly record ASM584v2, using *merlin*. After loading the genome into *merlin*, which is crucial for running *BIT*, both submissions were performed. The BLAST option 'Use specific BiGG models information' was used and only the BiGG model iAF1260 from the organism *Escherichia coli* str. K-12 substr. MG165 was selected in both submissions. It is a widely used model for genetic manipulation, highly cited by the scientific community and consists of:

- 1261 genes
- 2382 reactions (including transport reactions and pseudo-reactions)
- Gene-Protein-Reaction associations
  1944 reactions ($\approx$ 82% of model reactions)
- 1668 metabolites
- Three compartments
  extracellular space
  cytosol
  periplasm

Different settings were used to build the new *GSM* networks to validate the effect on the results. In the first submission, the 'Integrate reactions without rules' and 'Accept incomplete rules' plugin options were not enabled. After submission, 2:47 min were required to process and subsequently load the results into *merlin*. Using default BLAST configurations, the similarity search identified 1260 case-study genes, out of the 1261 present in the iAF1260 model, as orthologous. The generated draft metabolic network comprises 1260 genes,

1919 reactions (1919 GPRs) and 1634 metabolites in 3 compartments. In submission 2, the 'Integrate reactions without rules' and 'Accept incomplete rules' options were enabled. The results, loaded in 3:10 min, describe a draft metabolic network with 1260 genes, 2356 reactions (1919 with GPRs) and 1668 metabolites. Table 3 shows the comparison between the main components of the BiGG model iAF1260 and the new draft metabolic networks for *E. coli*, created by homology, verified with the genes from the iAF1260 model.

Table 3: **Main metabolic components of iAF1260 and the new draft genome-scale metabolic networks created.**

	***BIT*'s draft network 1**	***BIT*'s draft network 2**	**iAF1260**
Genes	1260	1260	1261
Reactions	1919	2356	2382
GPRs	1919	1919	1944
Metabolites	1634	1668	1668

According to table 3, the number of components in the iAF1260 model and the draft networks obtained using the *BIT* are quite similar. The most significant discrepancy is in the number of reactions integrated in the draft network 1. This result is due to the number of reactions in the iAF1260 model that have no association with BiGG genes (438 reactions). As *BIT* performs the assignment of BiGG reactions to case-study genes, taking into account the identification of sets of orthologous, these reactions cannot be identified by the tool, since the option 'Integrate reactions without rules' was not selected. Thus, focusing only on the 1944 reactions of the iAF1260 model with GPRs associations, it seems that approximately 99% of the BiGG reactions that comprise the actual pool of potential reactions, were indeed selected. In the second submission, the number of reactions integrated in the draft network increased to 2356 reactions (99% of the iAF1260 model's reactions), as all the reactions that are not associated with genes in the selected model, except the biomass reaction, were included. The remaining 25 reactions correspond to reactions associated with the s0001 gene, the only gene in the iAF1260 model not integrated in the new draft *GSM* networks. As this gene is not mapped to a genome annotation, both protein and DNA sequences are not available in BiGG Models; hence, it is not possible to identify an association with this gene using BLAST. The table S1 of the Supporting Material shows the 29 reactions associated with this gene. Note that four of these reactions (highlighted in the table) are also associated with genes other than s0001, and were integrated into both draft networks generated by *BIT*.

Bacillus subtilis

Likewise, *B. subtilis*, the best-characterised member of gram-positive bacteria, was reconstructed with *BIT*. After obtaining the *B. subtilis'* genome files (taxonomy identifier: 224308 - NCBI's Assembly record ASM904v1), two requests using different settings, were submitted with the *BIT* by selecting the model iYO844 from the *B. subtilis* subsp organism. subtilis str. 168, as the only model whose metabolic data must be used throughout the process. The BiGG model iYO844 encompasses:

- 844 genes
- 1250 reactions (including transport reactions and pseudo-reactions)
- Gene-Protein-Reaction associations
  904 reactions (≈ 72% of model reactions)
- 990 metabolites
- Two compartments
  extracellular space
  cytosol

In the first submission, the 'Integrate reactions without rules' and 'Accept incomplete rules' plugin options were not selected. The new draft metabolic network, generated in 2:10 min, consists of 840 genes, 900 reactions (900 GPRs) and 888 metabolites in 2 compartments, using the default BLAST configurations. The second draft network comprising 840 genes, 1247 reactions (900 with GPRs) and 989 metabolites, was obtained in 2:43 min. Both 'Integrate reactions without rules' and 'Accept incomplete rules' plugin options were selected, and the default BLAST configurations were used to obtain the latter results. Table 4 shows the comparison between the iYo844 model and the draft metabolic networks obtained for *B. subtilis*, using *merlin*'s *BIT*.

Table 4: **Main metabolic components of iYO844 and the new draft genome-scale metabolic networks created.**

	***BIT*'s draft network 1**	***BIT*'s draft network 2**	**iYO844**
Genes	840	840	844
Reactions	900	1247	1250
GPRs	900	900	904
Metabolites	888	989	990

The main differences between BiGG's iYO844 and *BIT*'s draft networks are the same as before. In this case, 346 reactions in the original model do not have GPRs; thus, the number of reactions available for the draft network 1, where the option 'Integrate reactions without

rules' was not selected, is 904. Hence, only 0.4% (4 reactions) of the reactions with GPRs present in the iYO844 model were not included in *BIT*'s draft network 1. On the other hand, the second draft network included, as expected, the 346 reactions not associated to genes in iYO844, exception for the biomass reaction. Furthermore, unlike submission 1, the reactions TECA4S_BS and TEICH45, for which *BIT* did not generate the Boolean rules were also included, as the option 'Accept incomplete rules' was selected. Note that there are four genes of the iYO844 model for which no orthologous were identified in BLAST (Table 5). As previously, gene BG12900 is not mapped to a genome annotation; thus, the LySLG_BS reaction was not included in any of the draft networks.

Regarding genes BSU22660, BSU35609 and BSU35690, their protein sequences are not present in the genome FASTA file retrieved from NCBI for *B. subtilis*; thus, no association with these genes was found in the BLAST search. Consequently, the IGPS reaction, uniquely associated with the BSU22660 gene, was not inserted into both new draft *GSM* networks. The TEICH45 and TECA4S_BS reactions (related to BSU35609 and BSU35690, respectively), are included in the second draft *GSM* network, as other genes for which orthologous were found are associated with this reaction. However, both reactions present a catalytic activity modelled by 'AND' logic and therefore, *BIT* was not able to generate the Boolean rules and, consequently, they were not included in the first draft *GSM* network.

Table 5: **Genes not integrated in *BIT*'s draft network**

Gene ID	Associated reactions	Gene reaction rule
BG12900	LySLG_BS	BG12900
BSU22660	IGPS	BSU22660
BSU35609	TEICH45	BSU35609 and BSU35600 and BSU35590 and BSU35570 and BSU35550 and BSU3540
BSU35690	TECA4S_BS	BSU35690 and BSU35680

5.3 CASE STUDIES

Five case studies were analysed using the *E. coli* genome (taxonomy identifier: 511145 - NCBI's Assembly record ASM584v2) to test *BIT* for different types of input. In each case, a distinct BLAST option was used to generate new draft *GSM* networks for *E. coli*, based on the association of metabolic data from different sets of BiGG models. The collection of main components that comprise the generated metabolic network (genes, reactions, metabolites and GPRs) will be analysed in each case study.

Case Study 1

In the first case study, the BLAST option 'Use specific BiGG organisms information' was used to generate a new draft *GSM* network for *E. coli*, using only the BiGG models available for the organism *E. coli* str. K-12 substr. MG1655. This common laboratory strain is the most represented organism in BiGG Models, with six *GSM* models currently available. Table 6 details the BiGG models for this strain. All models are mapped to the same genome annotation, sharing multiple reactions and metabolites. In total there are 1566 genes, 3117 reactions and 1954 metabolites with unique identifiers.

Table 6: **Main metabolic components of the BiGG models available for *Escherichia coli* str. K-12 substr. MG1655.**

BiGG model ID	Genes	Reactions	GPRs	Metabolites
e_coli_core	137	95	69	72
iAF1260	1261	2382	1944	1668
iAF1260b	1261	2388	1944	1668
iJO1366	1367	2583	2123	1805
iJR904	904	1075	873	761
iML1515	1516	2712	2266	1877

The models presented in Table 6, the default BLAST configurations, and enabling the plugin options 'Integrate reactions without rules' and 'Accept incomplete rules', allowed obtaining a draft network for *E. coli*. This model includes 1553 genes, 3075 reactions (2566 with GPRs) and 1953 metabolites. The execution time was about 2:30 min. As expected, there was a high level of similarity between the case-study genome and the selected BiGG models, with homology with 1552 BiGG genes out of the 1566 genes present in the six selected models (99% of the total number of genes). No similarity was found in the BLAST search for the 12 genes presented in Table S2. These genes are only available in the iJR904 model and are not mapped to a genome annotation. Despite that, the reactions associated with these genes were integrated in the draft *GSM* network obtained, as they are also related to other genes identified as orthologous.

Furthermore, no similarity with gene s0001 was found. This gene is present in all selected models, except for iJR904. From the reactions associated with this gene, 35 are not associated with other genes and therefore not included in the network. Gene b4104, from the iML1515 model, was also not included in the draft network, since its protein sequence is not present in the genome FASTA file retrieved from NCBI for *E. coli*, and so, no association with this gene was found in the BLAST search. Regarding the reactions integrated into the draft network (3075 reactions), 507 correspond to reactions not associated with genes in the selected models, as the option 'Integrate reactions without rules' was selected. All reactions that define the composition of the biomass (7 reactions) were not included. Finally, the only reactions for which no Boolean rule was generated by *BIT* were MEPNabcpp (b4104 + b4105 + b4106), from the iML1515 model and ARBabc (b1900 + b1898 + b1899 + b1901), from the iJR904 model. The Boolean rules for those reactions were not generated by *BIT* because both reactions present a catalytic activity modelled by 'AND' logic, and no similarity was found for genes b4104, b1898 and b1899. Despite that, both reactions were inserted due to the selection of the 'Accept incomplete rules' plugin option.

As shown in Figure 24, obtained from the analysis of *BIT*'s draft model's reactions set, 85% of the reactions (2622 reactions) are shared between the models, as expected. The remaining 15% are exclusive to four BiGG models. Thus, besides shared reactions, 264 unique reactions were provided by the iJR904 model, and 181 by the iML1515 model. Although not shown in the pie chart, four unique reactions from the iJO1366 and e_coli_core models were also identified. No single reactions from the iAF1260 and iAF1260b models have been found.

These results show that *BIT* is working as expected, as the model should include most reactions from the other models.

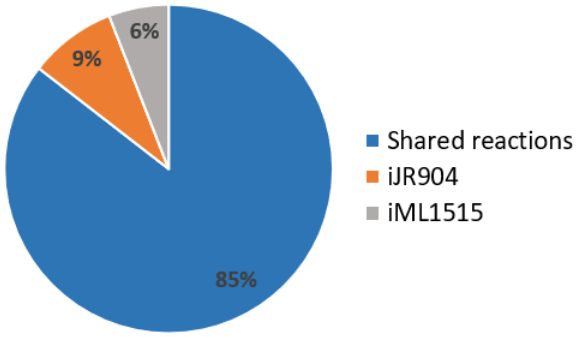


Figure 24: **Origin of the reactions incorporated in the draft network.**

Case Study 2

A new draft metabolic network was created for *E. coli*, using the BLAST option 'Use specific BiGG models information' from *BIT*. Here, three BiGG models from different strains of *E. coli* and three BiGG models of different species of *Shigella* were selected. Like *E. coli*, *Shigella* is a gram-negative and anaerobic facultative bacterial organism. The *Shigella* species are divided into four distinct serogroups (*Shigella dysenteriae*, *Shigella flexneri*, *Shigella boydii* and *Shigella sonnei*). Table 7 details the six BiGG models selected as input of the *BIT*.

Table 7: **Main metabolic components of the six BiGG models selected.**

BiGG model ID	Organism	Genes	Reactions	GPRs	Metabolites
iAPECO1_1312	*Escherichia coli* APEC O1	1313	2735	2200	1942
iUTI89_1310	*Escherichia coli* UTI89	1310	2725	2189	1940
iECS88_1305	*Escherichia coli* S88	1305	2729	2191	1942
iSF_1195	*Shigella flexneri* 2a str. 301	1195	2630	2082	1917
iSSON_1240	*Shigella sonnei* Sso46	1240	2693	2151	1936
iSBO_1134	*Shigella boydii* Sb227	1134	2591	2038	1908

After the submission, performed with the default plugin options ('Integrate reactions without rules' enabled and 'Accept incomplete rules' disabled), the results were loaded into *merlin* in approximately four minutes. Since models genetically close to *E. coli* are being used to generate the new draft *GSM* network, the query coverage parameter of the BLAST settings has been increased to 85%, to reduce the chance of an improper association of genes. With these parameters, the data returned by the *BIT* characterises a draft metabolic network formed by 1490 genes, 2708 reactions (2176 with GPRs), and 1949 metabolites. In the integrated 2708 reactions, 532 are not associated with genes in the selected BiGG models and were included due to the plugin option 'Integrate reactions without rules' being enabled. Furthermore, 27 reactions were discarded because no Boolean rule was generated by *BIT*.

The bidirectional BLAST search returned 5983 genes as orthologous of 1490 case-study genes, 550 of which were associated with the BiGG gene that corresponds to its exact match. The set of orthologous genes covers approximately 80% of the genes in the selected BiGG models (7464 BiGG genes in total).

A more detailed analysis reveals that 47% of the total of BiGG genes with similarity to the query genome belong to *Shigella*'s models. The distribution of the orthologous genes, obtained from the BiGG models, is shown in Figure 25. The percentage relative to *Shigella* species' genes is very similar to the values obtained for the *E. coli* models. Additionally, it was verified that the genes in the iSSON_1240 model from *S. sonnei* Sso46 exhibit over 85% of similarity with *E. coli* genes. This percentage proved to be higher than that obtained for all the selected BiGG models from *E. coli*, which is in line with several studies that determine

Shigella as genetically closer to *E. coli* than some of its species (Brenner et al. (1972); Van den Beld and Reubsaet (2012)).

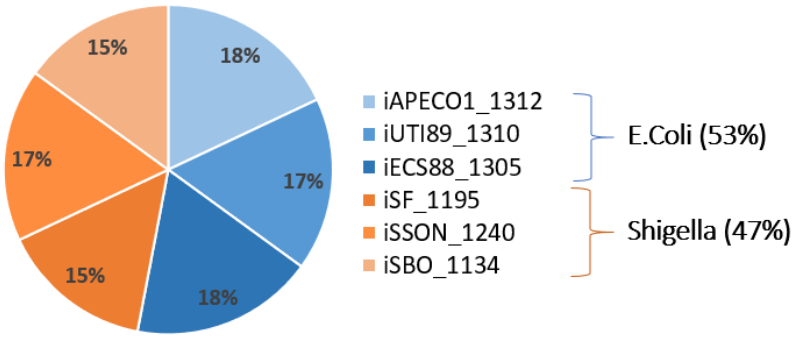


Figure 25: **Percentage of orthologous genes identified from each BiGG model.**

Case Study 3

The BLAST option 'Use specific BiGG models information' was used to build a draft *GSM* network from two BiGG models of organisms with different characteristics from *E. coli*. BiGG models iYO844, from *B. subtilis* subsp. subtilis str. 168 and iSB619, from the organism *Staphylococcus aureus* subsp. aureus N315. Both *B. subtilis* and *S. aureus* are gram-positive, facultative anaerobic bacterial organisms, and members of the phylum Firmicutes. Table 8 details both BiGG models selected in this case study. In total, there are 1463 genes, 1521 reactions and 1146 metabolites with unique BiGG identifiers.

Table 8: **Metabolic models components of BiGG models iYO844 and iSB619.**

BiGG model ID	Genes	Reactions	GPRs	Metabolites
iYO844	844	1250	904	990
iSB619	619	743	501	655

Considering that two models genetically distant from the organism under study are being used to generate the new draft network for *E. coli*, the options 'Integrate reactions without rules' and 'Accept incomplete rules' were both disabled, to ensure that only correct reactions are included. Using the default BLAST configurations, the *E. coli* draft network returned by the *BIT* comprised 809 genes, 749 reactions (749 with GPRs) and 820 metabolites. Boolean rules were not generated by *BIT* to 67 reactions; thus, none of these reactions were integrated into the draft network since the plugin option 'Accept incomplete rules' was disabled.

The bidirectional BLAST search identified similarities between 809 *E. coli* genes and 979 genes of the selected BiGG models. A more detailed analysis revealed that the case study's genome matched 68% and 65% of the iYO844 and iSB619 models' genes, respectively.

Moreover, 43% and 16% of the reactions of the draft network were inferred from the iYO844 and iSB619 models, respectively. The remaining 41% are reactions present in both BiGG models (Figure 26). In the total integrated reactions (749), approximately 60% corresponds to reactions shared with BiGG models from *E. coli* str. K-12 substr. MG1655.

This draft *GSM* network has a lower level of similarity when compared with the previous case studies, which was expected given the distinct characteristics between *E. coli* and the selected bacterias.

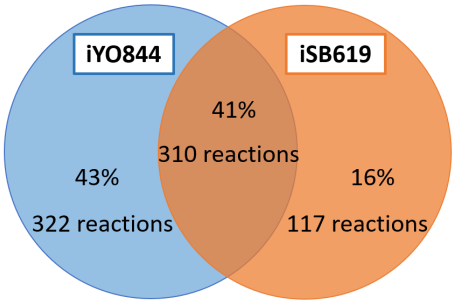


Figure 26: **Reactions incorporated in the draft network from iYO844 and iSB619.**

Case Study 4

The next case study evaluates the influence of using all metabolic data available in BiGG Models database, to build a draft network from *E. coli*'s genome. Currently, the BiGG Models cover more than 85 different species, for a total of 108 models. These include 103115 genes (39843 single sequence genes), 28547 reactions and 15721 metabolites with unique identifiers. The number of alignments increased considerably due to the large number of target genes (39843), which led to an extended execution time of about ten minutes (more than twice the processing time of the previous submissions).

The submission was performed with both options 'Integrate reactions without rules' and 'Accept incomplete rules' disabled and the query coverage was increased to 85%, to reduce the chance of an incorrect association of genes.*E. coli*'s new draft *GSM* network is comprised of 1886 genes, 3581 reactions (3581 GPRs) and 3019 compartmentalised metabolites. Boolean rule were not generated for 283 reactions; hence, these reactions were discarded by *BIT*, as the plugin option 'Accept incomplete rules' was disabled. Table 9 shows the ten models with the highest percentage of identified orthologous. As expected, all models belong to the *E. coli* species (similarities above 94%), with all models currently available in the BiGG Models for *Escherichia coli*, strain K-12, taking the top positions of the table. Among models not belonging to *E. coli* species, the highest percentage obtained was for the iSSON_1240 model from *S. sonnei* Sso46, that exhibited a percentage of identified orthologous even higher than some *E. coli* species, as shown in Case Study 2. Regarding the models that presented lower percentages of similarity, all belong to organisms genetically distant from *E. coli* (Table

10). Among the ten models with less similarity, models from mammals (iMM1415) and unicellular eukaryotes (iAM_Pf480, iAM_Pc455, iAM_Pb448, iAM_Pv461, iND750, iMM904, iLB1027_lipid, iIS312_Trypomastigote, iIS312_Epimastigote) were found.

The similarity search returned 33326 BiGG genes (4275 single sequences) identified as orthologous of 1886 *E.coli* genes. In the bidirectional BLAST search a perfect match was found for 1594 case-study genes.

Of the 108 BiGG models, 58 belong to different strains of *E. coli* species, which justifies the high number of reactions added to the draft metabolic network (3581 reactions). Furthermore, in BiGG Models, there are redundant reactions, i.e., reactions that are associated with different identifiers but represent the same chemical equation. The set of integrated reactions encompassed 3100 unique reactions. In the next iteration of *BIT*, a strategy should be developed to allow only unique reactions to be integrated into the draft *GSM* network.

Table 9: **BiGG models with the highest similarity percentage obtained.**

BiGG model ID	Organism	Genes	Similarity %
iJO1366	*Escherichia coli* str. K-12 substr. MG1655	1367	99,78%
iAF1260	*Escherichia coli* str. K-12 substr. MG1655	1261	99,76%
iAF1260b	*Escherichia coli* str. K-12 substr. MG1655	1261	99,76%
iML1515	*Escherichia coli* str. K-12 substr. MG1655	1516	99,74%
e_coli_core	*Escherichia coli* str. K-12 substr. MG1655	137	99,27%
iJR904	*Escherichia coli* str. K-12 substr. MG1655	904	98,67%
iEC1372_W3110	*Escherichia coli* str. K-12 substr. W3110	1372	96,28%
iY75_1357	*Escherichia coli* str. K-12 substr. W3110	1358	96,24%
iECDH10B_1368	*Escherichia coli* str. K-12 substr. DH10B	1327	94,88%
iBWG_1329	*Escherichia coli* BW2952	1329	94,81%

Table 10: **BiGG models with the lowest similarity percentage obtained.**

BiGG model ID	Organism	Genes	Similarity %
iAM_Pf480	*Plasmodium falciparum* 3D7	480	0,42%
iAM_Pb448	*Plasmodium berghei*	448	0,45%
iAM_Pv461	*Plasmodium vivax* Sal-1	461	0,65%
iAM_Pc455	*Plasmodium cynomolgi* strain B	455	0,66%
iMM1415	*Mus musculus*	1375	1,16%
iND750	*Saccharomyces cerevisiae* S288C	750	1,20%
iMM904	*Saccharomyces cerevisiae* S288C	905	1,33%
iLB1027_lipid	*Phaeodactylum tricornutum* CCAP 1055/1	1027	1,46%
iIS312_Trypomastigote	*Trypanosoma cruzi* Dm28c	312	1,60%
iIS312_Epimastigote	*Trypanosoma cruzi* Dm28c	312	1,60%

Case Study 5

In the last case study, a random set of BiGG models was selected (between 5 to 20 different models). This option can be very interesting for particular studies. Three analysis were performed with *BIT* to generate three new draft *GSM* networks. The results obtained in each submission are shown in Table 11. The default BLAST configurations and the default plugin options ('Integrate reactions without rules' enabled and 'Accept incomplete rules' disabled), were used to perform all the submissions.

Table 11: **BiGG data incorporated in the new *E. coli* draft networks.**

(a) iAT_PLT_636 (*Homo sapiens*); iAM_Pf480 (*Plasmodium falciparum* 3D7); iIS312 (*Trypanosoma cruzi* Dm28c); iSSON_1240 (*Shigella sonnei* Ss046); iND750 (*Saccharomyces cerevisiae* S288C); iIS312_Amastigote (*Trypanosoma cruzi* Dm28c)

(b) iEC1349_Crooks (*Escherichia coli* ATCC 8739); e_coli_core (*Escherichia coli* str. K-12 substr. MG1655); iEC1364_W (*Escherichia coli* W); iJR904 (*Escherichia coli* str. K-12 substr. MG1655); iIS312 (*Trypanosoma cruzi* Dm28c); iECIAI1_1343 (*Escherichia coli* IAI1); iECUMN_1333(*Escherichia coli* UMN026); iEC042_1314 (*Escherichia coli* 042); iECB_1328 (*Escherichia coli* B str. REL606); iAF1260 (*Escherichia coli* str. K-12 substr. MG1655)

(c) iS_1188 (*Shigella flexneri* 2a str. 2457T); iSF_1195 (*Shigella flexneri* 2a str. 301); iAM_Pv461 (*Plasmodium vivax* Sal-1); iETEC_1333 (*Escherichia coli* ETEC H10407); iEC1364_W (*Escherichia coli* W); iECW_1372 (*Escherichia coli* W); iEcDH1_1363 (*Escherichia coli* DH1); iEC1372_W3110 (*Escherichia coli* str. K-12 substr. W3110)

	Selected Models	Genes	Reactions	GPRs	Metabolites
Submission 5.1	(a)	1419	3682	2373	2385
Submission 5.2	(b)	1571	3261	2542	2186
Submission 5.3	(c)	1629	3173	2370	2323

Submission 5.1 returned a draft network comprised of 1419 genes, 3682 reactions (2373 GPRs) and 2835 metabolites, created from the random selection of six BiGG models. The high number of 'in model' reactions is due to the integration of reactions not associated with genes in the selected BiGG models (1309 reactions). In the *GSM* model curation, the user can decide whether to maintain these reactions. None of the selected models belong to the *E. coli* species. However, in Case Study 2, the high level of similarity between *S. sonnei* Ss046 (iSSON_1240 model) and *E. coli* had already been determined. All other models belong to organisms genetically distant from *E. coli*, and therefore with much lower levels of similarity.

Furthermore, the bidirectional BLAST search identified 350 genes that correspond to its exact match in the BiGG models, being all of these iSSON_1240 genes. Consequently, over 57% of the draft *GSM* network reactions originate from the iSSON_1240 model. Likewise, 75% (1199) of the genes identified as orthologous to *E. coli*'s genes (1605), belong to *S. sonnei* Ss046; thus, the remaining genes are shared between the other five models.

Submissions 5.2 and 5.3 are very similar, as most models randomly selected by *BIT* are closely related to *E. coli*. Regarding submission 5.2, all models belong to *E. coli* and *Shigella*

species, except for iIS312. This model belongs to an organism with low similarity to *E. coli*, and consequently, only about 9% of the draft network reactions are from the iIS312 model.

Similarly, in submission 5.3, only the iAM_Pv461 model belongs to a distant organism (*Plasmodium vivax* Sal-1), which provided only 17% of the draft model's reactions. Again, the high number of included reactions for submissions 5.2 and 5.3 is due to the integration of reactions without GPR in the selected BiGG models (719 and 803 reactions, respectively).

Despite the different number of models selected by the tool in each submission, the three draft *GSM* networks have similar size and characteristics. As all submissions registered at least one model with a high similarity to *E. coli*, sharing multiple reactions and metabolites between each other. As illustrated in the Venn diagram shown in Figure 27, 2571 reactions are simultaneously present in the three generated draft networks.

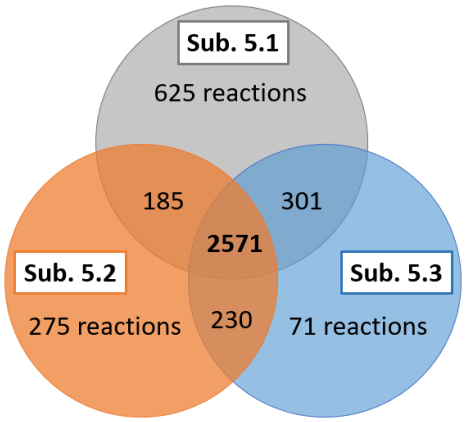


Figure 27: **Shared reactions between the three draft networks obtained.**

6

CONCLUSION

6.1 CONCLUSIONS

The work presented in this thesis aimed to develop new tools, as well as improve existing ones, in the software for the reconstruction of genome-scale metabolic models *merlin*. This work allowed enhancing *merlin*'s capabilities, providing a more comfortable use of the software, especially for information technologies non-specialists. The implementation of general platform improvements, described in this work, as well as the integration of the *merlin-settings* plugin, which allows the users to manage the configuration files within *merlin*, and the *merlin-exporter* plugin, which offers the users the possibility of backing up and restoring a workspace, prove to be very useful. As a whole, these implementations improved the software user-interface, freed users from repetitive tasks and from losing information gathered during the reconstruction process.

The main highlight of this work is *BIT*. This tool allows reconstructing draft networks from available BiGG *GSM* models, by accessing the BiGG Models database, and storing this information in an automatically updatable data structure. The *merlin-bigg* plugin allows users to use *BIT* with different configurations, namely how many and which models to use. The results are imported, in minutes, to the selected workspace. The integration of the *BIT* in *merlin* allows to reduce the requirement of the software on KEGG, as this was the main available option to retrieve metabolic data in the model assembly phase. The *BIT* is available at the GitLab repository at `https://gitlab.bio.di.uminho.pt/bigg-tool`.

6.2 LIMITATIONS AND FUTURE WORK

Although the main objectives of this work have been achieved, there are always improvements that can be made. Regarding the *BIT*, the presented implementation faces some limitations that may be overcome in a future work:

- Given the lack of information in BiGG Models regarding the pathways hierarchy, it was not possible to fill the PathwaysHierarchyContainer *merlin* container, and therefore, insertion of pathways in the database.
- Some genes in BiGG Models are not mapped to a genome annotation, with no protein sequence available, so reactions associated with these genes will not be included in the draft *GSM* network.
- A strategy should be developed so that only unique reactions are integrated into the draft *GSM* network generated by *BIT*.

Furthermore, the reconstruction of a *GSM* model using *BIT*, and its analysis, seems mandatory to validate the quality and consistency of models built with BiGG metabolic data.

BIBLIOGRAPHY

Rasmus Agren, Liming Liu, Saeed Shoaie, Wanwipa Vongsangnak, Intawat Nookaew, and Jens Nielsen. The raven toolbox and its use for generating a genome-scale metabolic model for penicillium chrysogenum. *PLoS computational biology*, 9(3), 2013.

Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

Rolf Apweiler, Amos Bairoch, Cathy H Wu, Winona C Barker, Brigitte Boeckmann, Serenella Ferro, Elisabeth Gasteiger, Hongzhan Huang, Rodrigo Lopez, Michele Magrane, et al. Uniprot: the universal protein knowledgebase. *Nucleic acids research*, 32(suppl_1):D115–D119, 2004.

Fankar Armash Aslam, Hawa Nabeel Mohammed, Jummal Musab Mohd, Murade Aaraf Gulamgaus, and PS Lok. Efficient way of web development using python and flask. *International Journal of Advanced Research in Computer Science*, 6(2), 2015.

Ramy K Aziz, Daniela Bartels, Aaron A Best, Matthew DeJongh, Terrence Disz, Robert A Edwards, Kevin Formsma, Svetlana Gerdes, Elizabeth M Glass, Michael Kubal, et al. The rast server: rapid annotations using subsystems technology. *BMC genomics*, 9(1):75, 2008.

Harald Barsnes, Marc Vaudel, and Lennart Martens. Jsparklines: Making tabular proteomics data come alive. *Proteomics*, 15(8):1428–1431, 2015.

Christian Bauer and Gavin King. *Hibernate in action*, volume 4. Manning Greenwich CT, 2005.

Deepak Bhatnagar, Kanniah Rajasekaran, Gary Payne, Robert Brown, Jiujiang Yu, and T(2008) Cleveland. The'omics' tools: genomics, proteomics, metabolomics and their potential for solving the aflatoxin contamination problem. *World Mycotoxin Journal*, 1(1):3–12, 2008.

Joost Boele, Brett G Olivier, and Bas Teusink. Fame, the flux analysis and modeling environment. *BMC systems biology*, 6(1):8, 2012.

Davide Bolchini, Anthony Finkelstein, Vito Perrone, and Sylvia Nagl. Better bioinformatics through usability analysis. *Bioinformatics*, 25(3):406–412, 2009.

Tim Bray. The javascript object notation (json) data interchange format. *RFC*, 7158:1–16, 2014.

Don J Brenner, GR Fanning, AG Steigerwalt, I Ørskov, and F Ørskov. Polynucleotide sequence relatedness among three groups of pathogenic escherichia coli strains. *Infection and immunity*, 6(3):308–315, 1972.

Thomas Brettin, James J Davis, Terry Disz, Robert A Edwards, Svetlana Gerdes, Gary J Olsen, Robert Olson, Ross Overbeek, Bruce Parrello, Gordon D Pusch, et al. Rasttk: a modular and extensible implementation of the rast algorithm for building custom annotation pipelines and annotating batches of genomes. *Scientific reports*, 5:8365, 2015.

Luis Caspeta, Saeed Shoaie, Rasmus Agren, Intawat Nookaew, and Jens Nielsen. Genome-scale metabolic reconstructions of pichia stipitis and pichia pastoris and in silico evaluation of their potentials. *BMC systems biology*, 6(1):24, 2012.

Ron Caspi, Hartmut Foerster, Carol A Fulcher, Pallavi Kaipa, Markus Krummenacker, Mario Latendresse, Suzanne Paley, Seung Y Rhee, Alexander G Shearer, Christophe Tissier, et al. The metacyc database of metabolic pathways and enzymes and the biocyc collection of pathway/genome databases. *Nucleic acids research*, 36(suppl_1):D623–D631, 2007.

Ron Caspi, Richard Billington, Carol A Fulcher, Ingrid M Keseler, Anamika Kothari, Markus Krummenacker, Mario Latendresse, Peter E Midford, Quang Ong, Wai Kit Ong, et al. The metacyc database of metabolic pathways and enzymes. *Nucleic acids research*, 46(D1): D633–D639, 2018.

Minh Thanh Chung, Nguyen Quang-Hung, Manh-Thin Nguyen, and Nam Thoai. Using docker in high performance computing applications. In *2016 IEEE Sixth International Conference on Communications and Electronics (ICCE)*, pages 52–57. IEEE, 2016.

Gene Ontology Consortium. Gene ontology consortium: going forward. *Nucleic acids research*, 43(D1):D1049–D1056, 2015.

Ileana M Cristea, Simon J Gaskell, and Anthony D Whetton. Proteomics techniques and their application to hematology. *Blood*, 103(10):3624–3634, 2004.

Zongjie Dai and Jens Nielsen. Advancing metabolic engineering through systems biology of industrial microorganisms. *Current opinion in biotechnology*, 36:8–15, 2015.

Emek Demir, Michael P Cary, Suzanne Paley, Ken Fukuda, Christian Lemer, Imre Vastrik, Guanming Wu, Peter D'eustachio, Carl Schaefer, Joanne Luciano, et al. The biopax community standard for pathway data sharing. *Nature biotechnology*, 28(9):935–942, 2010.

Oscar Dias, Andreas K Gombert, Eugénio C Ferreira, and Isabel Rocha. Genome-wide metabolic (re-) annotation of kluyveromyces lactis. *BMC genomics*, 13(1):517, 2012.

Oscar Dias, Miguel Rocha, Eugénio C Ferreira, and Isabel Rocha. Reconstructing genome-scale metabolic models with merlin. *Nucleic acids research*, 43(8):3899–3910, 2015.

Oscar Dias, Daniel Gomes, Paulo Vilaça, João Cardoso, Miguel Rocha, Eugénio C Ferreira, and Isabel Rocha. Genome-wide semi-automated annotation of transporter systems. *IEEE/ACM transactions on computational biology and bioinformatics*, 14(2):443–456, 2016.

Oscar Dias, Miguel Rocha, Eugénio Campos Ferreira, and Isabel Rocha. Reconstructing high-quality large-scale metabolic models with merlin. In *Metabolic Network Reconstruction and Modeling*, pages 1–36. Springer, 2018.

Sean R. Eddy. Profile hidden markov models. *Bioinformatics (Oxford, England)*, 14(9):755–763, 1998.

Jeremy S Edwards and Bernhard O Palsson. Systems properties of the haemophilus influenzaerd metabolic genotype. *Journal of Biological Chemistry*, 274(25):17410–17416, 1999.

Jeremy S Edwards, Markus Covert, and Bernhard Palsson. Metabolic modelling of microbes: the flux-balance approach. *Environmental microbiology*, 4(3):133–140, 2002.

Xueyang Feng, You Xu, Yixin Chen, and Yinjie J Tang. Microbesflux: a web platform for drafting metabolic models from the kegg database. *BMC systems biology*, 6(1):94, 2012.

Jon Ferraiolo, Fujisawa Jun, and Dean Jackson. *Scalable vector graphics (SVG) 1.0 specification*. iuniverse Bloomington, 2000.

Roy Fielding, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, and Tim Berners-Lee. Rfc2068: Hypertext transfer protocol–http/1.1, 1997.

Robert D Finn, Alex Bateman, Jody Clements, Penelope Coggill, Ruth Y Eberhardt, Sean R Eddy, Andreas Heger, Kirstie Hetherington, Liisa Holm, Jaina Mistry, et al. Pfam: the protein families database. *Nucleic acids research*, 42(D1):D222–D230, 2014.

Robert D Fleischmann, Mark D Adams, Owen White, Rebecca A Clayton, Ewen F Kirkness, Anthony R Kerlavage, Carol J Bult, Jean-Francois Tomb, Brian A Dougherty, Joseph M Merrick, et al. Whole-genome random sequencing and assembly of haemophilus influenzae rd. *Science*, 269(5223):496–512, 1995.

Ronan MT Fleming, Ines Thiele, and HP Nasheuer. Quantitative assignment of reaction directionality in constraint-based models of metabolism: application to escherichia coli. *Biophysical chemistry*, 145(2-3):47–56, 2009.

Christof Francke, Roland J Siezen, and Bas Teusink. Reconstructing the metabolic network of a bacterium from its genome. *Trends in microbiology*, 13(11):550–558, 2005.

Dirk Gevers, Klaas Vandepoele, Cedric Simillion, and Yves Van de Peer. Gene duplication and biased functional retention of paralogs in bacterial genomes. *Trends in microbiology*, 12 (4):148–154, 2004.

Daniel Glez-Peña, Miguel Reboiro-Jato, Paulo Maia, Miguel Rocha, Fernando Díaz, and Florentino Fdez-Riverola. Aibench: a rapid application development framework for translational research in biomedicine. *Computer methods and programs in biomedicine*, 98(2): 191–203, 2010.

Vladimir Gligorijević and Nataša Pržulj. Methods for biological data integration: perspectives and challenges. *Journal of the Royal Society Interface*, 12(112):20150571, 2015.

Joshua J Hamilton and Jennifer L Reed. Software platforms to facilitate reconstructing genome-scale metabolic networks. *Environmental microbiology*, 16(1):49–59, 2014.

Christopher S Henry, Matthew DeJongh, Aaron A Best, Paul M Frybarger, Ben Linsay, and Rick L Stevens. High-throughput generation, optimization and analysis of genome-scale metabolic models. *Nature biotechnology*, 28(9):977, 2010.

Richard P Horgan and Louise C Kenny. 'omic'technologies: genomics, transcriptomics, proteomics and metabolomics. *The Obstetrician & Gynaecologist*, 13(3):189–195, 2011.

Paul Horton, Keun-Joon Park, Takeshi Obayashi, Naoya Fujita, Hajime Harada, CJ Adams-Collier, and Kenta Nakai. Wolf psort: protein localization predictor. *Nucleic acids research*, 35(suppl_2):W585–W587, 2007.

Michael Hucka, Andrew Finney, Herbert M Sauro, Hamid Bolouri, John C Doyle, Hiroaki Kitano, Adam P Arkin, Benjamin J Bornstein, Dennis Bray, Athel Cornish-Bowden, et al. The systems biology markup language (sbml): a medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–531, 2003.

Jaime Huerta-Cepas, Damian Szklarczyk, Kristoffer Forslund, Helen Cook, Davide Heller, Mathias C Walter, Thomas Rattei, Daniel R Mende, Shinichi Sunagawa, Michael Kuhn, et al. eggnog 4.5: a hierarchical orthology framework with improved functional annotations for eukaryotic, prokaryotic and viral sequences. *Nucleic acids research*, 44(D1):D286–D293, 2016.

Jaime Huerta-Cepas, Kristoffer Forslund, Luis Pedro Coelho, Damian Szklarczyk, Lars Juhl Jensen, Christian Von Mering, and Peer Bork. Fast genome-wide functional annotation through orthology assignment by eggnog-mapper. *Molecular biology and evolution*, 34(8): 2115–2122, 2017.

Jaime Huerta-Cepas, Damian Szklarczyk, Davide Heller, Ana Hernández-Plaza, Sofia K Forslund, Helen Cook, Daniel R Mende, Ivica Letunic, Thomas Rattei, Lars J Jensen, et al.

eggnog 5.0: a hierarchical, functionally and phylogenetically annotated orthology resource based on 5090 organisms and 2502 viruses. *Nucleic acids research*, 47(D1):D309–D314, 2019.

Matthew D Jankowski, Christopher S Henry, Linda J Broadbelt, and Vassily Hatzimanikatis. Group contribution method for thermodynamic analysis of complex metabolic networks. *Biophysical journal*, 95(3):1487–1499, 2008.

Lars Juhl Jensen, Philippe Julien, Michael Kuhn, Christian von Mering, Jean Muller, Tobias Doerks, and Peer Bork. eggnog: automated construction and annotation of orthologous groups of genes. *Nucleic acids research*, 36(suppl_1):D250–D254, 2007.

Minoru Kanehisa and Susumu Goto. Kegg: kyoto encyclopedia of genes and genomes. *Nucleic acids research*, 28(1):27–30, 2000.

Minoru Kanehisa, Susumu Goto, Yoko Sato, Masayuki Kawashima, Miho Furumichi, and Mao Tanabe. Data, information, knowledge and principle: back to metabolism in kegg. *Nucleic acids research*, 42(D1):D199–D205, 2014.

Peter D Karp, Suzanne Paley, and Pedro Romero. The pathway tools software. *Bioinformatics*, 18(suppl_1):S225–S232, 2002.

Peter D Karp, Suzanne M Paley, Markus Krummenacker, Mario Latendresse, Joseph M Dale, Thomas J Lee, Pallavi Kaipa, Fred Gilham, Aaron Spaulding, Liviu Popescu, et al. Pathway tools version 13.0: integrated software for pathway/genome informatics and systems biology. *Briefings in bioinformatics*, 11(1):40–79, 2010.

Peter D Karp, Mario Latendresse, Suzanne M Paley, Markus Krummenacker, Quang D Ong, Richard Billington, Anamika Kothari, Daniel Weaver, Thomas Lee, Pallavi Subhraveti, et al. Pathway tools version 19.0 update: software for pathway/genome informatics and systems biology. *Briefings in bioinformatics*, 17(5):877–890, 2016.

Tae Yong Kim, Seung Bum Sohn, Hyun Uk Kim, and Sang Yup Lee. Strategies for systems-level metabolic engineering. *Biotechnology Journal: Healthcare Nutrition Technology*, 3(5): 612–623, 2008.

Zachary A King, Andreas Dräger, Ali Ebrahim, Nikolaus Sonnenschein, Nathan E Lewis, and Bernhard O Palsson. Escher: a web application for building, sharing, and embedding data-rich visualizations of biological pathways. *PLoS computational biology*, 11(8), 2015.

Zachary A King, Justin Lu, Andreas Dräger, Philip Miller, Stephen Federowicz, Joshua A Lerman, Ali Ebrahim, Bernhard O Palsson, and Nathan E Lewis. Bigg models: A platform for integrating, standardizing and sharing genome-scale models. *Nucleic acids research*, 44 (D1):D515–D522, 2016.

Hiroaki Kitano. *Foundations of systems biology*. The MIT Press Cambridge, Massachusetts London, England, 2001.

Hiroaki Kitano. Computational systems biology. *Nature*, 420(6912):206–210, 2002.

Amornpan Klanchui, Chiraphan Khannapho, Atchara Phodee, Supapon Cheevadhanarak, and Asawin Meechai. i ak692: A genome-scale metabolic model of spirulina platensis c1. *BMC systems biology*, 6(1):71, 2012.

Eugene V Koonin. Orthologs, paralogs, and evolutionary genomics. *Annu. Rev. Genet.*, 39: 309–338, 2005.

David M Kristensen, Yuri I Wolf, Arcady R Mushegian, and Eugene V Koonin. Computational methods for gene orthology inference. *Briefings in bioinformatics*, 12(5):379–391, 2011.

Akhil Kumar, Patrick F Suthers, and Costas D Maranas. Metrxn: a knowledgebase of metabolites and reactions spanning metabolic models and databases. *BMC bioinformatics*, 13(1):6, 2012.

Nicolas Le Novère, Andrew Finney, Michael Hucka, Upinder S Bhalla, Fabien Campagne, Julio Collado-Vides, Edmund J Crampin, Matt Halstead, Edda Klipp, Pedro Mendes, et al. Minimum information requested in the annotation of biochemical models (miriam). *Nature biotechnology*, 23(12):1509–1515, 2005.

Ivica Letunic, Tobias Doerks, and Peer Bork. Smart: recent updates, new developments and status in 2015. *Nucleic acids research*, 43(D1):D257–D260, 2015.

Nathan E Lewis, Kim K Hixson, Tom M Conrad, Joshua A Lerman, Pep Charusanti, Ashoka D Polpitiya, Joshua N Adkins, Gunnar Schramm, Samuel O Purvine, Daniel Lopez-Ferrer, et al. Omic data from evolved e. coli are consistent with computed optimal growth from genome-scale models. *Molecular systems biology*, 6(1), 2010.

Xu Li, Lina Chen, Liangcai Zhang, Wan Li, Xu Jia, Weiguo Li, Xiaoli Qu, Jingxie Tai, Chenchen Feng, Fan Zhang, et al. Rcm: A novel association approach to search for coronary artery disease genetic related metabolites based on snps and metabolic network. *Genomics*, 100(5):282–288, 2012.

Markus List. Using docker compose for the simple deployment of an integrated drug target screening platform. *Journal of integrative bioinformatics*, 14(2), 2017.

Hugo López-Fernández, Miguel Reboiro-Jato, Daniel Glez-Peña, José R Méndez Reboredo, Hugo M Santos, Ricardo J Carreira, José L Capelo-Martínez, and Florentino Fdez-Riverola. Rapid development of proteomic applications with the aibench framework. *Journal of Integrative Bioinformatics*, 8(3):16–30, 2011.

Hugo López-Fernández, Miguel Reboiro-Jato, Daniel Glez-Peña, Rosalía Laza, Reyes Pavón, and Florentino Fdez-Riverola. Gc4s: A bioinformatics-oriented java software library of reusable graphical user interface components. *PloS one*, 13(9), 2018.

Tom Madden. Chapter 16 : The blast sequence analysis tool. 2013.

Paulo Maia, Miguel Rocha, and Isabel Rocha. In silico constraint-based strain optimization methods: the quest for optimal cell factories. *Microbiol. Mol. Biol. Rev.*, 80(1):45–67, 2016.

Harald Mischak, Rolf Apweiler, Rosamonde E Banks, Mark Conaway, Joshua Coon, Anna Dominiczak, Jochen HH Ehrich, Danilo Fliser, Mark Girolami, Henning Hermjakob, et al. Clinical proteomics: a need to define the field and to begin to set adequate standards. *PROTEOMICS–Clinical Applications*, 1(2):148–156, 2007.

Rajesh Nair and Burkhard Rost. Mimicking cellular sorting improves prediction of subcellular localization. *Journal of molecular biology*, 348(1):85–100, 2005.

Jens Nielsen and Sabine Arnold. *Biotechnology for the Future*. Springer Science & Business Media, 2005.

Jens Nielsen and Michael C Jewett. Impact of systems biology on metabolic engineering of saccharomyces cerevisiae. *FEMS yeast research*, 8(1):122–131, 2008.

José Manuel Otero and Jens Nielsen. Industrial systems biology. *Biotechnology and bioengineering*, 105(3):439–460, 2010.

Ross Overbeek, Tadhg Begley, Ralph M Butler, Jomuna V Choudhuri, Han-Yu Chuang, Matthew Cohoon, Valérie de Crécy-Lagard, Naryttza Diaz, Terry Disz, Robert Edwards, et al. The subsystems approach to genome annotation and its use in the project to annotate 1000 genomes. *Nucleic acids research*, 33(17):5691–5702, 2005.

Ross Overbeek, Robert Olson, Gordon D Pusch, Gary J Olsen, James J Davis, Terry Disz, Robert A Edwards, Svetlana Gerdes, Bruce Parrello, Maulik Shukla, et al. The seed and the rapid annotation of microbial genomes using subsystems technology (rast). *Nucleic acids research*, 42(D1):D206–D214, 2014.

Stephan Pabinger, Robert Rader, Rasmus Agren, Jens Nielsen, and Zlatko Trajanoski. Memosys: Bioinformatics platform for genome-scale metabolic models. *BMC systems biology*, 5 (1):20, 2011.

Bernhard Ø Palsson. *Systems biology: properties of reconstructed networks*. Cambridge university press, 2006.

Yasset Perez-Riverol, Rui Wang, Henning Hermjakob, Markus Müller, Vladimir Vesada, and Juan Antonio Vizcaíno. Open source libraries and frameworks for mass spectrometry based proteomics: a developer's perspective. *Biochimica et Biophysica Acta (BBA)-Proteins and Proteomics*, 1844(1):63–76, 2014.

Dina Petranovic and Goutham N Vemuri. Impact of yeast systems biology on industrial biotechnology. *Journal of biotechnology*, 144(3):204–211, 2009.

Thomas Pfau, Maria Pires Pacheco, and Thomas Sauter. Towards improved genome-scale metabolic network reconstructions: unification, transcript specificity and beyond. *Briefings in bioinformatics*, 17(6):1060–1069, 2016.

Esa Pitkänen, Paula Jouhten, Jian Hou, Muhammad Fahad Syed, Peter Blomberg, Jana Kludas, Merja Oja, Liisa Holm, Merja Penttilä, Juho Rousu, et al. Comparative genome-scale reconstruction of gapless metabolic networks for present and ancestral species. *PLoS computational biology*, 10(2), 2014.

Leslie Pray, David A Relman, Eileen R Choffnes, et al. *The science and applications of synthetic and systems biology: workshop summary*. National Academies Press, 2011.

Kim D Pruitt, Tatiana Tatusova, and Donna R Maglott. Ncbi reference sequences (refseq): a curated non-redundant sequence database of genomes, transcripts and proteins. *Nucleic acids research*, 35(suppl_1):D61–D65, 2007.

Amy R McCune and John C Schimenti. Using genetic networks and homology to understand the evolution of phenotypic traits. *Current genomics*, 13(1):74–84, 2012.

Isabel Rocha, Paulo Maia, Pedro Evangelista, Paulo Vilaça, Simão Soares, José P Pinto, Jens Nielsen, Kiran R Patil, Eugénio C Ferreira, and Miguel Rocha. Optflux: an open-source software platform for in silico metabolic engineering. *BMC systems biology*, 4(1):45, 2010.

Armin Ronacher. Jinja2 documentation. *Welcome to Jinja2—Jinja2 Documentation (2.8-dev)*, 2008.

Milton H Saier. A functional-phylogenetic classification system for transmembrane solute transporters. *Microbiol. Mol. Biol. Rev.*, 64(2):354–411, 2000.

Eric W Sayers, Jeff Beck, J Rodney Brister, Evan E Bolton, Kathi Canese, Donald C Comeau, Kathryn Funk, Anne Ketter, Sunghwan Kim, Avi Kimchi, et al. Database resources of the national center for biotechnology information. *Nucleic acids research*, 48(D1):D9, 2020.

Jan Schellenberger, Junyoung O Park, Tom M Conrad, and Bernhard Ø Palsson. Bigg: a biochemical genetic and genomic knowledgebase of large scale metabolic reconstructions. *BMC bioinformatics*, 11(1):213, 2010.

Jan Schellenberger, Richard Que, Ronan MT Fleming, Ines Thiele, Jeffrey D Orth, Adam M Feist, Daniel C Zielinski, Aarash Bordbar, Nathan E Lewis, Sorena Rahmanian, et al. Quantitative prediction of cellular metabolism with constraint-based models: the cobra toolbox v2. 0. *Nature protocols*, 6(9):1290, 2011.

Ida Schomburg, Antje Chang, Oliver Hofmann, Christian Ebeling, Frank Ehrentreich, and Dietmar Schomburg. Brenda: a resource for enzyme data and metabolic information, 2002.

Fabian Schreiber and Erik LL Sonnhammer. Hieranoid: hierarchical orthology inference. *Journal of molecular biology*, 425(11):2072–2081, 2013.

Daniel Segre, Dennis Vitkup, and George M Church. Analysis of optimality in natural and perturbed metabolic networks. *Proceedings of the National Academy of Sciences*, 99(23): 15112–15117, 2002.

Tomer Shlomi, Omer Berkman, and Eytan Ruppin. Regulatory on/off minimization of metabolic flux changes after genetic perturbations. *Proceedings of the national academy of sciences*, 102(21):7695–7700, 2005.

George Stephanopoulos, Aristos A Aristidou, and Jens Nielsen. *Metabolic engineering: principles and methodologies*. Elsevier, 1998.

Gregory Stephanopoulos. Metabolic fluxes and metabolic engineering. *Metabolic engineering*, 1(1):1–11, 1999.

Neil Swainston, Kieran Smallbone, Pedro Mendes, Douglas B Kell, and Norman W Paton. The subliminal toolbox: automating steps in the reconstruction of metabolic networks. *Journal of integrative bioinformatics*, 8(2):187–203, 2011.

Roman L Tatusov, Natalie D Fedorova, John D Jackson, Aviva R Jacobs, Boris Kiryutin, Eugene V Koonin, Dmitri M Krylov, Raja Mazumder, Sergei L Mekhedov, Anastasia N Nikolskaya, et al. The cog database: an updated version includes eukaryotes. *BMC bioinformatics*, 4(1):41, 2003.

Marco Terzer, Nathaniel D Maynard, Markus W Covert, and Jörg Stelling. Genome-scale metabolic networks. *Wiley Interdisciplinary Reviews: Systems Biology and Medicine*, 1(3): 285–297, 2009.

Ines Thiele and Bernhard Ø Palsson. A protocol for generating a high-quality genome-scale metabolic reconstruction. *Nature protocols*, 5(1):93, 2010.

James Turnbull. *The Docker Book: Containerization is the new virtualization*. James Turnbull, 2014.

MJC Van den Beld and FAG Reubsaet. Differentiation between shigella, enteroinvasive escherichia coli (eiec) and noninvasive escherichia coli. *European journal of clinical microbiology & infectious diseases*, 31(6):899–904, 2012.

Amit Varma and Bernhard O Palsson. Stoichiometric flux balance models quantitatively predict growth and metabolic by-product secretion in wild-type escherichia coli w3110. *Applied and environmental microbiology*, 60(10):3724–3731, 1994.

Goutham N Vemuri and Aristos A Aristidou. Metabolic engineering in the-omics era: elucidating and modulating regulatory networks. *Microbiol. Mol. Biol. Rev.*, 69(2):197–216, 2005.

Edwin C Webb et al. *Enzyme nomenclature 1992. Recommendations of the Nomenclature Committee of the International Union of Biochemistry and Molecular Biology on the Nomenclature and Classification of Enzymes.* Number Ed. 6. Academic Press, 1992.

Nancy Y Yu, James R Wagner, Matthew R Laird, Gabor Melli, Sébastien Rey, Raymond Lo, Phuong Dao, S Cenk Sahinalp, Martin Ester, Leonard J Foster, et al. Psortb 3.0: improved protein subcellular localization prediction with refined localization subcategories and predictive capabilities for all prokaryotes. *Bioinformatics*, 26(13):1608–1615, 2010.

Cheng Zhang and Qiang Hua. Applications of genome-scale metabolic models in biotechnology and systems medicine. *Frontiers in physiology*, 6:413, 2016.

A

SUPPORT MATERIAL

Table S1: **Reactions associated with BiGG gene s0001.**

Reaction ID	Gene reaction rule
ACALDtpp	s0001
ACONIs	s0001
AOBUTDs	s0001
ARBTNexs	s0001
ATPHs	s0001
CO2tpp	s0001
CPGNexs	s0001
DATPHs	s0001
DHPTDCs	s0001
FALDtpp	s0001
FALGTHLs	s0001
FE3HOXexs	s0001
FECRMexs	s0001
FEENTERexs	s0001
FEOXAMexs	s0001
G5SADs	s0001
GLYCtpp	**b3927** and s0001
GTPHs	s0001
H2Otex	**b1319** or **b0957** or **b0929** or s0001 or **b2215** or **b3875** or **b1377** or **b0241**
H2Otpp	**b0875** or s0001
H2St1pp	s0001
H2tpp	s0001
METOX1s	s0001
METOX2s	s0001
N2Otpp	s0001
NH4tpp	**b0451** or s0001
NOtpp	s0001
O2tpp	s0001
SO2tpp	s0001

Table S2: **iJR904 genes for which no similarity was found.**

Gene ID	Associated reactions
b3111	SERD_L
b3112	SERD_L
b1416	GAPD
b4229	RIBabc
b4228	RIBabc
b1417	GAPD
b3692	DDPGALA; GALCTND
b3767	ACHBS; ACLS
b1898	ARBabc
b1899	ARBabc
b2978	GLYCTO2; GLYCTO3; GLYCTO4
b3768	ACHBS; ACLS