FCT Fundação para a Ciência e a Tecnologia
REPÚBLICA PORTUGUESA | CIÊNCIA, TECNOLOGIA E ENSINO SUPERIOR
NORTE2020 PROGRAMA OPERACIONAL REGIONAL DO NORTE
PORTUGAL 2020
UNIÃO EUROPEIA Fundo Social Europeu

Blueprint: Documenting the complexity of metabolic regulation by reconstruction of integrated metabolic-regulatory models

Fernando João Pereira da Cruz

UMinho | 2022

Universidade do Minho
Escola de Engenharia

Fernando João Pereira da Cruz

Blueprint: Documenting the complexity of metabolic regulation by reconstruction of integrated metabolic-regulatory models

October 2022

Universidade do Minho
School of Engineering

Fernando João Pereira da Cruz

Blueprint: Documenting the complexity of metabolic regulation by reconstruction of integrated metabolic-regulatory models

Doctorate Thesis
Doctorate in Biomedical Engineering

Work developed under the supervision of:
Professor Oscar Dias
Professor Miguel Rocha
Professor José Pedro Faria

October, 2022

This document was created with the (pdf/Xe/Lua)LaTeX processor and the NOVAthesis template (v6.9.15) [1].

Acknowledgements

I would like to thank Fundação para a Ciência e Tecnologia for the Ph.D. studentship I was awarded with (SFRH/BD/139198/2018). This work is only possible due to their support. I would also like to thank the Centre of Biological Engineering at the University of Minho for hosting me and providing an excellent working environment.

Após estes anos, após esta viagem, após linhas e linhas de código, após linhas e linhas de artigos, após linhas e linhas de tese, serei finalmente simples e breve. Estes anos são viagem! E, portanto, guardo em mim todas as suas passagens com muito carinho e gratidão a todos que dela fizeram parte.

Agradeço aos meus orientadores, em especial ao professor Oscar Dias e professor Miguel Rocha, todo o apoio, inspiração e amizade nesta viagem.

Agradeço aos meus amigos, em especial Alexandre, Marta, Capela, Emanuel, Sequeira, Fernanda, Nuno, Diogo, Miguel, e Zé, toda a paciência e suporte nesta viagem.

Agradeço aos meus pais e irmã (pota lima ahah) todo apoio, carinho e amor que sempre tiveram para comigo. Sem sombra de dúvida são os grandes obreiros do melhor de mim.

Agradeço à Coralie toda a amizade, carinho, e amor! Mas, agradeço-te principalmente por me acompanheres nesta viagem!

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 Universidade do Minho.

________________________, ________________________________

(Place) (Date)

Assinado por: **FERNANDO JOÃO PEREIRA DA CRUZ**
Num. de Identificação: 14354375
Data: 2023.02.03 11:23:47+00'00'

____

(Fernando João Pereira da Cruz)

Resumo

Blueprint: Descrição da complexidade da regulação metabólica através da reconstrução de modelos metabólicos e regulatórios integrados

Um modelo metabólico consegue prever o fenótipo de um organismo. No entanto, estes modelos podem obter previsões incorretas, pois alguns processos metabólicos são controlados por mecanismos reguladores. Assim, várias metodologias foram desenvolvidas para melhorar os modelos metabólicos através da integração de redes regulatórias. Todavia, a reconstrução de modelos regulatórios e metabólicos à escala genómica para diversos organismos apresenta diversos desafios.

Neste trabalho, propõe-se o desenvolvimento de diversas ferramentas para a reconstrução e análise de modelos metabólicos e regulatórios à escala genómica. Em primeiro lugar, descreve-se o *Biological networks constraint-based In Silico Optimization (BioISO)*, uma nova ferramenta para auxiliar a curação manual de modelos metabólicos. O BioISO usa um algoritmo de relação recursiva para orientar as previsões de fenótipo. Assim, esta ferramenta pode reduzir o número de artefatos em modelos metabólicos, diminuindo a possibilidade de obter erros durante a fase de curação.

Na segunda parte deste trabalho, desenvolveu-se um repositório de redes regulatórias para procariontes que permite suportar a sua integração em modelos metabólicos. O *Prokaryotic Transcriptional Regulatory Network Database (ProTReND)* inclui diversas ferramentas para extrair e processar informação regulatória de recursos externos. Esta ferramenta contém um sistema de integração de dados que converte dados dispersos de regulação em redes regulatórias integradas. Além disso, o ProTReND dispõe de uma aplicação que permite o acesso total aos dados regulatórios.

Finalmente, desenvolveu-se uma ferramenta computacional no MEWpy para simular e analisar modelos regulatórios e metabólicos. Esta ferramenta permite ler um modelo metabólico e/ou rede regulatória, em diversos formatos. Esta estrutura consegue construir um modelo regulatório e metabólico integrado usando as interações regulatórias e as ligações entre genes e proteínas codificadas no modelo metabólico e na rede regulatória. Além disso, esta estrutura suporta vários métodos de previsão de fenótipo implementados especificamente para a análise de modelos regulatórios-metabólicos.

Palavras-chave: Modelo metabólico à escala genómica, Rede regulatória, Metabolismo, Regulação genética

Abstract

Blueprint: Documenting the complexity of metabolic regulation by reconstruction of integrated metabolic-regulatory models

Genome-Scale Metabolic (GEM) models can predict the phenotypic behavior of organisms. However, these models can lead to incorrect predictions, as certain metabolic processes are controlled by regulatory mechanisms. Accordingly, many methodologies have been developed to extend the reconstruction and analysis of GEM models via the integration of Transcriptional Regulatory Network (TRN)s. Nevertheless, the perspective of reconstructing integrated genome-scale regulatory and metabolic models for diverse prokaryotes is still an open challenge.

In this work, we propose several tools to assist the reconstruction and analysis of regulatory and metabolic models. We start by describing BioISO, a novel tool to assist the manual curation of GEM models. BioISO uses a recursive relation-like algorithm and Flux Balance Analysis (FBA) to evaluate and guide debugging of *in silico* phenotype predictions. Hence, this tool can reduce the number of artifacts in GEM models, decreasing the burdens of model refinement and curation.

A state-of-the-art repository of TRNs for prokaryotes was implemented to support the reconstruction and integration of TRNs into GEM models. The ProTReND repository comprehends several tools to extract and process regulatory information available in several resources. More importantly, this repository contains a data integration system to unify the regulatory data into standardized TRNs at the genome scale. In addition, ProTReND contains a web application with full access to the regulatory data.

Finally, we have developed a new modeling framework to define, simulate and analyze GEnome-scale Regulatory and Metabolic (GERM) models in MEWpy. The GERM model framework can read a GEM model, as well as a TRN from different file formats. This framework assembles a GERM model using the regulatory interactions and Genes-Proteins-Reactions (GPR) rules encoded into the GEM model and TRN. In addition, this modeling framework supports several methods of phenotype prediction designed for regulatory-metabolic models.

Keywords: Genome-scale metabolic model, Transcriptional Regulatory Network, Metabolism, Gene regulation

Contents

List of Figures

List of Tables

Acronyms

AdaM	Adaptation of Metabolism *(p. 30)*
API	Application Programming Interface *(pp. 59, 64, 67, 70, 71, 74–77, 100, 101, 107, 108)*
BioISO	Biological networks constraint-based In Silico Optimization *(pp. v, vi, x, xiii, 35, 37–57, 141, 142, 144, 162, 163)*
BPCY	Biomass-Product Coupled Yield *(pp. 113, 140)*
cDNA	Complementary Deoxyribonucleic Acid *(p. 9)*
CDS	Central Data Storage *(pp. xi, 61–68, 72, 74–87, 90, 93, 94, 99–101, 108, 163)*
ChIP	Chromatin Immunoprecipitation *(p. 9)*
ChIP-chip	Chromatin Immunoprecipitation-Microarray *(pp. 8–10, 13)*
ChIP-seq	Chromatin Immunoprecipitation Sequencing *(pp. 8–10, 13)*
COBRA	Constraint-Based Reconstruction and Analysis *(p. 1)*
CRUD	Create, Read, Update and Delete *(pp. 64, 67, 100, 101, 108)*
CSO	Computational Strain Optimization *(pp. 111, 126)*
CSV	Comma Separated Values *(pp. 68, 70, 100, 107, 114, 120, 121, 123, 130, 142)*
DBMS	Database Management System *(pp. 61, 64, 67, 101, 108)*
DBTBS	Database of Transcriptional regulation in *Bacillus subtilis* *(pp. 12, 13, 71, 73, 75, 84–86, 91)*
dFBA	Dynamic Flux Balance Analysis *(pp. 29, 30)*
DNA	Deoxyribonucleic Acid *(pp. 6–10, 56, 66, 76)*
DNA-seq	DNA-sequencing *(p. 1)*
DOI	Digital Object Identifier *(pp. 74, 105)*
EA	Evolutionary Algorithm *(pp. 112, 113, 129)*

EFM	Elementary Flux Mode *(p. 30)*
EGRIN	Environment and Gene Regulatory Influence Network *(p. 28)*
ETL	Extract-Transform-Load *(pp. x, xi, xiii, 60–62, 67, 68, 77, 78, 82–87, 89, 90, 142)*
FASTA	FAST-All *(pp. 104, 105, 107)*
FBA	Flux Balance Analysis *(pp. vi, 22, 28, 29, 35, 40–42, 45, 47, 113–115, 122, 125–127)*
FTP	File Transfer Protocol *(p. 75)*
FVA	Flux Variability Analysis *(pp. 22, 29, 113, 126)*
GA	Genetic Algorithm *(p. 113)*
GAMS	General Algebraic Modeling System *(p. 36)*
GEM	Genome-Scale Metabolic *(pp. vi, x, 1–3, 20–33, 35–37, 42–46, 48, 53, 112–116, 118–120, 122, 123, 129, 130, 134, 140–142, 144, 163, 164)*
GEO	Gene Expression Omnibus *(pp. 9, 13)*
GERM	GEnome-scale Regulatory and Metabolic *(pp. vi, xi–xiii, 113, 114, 116, 119, 121, 123, 124, 127–142, 144, 163, 164)*
GIMME	Gene Inactivity Moderated by Metabolism and Expression *(p. 30)*
GPR	Genes-Proteins-Reactions *(pp. vi, 20, 25, 27, 112, 114, 117, 120, 123, 125, 126, 128, 129)*
GUI	Graphical User Interface *(pp. 27, 111, 144)*
GX-FBA	Gene Expression Flux Balance Analysis *(pp. 29, 30, 32)*
HEB	Hierarchical Edge Bundling *(pp. xi, 103)*
HTML	HyperText Markup Language *(p. 70)*
HypE	Hypervolume Estimation algorithm *(p. 113)*
IDREAM	Integrated Deduced REgulation And Metabolism *(pp. 28, 31, 111, 128, 129)*
JSON	JavaScript Object Notation *(pp. 62, 63, 68, 70, 71, 74, 75, 84, 100, 107, 114, 116, 121, 123, 142)*
KEGG	Kyoto Encyclopedia of Genes and Genomes *(pp. 13, 36, 37, 45, 74, 77, 93)*
lMOMA	linear Minimization of Metabolic Adjustment *(pp. 113, 128)*
LP	Linear Programming *(pp. 22, 124)*
M3D	Many Microbe Microarrays Database *(p. 13)*
MADE	Metabolic Adjustment by Differential Expression *(pp. 27, 29, 31, 32)*

METRADE	Metabolic and Transcriptomics Adaptation Estimator *(pp. 30, 31)*
MIAME	Minimum Information About a Microarray Experiment *(pp. 9, 13)*
MILP	Mixed-Integer Linear Programming *(pp. 27, 29, 124, 126, 136)*
MINSEQE	Minimum Information About a Next-generation Sequencing Experiment *(p. 9)*
MOEA	Multi-objective Evolutionary Algorithm *(pp. 113, 129)*
MOMA	Minimization of Metabolic Adjustment *(pp. 113, 128)*
mRNA	Messenger Ribonucleic Acid *(pp. 6, 9, 66)*
MySQL	My Structured Query Language *(pp. 62, 101, 108)*
NCBI	National Center for Biotechnology Information *(pp. 14, 68, 70, 71, 74–78, 93, 101, 102, 104–106)*
NGS	Next Generation Sequencing *(pp. 1, 9, 13)*
NSGA	Non-sorting Genetic Algorithm *(p. 113)*
ODB	Operon DataBase *(pp. 70, 71, 73, 79, 84–86)*
ODE	Ordinary Differential Equation *(pp. 20, 21)*
OGM	Object Graph Mapping *(pp. 62, 63, 67)*
pFBA	Parsimonious Flux Balance Analysis *(pp. 22, 30, 33, 113–115, 122, 125, 127)*
PMID	PubMed Identifier *(pp. 66, 70, 71, 74, 79, 105)*
PROM	Probabilistic Regulation of Metabolism *(pp. xii, 28, 111, 113–115, 122, 126, 136, 138, 142)*
ProTReND	Prokaryotic Transcriptional Regulatory Network Database *(pp. v, vi, x, xi, xiii, 59, 61–69, 72, 74, 77, 79, 81–87, 89–94, 96–109, 141, 142, 144)*
PSSM	Position-Specific Scoring Matrix *(p. 77)*
PWM	Position Weight Matrix *(pp. 19, 77, 104, 105)*
RefSeq	NCBI Reference Sequence Database *(pp. 14, 75, 76, 93, 101)*
rFBA	Regulatory Flux Balance Analysis *(pp. xii, 27, 111, 113–115, 122, 125, 126, 135, 136, 142)*
RIPTiDe	Reaction Inclusion by Parsimony and Transcript Distribution *(pp. 30, 31)*
RNA	Ribonucleic Acid *(pp. 6–10, 56, 69)*
RNA-seq	RNA-sequencing *(pp. 1, 8–10, 13, 30)*
ROOM	Regulatory on/off Minimization of Metabolic flux changes *(pp. 113, 128)*
SA	Simulated Annealing *(p. 113)*
SBML	Systems Biology Markup Language *(pp. 21, 27, 37, 42, 114, 120–123, 130, 142)*
SBML-qual	SBML Qualitative *(pp. 27, 120, 123)*
SMD	Stanford Microarray Database *(p. 13)*

SPEA2	Strength Pareto Evolutionary Algorithm *(p. 113)*
SR-FBA	Steady-state Regulatory Flux Balance Analysis *(pp. xii, 27, 111, 113–115, 122, 124, 126, 133, 135–137, 142)*
SRA	Sequence Read Archive *(p. 13)*
sRNA	small RNA *(pp. 66, 69, 71, 86)*
STRING	Search Tool for Retrieval of Interacting Genes/Proteins *(p. 13)*
TEAM	Temporal Expression-based Analysis of Metabolism *(p. 30)*
TF	Transcription Factor *(pp. 7, 9, 10, 12–17, 19, 20, 28, 59, 66, 69, 71, 72)*
tFBA	Transcriptional-controlled Flux Balance Analysis *(p. 29)*
TFBS	Transcription Factor Binding Site *(pp. xi, 7–10, 12, 13, 19, 20, 33, 59, 66, 67, 69–74, 77, 79–81, 84–86, 89–92, 94–98, 102, 104, 105, 108)*
TIGER	Toolbox for Integrating Genome-scale Metabolism *(pp. 27, 31, 111)*
TRANSFAC	TRANScription FACtor database *(pp. 104, 107)*
TrEMBL	Translated European Molecular Biology Laboratory *(p. 14)*
TRFBA	Transcriptional Regulated Flux Balance Analysis *(pp. 28, 31, 111)*
TRN	Transcriptional Regulatory Network *(pp. vi, x, 2, 3, 9–11, 13–20, 23–29, 31–33, 59, 61, 68, 70, 71, 75, 84, 85, 87, 89, 90, 93, 94, 96, 97, 101–104, 107, 114–116, 118–120, 122, 123, 128, 130, 132, 134, 140–142, 144, 163, 164)*
tRNA	Transfer RNA *(p. 65)*
TSV	Tabular Separated Values *(pp. 70, 71)*
TXT	Text File *(pp. 68, 123)*
UML	Unified Modeling Language *(pp. xi, 116, 119, 121, 124)*
UniProt	Universal Protein Resource *(pp. 14, 75–79, 101, 102, 104–106)*
UniProtKB	Universal Protein Resource Knowledgebase *(pp. 14, 68, 70, 76, 77, 79)*
XLSX	Excel Microsoft Office Open XML Format Spreadsheet File *(pp. 71, 100, 107)*
XML	Extensible Markup Language *(pp. 100, 107, 123)*
YEASTRACT	YEAst Search for Transcriptional Regulators And Consensus Tracking *(p. 28)*

1 Introduction

1.1 Context and Motivation

The volume of omics data is rising at an unprecedented scale due to the advent of Next Generation Sequencing (NGS) techniques, such as DNA-sequencing (DNA-seq) and RNA-sequencing (RNA-seq) [2]. As a result, high-throughput large-scale omics experiments are nowadays highly used to study the molecular machinery of many organisms at the systems level. For instance, the increase of publicly available genome sequences and gene expression data has enabled systems biology to thrive in this high-throughput era [3].

Systems biology is often described as a multidisciplinary field that provides quantitative and qualitative descriptions of biological systems. The reconstruction of *in silico* networks and models, leveraged by the large volume of omics data, allows modeling cells' behavior over time in a wide range of different conditions [4].

The reconstruction of comprehensive metabolic models at the genome scale using genomics data is common practice in systems biology [5]–[11]. Nevertheless, the reconstruction process is still challenging for high-quality reconstructions [12]. In the bottom-up reconstruction approach, model validation and manual curation can be laborious tasks and most bottlenecks derive from accumulated errors requiring complex and unique solutions [13]. On the other hand, the top-down reconstruction approach can quickly generate a gapless simulation-ready model [14]. Yet, the latter approach adds several artifacts to the metabolic network.

A major drawback of GEM models is the inability to account for gene regulation [15]. This can lead to incorrect phenotype predictions for prokaryotes in several environmental conditions, as the proteome of these organisms usually changes as the environmental conditions do. A regulatory layer can be added to GEM models to overcome this limitation. This additional regulatory layer can be integrated into GEM models for tailoring tendencies in the flux distributions, thus preventing false positive phenotype simulations. Despite the efforts for the integration of gene regulation into GEM models, most of the methods still fail to outperform Constraint-Based Reconstruction and Analysis (COBRA) methods [16]. This suggests that promising results reported by these methods can be artifacts that do not scale well to different

GEM models and regulatory data. Hence, reconstructing integrated regulatory-metabolic models for many prokaryotes is still a complex endeavor.

The reconstruction of TRNs has become a hot topic in computational biology [17]–[19]. However, the inference of TRNs from gene expression data is an underdetermined problem where many solutions can explain the data equally well [15]. More importantly, most of these methodologies are unavailable to all organisms due to the lack of transcriptomics data. To overcome this limitation, approaches based on comparative genomics tools can extrapolate well-known TRNs of model organisms to poorly studied organisms. For that, state-of-the-art regulatory information is mandatory, as these would ease the development of automated tools for reconstructing TRNs.

The following chapters will address the implementation of computational tools to assist the reconstruction and analysis of integrated regulatory-metabolic models at the genome scale. In the first phase, we implemented a computational tool to assist the curation of GEM models, as gaps in the metabolic network might hinder the integration of regulatory networks. In the second phase, we will detail the development of a repository of regulatory information for prokaryotic organisms. This user-friendly resource of regulatory data supports TRN inferring tools, thus easing the development of novel regulatory networks. Moreover, such a repository is also a useful resource of TRNs for reconstructing integrated regulatory-metabolic models. The third phase of this work will focus on developing methods for the reconstruction and analysis of integrated regulatory-metabolic models. These methods allow integrating TRNs into existing GEM models to perform phenotype prediction in complex environmental and genetic conditions. Ultimately, these approaches will be used to obtain new insights regarding the complexity of metabolic regulation in prokaryotic organisms.

1.2 Research Objectives

The main goal of this work is to develop computational tools for the reconstruction and analysis of integrated regulatory-metabolic models. First, we will describe a user-friendly tool to assist the curation of high-quality GEM models in a bottom-up reconstruction approach. Next, the collection of multiple TRNs into a unified resource of regulatory information will be addressed. Finally, this project will address the development of several tools for integrating TRNs into GEM models. As the control of gene expression is a complex process in prokaryotic organisms and even more in eukaryotes, all tools to be developed will mainly focus on the regulation of metabolism in prokaryotic organisms. However, future endeavors may extend the scope of this platform.

The computational tools developed in the context of this project will be available to the scientific community easing access to a framework for obtaining and integrating regulatory data into metabolic models. Ultimately, this framework can be extended to support the inference of novel *in silico* networks leading to new insights regarding the cellular mechanisms that control gene expression and metabolism in prokaryotes.

For that, the following objectives are to be accomplished:

- To develop an objective-oriented application for easing manual curation steps when debugging *in silico* metabolic networks;
- To collect relevant regulatory information and TRNs for prokaryotic organisms into a unified and integrated data repository;
- To provide easy access to the repository of regulatory data capable of supporting TRN inferring tools and integration with GEM models;
- To develop computational tools for extending GEM models with TRNs and representing integrated regulatory-metabolic models;
- To collect and implement appropriate phenotype prediction methods for integrated regulatory-metabolic models.

1.3 Outline

This document is divided into six chapters and outlined as follows:

- The current chapter consists of the context and motivation for the reconstruction and analysis of integrated regulatory-metabolic models. In addition, the main objectives associated with this project are also briefly introduced together with the outline of this thesis;
- Chapter 2 consists of a detailed collection of concepts and subjects fundamental for understanding the regulation of metabolism in prokaryotic organisms. The first phase will address regulatory resources and the reconstruction of TRNs, while the second phase addresses the reconstruction of GEM models using the bottom-up approach. Finally, the integration of a regulatory layer into GEM models will be thoroughly discussed;
- Chapter 3 addresses the implementation of a user-friendly computational tool to assist the manual curation of GEM models. The potential of this tool to guide the debugging of model reconstructions is then assessed with the results of two other state-of-the-art tools;
- In chapter 4, the reconstruction of a unified and integrated repository of regulatory information for prokaryotes will be addressed. Likewise, this chapter also encompasses the design and implementation of a web application to access multiple TRNs and to ease the curation of the collected regulatory data;
- Chapter 5 comprehends the development of computational tools for representing regulatory- metabolic models. This chapter also includes the implementation of several methods for phenotype prediction with integrated models;

- Chapter 6 includes a brief synopsis of this project. An assessment of the results obtained in this work is also provided together with new strategies to improve the presented computational tools.

2 Background

The work presented in this chapter has also been partially published in the following publications:

- *Cruz, F., Lima, D., Faria, J. P., Rocha, M., Dias, O. (2020). Towards the Reconstruction of Integrated Genome-Scale Models of Metabolism and Gene Expression. In: Fdez-Riverola, F., Rocha, M., Mohamad, M., Zaki, N., Castellanos-Garzón, J. (eds) Practical Applications of Computational Biology and Bioinformatics, 13th International Conference. PACBB 2019. Advances in Intelligent Systems and Computing, vol 1005. Springer, Cham.*
- *Cruz, F., Faria, J. P., Rocha, M., Rocha, I., Dias, O. (2020). A review of methods for the reconstruction and analysis of integrated genome-scale models of metabolism and regulation. Biochemical Society Transactions.*

2.1 The control of gene expression

In prokaryotic organisms, the proteome usually changes as the environmental conditions do, thereby reflecting the cell response to the ever-changing environment by making use of a variety of regulatory mechanisms.

As depicted in figure 1, the control of gene expression can take place at several stages of regulation [20]. However, the regulation of transcription initiation is most likely the primary stage for controlling gene expression in prokaryotic cells [21].

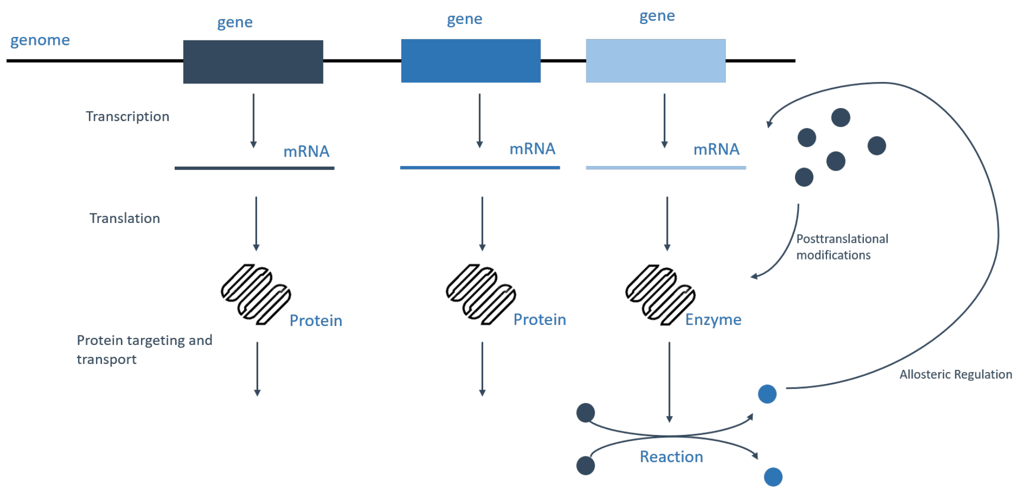


Figure 1: Different potential stages of regulation in prokaryotic organisms. Each stage can be associated with several regulatory mechanisms that influence the cellular concentration of a given protein. Transcription initiation, translation, posttranslational modifications of proteins, protein targeting, and allosteric regulation are examples of potential stages and mechanisms of regulation in prokaryotic organisms.

Transcription is initiated when the holoenzyme Ribonucleic Acid (RNA) polymerase binds to a specific region of Deoxyribonucleic Acid (DNA) known as the promoter [21]. Promoters are usually characterized by a DNA consensus sequence, even if these sequences vary considerably within the genome. The binding affinity of RNA polymerase is influenced by the variety of promoter sequences and the rate at which transcription is initiated. Hence, promoters are often classified as strong or weak promoters due to this control over the initiation of transcription.

Many of the principles assumed in regulating prokaryotic gene expression are based on the fact that genes are clustered into operons, thus being regulated together [21]. An operon stands for a group of genes linearly and sequentially disposed of, such that a single functional Messenger Ribonucleic Acid (mRNA) molecule that contains the information for the synthesis of multiple related proteins is transcribed at once. According to the organization of a prokaryotic genome, operons can be considered the primary units of regulation of gene expression. A well-known example, namely the *lac* operon in *Escherichia coli* [22], [23], is described in figure 2. In this case, the operon contains the genes required to consume lactose inside the cell.

Operons often comprise additional regulatory DNA sequences known as *cis*-regulatory, *cis*-acting elements or TFBS. These specific sites, where gene transcription regulatory proteins bind, also take part directly or indirectly in the initiation of transcription [21]. In this primary mechanism for controlling gene expression, the other peers that bind to TFBS are known as regulators or Transcription Factor (TF). These regulatory proteins are *trans*-regulatory or *trans*-acting elements that either induce or repress the expression of a given gene. Besides the fundamental biological machinery described so far, there are many other regulatory mechanisms for controlling gene expression in prokaryotes, such as transcriptional attenuation and gene regulation by recombination [20] or regulators of the holoenzyme RNA polymerase called global regulators (e.g. sigma factors) [24]. In the case of the *lac* operon in *E. coli* [22], [23], the *lacI* regulator represses the transcription of the operon by binding to the TFBS located inside the operon (figure 2).

The regulation of transcription initiation is fundamentally different in prokaryotic and eukaryotic organisms [24], [25]. In bacteria, a given regulator might influence the expression of one or more operons. As such, a new level of abstraction called regulon was created for describing a network of operons being controlled by the same regulator or TF [21]. Simple regulons (comprising only a few operons) tend to be co-expressed together under several conditions (if not all), thereby revealing expression modularity [26], [27]. Hence, the bacterial regulatory machinery is robust and often maintains reproducible responses to an ever-changing environment [28].

On the other hand, the presence of intra-operonic promoters may result in gene-intraspecific expression patterns (and not regulon-specific) for specific environmental conditions [29], [30]. Moreover, complex regulatory mechanisms may also exhibit highly specific expression behaviors, which may also provide flexibility to bacterial regulatory networks [31].

Regarding the ubiquity of global regulators, these are TFs that directly bind to the RNA polymerase controlling gene expression at the promoters, and thus downregulating or upregulating global transcription [24].

Metabolism and regulation of gene expression are two of the main biological systems of a prokaryotic cell. According to previous findings, these two systems are intrinsically interconnected, as metabolism is controlled by enzyme levels, which in turn are regulated by the transcription initiation [23], [32], [33]. Furthermore, simple regulatory motifs can have specific functions inside the cell, such as controlling an entire metabolic pathway [34], [35]. Figure 2 highlights a well-known example where metabolism is regulated through the control of gene expression, namely the lactose catabolism in *E. coli* [23]. When lactose is absent, the transcription of the *lac* operon is inhibited by the binding of the *lacI* regulator to the TFBS located inside the operon. Nevertheless, other layers, such as post-transcriptional modifications, allosteric regulation, or thermodynamics, are also in control of the metabolism [36], [37]. It is also noteworthy that the role of transcription in regulating microbial metabolic fluxes can occasionally be minimal compared to the remarkable plasticity of the cellular metabolic state [38].

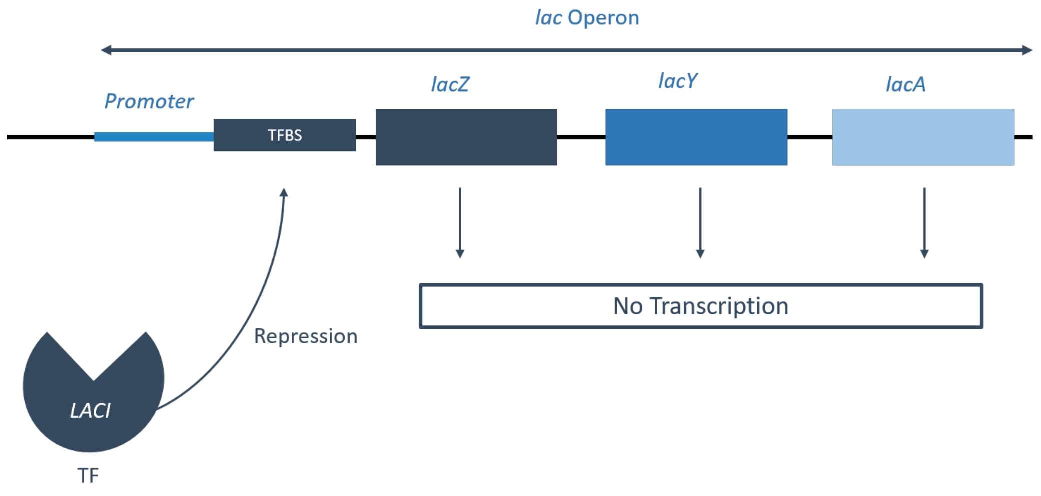


Figure 2: Regulation of the *lac* operon in *Escherichia coli*. The *lacI* regulator prevents the transcription of the operon by binding to the TFBS within the promoter region when lactose is absent inside the cell.

2.2 From high-throughput measurement techniques to the transcriptome

According to the experimental design, gene expression data can be classified in steady-state [39]–[43] or dynamic/time-series [44], [45], in which gene expression levels can be measured at a single time point or during consecutive ones, respectively.

Although dynamic gene expression datasets offer more detail about the transcriptome, these are *per se* harder to conduct, involving laborious computational analysis. Perturbations comprising metabolic, proteomic, or genetic manipulations can also be added to the biochemical assays [46], [47].

Techniques for measuring genome-wide gene expression are usually based on RNA profiling. Example of these techniques are DNA microarrays [48]–[51] and RNA-seq [52]–[56], as shown in figure 3. On the other hand, genome binding and occupancy experiments provide additional insights regarding the cell regulatory mechanism, namely, the DNA spots where proteins bind are less used. Examples of these techniques are Chromatin Immunoprecipitation-Microarray (ChIP-chip) [57]–[59] and Chromatin Immunoprecipitation Sequencing (ChIP-seq) [60], [61], as illustrated in figure 3.

DNA microarrays were the first high-throughput technology aimed at simultaneously measuring a large number of gene expression levels. This technique was, therefore, responsible for the first impulse towards understanding the control of gene expression in prokaryotic cells [48], [50]. DNA microarrays are based on the hybridization of thousands of short DNA fragments (probes), disposed on a substrate chip, with thousands of genomic regions of the organism of interest. As a result, one can retrieve an estimated concentration of the transcripts of a cell population. The probes' design can introduce bias to the assay, eventually leading to some limitations. Besides the bias problem, DNA microarrays are also extremely sensitive to the quality of the experiments, as results obtained from similar experiments vary from laboratory to laboratory [49] or have high noise-to-signal ratio levels [51].

ChIP-chip overcomes some of the expression profiling limitations [59]. This technique combines Chromatin Immunoprecipitation (ChIP) with array hybridization. The immunoprecipitation experimental technique ChIP allows determining whether a given set of proteins interact with DNA. This can be of great use to identify which genome regions TFs bind to. Thus, protein-DNA interactions can be simultaneously identified and quantified on a genome-wide scale [57], [58]. Despite the advantage of identifying TFBS, ChIP-chip-based studies are rare and considerably expensive, when compared to the counterpart technique DNA microarray.

The advent of NGS technologies revolutionized molecular biology research, giving rise to an unprecedented amount of public available genome sequences for a wide variety of organisms [2]. As a result, RNA-seq has also emerged as a RNA profiling technique capable of measuring transcript abundances [52]–[56]. RNA-seq is based on the reverse transcription of fragmented RNA, previously obtained from a cell population in a given state. Usually, after sequencing and mapping the resulting Complementary Deoxyribonucleic Acid (cDNA) to a reference genome, the number of fragments mapping a given gene (usually normalized) stands for a raw level of gene expression [62]. In addition to the straightforward measurement of raw gene expression levels, this technology can also be used to directly measure other forms of RNA besides mRNA [56]. Moreover, RNA-seq opens the possibility to a vast variety of experiments and biochemical assays, while avoiding the drawbacks of DNA microarrays, namely the bias associated with probe design and the high noise-to-signal ratio levels [63].

ChIP-seq is the result of applying NGS technologies to genome binding and occupancy experiments, thereby enabling whole-genome ChIP assays at a faster pace [60]. This technique is of particular interest for reconstructing TRN, as it allows identifying where regulators bind in the genome sequence of the organism of interest and the potentiality of quantifying these regulatory interactions [61].

The rise of experiments for measuring gene expression levels across various environmental and genetic conditions also brought new problems and challenges, namely the lack of standards and guidelines for publishing and sharing data from heterogeneous sources. Following the standards proposed in both Minimum Information About a Microarray Experiment (MIAME) [64] and Minimum Information About a Next-generation Sequencing Experiment (MINSEQE), gene expression data from heterogeneous sources can now be stored efficiently in both Gene Expression Omnibus (GEO) [65] and ArrayExpress [66].

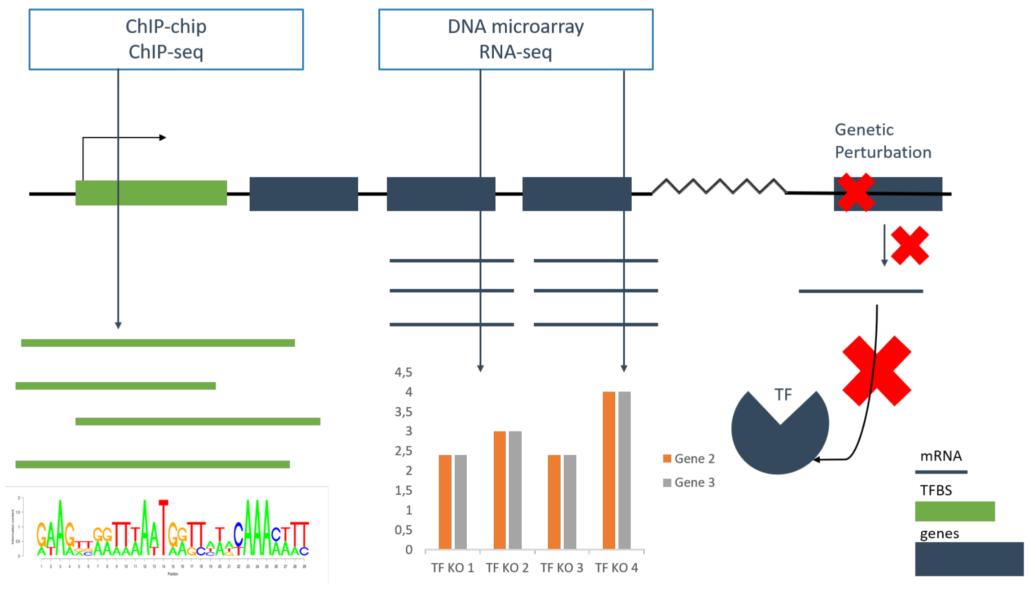


Figure 3: Available techniques for measuring the transcriptome. Available techniques for measuring the transcriptome of prokaryotic cells, namely DNA microarrays, RNA-seq, ChIP-chip and ChIP-seq. In this example, the experimental design is based on the genetic perturbation of several TFs in the organism of interest.

All transcriptomics techniques mentioned so far can yield gene expression data extremely useful for reconstructing a TRN of a given organism. These methodologies cover and measure distinct aspects of the organisms' regulatory machinery. Single gene expression levels or co-expression patterns can be inferred from simple steady-state gene expression datasets obtained with RNA profiling techniques [39], [40], [42], [43]. The causality of regulatory interactions, on the other hand, can be inferred from steady-state gene expression datasets obtained with genome binding and RNA profiling techniques while introducing genetic perturbations to the organism of interest [46], [47]. Moreover, new layers of detail such as the dynamics of several regulatory processes can also be achieved by the analysis of dynamic gene expression datasets [44], [45].

2.3 Resources of regulatory data

The resources of regulatory data can be divided into three groups:

- Databases of transcriptional information;
- Databases of gene expression data;
- Literature.

Many databases contain transcriptional data regarding the biological elements responsible for controlling gene expression, including valuable information regarding promoters, TFs, TFBS and target genes.

On the other hand, a restricted number of databases of gene expression data centralizes most of the gene expression data currently available. Both types of databases (transcriptional information and gene expression data) must be seen as complementary resources of regulatory information pivotal for the reconstruction of prokaryotic TRNs.

Databases of prokaryotic transcriptional information provide information regarding regulatory events inside the cell. Databases of transcriptional information can be classified as organism-specific and non-organism-specific. Table 1 presents relevant databases containing transcriptional information, highlighting the type of regulatory data and its representativeness.

Table 1: Online resources having transcriptional information. Summary of the most relevant databases of transcriptional information, classified by their representativeness and type of regulatory data.

Database	**Organism**	**Regulatory information**
RegulonDB [67]	*Escherichia coli*	promoters, TFs, target genes, TFBS, operon, regulatory interactions and sigma factors
EcoCyc [68]	*Escherichia coli*	regulatory and metabolic integration
DBTBS [69]	*Bacillus subtilis*	TFs, target genes, TFBS, operon, regulatory interactions and sigma factors
MTRBRegList [70]	*Mycobacterium tuberculosis*	promoters, TFs, target genes, TFBS
CoryneRegNet [71]	*Corynebacterium*	Corynebacterial gene regulatory networks with TFs and target genes
cTFbase [72]	*Cyanobacteria*	putative TFs
CollecTF [73]	multiple	TFs, target genes, TFBS and regulatory interactions
RegPrecise [74]	multiple	TFs, target genes, TFBS and regulatory interactions
PRODORIC2 [75]	multiple	TFs, target genes, TFBS
Abasy [76]	multiple	global gene regulatory networks with TFs and target genes
SwissRegulon [77]	multiple	TFs and TFBS
ODB [78]	multiple	operons
footprintDB [79]	multiple	TFs

Most organism-specific databases describe thoroughly well-known bacteria, such as *E. coli*, *Bacillus subtilis* and *Mycobacterium tuberculosis*, or groups of bacteria, such as Gamma-proteobacteria, Mycobacteria, and Cyanobacteria. For instance, RegulonDB comprises a set of curated regulatory interactions taking place in *E. coli* [67]. The authors present this database as a unified resource for gram-negative bacterium transcriptional regulation. Likewise, Database of Transcriptional regulation in *Bacillus subtilis*

(DBTBS) also comprises a collection of experimentally validated regulatory interactions for *B. subtilis* [69].

Organism-specific databases can be seen as gold standards of known regulatory interactions taking place in a given prokaryotic organism. Therefore, the contents of these databases can be used to assess state-of-the-art methods aimed at inferring high-quality TRNs.

On the other hand, non-organism-specific databases of transcriptional information offer limited information, even if for vast phylogenetic clades. In fact, these databases are considered less comprehensive, as the retrieved regulatory information is usually limited to TFs, TFBS, operons, or target genes. For example, RegPrecise holds transcriptional information for at least 14 taxonomic groups of bacteria [74]. This collection of transcriptional regulons, inferred from high-quality manually-curated data, is also complemented with comparative-genomics approaches [74]. Collectf is an online database of experimentally validated TFBSs in several prokaryotic organisms [73].

Besides databases storing regulatory data only, other online resources, such as Search Tool for Retrieval of Interacting Genes/Proteins (STRING) [80], Kyoto Encyclopedia of Genes and Genomes (KEGG) [81], and BioCyc [82], store relevant transcriptional information for either single or large collections of prokaryotic organisms while providing several other forms of biological data.

Databases of gene expression data can house large datasets for a wide diversity of organisms, including bacteria. GEO [65] and ArrayExpress [66] are two of the most widely known databases of gene expression data. Both GEO and ArrayExpress have the advantage of being MIAME compliant and thus being the major hub for sharing the transcriptome of prokaryotic organisms. These databases provide query and browsing tools for analyzing and retrieving gene expression data.

Other databases of gene expression data derived from microarray and RNA-seq experiments are COLOMBOS [83] and Many Microbe Microarrays Database (M3D) [84]. These databases comprise a comprehensive compendium of processed bacterial gene expression data useful to reconstruct TRNs. Stanford Microarray Database (SMD) is one of the oldest online resources for storing raw and normalized data, obtained from microarray experiments [85].

Recently, the Sequence Read Archive (SRA) platform has emerged as a prominent resource of biological sequence data, due to the development of NGS technologies [86]. This database provides a public archive of sequenced reads for a large variety of organisms, including prokaryotes. One may access the SRA platform for finding new experiments involving the measurement of prokaryotic transcriptomes using either RNA-seq and ChIP-seq.

Several TRNs have been published throughout the years for prokaryotic organisms. Well-known prokaryotes such as *E. coli*, *B. subtilis* and *M. tuberculosis* are well represented in the literature, having comprehensive TRNs with hundreds of regulators and thousands of target genes. For instance, several authors have reconstructed high-quality TRNs for *Escherichia coli* K-12 MG1655 [46], [87] by combining experimental ChIP-chip and ChIP-seq data with the state-of-the-art network available in RegulonDB [67]. Likewise, two published TRNs of *Bacillus subtilis* 168 encompass more than 200 regulators each. These networks have been assembled using literature, DBTBS [69] and transcriptomics data. Turkarslan *et al* [47] has published a TRN for *Mycobacterium tuberculosis* H37Rv using ChIP-seq and gene expression data analysis. Interestingly, several TRNs have also been published for non-model prokaryotic organisms [17], [45], [88]–[97]. Nevertheless, most of these TRNs are highly based in data-driven approaches missing the proficiency of manual curation or a characterization at the genome-scale.

2.4 Resources of genomics and proteomics data

Resources of genomics and proteomics data can greatly help extend traditional TRNs. The characterization of TRNs at the genome scale is pivotal for reconstructing novel networks and multi-scale models. Universal Protein Resource (UniProt) and National Center for Biotechnology Information (NCBI) are usually considered two central hubs for genomics and proteomics data.

Universal Protein Resource Knowledgebase (UniProtKB) consists of a collection of functional information for proteins [98]. This repository contains amino acid sequences, protein functional and structural annotation, and links to external yet useful resources. The UniProtKB is divided into two sections, namely Swiss-Prot and Translated European Molecular Biology Laboratory (TrEMBL). While the former encompasses only high-quality records for many proteins, the latter contains a massive collection of computationally annotated records lacking revision and curation.

NCBI is an online resource that provides a series of databases containing several forms of biological data and relevant bioinformatics tools [99]. GenBank is an open-access widely used database of nucleotide sequences available at NCBI [100]. On the other hand, NCBI Reference Sequence Database (RefSeq) database contains annotated and curated nucleotide sequences and the corresponding protein products [101]. These repositories provide public access to relevant information for the structural and functional annotation of many genomes. Another relevant resource available at NCBI is PubMed. This database consists of a large collection of biomedical literature. NCBI also includes a database that attempts to incorporate phylogenetic and taxonomic knowledge for many organisms named NCBI Taxonomy Database [102].

2.5 Transcriptional regulatory networks

A TRN can be represented by a bipartite graph model according to definition 2.5.1 [26]. In this bipartite graph, the vertices correspond to either TFs or target genes, while edges determine the links between these two regulatory elements, often a causal relationship. Figure 4 depicts the transformation of a TRN, comprising one TF controlling the expression of two target genes, into a graph-based structure.

Theorem 2.5.1. ***Definition of a Transcriptional Regulatory Network***

Let $G = \{E, V\}$ be the bipartite graph of a given transcriptional regulatory network where V is the set of vertices that stand for either TFs or target genes and E the set of edges defining the links between the vertices (usually the causal relationship between TFs or target genes).

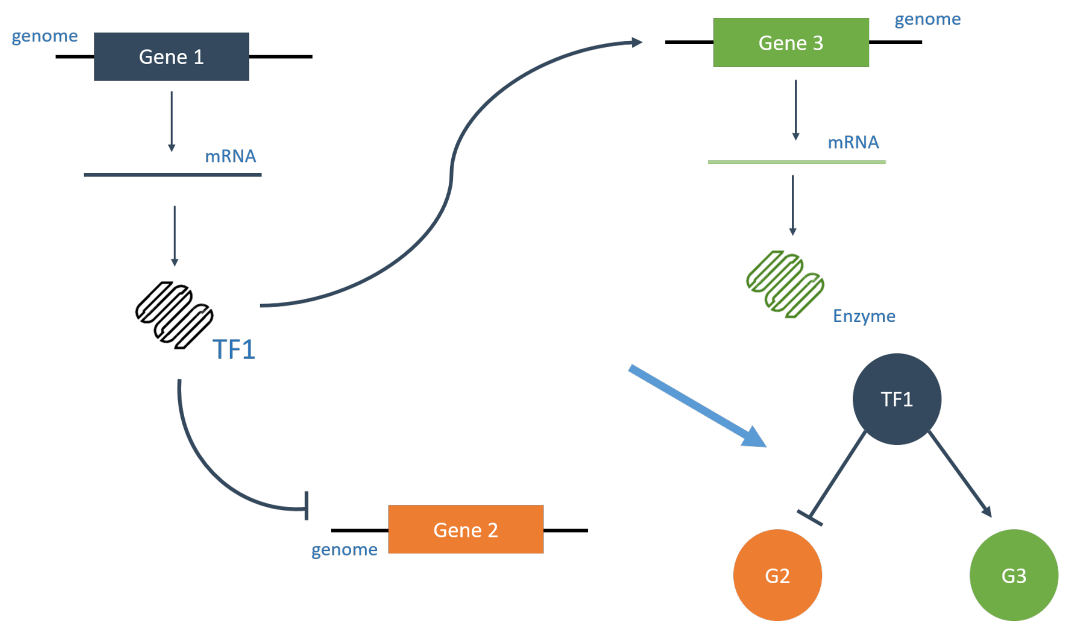


Figure 4: Example of a TRN. Example of the transformation of a TRN, comprising one TF acting over the expression of two target genes, into a TRN bipartite graph model.

TRN graphs are usually classified according to the causality of the relationships between the vertices [19]. That is if there are enough pieces of evidence on a regulatory interaction between two vertices, namely one TF that exerts control over a target gene, this TRN sub-graph can be classified as directed, regardless of the nature of this relationship (inhibition or activation). However, if no direction can be inferred between the vertices, this sub-graph is called undirected. In this case, there are no pieces of evidence to infer if a gene exerts control or influences the expression of another gene and vice versa. Ideally, a weighted TRN graph is obtained when both the nature and direction of the causality between two vertices are known. For that, one should infer whether a TF activates or silences the target gene.

TRNs can also be divided according to the temporal dimension. Whereas dynamic TRNs are usually inferred from time-series gene expression data, static TRNs can be obtained from regular transcriptomics datasets [103]. Figure 5 illustrates all categories of TRNs, namely directed, undirected, weighted and dynamic graphs. Note that, the first tree examples can also be thought of as static graphs.

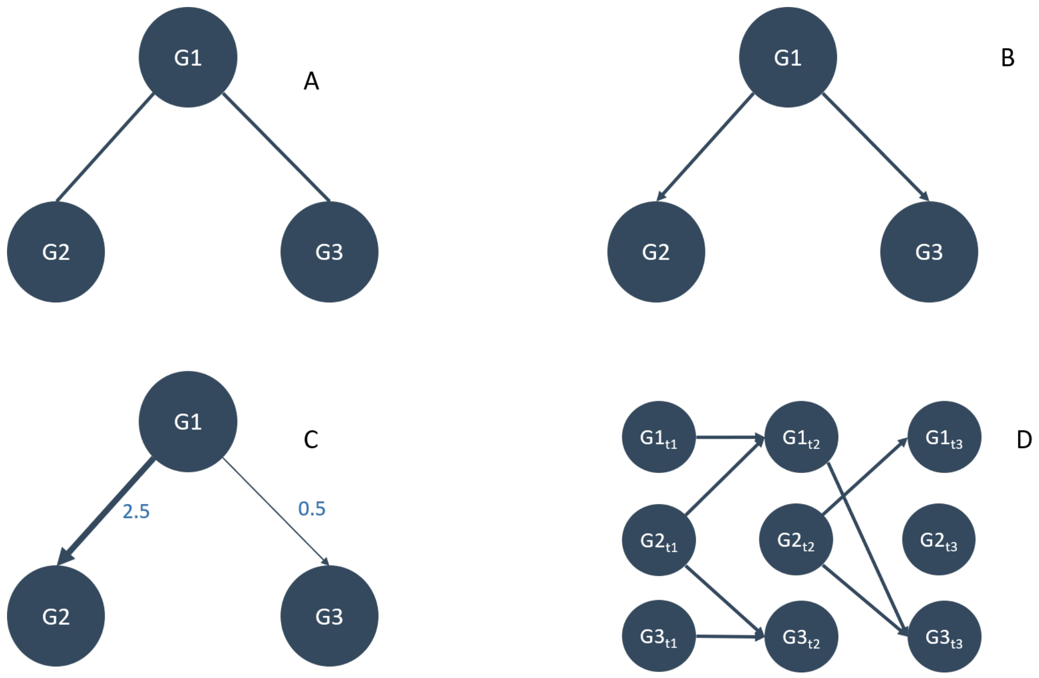


Figure 5: Different TRNs graphs categories. Example of TRNs graphs categories: undirected (A); directed (B); weighted (C); dynamic (D). Each of the first tree examples can also be thought of as static TRNs graphs.

As shown in figures 6 A and B, TRNs can also be decomposed by the inherent structure and regulatory function [104], respectively. Whereas the structure or topology of the TRN reveals which TFs control the expression of the target genes, the regulatory function, usually represented by a mathematical formulation, determines how one or more TFs exert such control, describing or quantifying the expression levels of target genes. The analysis of the TRN structure can suggest the presence of regulatory motifs or modules, namely groups of co-expressed genes associated with a shared regulatory program or subnetworks of interconnected regulators and target genes. Regulatory functions, on the other hand, are used for determining the expression level of target genes as a function of the TF expression level or state.

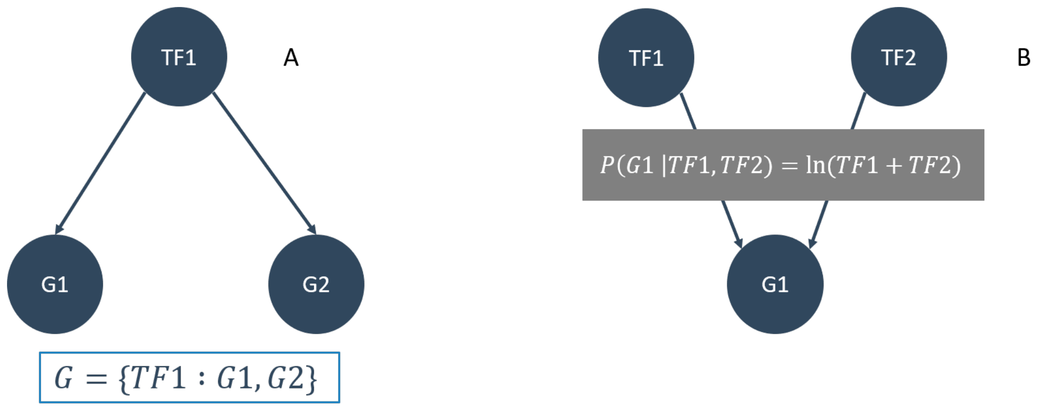


Figure 6: Difference between structure and function of TRNs graphs. The structure (A) of a TRN graph defines which TFs are connected to a given target gene. The function (B) of a TRN graph determines how a TF is connected to a target quantifying the expression levels of the target as a result of the connections.

Inferring TRNs is fundamentally an underdetermined problem associated with a large search space where many solutions explain the data equally well [15], [26], [105]. High-quality transcriptional information is scarce in databases or literature, being narrowed down to a few well-studied organisms (see section 2.3). The number of potential regulatory interactions between a TF and target genes is considerably larger than the actual true biological interactions. Given the example of the latest TRN for *B. subtilis* [106], with 275 regulators out of 4237 protein encoding genes, 1165175 regulatory interactions (275 regulators x 4237 genes) would be possible without combinatorial regulation. Even if filtering out potential regulatory interactions that poorly explain the gene expression data of this bacterium, the resulting number of solutions is still prohibitively large. In addition to the large search space, prokaryotic TRNs are typically very sparse, meaning that the number of false positive regulatory interactions predicted by the reconstruction methods is also a subject of concern.

To tackle the underdetermined problem, most approaches implement optimization strategies, with different mathematical formulations filtering the most inadequate solutions in a time-efficient way. Another approach regularly used to overcome the large search space is the integration of different layers and sources of regulatory information.

2.6 Reconstruction of transcriptional regulatory networks

Methods for reconstructing TRNs have been extensively reviewed in the literature [15], [18], [19], [26], [104], [107], [108]. In fact, there is a panoply of classification systems and procedures. As new methods are released each year, the complexity increases. Hence, assigning classes to these new approaches can be a complicated task. More importantly, this reveals that standard platforms and methodologies to assemble TRNs using diverse sources of regulatory information, such as gene expression data [65], [66] or transcriptional information [67], [69], [109], are still missing.

Nevertheless, an integrative workflow for reconstructing bacterial TRNs has been proposed by Faria *et al* [15]. The authors suggested that comparative-genomics approaches, namely the inference of TRNs using template curated networks and the prediction of *cis*-regulatory elements, can be integrated with the output of *de novo* reverse engineering tools. The workflow depicted in figure 7 addresses the possibility of reconstructing TRNs for poorly described prokaryotic organisms using a variety of sources of regulatory information.

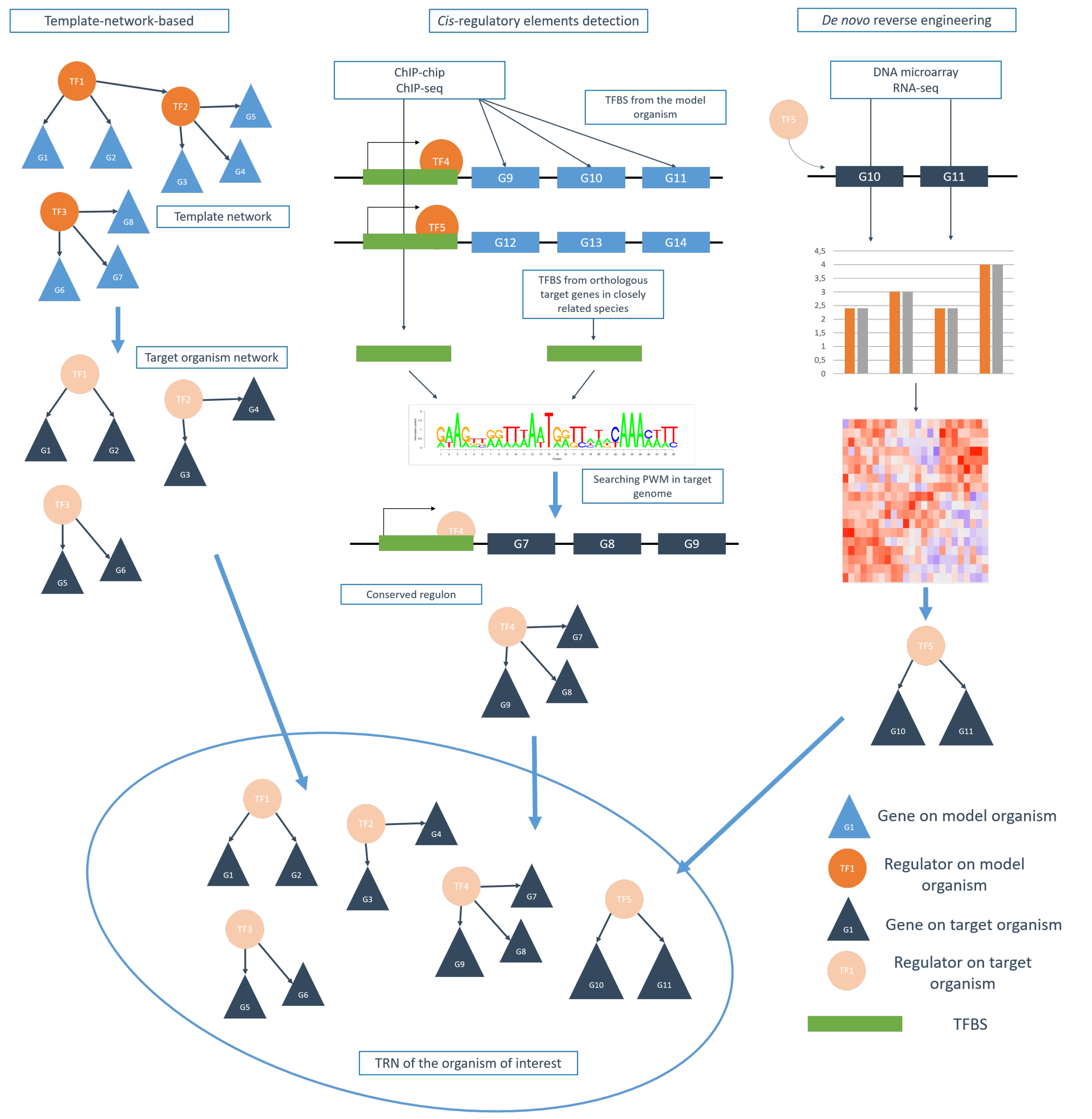

Template-network-based
Cis-regulatory elements detection
De novo reverse engineering
Template network
Target organism network
ChIP-chip
ChIP-seq
TFBS from the model organism
TFBS from orthologous target genes in closely related species
Searching PWM in target genome
Conserved regulon
DNA microarray
RNA-seq
TRN of the organism of interest
Gene on model organism
Regulator on model organism
Gene on target organism
Regulator on target organism
TFBS

Figure 7: Workflow for reconstructing TRNs using integrative approaches. Workflow for reconstructing TRNs using *de novo* reverse engineering tools combined with comparative-genomics approaches, namely template-network-based and *cis*-regulatory elements detection, as suggested by Faria *et al* [15]. Template-network-based methods rely mostly on experimentally defined networks to perform a search for orthologous genes in the genome of the organism of interest. Methods based on the detection of *cis*-regulatory elements are aimed at identifying regulatory interactions between TF and target genes using collections of TFBS to create PWMs. These PWMs are then used to search the corresponding interactions in the genome of interest. *de novo* reverse engineering tools use gene expression data to determine regulatory interactions.

Template-network methodologies are based on the conservation of prokaryotic TRNs across evolution [31], [110], [111]. As described by Faria *et al* [15], template-network-based methods usually perform a search for orthologous genes in the genome of the organism of interest to propagate TRNs to strains of well-characterized model organisms or closely related ones.

Cis-regulatory elements detection relies on the assumption that a regulatory interaction between a given TF and target gene can be inferred from the detection of the TFBS. The prediction of these *cis*-regulatory elements is a problem in which computational methods can assist [104], [112]. Although these computational tools are unable to infer a complete TRN from binding site data, they can be integrated into the following workflow towards such a goal. The principles of this methodology were implemented by Alkema *et al* [113] in Regulogger and the RegPredict web-based platform [114].

De novo reverse engineering tools are widely used for inferring TRNs from gene expression data. A vast repertoire of computational tools based on the *de novo* reverse engineering approach can be found in the literature, and consequently numerous ways to classify these tools [18], [19], [107]. Nevertheless, *de novo* reverse engineering methods are usually classified by mathematical formulation. Data-driven methods are usually based on the following:

- Correlation (e.g. *COREGNET* [115]);
- Information-theoretic (e.g. Faith *et al* [94]);
- Boolean algebra (e.g. *ModEnt* [116]);
- Regression-based (e.g. *GENIE3* [117]);
- Ordinary Differential Equation (ODE)s (e.g. *Inferelator* [105]);
- Bayesian models (e.g. Gat-Viks *et al* [118]).

2.7 Reconstruction of genome-scale metabolic models

The generation of GEM models is a common practice in systems biology. The reconstruction of these comprehensive models, through modeling techniques and genomics data, allows predicting cells' metabolic behavior [12], [13], [119].

As presented in figure 8, a GEM model is an *in silico* representation of the biochemical reactions taking place within the metabolism of a given organism [120]. A genome-wide functional annotation that provides the required metabolic information over the organism of interest should be performed to assemble this representation. This information is linked to existing metabolic knowledge retrieved essentially from biochemical databases and literature. These steps help to create the reaction set, upon which the metabolic network is assembled.

The link from metabolic genes to proteins (mainly enzymes or membrane transporter proteins), as well as from proteins to reactions, is established by GPR associations. GPR associations must be cautiously defined during the reconstruction, taking into account isoenzymes, protein complexes, and cascade reactions, through the use of *AND* or *OR* Boolean rules [13].

In the next iteration, biomass and organism-specific constraints are formulated from the retrieved knowledge to assemble a final stoichiometric model. The final GEM model may then be exported in a standard format, such as the Systems Biology Markup Language (SBML) [121]. Several platforms, such as merlin [122], ModelSEED [123], RAVEN [124], and CarveMe [14], have been developed specifically for performing or assisting in the reconstruction of these models [125].

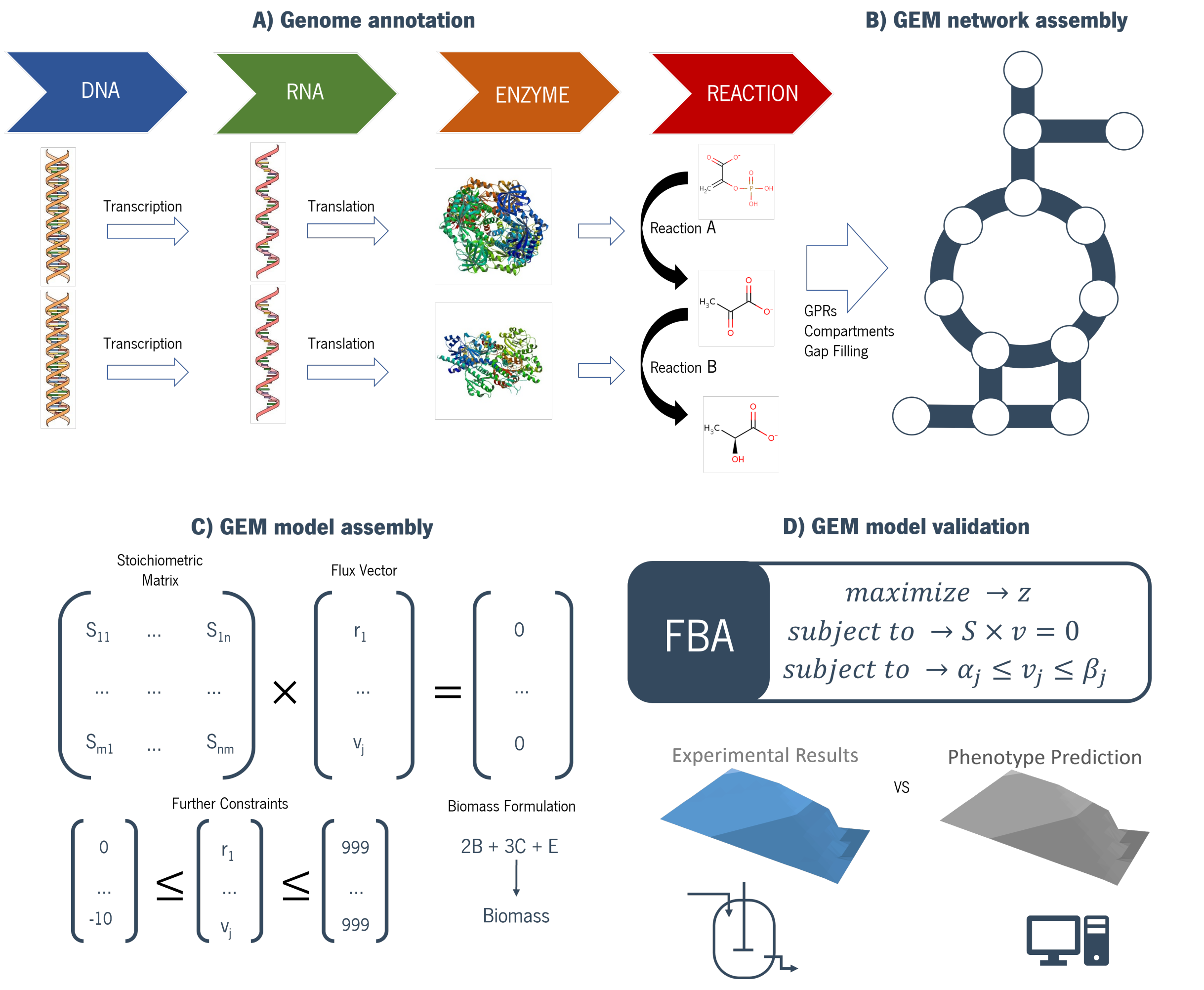


Figure 8: Reconstruction of a GEM model. Reconstruction of a GEM model involving four steps: (A) Genome annotation; (B) GEM network assembly; (C) GEM model assembly; (D) GEM model validation.

The classic principles of chemical engineering are used to infer the dynamic mass balances of all metabolites in the Genome-scale metabolic network. A single ODE is created for each metabolite, accounting for its stoichiometry in the whole reaction set. Due to the lack of kinetic rates for all reactions in the ODE set, a steady-state approximation is used to reduce the mass balances to a set of linear equations. In a pseudo-steady-state paradigm, the concentration of a metabolite is assumed to remain constant throughout time [12].

When used to determine flux values, the set of linear equations defines a linear system, typically underdetermined, as the number of fluxes is much higher than the number of mass balance constraints, also referred to as the null space of S [126]. Additional mass balance constraints can be added to the system to limit the flux that each reaction can accommodate by fixing both lower and upper bounds.

The system can be solved mathematically transforming it into an optimization problem, using several constraint-based approaches to predict the phenotypic behavior of the organism under a wide variety of environmental and genetic conditions. One of the most popular approaches is the FBA framework [126]. FBA can compute an optimal solution, out of the feasible space determined by both mass balance and flux constraints using Linear Programming (LP). FBA requires the definition of an objective function, which should be relevant to the undergoing problem. This is commonly defined as the maximization or minimization of a specific metabolic flux (e.g., biomass reaction), and quantitatively determines how much each reaction contributes to a phenotype [12]. Notation 2.1 shows the definition of an objective function, whereas notations 2.2 and 2.3 represent the optimization problem constraints:

$$maximize \rightarrow Z \tag{2.1}$$

$$subject \rightarrow S * v = 0 \tag{2.2}$$

$$subject \rightarrow \alpha_j \leq v_j \leq \beta_j, j = 1, ..., N \tag{2.3}$$

Where:

- Z is the linear objective function;
- v is the flux vector (includes exchange fluxes);
- S is the stoichiometric matrix (columns represent reaction fluxes and rows the metabolites mass balances);
- α_j and β_j are the lower and upper bounds, respectively.

Parsimonious Flux Balance Analysis (pFBA) [127] and Flux Variability Analysis (FVA) [128] are alternative mathematical frameworks that also employ LP to analyze *in silico* flux distributions. This set of tools is extremely helpful for validating a reconstructed model using experimental data of the organism of interest (figure 8). COBRA Toolbox [129], COBRApy [130], OptFlux [131], and ReFramed (https://github.com/cdanielmachado/reframed) are prominent computational tools that have implemented these methods.

Although GEM models have proven to be valuable throughout the years [5]–[10], [132], there are limitations. Indeed, they are not yet capable of accounting for biological regulatory phenomena, such as the control of gene expression [15]. The lack of the regulatory layer in these models can lead to erroneous *in silico* phenotype simulations, due to the lack of constraints that allow reaching the most accurate flux distribution according to experimental data.

Several methods have been proposed to improve phenotype simulations obtained from GEM models, which will be herein surveyed. Most of these new methodologies are aimed at combining additional layers of omics data, namely transcriptomics, to limit the cone of allowable flux distributions. Also, these methods often resort to the integration of gene expression data and/or regulatory information obtained from TRNs being, therefore, prominent efforts made towards the reconstruction of integrated metabolic-regulatory models. The utilization of these integrated models can be useful to improve phenotype simulations or extend the analysis of regular GEM models.

2.8 Integrated models

Combining regulatory elements with information on metabolic stoichiometry is a complex task. There are many ways of controlling metabolism [20], which are well represented in the large diversity of methods proposed to quantify such influence [15], [16], [36], [133]–[143]. Nevertheless, the common denominator is that most methods start with GEM models.

In detail, several of these methods integrate complete functional TRNs [144]–[152] or gene expression data [153]–[163] into GEM models, whereas others impose additional constraints using the information on allosteric and post-translational modifications [36], [142], [143]. A different strategy is a combination of multiple layers of regulation [133], [137], [138], [141]. For higher eukaryotes such as humans, the control of gene expression also plays an essential role in the differentiation between different tissues or cell types. Thus, algorithms for tailoring a GEM models according to a specific cell line or tissue, commonly referred to as context-specific models, have been proposed [139], [164]–[172]. These principles and their main implementations are depicted in Figure 9.

Surveying all approaches is out of the scope of this project. The following sections will cover the integration of TRNs or gene expression data into GEM models, focusing on controlling gene expression at the transcriptional level. Figure 10 highlights both approaches, namely the integration of TRNs (Figure 10 A) and gene expression data (Figure 10 B) into GEM models.

The differences between the integration of TRNs and gene expression data into GEM models are associated with the type and amount of data that these sources can offer to the metabolic landscape of GEM models.

Methods capable of integrating TRNs into GEM models provide comprehensive knowledge regarding the metabolic and regulatory events occurring inside the cell at the genome-scale. As a result, both regulatory and metabolic networks can be analyzed together at the genome-scale, extending the range of applications of a regular GEM model. On the other hand, gene expression data comprises a set of snapshots of the transcriptome for several experimental conditions. Thus, a gene expression dataset can solely offer gene expression levels at a given experimental condition.

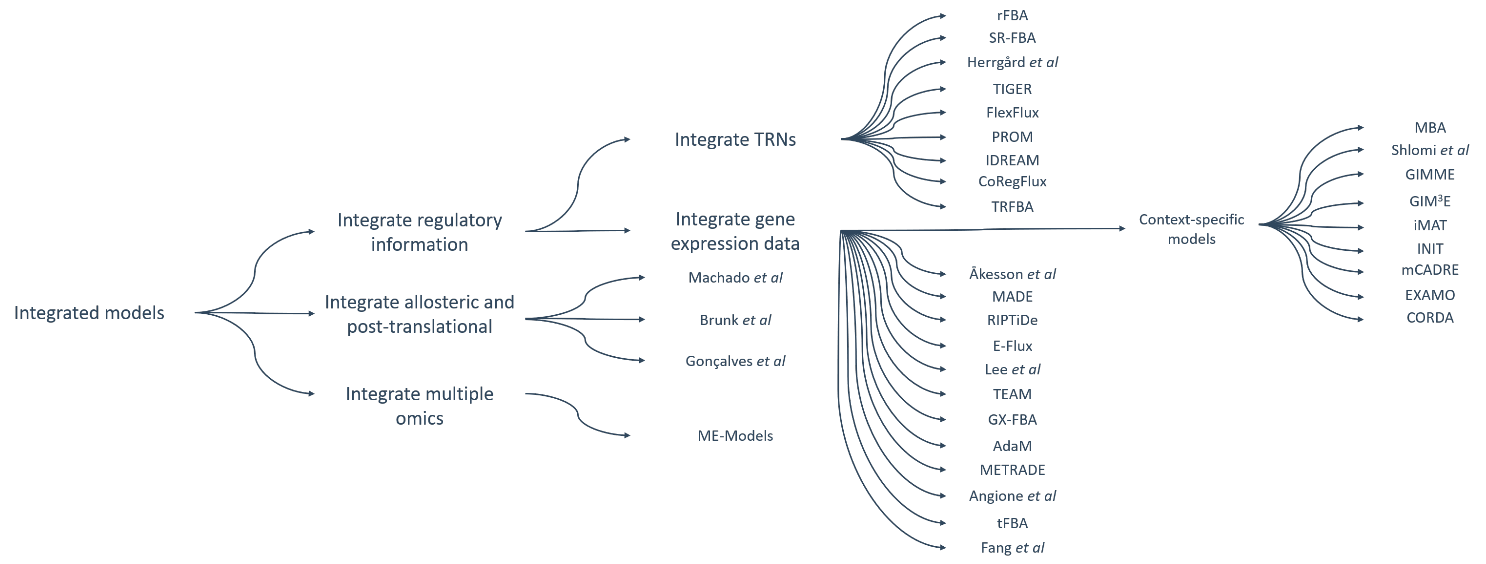


Figure 9: Overview of several methods for integrating additional constraints into GEM models based on the regulation of metabolism. Whereas some methods integrate complete functional TRNs or gene expression data into GEM models, others impose further constraints based on allosteric and post-translational modifications. Additionally, other methods integrate multiple omics layers of regulation of metabolism. For higher eukaryotes such as humans, context-specific models have also been based on tailoring the flux cone of solutions.

The group of methods aimed at integrating gene expression data with GEM models comprises methods using only transcriptomics data for tailoring the flux distributions, so no structure or rules describing the regulatory interactions are observed in this class of methods. Thus, the integration of gene expression data focuses on improving the prediction of flux distributions, rather than the study and analysis of an additional biochemical network at the genome-scale.

Methods have also been classified according to the main formulations, as previously suggested by Machado *et al* [16]. Organizing methods into containers, according to their main formulations and features, facilitates the decision process when selecting an adequate method for the existing constraints and data sources. Hence, methods were classified into discrete (Figure 10 C) or continuous (Figure 10 D), according to whether phenotype simulations were performed with discrete, namely Boolean logic ("ON/OFF"), or continuous constraints.

Accordingly, a method is systematically classified as discrete if the result of the integration is a Boolean value (e.g., 1 for "ON" and 0 for "OFF"), imposing additional constraints on the system. These methods are also referred to as "switch" methods since TRN or gene expression data switch reactions on or off. The state of a given metabolic gene is determined by evaluating the Boolean regulatory rule or thresholding the gene expression level. Then, metabolic reactions mapped to metabolic genes are accessed according to the GPR rules to determine the resulting states. Thus, reactions having a one-to-one direct GPR rule are active/inactive according to the state of the metabolic gene. Reactions catalyzed by enzyme complexes, encoded by multiple yet mandatory genes, are considered inactive if at least one metabolic gene is unavailable. In contrast, reactions catalyzed by isoenzymes, namely multiple enzymes catalyzing the same reaction, are considered active if at least one metabolic gene is active.

Alternatively, there are methods aimed at circumventing the rigid Boolean logic, called "valve" methods, which impose continuous constraints to adjust a given flux distribution gradually and according to penalties, expression scores, or normalized expression levels obtained from the gene expression data. Typically, continuous integration is performed by implementing slack variables in the constraints' formulations, altering the reactions' bounds. The slack variable represents penalties, expression scores, or normalized expression levels retrieved from gene expression data for the metabolic genes associated with a given reaction. As before, the reactions' state is obtained from the metabolic genes using the GPR rules, selecting the best penalty, expression score, or normalized expression level for the slack variable.

When a set of isozymes catalyzes a given reaction, the methodology for assigning a value to the slack variable comprises several distinct approaches. These include: methods in which the slack variable assumes the maximum expression score of the associated genes; methods where the slack variable takes the sum of expression scores of all genes encoding the isozymes catalyzing a single reaction; and, methods in which the reaction is replicated, according to the number of isozymes, and each new reaction is associated with one, and one only, gene.

Regarding reactions catalyzed by an enzyme complex, a group of methods establishes that the minimum expression score of all encoding genes is assigned to the slack variable. In contrast, other methods define the utilization of the geometric or arithmetic mean of the expression score of all genes associated with an enzyme complex or isozymes.

Furthermore, methods capable of integrating gene expression data into GEM models were also divided into single-condition (Figure 10 A) or multi-condition (Figure 10 B). Notice that this classification was not used to classify those methods aimed at integrating TRNs into GEM models, as will be explained next.

Methods were classified as single-condition (Figure 10 A) or multi-condition (Figure 10 B) according to whether phenotype simulations were performed for one or more conditions/states in the gene expression dataset, respectively. For instance, a given method is considered multi-condition if it adjusts the flux cone of solutions by considering all conditions in the gene expression dataset or the gene differential expression between two conditions. Otherwise, the methods are classified as single-condition.

Notice that the latter classification was not used to classify those methods aimed at integrating TRNs into GEM models. Methods capable of integrating TRNs into GEM models do not require a gene expression dataset, thus classifying them into single- or multi-condition would be meaningless. Other methods capable of assembling and integrating TRNs into GEM models often use the whole dataset and perform condition-specific phenotype simulations. Hence, classifying these methods as single-condition would be misleading.

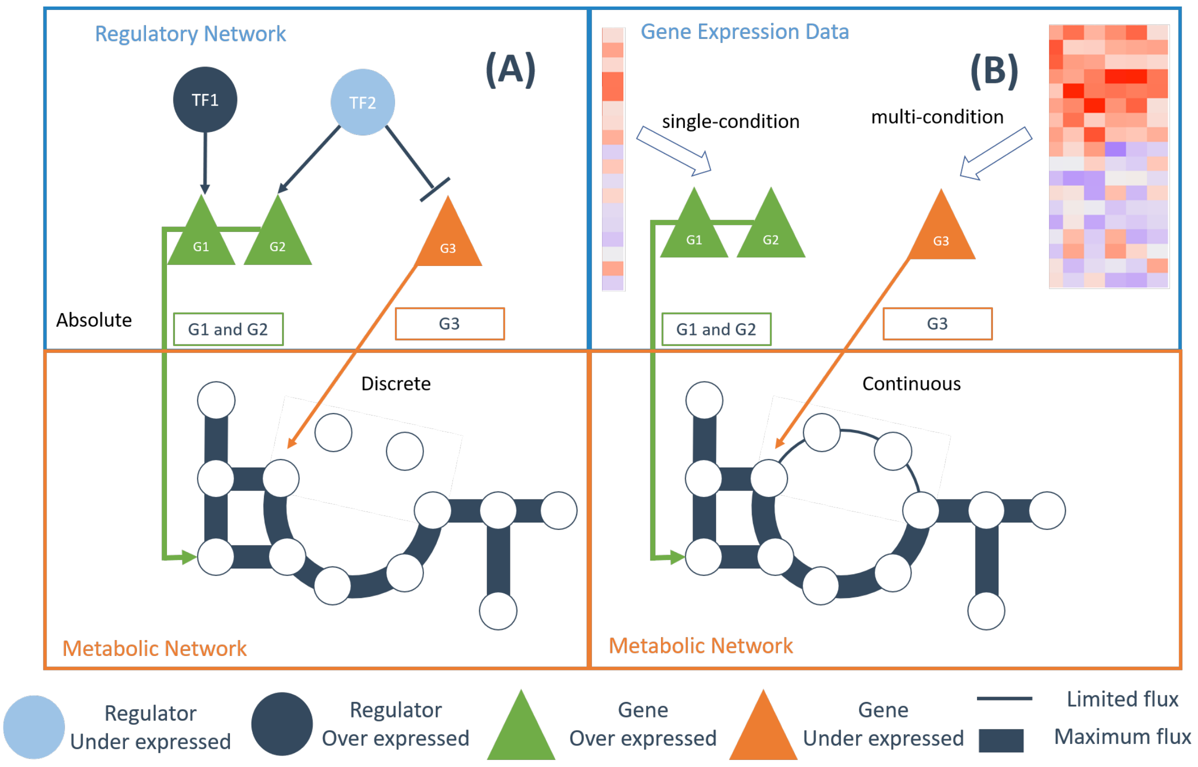


Figure 10: Two examples of the integration of TRNs (A and C) or gene expression data (B and D) into GEM models. The integration of TRNs (A) does not require gene expression data, while methods that integrate gene expression data (B) are capable of tailoring the flux cone of solutions by accounting for one (single-condition) or more (multi-condition) conditions in the gene expression dataset. Both types of integration can be mediated by discrete (C) or continuous (D) variables.

An analysis of these methods, encompassing the year of publication, availability of a tool with a user-friendly interface (namely a Graphical User Interface (GUI) without the requirement of coding skills), type of reaction constraint formulation, as well as the organism used for proof of concept has also been conducted. This information is available in supplementary material I.1. Figure 11 provides, on the other hand, a complete understanding of the methods described next, as well as their categorization according to the classification axes described above.

2.9 Integrating transcriptional regulatory networks

For simulation purposes, the first attempts to integrate TRNs within GEM models, namely rFBA [32], [144], [173], [174], SR-FBA [145], and the method proposed by Herrgård *et al* [147], are based on the switch approach, to complement the metabolic system with additional constrains outlining which genes are activated or silenced in the network for specific *stimuli*.

As proof of concept, rFBA was successfully used to create the first integrated genome-scale model of metabolism and gene expression for *E. coli* [173], [174]. In this reconstruction, as well as in the integrated network of *S. cerevisiae* provided by Herrgård *et al* [147], Boolean networks collected from literature were integrated through a set of GPR rules with the GEM model imposing regulatory events as additional time-dependent constraints.

On the other hand, SR-FBA performs steady-state simulations by including all valid metabolic and regulatory constraints in the system in a single step through a Mixed-Integer Linear Programming (MILP) formulation. For that, nested Boolean expressions are formulated as a set of linear constraints by recursively iterating over the structure of the regulatory layer and GPR rules to add auxiliary variables representing intermediate Boolean terms [145]. As shown in Figure 11, these methods have been classified as discrete, and none provides a user-friendly interface without the requirement of coding skills.

Two platforms, namely Toolbox for Integrating Genome-scale Metabolism (TIGER) [152] and FlexFlux [151], have been developed for integrating Boolean-based TRNs into models. TIGER can convert a series of logic Boolean rules, which can be thought of as a Boolean TRN, into a set of mixed-integer inequalities. Then, several algorithms for integrating gene expression data into the metabolic model and simulating phenotypic behavior can be implemented in the toolbox. Other implementations already available in this toolbox, such as Metabolic Adjustment by Differential Expression (MADE) [162], can be used for simulations.

FlexFlux differs from TIGER insofar as it is the only tool that provides a user-friendly interface for integrating TRNs into GEM models. This computational tool developed in Java® allows the input of SBML [121] with the SBML Qualitative (SBML-qual) extension. SBML-qual is the standard file format extension for storing and sharing qualitative multi-state TRNs [175]. In this way, a regular SBML file can hold a computer representation of qualitative models of biological networks. Qualitative multi-state regulatory networks can then be used to determine multi-state qualitative constraints for metabolic flux analyses using FBA. Furthermore, FlexFlux allows the translation of the discrete qualitative states into continuous intervals, thereby constraining a reaction flux continuously or discretely [151].

PROM [146], PROM2.0 [176], and Integrated Deduced REgulation And Metabolism (IDREAM) [150] are all based on a probabilistic model for TRNs, which are integrated with a constraint-based model using a continuous method. PROM and PROM2.0 were the first attempts to circumvent the previous rigid discrete constraints added to a GEM model by setting the reactions' flux bounds proportional to the probabilities of their associated metabolic genes. In turn, the probability of a metabolic gene being activated in the whole set of conditions is defined together from the TRN and gene expression dataset provided as input. In short, PROM approaches can determine the probability of a given gene being or not activated when the set of regulating TFs is either activated or silenced. The probability is calculated according to the frequency of each gene active in the dataset (of either perturbed or over/under-expressed TFs). Likewise, the effect of perturbations on the regulatory network can also be robustly predicted.

Although PROM-based approaches are probably the best examples for integrating both TRNs and gene expression data into a GEM model, the gene expression dataset must have a large number of measurements per condition. PROM and PROM2.0 have been validated with *E. coli* and *M. tuberculosis* experimental gene expression data and the respective TRNs.

The IDREAM method resulted from the combination of Environment and Gene Regulatory Influence Network (EGRIN) [95], [177] and PROM frameworks to create an enhanced genome-scale model of metabolism and gene expression for *Saccharomyces cerevisiae* [150]. Contrariwise to the previous approaches, this methodology has used a *de novo* reverse engineering method called EGRIN to complement the yeast TRN collected from the database YEAst Search for Transcriptional Regulators And Consensus Tracking (YEASTRACT) [178]. Then, the phenotype simulations are conducted similarly as in the PROM-based approaches.

Transcriptional Regulated Flux Balance Analysis (TRFBA) [149] and CoRegFlux [148] also provide a framework for the integration of gene expression data and TRNs in a continuous manner. Whereas the former requires a TRN for the organism of interest, the latter provides tools for inferring the regulatory network from gene expression data using CoRegNet [115]. Nevertheless, CoRegFlux allows us to use a curated TRN rather than using the provided data-driven method.

Regarding the TRFBA methodology, this FBA-based approach considers gene expression levels as two additional types of continuous constraints. The first is represented by a constant parameter that converts the gene expression levels to the upper bounds of the reactions. The second type of linear constraint to be added to the system can be thought of as the linear regression of each target gene from the regulating TFs.

CoRegFlux differs from TRFBA in that it uses a statistical reverse engineering method to infer targets of a given set of regulators at the genome scale. Then, the influence score (similar to correlation scores for activation or repression) of each regulator in the set of target genes is calculated with CoRegNet from a large gene expression training dataset. Influence scores are used to train a linear model capable of predicting the gene expression of metabolic genes using a new gene expression dataset. These predicted levels of expression are then translated into flux bounds for the phenotype simulations using FBA or Dynamic Flux Balance Analysis (dFBA) [179].

The methods capable of integrating TRNs into GEM models addressed herein are available in supplementary material I.1.

2.10 Integrating gene expression data

The method proposed by Åkesson *et al* [163] and MADE [162] were the earliest approaches for tailoring the flux cone of solutions using discrete variables obtained solely from gene expression data. In the case of the method developed by Åkesson *et al* [163], a reaction is simply switched "off" with a zero flux bound if the associated genes are found to be under-expressed in the corresponding condition (single-condition method). MADE, on the other hand, tries to surpass the problem of arbitrary thresholding under-expression by considering multiple conditions (multi-condition method). Statistical significance between changes in gene expression levels across sequential conditions is calculated to infer whether a gene is activated [162].

E-Flux [180] and the method proposed by Lee *et al* [161] have introduced several novelties when compared with the previous methodologies. These methods were the first attempts to constrain an FBA-based model using continuous variables. Nevertheless, these approaches are radically different. E-Flux directly maps gene expression levels into flux-bound constraints, assuming the maximum flux of a given reaction to be a linear function of the expression of the associated genes in the same condition (single-condition method). Lee and coworkers [161] do not introduce or alter flux bound constraints directly into the GEM model. An alternative objective function that minimizes the distance between flux distributions and gene expression data is applied for each phenotype simulation (single-condition method).

The Transcriptional-controlled Flux Balance Analysis (tFBA) method, proposed by van Berlo *et al* [160], is aimed at overcoming the problem of setting an arbitrary threshold to determine whether a gene is activated or not. The tFBA assumption is that differential gene expression between two conditions should also be reflected in the flux of the reactions associated with this gene. For that, the authors formulated constraints defining upper and lower limits for fluxes according to the gene expression, assuming their transgression is possible. The optimization problem (MILP formulation) consists of finding the flux distribution that minimizes the number of transgressions.

The method developed by Fang *et al* [159] is based on the differential gene expression between two conditions, namely reference and perturbed conditions. This method assumes that the flux distribution of a reference condition can be determined using the FBA or FVA frameworks, while the differential gene expression between the reference and perturbed conditions is used for tailoring the flux distribution of the perturbed one. Also, this method considers the variation of the biomass composition between reference and perturbed conditions.

As with tFBA [160] and the method proposed by Fang *et al* [159], the Gene Expression Flux Balance Analysis (GX-FBA) method [158] also determines the flux distribution for the reference condition using FBA.

Then, GX-FBA employs a new objective function and new constraints derived from the difference between reference and perturbed states to perform the *in-silico* phenotype simulation of the latter state. A wide range of phenomena associated with temperature and known to induce virulence in the gram-negative bacterium *Yersinia pestis* was used as proof of concept.

Temporal Expression-based Analysis of Metabolism (TEAM) [157] and Adaptation of Metabolism (AdaM) [156] are the only methods developed for integrating time-series gene expression data into constraint-based models. The former uses dFBA [181] to predict time-series flux distributions based on temporal gene expression profiles. Using a cost minimization scheme similar to the strategy proposed in the context-specific Gene Inactivity Moderated by Metabolism and Expression (GIMME) method [166], TEAM is capable of determining the flux distribution of a GEM model, constrained with gene expression levels of each time step in the dataset. TEAM was tested with time-series gene expression data from *Shewanella oneidensis*.

AdaM consists of a flux-based bilevel optimization problem that extracts minimal operating networks from a given GEM model [156]. This algorithm infers minimal operating networks in agreement with the differential gene expression pattern between time steps. Then, Elementary Flux Mode (EFM)s [182] are computed with these minimal operating networks rather than computing the flux distributions at each time step. Reactions are weighted according to the number of EFMs in which these are present. The optimization problem consists of finding the minimal network having the largest weight.

Angione *et al* [154], [155] formulated methods, for example, the Metabolic and Transcriptomics Adaptation Estimator (METRADE), aimed at measuring the adaptability to a changing environmental condition over time. These approaches have provided equally valid methodologies for integrating gene expression data in metabolic networks. In short, these methods have modeled both the upper and lower bounds of each reaction as a continuous logarithmic function of the associated gene expression levels.

Reaction Inclusion by Parsimony and Transcript Distribution (RIPTiDe) [153] is aimed at circumventing the assumption that reaction fluxes are directly related to the gene expression levels for a given condition. Instead, the authors have proposed an unsupervised method that assigns weights (continuous variables) to reactions according to the normalized expression levels of associated genes over the entire dataset. Then, a pFBA simulation considering these linear coefficients is performed. The novelty of this method consists of its validation with precise transcript abundance obtained with RNA-seq.

The methods capable of integrating gene expression data into GEM models addressed herein are available in supplementary material I.1.

2.11 Synopsis

The reconstruction of GEM models is common practice in systems biology nowadays. The advent of the GEM model reconstruction for many organisms was facilitated by the adoption of standard protocols [13], as well as the existence of user-friendly computational tools [122], [123], capable of assembling these models from different genomic, enzymatic, and stoichiometric data. Nevertheless, the simulation of GEM models still presents today false-positive phenotypes for several environmental conditions.

The reconstruction of TRNs is a well-known strategy in systems biology for understanding the regulatory machinery of a given organism [19], [26], [107]. Although there are many methodologies for assembling a TRN, standard protocols and computational platforms are yet missing to support the reconstruction of TRNs for less described organisms using different data sources. The workflow suggested by Faria *et al* [15] highlighted several methodologies that can be combined to extend the reconstruction of TRNs to more bacterial species. To the best of our knowledge little progress has been made to provide a user-friendly platform capable of achieving such a goal. More importantly, the reconstruction of genome-scale TRNs using such integrative workflow would be pivotal for the reconstruction and simulation of integrated models.

The integration of the control of gene expression into GEM models has been surveyed in this work. A systematic classification that grasps the difference between the several methodologies, capable of integrating and simulating regulatory events into GEM models was proposed herein. Although some reviewed methods have already been surveyed before [15], [16], [134]–[136], [140], [183], TIGER, FlexFlux, ME-TRADE, IDREAM, TRFBA, CoRegFlux, RIPTiDe, and the method proposed by Angione *et al* have never been addressed elsewhere in reviews, to the best of our knowledge. Moreover, a detailed categorization that highlights the methodologies used to perform the integration of the regulatory layer into GEM models has not been provided. This systematic categorization can guide the decision process of selecting the most adequate method of integration and simulation.

As shown in Figure 11, there are several methods and toolboxes capable of integrating and simulating TRNs into GEM models using a discrete approach [144], [145], [147], [151], [152]. The TRNs used by these methods and toolboxes were mainly reconstructed from literature, which might be a time-consuming approach. The remaining methods allow assembling TRNs from gene expression data using *de novo* reverse engineering methods. The resulting TRNs can be integrated and simulated with a given GEM model. FlexFlux is the prominent exception as it can perform the integration of the TRN in the GEM model using either discrete or continuous variables.

To date, only two prokaryotic organisms, *E. coli* [144]–[146], [149], [151] and *M. tuberculosis* [146], [176], and the yeast *S. cerevisiae* [147]–[150], [152], have integrated genome-scale models as a result of the integration of complete TRNs into a metabolic network. Nevertheless, some of these reconstructions still require gene expression datasets, namely several methods in the continuous sub-group.

Regarding the methods for integrating gene expression data, most of these have provided means for integrating transcriptomics data as continuous constraints from one or more conditions (Figure 11). Only the method proposed by Åkesson *et al* [163], as well as MADE [162], use discrete variables to simulate integrated models of metabolism and gene expression. Besides *E. coli*, *M. tuberculosis*, and *S. cerevisiae*, methods for integrating gene expression data have also provided integrated models for *S. oneidensis* [157] and *Y. pestis* [158].

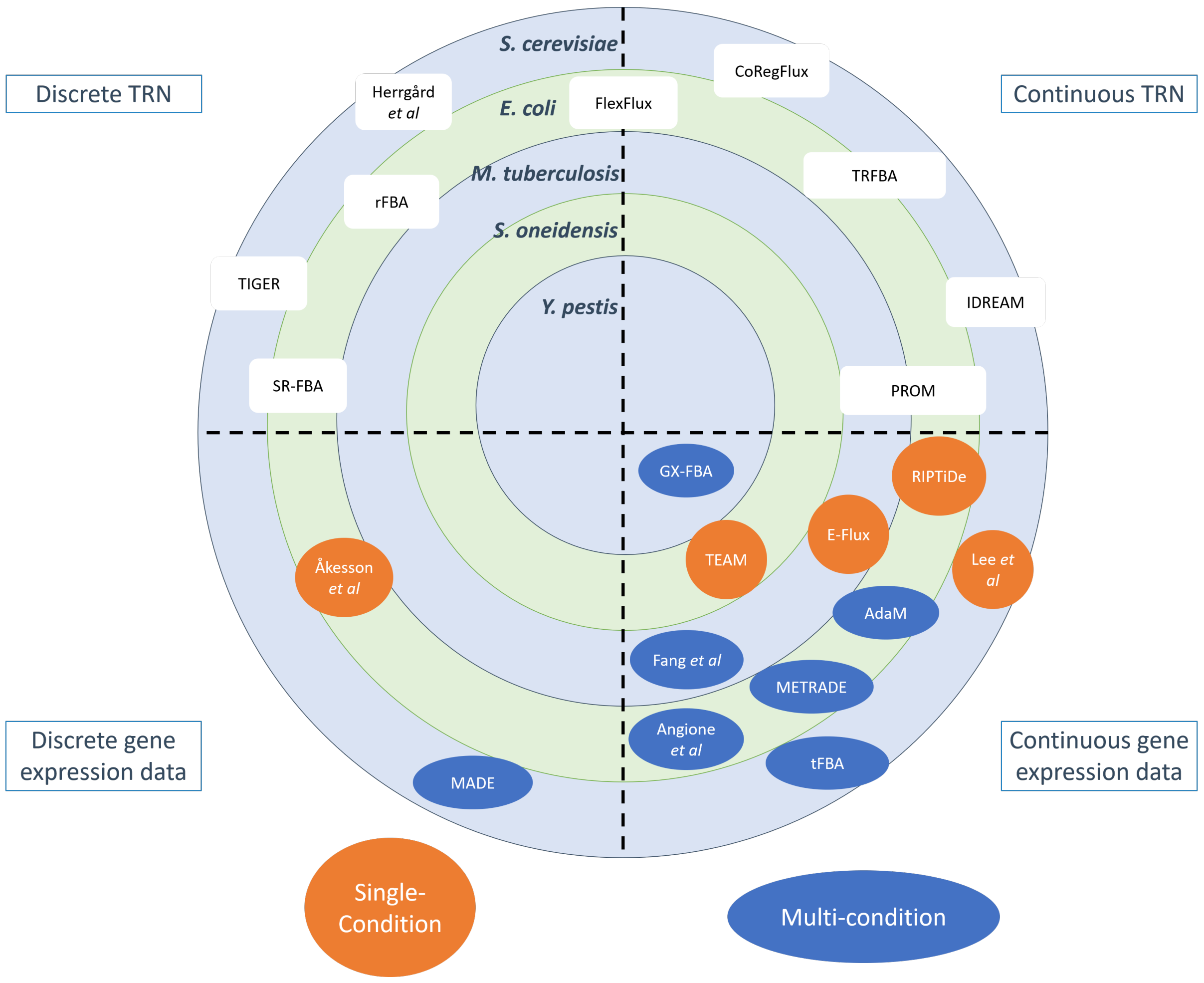


Figure 11: Classification of methods aimed at the reconstruction of integrated genome-scale models of metabolism and gene expression. These methods have been divided according to the integration of TRNs (white boxes) or solely gene expression data into GEM models. Discrete and continuous categories were used to classify these methods according to the usage of discrete, namely Boolean logic ("on/off"), or continuous constraints. Methods capable of integrating gene expression data into GEM models have been further divided into single-condition (orange circles) and multi-condition (blue ellipses) whether phenotype simulations were performed for one or more conditions/states in the gene expression dataset, respectively. Each inner circle stands for a prokaryotic organism, while the outer circle stands for the baker's yeast *Saccharomyces cerevisiae*.

A vast diversity of methods for the integration of gene expression data in GEM models has been found. Yet, most methods require large gene expression datasets to be robust, which might not be the case for all organisms. Other methods resort to mapping levels of gene expression directly with the reactions bounds, which again might not be the best approach [16], [38], [183].

Furthermore, the methods for integrating gene expression data with metabolic models previously evaluated by Machado and coworkers [16], namely E-Flux, MADE, GX-FBA, and the method developed by Lee *et al* [161] have shown to perform poorly in the designed benchmark. None of the methods have outperformed each other in the phenotype simulations or pFBA, which indicates that the promising results reported by these methods seem to be mere artifacts related to rigid constraints created around the nature of the gene expression dataset.

The reconstruction of integrated models using TRNs is, in theory, more useful than merely integrating gene expression data into GEM models. Integrated models that result from the integration of TRNs provide comprehensive knowledge regarding the metabolic and regulatory events happening inside the cell, thus leading to a broader range of applications when compared to a regular GEM model [184], [185].

Moreover, the diversity of methods for reconstructing TRNs using different data sources, such as gene expression, TFBS, or comparative genomics analysis, eases the reconstruction of TRNs for most prokaryotic organisms having a sequenced genome [15]. However, the absence of a user-friendly computational tool based on the ensemble of these different approaches is missing. In contrast, the same strategy has yielded results in the reconstruction of GEM models [14], [122]–[125], [129], [186]. In short, the existence of standardized protocols and easy-to-use computational tools for the generation of GEM models has eased its practice in systems biology to study the metabolism of many organisms. In contrast, the absence of computational tools that ease the reconstruction of TRNs from different sources of regulatory data hindered a similar approach.

The major obstacle when using the methods described in this survey to simulate integrated genome-scale models of metabolism and gene expression is not reproducing their results, but rather extending their implementations to other organisms and case studies. This hurdle poses a stiff challenge for using these methods out of the scope they were aimed at during development. The requirement for large gene expression datasets with specific experimental conditions, the usage of TRNs reconstructed solely from literature, and the output of biased results strictly related to rigid constraints, are specific indicators of issues preventing the scaling-up of the reconstruction and analysis of integrated models. In short, there is a vast diversity of methods capable of integrating and simulating the effect of regulation into the metabolism, though few approaches ease the reconstruction of these integrated models.

Hence, the perspective of reconstructing integrated metabolic-regulatory models for diverse prokaryotes is still a complex endeavor. The implementation of a user-friendly computational framework that does not require coding skills and is capable of running a semi-automated pipeline for reconstructing TRNs or analyzing gene expression data, and performing its integration into standard GEM models, would be a clear breakthrough towards the reconstruction and simulation of integrated genome-scale models of metabolism and gene expression. This hypothetical computational tool should be able to combine different sources of regulatory information that are seldom combined.

3 Assisting the curation of genome-scale metabolic models

The work presented in this chapter is currently pending review:

- *Cruz, F., Capela, J., Ferreira, E. C., Rocha, M., Dias, O. BioISO: an objective-oriented application for assisting the curation of genome-scale metabolic models. Submitted. Pre-print available at: https://doi.org/10.1101/2021.03.07.434259*

3.1 Introduction

The reconstruction of GEM model models is a challenging process [13]. Model validation and manual curation can be laborious tasks [12] and most bottlenecks derive from accumulated errors, which require complex and unique solutions. For instance, when a metabolic network is converted into a stoichiometric model, FBA often mispredicts the organism's experimental growth rate due to errors like missing or blocked reactions and dead-end metabolites (gaps), among others.

The reconstruction of GEM models can follow two diverse paradigms: bottom-up [13] and top-down [14].

The fast and automated top-down paradigm does not resort to gap-filling procedures. This approach reconstructs a universal GEM model curated previously for the most common errors [14]. This universal simulation-ready model is then converted to an organism-specific model by carving reactions and metabolites for which evidence is missing. Thus, the top-down paradigm can be extremely useful in creating microbial community models by merging the automated single-species models into community-scale networks [14].

Unlike the top-down paradigm, the widely-used bottom-up paradigm consists of four main steps: draft reconstruction based on genome functional annotation; refinement and curation of the draft reconstruction; conversion to stoichiometric model; model validation [13]. The last steps of a bottom-up reconstruction usually include several time-consuming and repetitive tasks to fix errors that emerged during the draft reconstruction, thus solving the discrepancy between the predicted phenotype and experimental results. While mistakes can be solved using manual curation, there are several gap-find and gap-fill tools to accelerate the debugging process. Gap-finding algorithms aim to find either missing or blocked reactions and dead-end metabolites in a draft reconstruction, whereas gap-filling ones are responsible for finding potential solutions to the errors mentioned above.

To the best of our knowledge, the reconstruction of high-quality GEM models is frequently based on the bottom-up approach involving manual curation and human intervention. In our view of a parsimonious bottom-up reconstruction, the metabolic network can be divided into smaller yet insightful modules based on the studied phenotype. Then, recursive relations can be used to divide metabolic networks into smaller modules directly associated with the objective phenotype. FBA simulations applied over surrogate reactions designed explicitly for each module can unveil the minor manual curation tasks that often increment the reconstruction's quality and resolve the metabolic gaps for such a module.

With this methodology in mind, BioISO was designed to automatize the search for reactions and metabolites associated with a given objective, narrowing the search space. More importantly, BioISO is a user-friendly gap-finding tool that allows users to analyze gaps and errors quickly. Reactions and metabolites are appropriately evaluated by BioISO that uses FBA over surrogate reactions. Then, the results are presented in a graphical user interface embedded in both a web server and merlin so that the debugging process can be easy to follow and repeat.

3.2 Survey of gap-find and gap-fill tools

Most state-of-the-art tools for debugging draft reconstructions comprehend both automated gap-finding and gap-filling procedures [187]–[190]. Nevertheless, other tools were developed for only one of these procedures [129], [191], [192]. Besides, gap-find and gap-fill tools can also be separated according to the gap-finding and gap-filling methodology. Several gap-find and gap-fill state-of-the-art tools have been described with further detail being given in the supplementary material I.2 and I.3.

Regarding gap-finding algorithms, tools such as *biomassPrecursorCheck* [129], and Meneco [187] are based on guided-search algorithms to identify gaps or errors directly associated with a given objective/reaction. Both tools check the metabolic network topological features to find gaps, asserting the existence of a given metabolite's predecessors and successors. The COBRA Toolbox's *BiomassPrecursorCheck* tool searches for predecessors immediately upstream of the biomass reaction of a given model, whereas Meneco performs the gap search according to a set of seed and target metabolites provided as input. However, the search depth of the latter may encompass the whole metabolic network.

On the other hand, gapFind/gapFill [188], fastGapFill [189], and Gauge [190] are based on exhaustive searches. Thus, these methodologies identify gaps all over the metabolic network, regardless of a given objective. GapFind/gapFill and fastGapFill highlight gaps using a stoichiometry-like approach. These methods search the stoichiometric matrix for no-production and no-consumption metabolites. Alternatively, Gauge combines Flux Coupling Analysis and gene expression data to propose gaps in a draft GEM model.

Regarding gap-fill, all available tools require a dataset of metabolic reactions, usually retrieved from a biochemical database (e.g., KEGG [81], BiGG Models [193], or MetaCyc [82]), to resolve metabolic gaps [187]–[192]. Besides a database of metabolic reactions, both Gauge [190] and Mirage [192] require gene expression data. Smiley [191] relies on additional growth phenotype data to identify minimal environmental conditions for which the model mispredicted growth and non-growth phenotypes. Meneco, gapFind/gapFill, fastGapFill, Gauge and Smiley consider the minimal reaction set of the whole dataset to resolve every single gap. Alternatively, Mirage considers a pan-metabolic network that assures flux through all metabolites, followed by a pruning step to reduce the large set of solutions. Thus, the solution set is often the result of two very different gap-filling approaches, namely the parsimonious and pruning approaches.

Most state-of-the-art tools for debugging draft reconstructions rely on proprietary software, such as MATLAB (Mathworks®) or General Algebraic Modeling System (GAMS). From the above-mentioned tools, Meneco is the only freely available to the community, as it is available as a Python package. It is worth noticing, though, that all tools require coding skills. More importantly, most tools require and return excessively verbose outputs, such as large arrays of missing metabolites and even greater sets of potential solutions. The analysis of these results can be challenging for wet-lab scientists without coding skills or data analysis expertise.

Furthermore, gap-filling tools usually warn that gaps might result from missing mappings between the metabolites' abbreviations and the reference database identifiers. Besides the mapping's limitations, several tools require different format files for the metabolic data, such as SBML (e.g., Meneco), KEGG reaction database LST format file (e.g., fastGapFill), customised text files (e.g., gapFind/gapFill), or data structures (e.g., Gauge). On the other hand, other tools lack information on how a different source of solutions can be used (e.g., Mirage and Smiley).

The introduction of artifacts in metabolic networks can hinder GEM model's applications, such as metabolic engineering and drug-targeting tasks. These issues may be extremely relevant for organisms that have evolved due to a combination of extensive loss-of-function events and acquisition of key genes via horizontal gene transfer during co-evolution with well-defined and constant ecological niches [194]–[196]. Moreover, although loss-of-function genetic variants are frequently associated with severe clinical phenotypes, several events are also present in healthy individuals' genomes, making it essential to assess their impact [197].

Hence, automated approaches, and especially gap-fill tools, must be used very carefully, taking into consideration the issues raised above. Otherwise, the offered automation can be a counterproductive solution for the manual curation steps performed during high-quality reconstructions. Furthermore, the usability of gap-fill approaches can be vastly improved.

3.3 BioISO implementation

3.3.1 BioISO's algorithm

BioISO requires a constraint-based model and the reaction to be evaluated, which defines the linear programming problem's objective function. A recursive relation-like algorithm is then used to build a hierarchical structure according to the metabolites and reactions associated with this objective.

BioISO is herein showcased through the analysis of a small-scale metabolic network, represented in figure 12, having 12 intracellular and two extracellular metabolites, 12 reactions, and two compartments (extracellular and intracellular). In this metabolic network, the reaction identified by R_8 is considered missing, blocked, or incorrectly formulated, while the identifier R_{11} refers to the reaction to be evaluated.

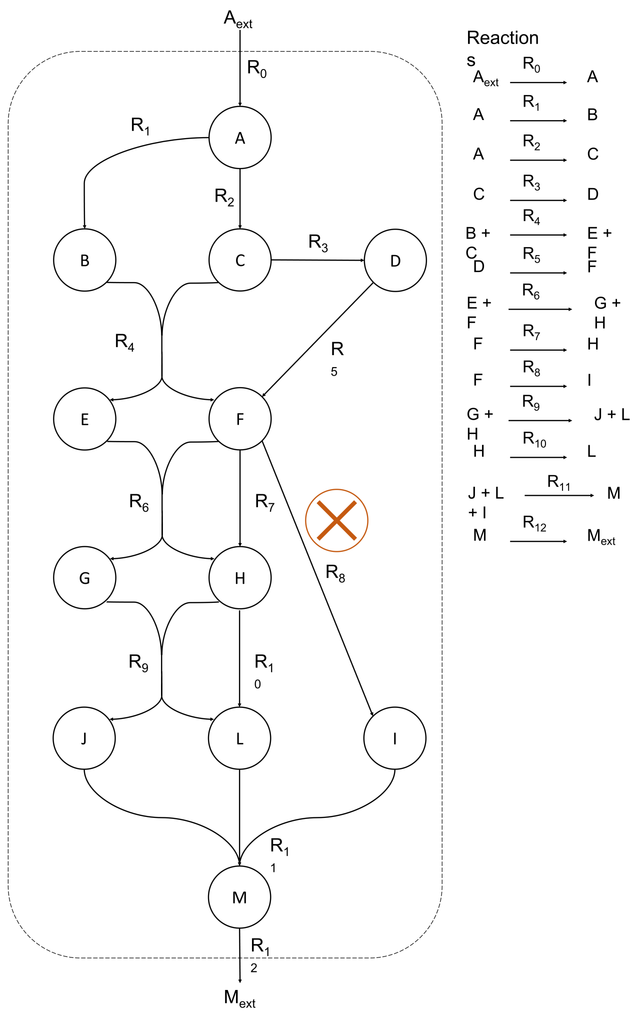


Figure 12: A small-scale metabolic network to showcase BioISO's algorithm. Metabolites and reactions are represented in the metabolic network as white nodes and black-directed arrows. The extracellular boundary is represented as a dashed line. The reactions are listed alongside the metabolic network. In this metabolic network, the reaction identified by R_8 is considered missing, blocked, or incorrectly formulated..

BioISO starts by finding the set of metabolites associated with the reaction submitted for evaluation, namely reaction R_{11}. The tool will discover metabolites *J*, *L*, *I*, and *M* as the subsequent nodes since these metabolites are involved in R_{11} (figure 13). A set of reactions is then created for each node and populated with the reactions associated with each metabolite. Thus, BioISO will retrieve four reactions sets, one for each metabolite (figure 13).

Meanwhile, BioISO assembles a hierarchical tree-based structure, depicted in figure 14. The tool identifies as precursors (reactants) or successors (products) the metabolites associated with the submitted reaction (R11). Thus, *J*, *L*, and *I* (reactants) and *M* (product) involved in rection R_{11} were separated into two different branches: precursors and successors, respectively.

In the next recursive call, BioISO retrieves metabolites *G, H, F, M, J, L, I,* and M_{ext} from reactions R_8, R_9, $R_{10,}$ and R_{12} (figure 13), while adding the precursors *G, H, F,* and *M*, and the successors *J, L, I,* and M_{ext} to the tree-based structure (figure 14). These reactions are either consuming or producing the metabolites identified in the previous step.

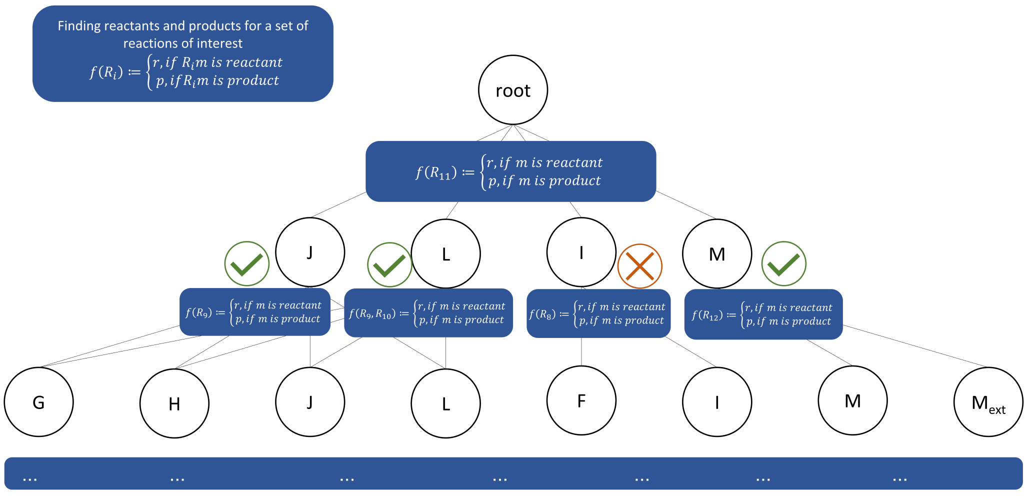

Finding reactants and products for a set of reactions of interest
$f(R_i) \coloneqq \begin{cases} r, if\ R_i m\ is\ reactant \\ p, if\ R_i m\ is\ product \end{cases}$
root
$f(R_{11}) \coloneqq \begin{cases} r, if\ m\ is\ reactant \\ p, if\ m\ is\ product \end{cases}$
J
L
I
M
$f(R_9) \coloneqq \begin{cases} r, if\ m\ is\ reactant \\ p, if\ m\ is\ product \end{cases}$
$f(R_9, R_{10}) \coloneqq \begin{cases} r, if\ m\ is\ reactant \\ p, if\ m\ is\ product \end{cases}$
$f(R_8) \coloneqq \begin{cases} r, if\ m\ is\ reactant \\ p, if\ m\ is\ product \end{cases}$
$f(R_{12}) \coloneqq \begin{cases} r, if\ m\ is\ reactant \\ p, if\ m\ is\ product \end{cases}$
G
H
J
L
F
I
M
M_{ext}
...

Figure 13: Evaluation of reaction R_{11} with BioISO. BioISO finds the set of metabolites associated with reaction R_{11}, which in this case corresponds to metabolites *J, L, I,* and *M*. For the next call, BioISO finds metabolites *G, H, J, L, F, I, M,* and M_{ext} in the reactions R_8, R_9, R_{10} and R_{12} and so forth.

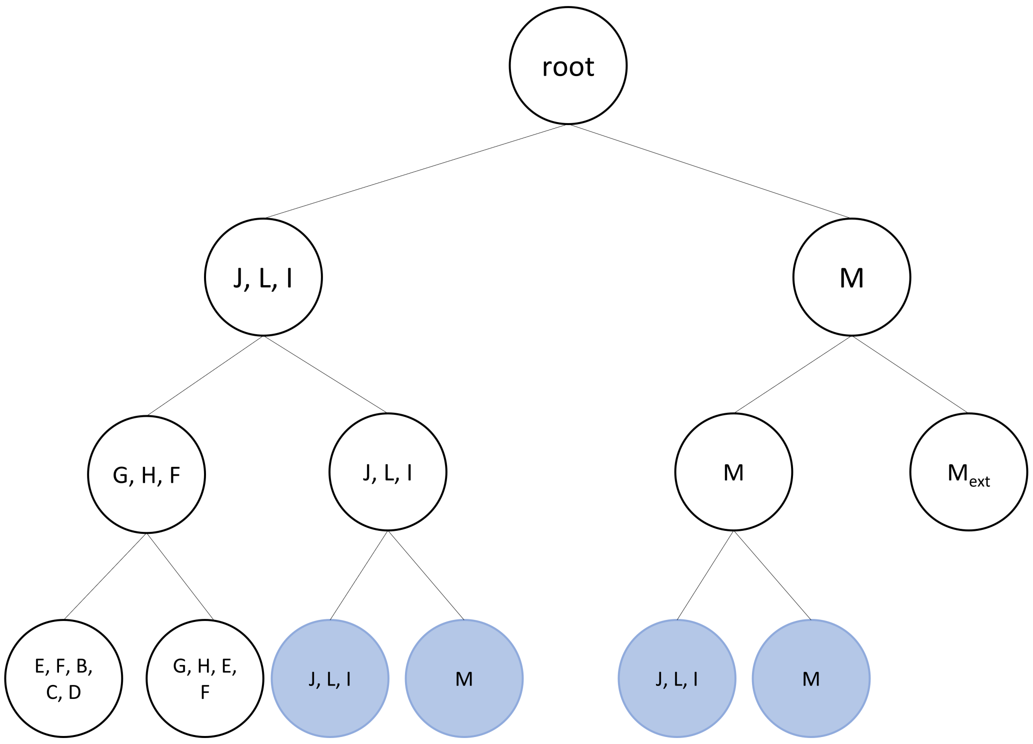


Figure 14: The hierarchical tree-based structure imposed by the recursive relation-like computational method implemented in BioISO. The hierarchical tree-based structure, imposed by the recursive relation-like computational method implemented in BioISO, is outlined. BioISO finds the set of precursors and successors in the first level, which in this case correspond to metabolites *J*, *L*, *I*, and *M*, respectively. For the next level, BioISO finds the precursors *G*, *H*, *F* for the previous precursors *J*, *L*, *I*, which are also successors of themselves. On the other branches, *M* is its precursor, while M_{ext} is the successor. BioISO has implemented a cache memory system of all simulations performed during the recursion. Thus, nodes colored in blue are only evaluated once, as they were already evaluated in those specific conditions.

The stopping condition, namely BioISO's depth, represents the number of recursive calls performed during the metabolic network analysis. For instance, varying BioISO's depth from 1 to 3 allows running the tool from shallow to guided or nearly exhaustive searches, depending on the metabolic network's size and arborescence.

The methodology for finding and assessing metabolites and reactions is detailed in algorithms 1-4 of supplementary material I.4. The first algorithm, labeled BioISO (algorithm 1 of supplementary material I.4), is the core logic supporting the methodology proposed in this work. BioISO uses the algorithm 2 of supplementary material I.4 to find and evaluate (using FBA) reactions associated with nodes. More importantly, BioISO uses a more comprehensive approach to assess reactants (precursors) and products (successors), as demonstrated in algorithms 3 (*testReactant*) and 4 (*testProduct*) of supplementary material I.4, respectively.

In detail, a **precursor (reactant)** is considered a positive assessment if the metabolic model can

produce it (connected metabolite). Thus, a reactant is a product elsewhere in the metabolic network; otherwise, it would not be available for the objective reaction. Hence, BioISO evaluates an unbalanced reaction explicitly designed to allow metabolite accumulation in the metabolic model. The evaluation is successful if the model can attain non-zero flux in the FBA solution for this surrogate reaction.

As described in notation 3.1, 3.2, 3.3, and 3.4, when evaluating the reaction *R11*, BioISO will evaluate the precursor *I* by adding an unbalanced reaction *R13*, which takes *I* as a reactant and whose lower and upper bounds are set to zero and plus-infinity, respectively.

$$maximize \longrightarrow v_{13} \tag{3.1}$$

$$subject \longrightarrow S * v = 0 \tag{3.2}$$

$$subject \longrightarrow \alpha_j \leq v_j \leq \beta_j, j = 1, ..., N \tag{3.3}$$

$$subject \longrightarrow 0 \leq v_{13} \leq +\infty \tag{3.4}$$

Where:

- v_{13} is the linear objective function for maximization of reaction *R13*.
- v is the flux vector.
- S is the stoichiometric matrix (columns represent reaction fluxes and rows the metabolites' mass balances).
- α_j and β_j are the lower and upper bounds, respectively.

Furthermore, a similar reaction is included in the model for each reactant to prevent the seldom cases in which all reactants are forcibly produced by reactions that produce/synthesize the assessed metabolite.

Likewise, BioISO creates unbalanced reactions that allow the uptake of all products associated with the evaluated reaction. These reactions are included in the model to prevent the unlikely scenario that the model forcibly needs to consume/metabolize such products to synthesize the precursor.

On the other hand, a **successor (product)** is considered a positive assessment if the metabolic model can **consume** it (connected metabolite). Thus, a product is a reactant elsewhere in the metabolic network. As described in the testing of precursor *I*, BioISO also creates an unbalanced reaction for the successor. However, this reaction is now explicitly designed to allow the metabolite **uptake** in the metabolic model. Thus, the minimization of this uptake reaction is now the objective function of the FBA simulation. In other words, the model should metabolize/consume the precursor metabolite, obtaining an optimal non-zero flux solution through the unbalanced reaction.

A detailed description of the BioISO workflow to search and assert gaps is provided in supplementary material I.4. In short, the procedure to split the objective into two sub-problems and evaluate both metabolites and reactions follows the workflow below:

- collect the reactions associated with each metabolite to be evaluated.
- maximize/minimize the reactions and assess the outcome of the FBA solution.
- from such reactions, find the precursor (reactants) and successor (products) metabolites.
- create unbalanced reactions allowing accumulation or uptake of the metabolites.
- maximize/minimize the unbalanced reactions and assess the outcome of the FBA solutions.

An analysis of BioISO's relation-like algorithm's complexity, together with the recursion tree method visualization, is also provided in supplementary material I.4.

3.3.2 BioISO's applications

BioISO is a package developed in Python 3 using the FBA framework implemented in COBRApy [130]. BioISO relies on COBRApy to read GEM models written in the SBML [198]. The IBM CPLEX solver (v. 1210) is used by default to solve multiple linear programming problems formulated with the FBA framework, although any solver supported by COBRApy can be used. BioISO's source code, validation procedures, and examples can be obtained from the GitHub repository at https://github.com/BioSystemsUM/bioiso.

A Dockerised Flask application has been implemented to make BioISO available to all scientific community at bioiso.bio.di.uminho.pt. This web service allows users to submit a GEM model in the SBML file format and evaluate a specific reaction in the model. BioISO's web service will then return a user-friendly web page highlighting the metabolic network's blocked reactions and dead-end metabolites according to the submitted reaction. Finally, the user is encouraged to navigate through the set of dead-end metabolites intuitively.

Besides the web service application, BioISO is also available as a plugin for merlin [199], an open-source and user-friendly resource that hastens the reconstruction of GEM models. This plugin allows BioISO to supply an equally user-friendly view of the errors associated with a given model reconstructed within merlin.

Finally, instructions to run BioISO in the available applications and interpret the expected results are also available at bioiso.bio.di.uminho.pt/tutorial.

3.4 Materials and Methods

3.4.1 BioISO's algorithm depth analysis

BioISO's algorithm depth analysis was performed in parallel for both objective functions, namely growth and compound production maximization. This assessment allowed setting shallow, guided, or nearly exhaustive searches with BioISO to assess the algorithm's robustness. BioISO's algorithm depth analysis was performed in five state-of-the-art GEM models: iDS372 (*Streptococcus pneumoniae*) [8]; iJO1366 (*Escherichia coli*) [11]; iBsu1103 (*Bacillus subtilis*) [10]; iTO977 (*Saccharomyces cerevisiae*) [200]; iOD907 (*Kluyveromyces lactis*) [9].

For instance, five incomplete models were created for the growth maximization analysis by removing the following reactions from the *E. coli* iJO1366 model ([11]), one at a time: *SDPTA*; *IMPC*; *MEPCT*; *NNDPR*; *SERAT*. The incomplete models were evaluated by setting BioISO's algorithm depth level at 1 and growth maximization as the objective function (Ec_biomass_iJO1366_core_53p95M). This procedure was repeated for the remaining models, with depth levels of 2 and 3, and compound production maximization analysis.

Then, two metrics were proposed to evaluate the gap-finding performance: the ratio of dead-end metabolites and the ratio of blocked reactions. These metrics were used to quantify the search space associated with debugging gaps and errors. As shown in definition 3.5, the ratio of dead-end metabolites for the objective-oriented search space (dem_{ooss}) is a function of the number of metabolites that a guided-search tool evaluates as unsuccessful (dead-end metabolite) divided by the size of the objective-oriented search space (*ooss*). The ratio of dead-end metabolites for the whole search space (dem_{wss}) is a function of the number of found dead-end metabolites divided by the *wss*, as described in definition 3.6. Definitions 3.7 and 3.8 describe a similar approach to calculate the ratio of blocked reactions for the objective-oriented search (br_{ooss}) and the whole search (br_{wss}) spaces, respectively.

Dead-end metabolite of the objective-oriented search space

$$dem_{ooss} = \frac{\sum DeadEndMetabolites}{\sum CoveredMetabolies} \tag{3.5}$$

Dead-end metabolite of the whole search space

$$dem_{wss} = \frac{\sum DeadEndMetabolites}{\sum Metabolies} \tag{3.6}$$

Blocked reaction of the objective-oriented search space

$$br_{ooss} = \frac{\sum BlockedReactions}{\sum CoveredReactions} \tag{3.7}$$

Blocked reaction of the whole search space

$$br_{wss} = \frac{\sum BlockedReactions}{\sum Reactions} \tag{3.8}$$

Objective functions, settings, evaluation metrics, and methodologies used to introduce gaps in state-of-the-art GEM models used during the algorithm depth analysis can also be consulted in detail at supplementary materials I.5 and I.6.

3.4.2 Exhaustive-search versus guided-search

The exhaustive-search versus guided-search analysis was performed for both iDS372 [8] and iJO1366 [11] models to assess the relevance of guided (BioISO and Meneco [187]) and exhaustive searches (fastGapFill [189]) for gap-finding.

BioISO was used as described in the previous section for a depth level of 2.

All metabolites available in the extracellular compartment of the iDS372 or iJO1366 models were used as seed metabolites to run Meneco gap-finding methodology. Likewise, the set of target metabolites was comprised of precursors and successors of the evaluated reactions. The set of dead-end metabolites was determined through Meneco's *get_unproducible* method. Note that Meneco cannot assert blocked reactions. Thus, the set of blocked reactions could not be determined.

fastGapFill was used to assess the whole search space by accounting for errors and gaps. fastGapFills' *gapFind* and *findBlockedReaction* methods were used to determine dead-end metabolites and blocked reactions, respectively, in the incomplete models.

The metrics described in the previous section were then used to assess the tools' performance: the ratio of dead-end metabolites and the ratio of blocked reactions. Supplementary materials I.4 and I.5 present all details about the assessment of BioISO with Meneco and fastGapFill.

3.4.3 BioMeneco - embedding BioISO in Meneco

The BioMeneco analysis was performed for the iDS372 [8] and iJO1366 [11] models to assess the integration of BioISO as Meneco's gap-finding algorithm.

Meneco's topological search finds dead-end metabolites, so the gaps associated with them can be filled with reactions from a universal database. The novelty of Meneco is that it allows selecting which gaps should be filled by tweaking the set of target metabolites. Hence, BioMeneco, BioISO's integration with Meneco, was performed to assess whether BioISO can suggest the right set of targets to be used as input in Meneco.

Reactions *R04568_C3_cytop* and *SO4tex* were removed from the iDS372 and iJO1366 models, respectively, to perform this assessment for growth validation. Meneco was then used to generate potential solutions for both models using two sets of target metabolites in parallel:

- The set of target metabolites comprised precursors and successors of the evaluated reaction in each model.
- The set of target metabolites was formulated based on identifying dead-end metabolites by BioISO.

BiGG Models [193] universal database was used as the source of metabolic reactions for the test iJO1366 model, while KEGG [81] was used to solve gaps in the iDS372 model.

3.5 Results

3.5.1 Overview of BioISO's assessment

BioISO aims to identify errors that emerge during the bottom-up reconstruction of high-quality GEM models. Errors such as missing or blocked reactions and dead-end metabolites are often met during model debugging and refinement. Thus, BioISO is based on a recursive-like algorithm to guide the search for metabolic gaps associated with a given objective. Throughout BioISO's objective-oriented search, multiple FBA simulations are used to assert real metabolic gaps. Hence, we propose a tool capable of reducing large search spaces and asserting real metabolic gaps to accelerate time-consuming and laborious manual curation tasks.

Most state-of-the-art tools for debugging draft reconstructions aim to find and solve a wide range of problems. These tools are commonly used in automatic gap-find and gap-filling routines. For instance, Meneco, gapFind/gapFill, fastGapFill, Gauge, Smiley and Mirage are gap-fill tools aimed at finding and solving errors accumulated during the draft reconstruction.

GapFind/gapFill, fastGapFill, and Gauge exhaustive-search tools attempt to assert gaps throughout the whole metabolic network. Then, these tools enumerate minimal solutions (set of reactions) to solve the highlighted gaps. Alternatively, Mirage and Smiley add new reactions to the model without an initial gap scan, forcing model predictions to match the experimental data.

On the other hand, Meneco's guide-search algorithm searches for gaps according to a set of seed and target metabolites. Then, this tool enumerates a minimal set of reactions that can restore the flux to all dead-end metabolites identified during the topological search.

Likewise, BioISO seeks dead-end metabolites downstream and upstream of a user-defined objective. Additionally, BioISO performs multiple FBA simulations of custom unbalanced reactions during the topological search to evaluate whether a given metabolite is being consumed or produced.

More importantly, BioISO is the only tool freely available to all scientists. That is, BioISO is the only user-friendly gap-finding tool, providing a graphical user interface embedded in both a web server and merlin. Thus, our tool allows users to analyze gaps and errors without requiring coding skills or additional metabolic data such as growth phenotype data or biochemical databases. Moreover, BioISO is a ready-to-use and relatively fast method, allowing users to run this tool iteratively during model reconstruction.

A summary of all features used to compare BioISO with several gap-find and gap-filling tools is available in supplementary materials I.2 and I.3.

BioISO's validation includes three assessments:

- BioISO's algorithm depth analysis.
- Exhaustive-search *versus* guided-search.
- BioMeneco – embedding BioISO in Meneco [187].

The first analysis was aimed at assessing BioISO's shallow, guided, or nearly exhaustive searches for metabolic gaps in five state-of-the-art models. The second analysis allowed us to assess the relevance of guided- and exhaustive-searches for gap-finding. In this assessment, we have compared BioISO and Meneco guide-searches against fastGapFill exhaustive-search. Finally, the last analysis showcases the outcome of setting BioISO as Meneco's gap-finding algorithm.

3.5.2 BioISO's algorithm depth analysis

BioISO was used to analyze several published GEM models for two objective functions: growth and compound production maximization. The workflow and methodology used to assess BioISO's algorithm robustness are described above together with supplementary materials I.5 and I.6.

BioISO's analysis included different settings, namely varying the algorithm's depth from 1 to 3, allowing to set BioISO for shallow, guided, or nearly exhaustive searches.

BioISO's depth level of 1 assesses the nearest neighbors (successors and precursors metabolites) and their associated reactions. According to figure 15, BioISO analyses less than 50% of all reactions for growth maximization. The number of blocked reactions is significantly reduced at depth level 1 (less than 12%), except for the iJO1366 [11] and iBsu1103 [10] models (figure 15 and supplementary material I.6). Likewise, BioISO only covers less than 10% of all metabolites for growth maximization (figure 15). As a result, the number of dead-end metabolites found by BioISO at a depth level of 1 is less than 3 for all models (figure 15 and supplementary material I.6). The level of insight provided by BioISO for shallow searches is significantly reduced and similar to the *biomassPrecursorsCheck* tool from COBRA Toolbox [129] or Meneco [187].

Increasing the depth level to 2 allows the evaluation of more reactions. As demonstrated in figure 15, 50% or more of the reactions are assessed in all models. Whereas BioISO analyses nearly a quarter of all metabolites in the iOD907 [9] and iTO977 [200] models, this coverage increases up to 60% in the iJO1366 [11] and iBsu1103 [10] models. In contrast, only 15% of all metabolites have been covered by BioISO in the iDS372 [8] model. BioISO also detects more blocked reactions and dead-end metabolites for guided searches. The percentage of blocked reactions varies between 20% and 40%, and the percentage of dead-end metabolites between 2% and 12% (figure 15 and supplementary material I.6).

At a depth level of 3, a nearly exhaustive search is performed as BioISO analyses more than 70% of both metabolites and reactions (figure 15 and supplementary material I.6). Likewise, the percentage of detected blocked reactions and dead-end metabolites increases up to 65% and 30%, respectively.

As detailed in supplementary material I.6, similar results were obtained for the maximization of compound production. However, the number of metabolites covered in the iTO977 model is considerably lower than the remaining models at depth levels 2 and 3.

Figure 15 also presents BioISO's computation time for each model during growth maximization as a function of the depth level. BioISO was considerably faster for shallow searches (depth level of 1) in all models for growth (figure 15 and supplementary material I.6) and compound production (supplementary material I.6) maximization. According to figure 15, BioISO required computation time for a depth level of 2 varies between 2 and 144 seconds during growth maximization. During the compound production maximization, BioISO takes between 4 and 74 seconds (supplementary material I.6). The computation time of BioISO increases significantly at the depth level of 3. At this depth, BioISO's computation time can attain around 600 and 405 seconds when maximizing growth (figure 15 and supplementary material I.6) and compound production (supplementary material I.6), respectively.

Hence, BioISO's computation time significantly depends on the size of the covered search space. In turn, the covered search space increases with the depth of the search and the model's size. Although navigating the network through the new metabolites and reactions might not be time-consuming, evaluating numerous metabolites and reactions using the FBA framework requires time.

The dead-end metabolites and blocked reactions ratios were calculated as denoted in definitions 3.5, 3.6, 3.7, and 3.8. Figure 16 highlights the ratios of dead-end metabolites obtained for growth and compound production analysis in all models. Both dead-end metabolites and blocked reactions ratios are also available in the supplementary material I.6.

At a depth level of 1, the ratio of blocked reactions for the objective-oriented search (br_{oos}) varies between 0.3 and 0.9. In contrast, the homologous ratio for the whole-space search (br_{wss}) varies between 0.02 and 0.25 (supplementary material I.6).

Regarding the ratios of dead-end metabolites, BioISO attains markedly small ratios for the whole-space search (dem_{wss}) at a depth level of 1, namely obtaining ratios smaller than 0.1 in all models for both objective functions (figure 16 and supplementary material I.6). However, the ratio of dead-end metabolites for the objective-oriented search (dem_{oos}) can peak up to 0.35 (figure 16 and supplementary material I.6) and 0.85 (figure 16 and supplementary material I.6) in the growth and compound production analysis, respectively.

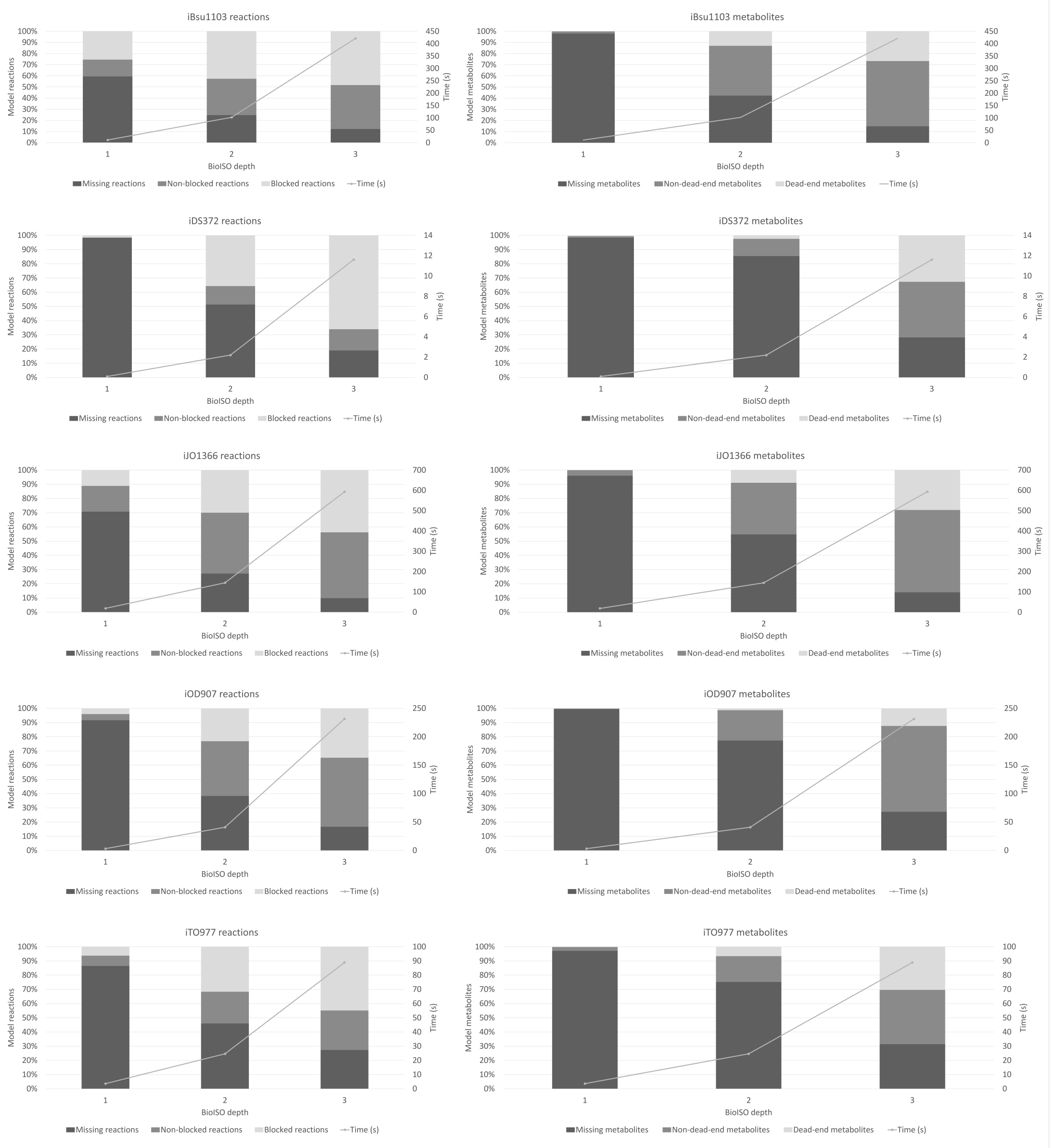


Figure 15: Summary of the reactions (left panel) and metabolites (right panel) analyzed by BioISO for published GEM models. BioISO was used to analyze 5 state-of-the-art models (iBsu1103, iDS372, iJO1366, iOD907, and iTO977) with added gaps. BioISO's analysis included different algorithm settings, namely varying the depth from 1 to 3 for each objective function, which will control the number of recursive calls for precursors and successors. This allowed running BioISO's algorithm for shallow, guided, or nearly exhaustive searches, depending on the size and arborescence of the metabolic network. BioISO's computation time was recorded in seconds (s) together with missing (non-covered by BioISO) reactions and metabolites, non- and dead-end metabolites, non- and blocked reactions.

The br_{wss} ratios increase significantly when raising the depth to level 2 (BioISO guided search), whereas br_{oos} ones remain roughly the same as in the previous level. The br_{wss} ratio can vary from 0.23 to 0.43 (supplementary material I.6) and from 0.23 to 0.36 (supplementary material I.6) for the growth and compound production analysis, respectively.

The dem_{oos} ratio tends to increase as a response to BioISO's guided search (depth level of 2) in the iBsu1103, iJO1366, and iTO977 models during the growth maximization analysis (figure 16). In contrast to the previous trend, the dem_{oos} ratio tends to decrease in the iDS372 and iOD907 models. Regarding the maximization of compound production, BioISO also attains higher dem_{oos} ratios in both iJO1366 and iTO977 models at a depth level of 2 (guided search). Nevertheless, the dem_{oos} ratio obtained in the iBsu1103 model is smaller in comparison to the value obtained for the shallow search (depth level of 1).

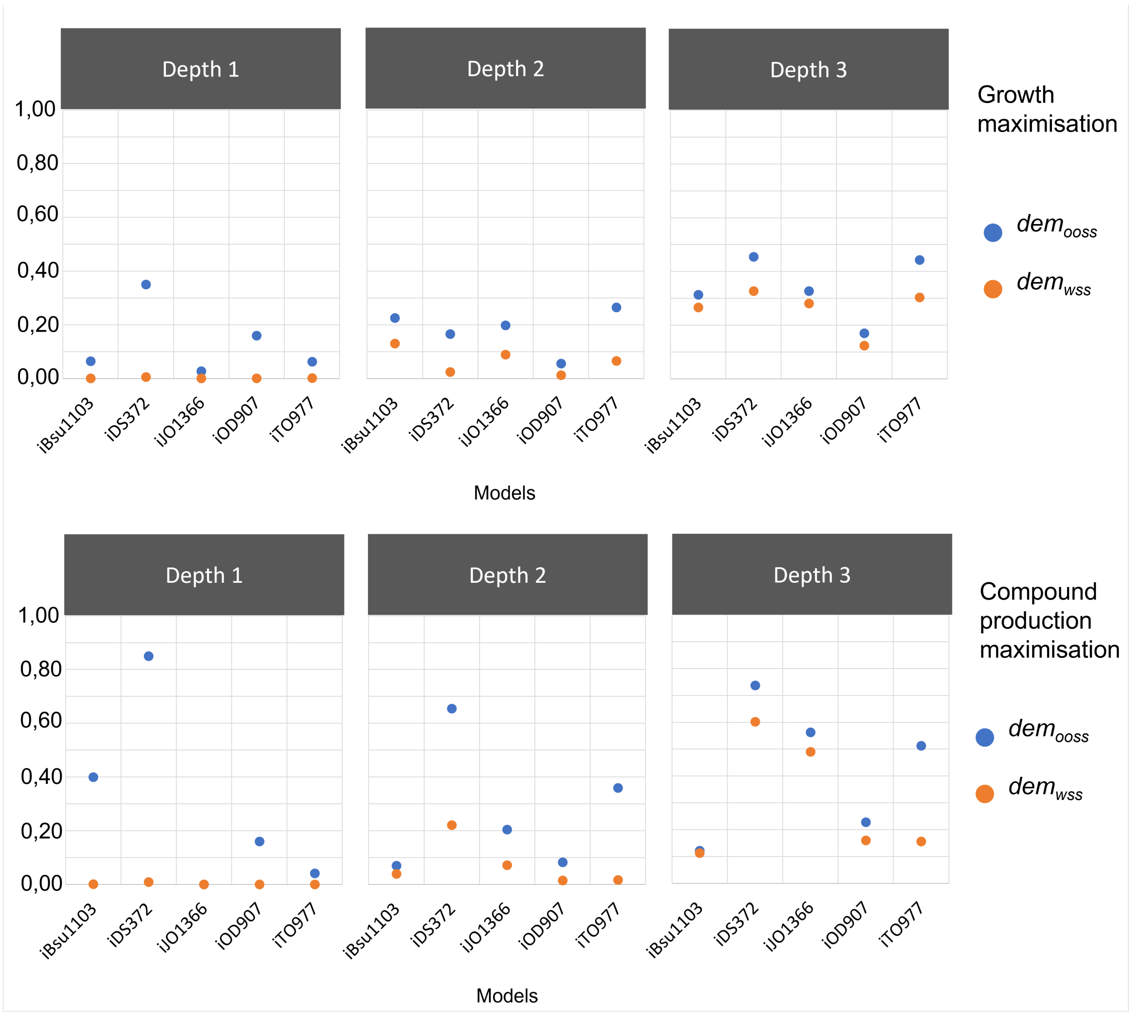


Figure 16: Calculated ratios of dead-end metabolites for the maximization of growth (upper panel) and compound production (bottom panel). BioISO was used to analyze 5 state-of-the-art models (iBsu1103, iDS372, iJO1366, iOD907, and iTO977) with added gaps. BioISO's analysis included different algorithm settings, namely varying the depth from 1 to 3 for each objective function, which will control the number of recursive calls for precursors and successors. This allowed running BioISO's algorithm for shallow, guided, or nearly exhaustive searches, depending on the size and arborescence of the metabolic network. dem_{ooss} and dem_{wss} stand for the ratios of dead-end metabolites for the objective-oriented search and whole search spaces.

In general, the dem_{wss} ratio tends to increase as a response to BioISO's guided search (depth level of 2) in all models for both objective functions, though not exceeding 0.221. As shown in figure 16 and supplementary material I.6, the dem_{wss} ratio ranges between 0.01 (iOD907 model) and 0.13 (iBsu1103 model) for the growth maximization analysis. During the compound production maximization analysis, the dem_{wss} ratio is less than 0.1 in all models except for the iDS372 model, where it peaks at 0.221 (figure 16 and supplementary material I.6).

Using BioISO for nearly exhaustive searches (depth level of 3) returns br_{oos} ratios between 0.41 and 0.81, whereas the br_{wss} ratio varies between 0.35 and 0.66 for both objective functions (supplementary material I.6). As for detecting dead-end metabolites during the growth maximization analysis, BioISO's nearly exhaustive search attains the highest dem_{oos} and dem_{wss} ratios of 0.455 and 0.327 in the iDS372 model (figure 16), respectively. Regarding the compound production maximization, BioISO peaked for a depth level of 3 dem_{ooss} and dem_{wss} ratios of 0.737 and 0.602 in the iDS372 model (figure 16), respectively.

Although the br_{oos} ratio oscillates when rising depth, the br_{wss} tends to increase steadily. Similarly, the dem_{wss} also tends to increase with depth for both objective functions, whereas the dem_{oos} ratio mimics the oscillatory behaviour of the br_{oos} ratio. The oscillatory behavior of the br_{oos} and dem_{oos} ratios is heavily pronounced between depths 1 and 2, which can be associated with the reduced level of detail that BioISO can provide for shallow searches.

The high br_{wss} ratios obtained for all levels are likely associated with the fact that BioISO does not prevent circular dependencies nor by-products accumulation when testing reactions. The interactive output of BioISO in both the webserver and merlin guides the user through the dead-end metabolites (precursors and successors having an unsuccessful evaluation) while evaluating the reactions for guidance and further insight.

When testing metabolites, BioISO's strategy to prevent circular dependencies and by-product accumulation, as well as the isolated evaluation of precursors and successors, seems to have a greater impact on reducing the set of dead-end metabolites. The dem_{wss} ratio is significantly smaller for shallow and guided searchers across all models for both objective functions.

Furthermore, although BioISO has attained dem_{wss} ratios higher than 0.4 for two models during the compound production maximization analysis with a depth level of 3, this ratio remains below 0.33 in all models during the growth maximization analysis. The dem_{wss} ratios higher than 0.4 obtained with nearly exhaustive searches of BioISO may be associated with the factual metabolic gaps that do not need to be corrected or might not be associated with the desired phenotype.

For example, BioISO has systematically attained higher ratios for all metrics when assessing the iDS372 incomplete models for both objective functions. These higher scores may be associated with poor connectivity of most metabolites involved in the metabolic pathways analyzed by BioISO, as parasitic organisms evolve in rich media, thus developing auxotrophies [8], [194], [201]. Hence, it is worth noticing that the identified dead-end metabolites might be associated with real metabolic gaps that should not be gap-filled.

In short, BioISO scores most of the smaller dem_{wss} ratios at the depth level of 2 (guided search). Moreover, the gap between dem_{wss} and dem_{oos} ratios also starts to narrow for BioISO's guided search.

The small difference between both metrics suggests that most dead-end metabolites suggested by BioISO are a direct outcome of the network gaps introduced during the validation. Hence, BioISO can suggest a higher number of dead-end metabolites associated with the objective while maintaining the curation efforts at a minimum.

Therefore, a depth level of 2 was selected as the default level for running BioISO after analysing all dem_{wss}. Using this depth value, BioISO can guide the search to identify errors in a given metabolic network, without evaluating only the direct precursors and successors, such as *biomassPrecursorsCheck* [129] and Meneco [187], or the burden of evaluating the whole network, such as fastGapFill [189] and gapFind/gapFill [188].

Furthermore, a significant part of the metabolic network associated with a given objective is analyzed by the tool for a guided search (depth level of 2), while the required computation time is significantly lower.

3.5.3 Exhaustive-search versus guided-search

The exhaustive-search versus guided-search assessment was designed to compare the results of two guided-search tools, namely BioISO (guided search – depth level of 2) and Meneco [187], with fastGapFill [189] exhaustive-search application. The iJO1366 and iDS372 models obtained for the growth maximization analysis were used in this assessment. The workflow and methodology used to compare exhaustive-searches against guided-searches are described above together with supplementary materials I.5 and I.6. Figure 17 exhibits a summary of dead-end metabolites and blocked reactions ratios calculated for each tool.

According to figure 17 and supplementary material I.6, Meneco performed the poorest in identifying metabolic gaps. Besides the poor performance in assessing dead-end metabolites, it does not provide insights into blocked reactions.

The number of covered metabolites when using Meneco is the same as the number of metabolites selected for target metabolites, as this tool only evaluates target metabolites. No information is provided about other metabolites in the metabolic network.

Hence, although Meneco obtained the lowest dem_{wss} ratios in both models (figure 17), the tool suggests the absence of biosynthetic pathways to synthesize all metabolites in the covered search space, thus obtaining the highest dem_{oos} ratios in both models (figure 17). In short, most metabolites analyzed by this tool were evaluated as dead-end metabolites in the incomplete models.

fastGapFill provides, on the other hand, a level of insight much larger than Meneco, analyzing all reactions and metabolites in all models (figure 17 and supplementary material I.6). However, the exhaustive-search tool is associated with two significant drawbacks. Firstly, fastGapFill is the slowest tool (supplementary material I.6). Secondly, this gap-filling tool highlights numerous blocked reactions and dead-end metabolites, according to figure 17, which might hinder a fast and precise identification of a *de facto* error in the network, such as the ones introduced in this validation procedure.

For example, fastGapFill has attained higher dem_{wss} ratios (figure 17) for the model of the less described and smaller genome's organism (*Streptococcus pneumoniae*), which has probably evolved through a combination of extensive loss-of-function events during the co-evolution with well-defined and constant ecological niches [8], [194], [201].

Although Meneco has obtained smaller dem_{wss} scores than BioISO, the level of insight provided by the gap-filling tool for both metabolic networks are significantly lower than the detail provided by our tool. When comparing the dem_{oos} ratios, it is clear the lack of insight provided by Meneco, as most of the metabolites covered by Meneco are highlighted as dead-end metabolites. On the other hand, BioISO can be more effective and precise by suggesting fewer dead-ends out of the examined metabolites pool (figure 17 and supplementary material I.6).

According to figure 17, BioISO attained lower br_{wss} and dem_{wss} ratios than fastGapFill in both models. Hence, such smaller br_{wss} and dem_{wss} ratios suggest that BioISO is more capable of reducing the whole-space search to fewer dead-end metabolites than the gap-filling tool.

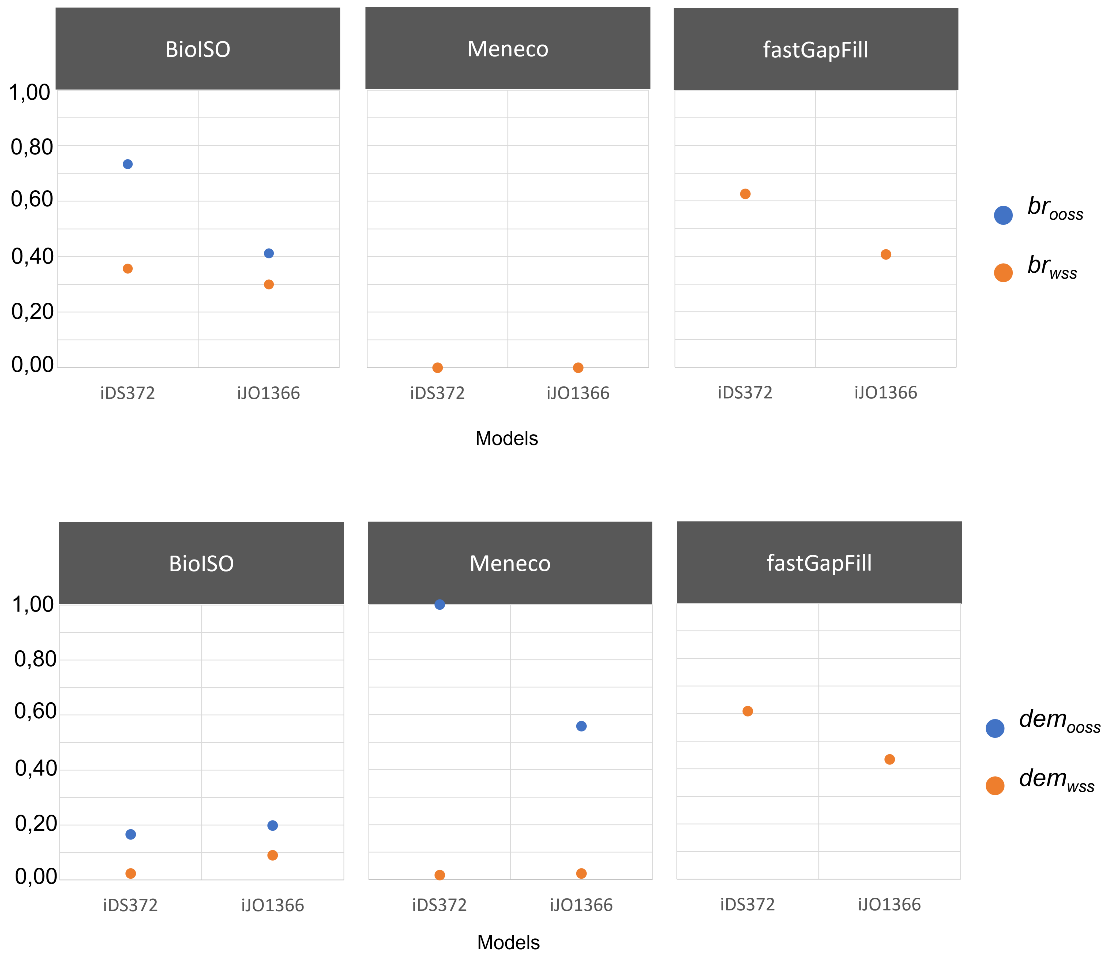


Figure 17: Assessment of the relevance of guided (BioISO and Meneco) versus exhaustive searches (fastGapFill) for gap-finding. BioISO, Meneco, and fastGapFill were used to highlight gaps in two state-of-the-art models (iDS372 and iJO1366), with added gaps. The ratios of dead-end metabolites (dem_{ooss} and dem_{wss}) and blocked reactions (br_{ooss} and br_{wss}) for both objective-oriented search (*ooss*) and whole search (*wss*) spaces were then calculated for each tool.

When debugging and validating the model for specific objective functions, such as growth maximization, BioISO seems better suited for reducing the search space for errors and gaps in metabolic networks than the other tools analyzed in this assessment. This advantage allows spending less time debugging unrealistic errors or gaps. Furthermore, as BioISO reduces the search space for errors, it also favors parsimonious alterations to the draft metabolic network. As a result, BioISO can be of paramount importance for the high-quality bottom-up reconstruction of GEM models during the manual curation stage. However, it should be noticed that Meneco and fastGapFill have been designed to be essentially gap-fill tools. Thus, these tools use different approaches than BioISO to find errors.

3.5.4 BioMeneco - embedding BioISO in Meneco

BioMeneco, BioISO's integration with Meneco [187], was developed to determine whether the former can improve the latter results by narrowing the search space for the gap-filling task. For that, reactions *R04568_C3_cytop* and *SO4tex* were removed from the iDS372 and iJO1366 models, respectively. Meneco was then used to generate potential solutions for restoring models' prediction of a growth phenotype based on BioISO suggestions for the set of targets (primary input for Meneco).

All metabolites identified as not being produced or consumed by BioISO in the iDS372 model are reported in table 2. These metabolites have been selected for the set of target metabolites after a brief analysis of the BioISO's output.

In the iDS372 model, BioISO indicated that reaction *R04568_C3_cytop* might be associated with the synthesis of a precursor of the lipidic and lipoteichoic acid pathways. Most dead-end metabolites identified by BioISO were associated with metabolites *C01356_cytop* and *C06042_cytop*, which are biomass precursors representing the lipid and lipoteichoic acid cellular biomass fractions, respectively. These suggestions are in agreement with the metabolites being synthesized by the reaction removed from the model. Reaction *R04568_C3_cytop* is associated with the synthesis of *trans-Tetradec-2-enoyl-(acp)* (*C05760_cytop*), which in turn is a precursor of the compound *Tetradecanoyl-(acp) C05761_cytop*) involved in the fatty acid biosynthesis pathway.

Table 2: Dead-end metabolites identified by BioISO in the iDS372 model. Metabolites identified by BioISO as not being produced or consumed (dead-end metabolites) by the iDS372 model missing R04568_C3_cytop reaction.

Dead-End metabolite	**Connected dead-end metabolites**
C01356_cytop (Lipid)	C05980_cytop C06040_cytop C04046_cytop C00344_cytop
C06042_cytop (Lipoteichoic acid)	C00116_cytop C00162_cytop

Besides the dead-end metabolites shown in table 2, BioISO suggested an unsuccessful evaluation of all remaining biomass precursors and successors. Nevertheless, only the lipid and lipoteichoic acid compounds were identified as dead-end metabolites. The remaining biomass precursors and successors refer to the special cases reported in the tutorial at bioiso.bio.di.uminho.pt/tutorial. Briefly, these metabolites were unsuccessfully evaluated due to a missing or impaired reaction downstream, namely the biomass reaction.

The corresponding precursors and successors of all neighbor metabolites were classified as non-dead-end metabolites, except for several precursors and successors of the lipid and lipoteichoic acid compounds.

All metabolites identified as not being produced or consumed by BioISO in the incomplete iJO1366 model, reported in table 3, were selected for the set of target metabolites.

BioISO has indicated that the missing *SO4tex* reaction might be associated with synthesizing a precursor for sulfur metabolism. Most dead-end metabolites identified by BioISO were linked to the iron-sulfur clusters, biotin, bis-molybdopterin guanine dinucleotide, and sulfate biomass precursors, which are all associated with the sulfur requirements of *E. coli*. These results are in line with the transport of sulfate to the periplasm by the removed reaction. The *SO4tex* reaction is responsible for transporting sulfate from the extracellular medium to the periplasm, which is then transported to the cytoplasm.

BioISO suggested more dead-end metabolites than the precursors described in table 3, absent from the set of target metabolites for the iJO1366 model. BioISO negatively evaluated the biomass precursors *mobd_c*, *sheme_c*, *cl_c*, and *2ohph_c*. Nevertheless, these metabolites have been ignored as dead-end metabolites, as they refer to the special cases reported in the tutorial at bioiso.bio.di.uminho.pt/tutorial. The precursors of these metabolites are being produced, and the successors are consumed, except for the biomass. Thus, BioISO highlighted these metabolites because the biomass reaction is, actually, the only consumption site available. These metabolites are an example of unsuccessful evaluations that should be easily detected in the user-friendly output returned by the web server and merlin. Moreover, the automatic tools would deal with such cases as regular gaps and incorrectly resolve them.

Table 3: Dead-end metabolites identified by BioISO in the iJO1366 model. Metabolites identified by BioISO as not being produced or consumed (dead-end metabolites) by the iJO1366 model missing SO4tex reaction.

Dead-End metabolite	**Connected dead-end metabolites**
2fe2s_c ([2Fe-2S] iron-sulphur cluster)	4fe4s_c lipopb_c iscu_DASH_2fe2s_c iscu_c sufbcd_DASH_2fe2s_c sufbcd_c
4fe4s_c ([4Fe-4S] iron-sulphur cluster)	iscu_DASH_4fe4s_c sufbcd_DASH_4fe4s_c 3fe4s_c
bmocogdp_c (bis-molybdopterin guanine dinucleotide)	bmoco1gdp_c
btn_c (Biotin)	btn_p btnso_c 2fe1s_c
so4_c (Sulphate)	so4_p

Metabolites identified by BioISO as not being produced or consumed (dead-end metabolites) by the iJO1366 model missing SO4tex reaction.

The gap-filling solutions suggested by Meneco for the incomplete iDS372 model are satisfactory. The proposed solutions could restore flux through the biomass reaction and thus through all sets of targets initially proposed. Meneco suggests adding reaction *R04568* (which was previously removed for this test) to restore the metabolic model.

Nevertheless, other solutions may add artifacts in the iDS372 model. Reactions *R11633*, *R09085*, *R11636*, *R11634*, *R11671* and *R00183* are equally recommended to restore flux throughout the set of targets. However, these reactions are not involved in synthesizing or consuming missing biomass precursors or successors. Most reactions are involved in the synthesis of biomass precursors not affected by the removed reaction, such as the *R11636* (*dCTP* synthesis), *R09085* (carbon metabolism), *R11634* (*dATP*synthesis), and *R11633* (*dGTP* synthesis). Other reactions are involved in synthesizing metabolites not required for *S. pneumoniae's* growth. Only reactions suggested in the pool named *One minimal completion* were considered. Nevertheless, Meneco provides a complete enumeration of all combinations of minimal completions.

BioMeneco recommended, on the other hand, a reduced pool of gap-filling solutions. In this case, BioMeneco suggested reaction *R04568*, but now only three reactions (*R11671*, *R00182*, and *R09085*) have been equally proposed. As neither RNA nor DNA were included in the set of target metabolites, all reactions previously suggested to restore the synthesis of purines and pyrimidines have been discarded.

Meneco restored the test iJO1366 model for six biomass precursors while indicating 35 *unreconstructable targets*. Meneco identified the *so4_c* metabolite, one of the biomass precursors affected by the removal of the *SO4tex* reaction, as *reconstructable*. Nevertheless, the removed reaction did not affect the remaining metabolites for which Meneco could restore flux.

More importantly, Meneco's output does not comprise the *SO4tex* reaction in the set of gap-filling solutions to restore flux through all biomass precursors. More surprisingly, the *SO4tex* reaction was not included in any combination of minimal completions obtained through the complete enumeration of solutions. Furthermore, other potential solutions can lead to the introduction of artifacts in the iJO1366 model. For example, all reactions included in the *One minimal completion* set of solutions are transport reactions for biomass precursors not affected when reducing the iJO1366 model.

Interestingly, only the reaction *SO4tex* has been suggested by BioMeneco to restore flux through all missing biomass precursors and successors in the test iJO1366 model. Moreover, as none of the other biomass precursors was included in the set of target metabolites, all reactions involved in the transport of co-factors, ions, and amino acids were discarded from the *One minimal completion* pool.

Therefore, BioISO can be used to decrease large search spaces associated with model debugging procedures. Besides proposing a user-friendly application to guide the search for dead-end metabolites, we have showcased that BioISO can also facilitate high-quality bottom-up reconstructions by adjusting the guided-search gap-filling tool Meneco. For that, we suggest BioMeneco as an iterative process comprising two separate tasks:

- running BioISO to identify the set of metabolites not being produced or consumed (dead-end metabolites).
- running Meneco using the set of metabolites highlighted earlier as target metabolites to obtain parsimonious solutions to complete draft metabolic networks.

4 Database of prokaryotic transcriptional regulatory networks

The work presented in this chapter corresponds to the following publications:

- *Lima, D., Cruz, F., Rocha, M., Dias, O. (2021). Reconciliation of Regulatory Data: The Regulatory Networks of* Escherichia coli *and* Bacillus subtilis. *In: Panuccio, G., Rocha, M., Fdez-Riverola, F., Mohamad, M., Casado-Vara, R. (eds) Practical Applications of Computational Biology & Bioinformatics, 14th International Conference (PACBB 2020). PACBB 2020. Advances in Intelligent Systems and Computing, vol 1240. Springer, Cham.*

- *Cruz, F., Lima, D., Rocha, M., Dias, O. ProTReND: a database of prokaryotic genome-scale TRNs. In preparation.*

4.1 Introduction

The number of databases storing biological data has increased significantly with the development of high-throughput techniques to measure the transcriptome of living organisms and the implementation of new algorithms in bioinformatics. For prokaryotes, in particular, there are several databases of regulatory information. Over the last decades, databases of regulatory information [67], [69], [71], [73], [74], [76] have become significantly more extensive, containing information on promoters, TFs, genes, TFBS, operons, regulatory interactions, effectors, and gene expression data.

Although the proliferation of resources of regulatory data can contribute significantly to understanding gene regulation in prokaryotes, it can also lead to undesirable side effects in the regulatory data ecosystem. This proliferation can be considered a significant drawback in biological data management [202]. In general, there are a series of reasons for the expansion of resources of regulatory data:

- The access to most regulatory data is hampered due to the absence of Application Programming Interface (API)s;
- The resource is closed to external researchers, preventing community contributions to curate and improve regulatory data;
- The database is strictly focused on a specific topic of research (e.g., an organism or type of regulatory data), leading to data duplication in other resources;
- The resource maintenance is inconsistent throughout time, in that one experiences considerable periods of downtime and fungible records in new releases.

Also, as a consequence of this rich ecosystem, one can easily find incoherent information, dead records, and circular references among these databases of regulatory information. Besides, maintaining this large ecosystem requires time-consuming human effort to keep all data clean and updated. Hence, data management systems are now highly relevant to store a large volume of biological data, avoiding its dispersion [203].

With this in mind, we have developed a framework capable of assembling an integrated database of TRNs. ProTReND comprises several tools to extract and process regulatory information dispersed into several data sources. The data integration system includes several rules to unify the transformed regulatory data, resolving the same entities to a common reference space. In addition, a dedicated web application has been created for ProTReND, allowing users easy access to the integrated regulatory data. This web application also enables community contributions to curate the available regulatory data or to add new regulatory interactions.

4.2 Data integration system

4.2.1 Data warehousing

The assembly of an integrated data management system is often motivated by the need for centralized access to multiple data sources, easing the analysis of different sources, but sharing identical attributes. Notably, an integrated database often supports new computational tools.

The data integration process requires meticulous planning and schematization to handle several levels of data heterogeneity. Hence, several standardized procedures to ease the data integration of heterogeneous sources are currently available. Data warehousing allows the unification of dispersed data into an integrated structure [204]. This methodology comprehends several steps based on ETL tools [205]. ETL tools must extract data from heterogeneous sources, transform and map the collected data to the correct format and structure, and ultimately load the transformed data into a unified database. Figure 18 describes a simple pipeline to populate a data warehouse using ETL tools.

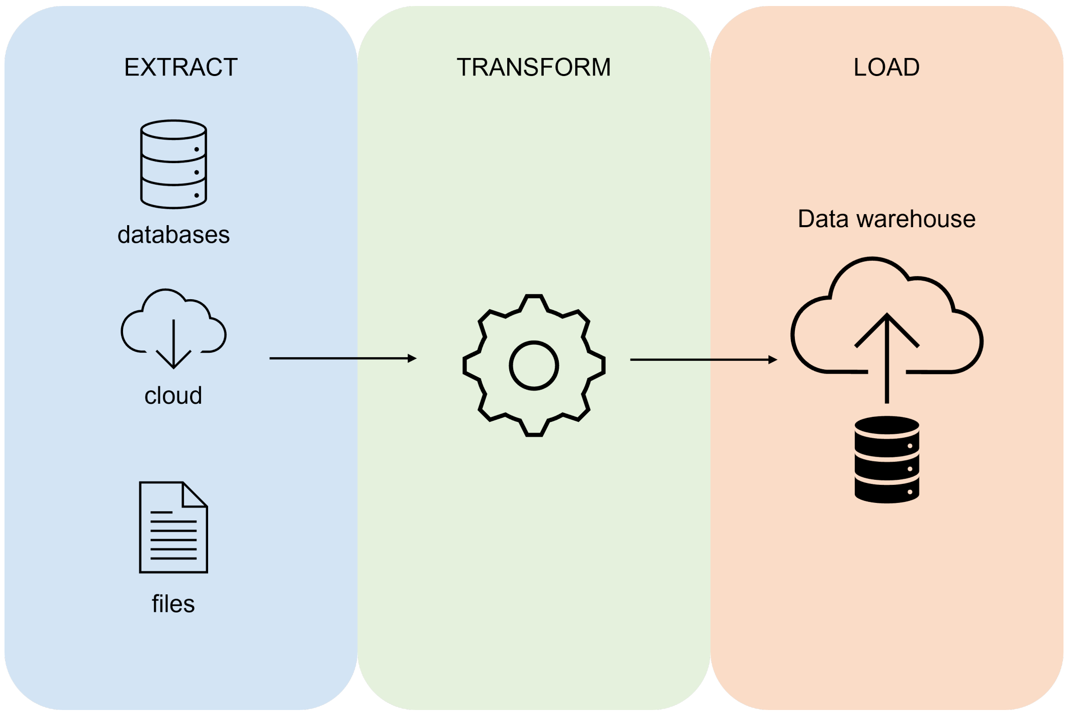


Figure 18: An ETL-based pipeline to populate a data warehouse. An ETL tool extracts, transforms, and loads heterogeneous data from different sources into a data warehouse.

There are several advantages to using ETL tools to assemble an integrated data management system. For instance, ETL tools implement a standardized protocol for data maintenance that allows automated periodic updates. In addition, other data sources can be integrated without breaking the existing system. Hence, the ETL pipeline provides a generic architecture for data warehousing. However, one must implement new tools to extract and transform a new data source. Furthermore, the integration rules to resolve data heterogeneity must still rely on domain-specific knowledge.

4.2.2 Overview of the data integration system

The data integration system proposed in this work comprises an ETL tool to populate a data warehouse with relevant regulatory information. The database of integrated prokaryotic TRNs consists of a graph store model using Neo4j Database Management System (DBMS). The ProTReND repository comprises the integrated database and all tools implemented in the data integration system. These tools are available at the https://github.com/cruz-f/protrend-database GitHub repository.

The ProTReND repository has the following sub-systems:

- Data extraction sub-system;
- Data transformation sub-system;
- Knowledge expansion sub-system;
- Data integration sub-system;
- Data loading sub-system;
- Data lake sub-system;
- CDS sub-system.

In addition to the data integration system tools, the ProTReND repository also comprehends a web application with access to the CDS. Furthermore, we have added a contributing system to this web application allowing users to submit new records or update existing ones.

Figure 19 describes the architecture of the data integration system. In the first phase, data is extracted from relevant resources of regulatory data and saved in the data lake sub-system. Then, the transformation sub-system processes the collected data. Meanwhile, this sub-system also identifies relevant biological entities (e.g. organisms, regulators, genes, etc) and related properties to be annotated in the knowledge expansion sub-system. Once the enrichment is complete, the integration sub-system applies a series of integration rules. In addition, this sub-system also tries to resolve new or existing objects in the CDS. In the meantime, extracted, transformed, and integrated objects are stored in the data lake sub-system. Finally, the data loading sub-system is responsible for creating or updating records in the CDS according to the results obtained in the integration sub-system.

Regarding the architecture of the ETL tools, the ProTReND framework comprises several Python modules implementing the data integration system defined in figure 19. The data extraction module (A) contains web-scraping tools to collect and store regulatory data in the data lake (B), a file storage system. Data processing is available in the data transformation module (C). This module contains methods and algorithms designed to transform and clean objects found in a specific resource. The transformation tool exchanges objects with the knowledge expansion sub-system (D). This system comprises *Bioapis*, *Annotation*, *BindingSiteAlignment*, and *Motif* modules containing several tools to enrich and annotate regulatory objects using biochemical databases and bioinformatics algorithms. The transform tool then stores the results in the data lake as JavaScript Object Notation (JSON) files. The data integration sub-system (E) applies a set of rules to the objects stored in the data lake. This tool determines which objects should be created or updated on the CDS. ProTReND identifiers are assigned to new objects by the data integration tool. The data loading sub-system (F) uses neomodel Object Graph Mapping (OGM) to load objects and relationships into the CDS sub-system (G). Finally, the CDS sub-system contains the *Model* tool, which implements the graph store data model.

The *Pipeline* tool is responsible for running the ETL tool in the correct order. In addition, the data integration system comprises other utilities, such as the *Logging* and *Report* tools.

In parallel, the web application (H) provides user-friendly tools to access and visualize the regulatory data in the CDS. This tool is based on the Django, Django REST, and My Structured Query Language (MySQL) frameworks.

The ProTReND repository consists of autonomous applications that have been containerized using Docker. In detail, the CDS sub-system is available as an independent app running in a Docker container. The remaining ETL tools are also available in another Docker container running in parallel.

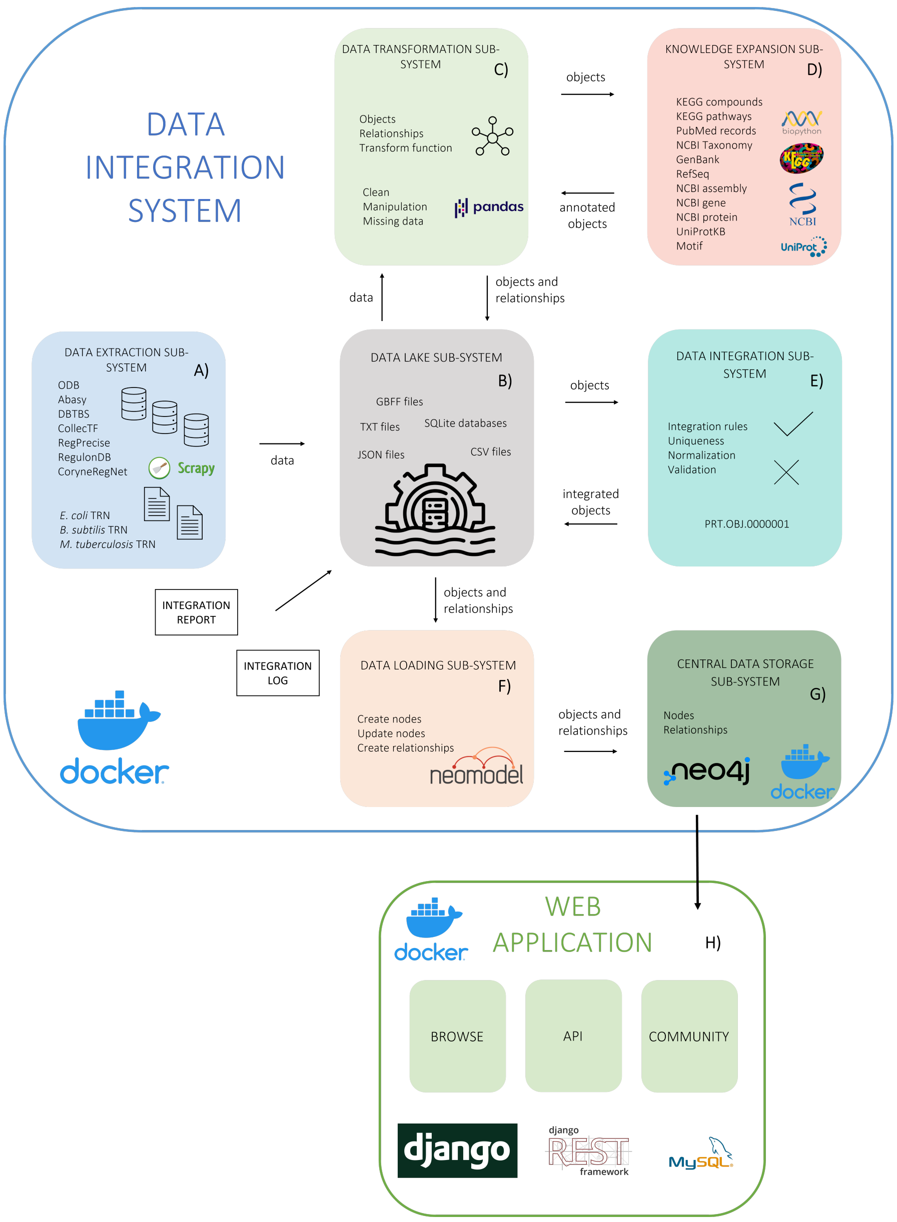


Figure 19: Architecture of the data integration system implemented in the ProTReND repository. The extraction sub-system is responsible for extracting relevant data sources using scraping techniques or manual download. The extracted data is automatically saved in the data lake sub-system as JSON files, among others. Regulatory data is transformed and processed in the transformation sub-system using pandas Python package. Transformed objects are annotated and enriched next in the knowledge expansion sub-system using the Biopython Python package and biochemical databases. The integration sub-system is responsible for applying a series of rules to objects that ensure uniqueness and normalization. Finally, the loading sub-system uploads integrated objects into the CDS using neomodel, a Neo4j OGM. The web application contains several tools for data access and visualization using the CDS. The data integration system, CDS, and web application are autonomous systems containerized using Docker.

4.2.3 Central data storage sub-system

The CDS sub-system contains the ProTReND graph database, which stores key biological entities associated with gene regulation in prokaryotic organisms. A DBMS consists of an interface that allows storing, accessing, organizing, and querying a database. For that, DBMS support standard APIs performing Create, Read, Update and Delete (CRUD) operations to the data stored in the database. In a graph database, data is represented through graphs using nodes and edges. Both nodes and edges can represent entities, as these can hold attributes of a specific data type. Nevertheless, nodes usually represent entities with as many properties as required, while edges represent the relationships between entities and can also hold properties. Neo4j is one of the most common DBMSs using the graph store model [206]. Cypher is the standard API for performing CRUD operations in Neo4j. Graph databases can be used to support data warehousing, as these DBMSs can easily be extended to new domains storing other entities with minor effort.

The CDS follows several generic rules adapted from graph theory that define the graph structure in the CDS. The following symbols will be used in these generic rules.

- Parenthesis, $()$, are used to define a tuple;
- Braces, , are used to define a dictionary of attributes;
- Double quotes, ", are used to define the real values of an attribute;
- $\equiv$ is used to define an equivalent object;
- n is used to define namespace;
- $attr$ is used to define attribute;
- id is used to define unique identifier attribute;

Following object definition (4.2.1), an object is an instance of any entity, such as an organism, regulator, or gene. The object has a unique identifier assigned in the data warehouse methodology and must be associated with a namespace according to the entity. Finally, an object can have many attributes or properties that can hold data of many types.

Theorem 4.2.1. *Object definition*

An object is a tuple $o = (id, n, attr)$, such that,

- *id stands for the unique identifier*
- *n stands for the object namespace*
- *$attr$ stands for the set of attributes/properties associated with the object*

Following definition 4.2.2, a relationship is a directed edge between two objects, which must be distinct from each other. Furthermore, a relationship is associated with a namespace according to the type of interaction between the two objects (e.g., *HAS*). Finally, a relationship can have many attributes or properties that can hold data of many types. Note that two objects can have multiple relationships in the same namespace.

Theorem 4.2.2. ***Relationship definition***

A relationship is a tuple $e = (o1, o2, n, attr)$, such that,

- *e is a directed edge between two objects, namely $o1$ and $o2$*
- *$o1$ and $o2$ must be distinct so $o1 \neq o2$*
- *n stands for the relationship namespace*
- *$attr$ stands for the set of attributes/properties associated with the relationship*

Following definition 4.2.3, the CDS comprises a single universal graph that includes all objects and relationships. In the CDS, two objects are differen only if their identifiers and namespaces are different (definition 4.2.4).

Theorem 4.2.3. ***Central Data Storage definition***

The Central Data Storage is a tuple $G = (V, E)$, such that,

- *G is a universal graph*
- *V is the set of all objects*
- *E is the set of all edges*

Theorem 4.2.4. ***Different objects definition***

Let $o_a = (id_a, n_a, attr_a) \in V$ and $o_b = (id_b, n_b, attr_b) \in V$, then $o_a \neq o_b$ if $id_a \neq id_b \land n_a \neq n_b$

ProTReND's database comprehends 13 entities. Each entity represents a namespace in the CDS universal graph, and all objects available in the database are associated with a single namespace as defined above in section 4.2.1. In detail, the following entities are available in ProTReND's database:

- Effector - An effector object can represent a metabolic compound, Transfer RNA (tRNA), or protein sub-unit that belongs to a regulatory protein complex. Effectors can bind non-covalently to an allosteric regulatory site influencing interactions between regulators and genes. Alternatively, effectors can also represent biological processes or environmental conditions that alter the regulatory effect of regulatory interactions.

- Evidence - An evidence object represents the experimental or computational technique used to infer a given regulatory interaction. There can be specific evidence objects, such as directed mutagenesis or gene expression profiling, and generic evidence objects, such as predicted or experimental.
- Gene - A gene object represents a DNA segment transcribed and translated into a protein. In the CDS, most genes are target genes in that expression levels are determined by regulators.
- Motif - A motif object represents a series of aligned fixed-length TFBS associated with a single regulator.
- Operon - An operon object represents a set of genes regularly transcribed as a single functional mRNA molecule encoding the genetic information for synthesizing one or more proteins.
- Organism - An organism object represents an organic living system. Note that the ProTReND repository only contains prokaryotic organisms.
- Pathway - A pathway object represents a biochemical pathway associated with a given regulator or gene.
- Publication - A publication object represents a journal or book citation referenced with a PubMed Identifier (PMID).
- Regulator - A regulator object represents a regulatory protein of one of the following types: TF; transcription attenuator; transcription terminator; sigma factor; small RNA (sRNA). Regulators are responsible for controlling gene expression upon binding specific sites in the genome.
- Regulatory Family - A regulatory family object represents a family of regulatory proteins that can share similar regulatory processes or sequence similarities.
- Regulatory Interaction - A regulatory interaction object represents the control of a gene's expression mediated by a regulator upon binding a given TFBS in association with a given effector. Regulatory interactions are associated with the regulatory effect, namely the activation or repression of the gene expression.
- Source - A source object represents a resource of regulatory data extracted in the ProTReND repository.
- TFBS - A TFBS object represents a physical DNA sequence also referred to as a binding site. Regulators bind to TFBS to control gene expression.

Figure 20 portrays the main entities and relationships in the CDS, while supplementary material I.8 contains the schema for the namespaces, attributes, and relationships. *Organism*, *Regulator*, *Gene*, and *TFBS* are highly interconnected in the CDS. Likewise, *Regulatory Interaction* comprises the central relationships in the CDS, also having connections with *Publication*, *Evidence*, and *Effector*. The CDS also contains *Regulator* - *Regulatory Family* and *Gene* - *Operon* relationships. *Motif* entity is mainly associated with *TFBS* and *Regulator*. Note that ProTReND's database contains more relationships than the ones defined in Figure 20. For instance, *Source* entity contains connections with all objects integrated in the CDS. Hence, we have selected the main entities and relationships in the CDS to ease the visualization of the database connectivity.

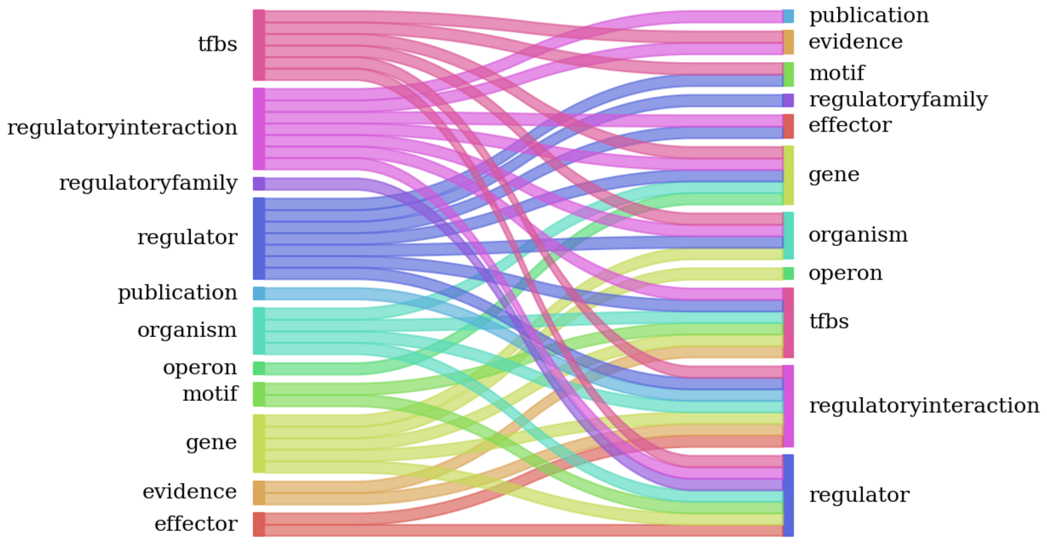


Figure 20: Main namespaces and relationships in ProTReND's CDS. Namespaces are depicted on the left and right axis representing the source and target nodes, respectively. The arcs between namespaces stand for relationships in the CDS. All relationships are bi-directional in the CDS. For instance, the *Regulatory Family* entity is associated with the *Regulator* entity as the *Regulator* entity is related to the *Regulatory Family*.

ProTReND database runs over the Neo4j DBMS. In detail, neomodel, an OGM for the Neo4j graph database built on top of Python's neo4j driver, was used to implement the graph data model defined above 4.2.3. An OGM is a technique based on the object-oriented programming paradigm. Thus, neomodel allows performing CRUD operations using simple Python objects. Neomodel's simple yet powerful API is used in all ETL tools instead of Neo4j's query language, Cypher.

4.2.4 Data lake sub-system

In the present context, a data lake is a schemaless and unstructured data storage technique to store raw data from several sources. Whereas data warehouses are analytical-oriented data storage systems that are usually optimized for a set of analyses and reports, data lakes can store all types of data without prior design and effort. Structured and unstructured raw data are often held in data lakes as extracted. Then, a particular share of the stored data can be further processed for a specific analysis.

ProTReND's data integration system comprises a data lake sub-system to store extracted data into several raw format files, such as JSON, Comma Separated Values (CSV), XLSX spreadsheet, and Text File (TXT). In addition, the data lake also contains data created by the transform, integration, and load tools. Data available in the knowledge expansion sub-system is also stored in the data lake in the form of SQLite databases and JSON format files.

ProTReND's data lake is organized into a hierarchical file storage system. In detail, the name and version of the data source are used to create the directories hierarchy. For instance, a directory named *regulondb-0.0.2* contains all files obtained from RegulonDB [67] in the second extraction. Many files are stored in several formats within each directory. Besides, these files can have been generated at each stage of the ETL pipeline or data integration system. For instance, the directory *bioapi* cache contains many SQLite databases with cached records of NCBI and UniProtKB used in the ETL tools.

In short, the data lake fundamentally differs from the CDS, which consists of a graph database optimized to analyze and visualize regulatory information. In contrast, the data lake contains all structured and unstructured files to assemble the CDS, which can also support particular queries and analyses.

4.2.5 Data extraction sub-system

The data extraction sub-system contains all tools to collect regulatory data from distinct sources. This system comprehends seven databases of regulatory data and three TRNs available from the literature. Table 4 contains the resources of regulatory information selected at this stage together with the type of extraction performed and entities referenced.

Table 4: Extracted resources of regulatory data. Summary of the resources extracted in ProTReND. The resources are identified by name, reference, type, taxonomic coverage, and data type.

Resource	**Type**	**Organism**	**Extraction**	**Data**
Abasy [76]	database	*Bacteria*	manual	TF and gene
CollecTF [73]	database	*Prokaryota*	scraping	regulon, gene, TFBS, evidence and publication
CoryneRegNet [71]	database	*Bacteria*	manual	TF, sRNA, sigma factor, gene, binding site, evidence and publication
DBTBS [69]	database	*B. subtilis*	scrapping	TF, sigma factor, gene, TFBS and family
ODB [78]	database	*Prokaryota*	manual	operon and gene
RegPrecise [74]	database	*Prokaryota*	scrapping	regulon, sRNA, gene, TFBS, effector, pathway, RNA family, TF family and publication
RegulonDB [67]	database	*E. coli*	manual	TF, sigma factor, sRNA, gene, binding site, effector, evidence and publication
Fang *et al* [207]	literature	*E. coli*	manual	TF and gene
Faria *et al* [106]	literature	*B. subtilis*	manual	TF, sigma factor, sRNA, gene and metabolite
Turkarslan *et al* [47]	literature	*M. tuberculosis*	manual	TF and gene

Third-party resources provide data in different formats. Furthermore, external resources may have different notations to the same entity or property. Hence, the data extraction sub-system comprehends an extraction tool for each data source. Web scraping is used to collect regulatory data from CollecTF [73], RegPrecise [74], and DBTBS [69], as these resources do not have a RESTfull API or user-friendly tool to download all data. The copyright licences and web robots file (*robots.txt*) have been consulted before the web scraping process to verify copyright infringements.

On the other hand, the data sources Abasy [76], CoryneRegNet [71], Operon DataBase (ODB) [78], and RegulonDB [67] have been obtained manually from the resources' websites. The copyright licenses have been consulted before downloading the regulatory data to verify copyright infringements. In this case, the raw format files downloaded from these resources were added directly to the data lake. Finally, the published TRNs of *E. coli*, *B. subtilis*, and *M. tuberculosis* were available in the supplementary materials of each work. The following paragraphs contain a detailed description of the entities and properties extracted for each data source.

Abasy - The Abasy database [76] allows downloading state-of-the-art TRNs in the JSON format file. We have downloaded nine TRNs containing many regulatory interactions between regulators and target genes identified by the gene locus tag or name. In addition to the TRN JSON file, the Abasy database also provides a Tabular Separated Values (TSV) format file containing the following information for all regulators and target genes: gene locus tag; gene name; NCBI gene identifier; UniProtKB accession; synonyms; product. Supplementary material I.9 fully describes the regulatory information collected from Abasy.

CollecTF - The CollecTF database [73] contains regulatory interactions and binding site motifs for several bacteria. Although this database allows downloading regulatory interactions by taxonomy, we have implemented a web scraping tool to collect the data automatically. This web scraping tool is based on Scrapy, an open-source web-crawling tool in Python. The web scraping tool starts by listing all bacteria documented in CollecTF using a custom parser to process the downloaded HyperText Markup Language (HTML) content. Then, a new web crawling process retrieves regulators, operons, genes, TFBSs, and evidence information for each organism's web page. Finally, the web scraping tool automatically yields a JSON report of all items gathered in CollecTF. Supplementary material I.9 contains the web-crawled content for organisms, regulators, operons, genes, TFBS, and experimental evidence.

CoryneRegNet - The CoryneRegNet database [71] is a collection of experimentally validated and computationally predicted TRNs for *Corynebacteria*. In addition, this database comprises state-of-the-art regulatory networks for *E. coli*, *B. subtilis*, and *M. tuberculosis*. We have retrieved experimentally validated TRNs of *Corynebacterium glutamicum*, *E. coli*, *B. subtilis*, and *M. tuberculosis* from the database's website. The TRNs are in CSV format files, including many regulatory interactions between regulators and target genes. Locus tags, regulatory effect, operon, TFBS, evidence, and PMID are associated with each regulatory interaction. Supplementary material I.9 fully describes the regulatory information available in the CoryneRegNet database.

DBTBS - The DBTBS database [69] contains experimentally validated regulatory interactions and TFBS for *B. subtilis*. However, to our knowledge, DBTBS does not have an API to obtain the regulatory data. Thus, we have implemented another web scraping tool to collect the bacterium regulatory information automatically. This tool parses all regulators associated with the bacterium and retrieves gene and TFBS information from each regulator web page. Finally, all items gathered in the DBTBS database are stored into a JSON report file. Supplementary material I.9 contains the web-crawled content for regulators, genes, and TFBS.

ODB - The ODB [78] is a compendium of known and predicted operons in microbial genomes. The data manually collected from ODB contains a list of operons, including the following information: ODB identifier; NCBI taxonomy identifier; operon name; locus tags of the operon genes; product; PMIDs. The supplementary material I.9 contains all information regarding the operons obtained at the ODB.

RegPrecise - RegPrecise database [74] is a comprehensive collection of regulons found in prokaryotic organisms. This database is the largest resource of regulatory data addressed in this work. Although RegPrecise allows downloading regulators, genes, and TFBS, there is no programmatic access via an API to collect this content automatically. Hence, we have implemented another web scraping tool to automatically extract the regulatory information available in RegPrecise. This tool first parses the taxonomy collection web page containing a list of all microbial genomes. Then, the web crawling procedure downloads the regulons associated with each organism. In RegPrecise, the regulon web page contains relevant information about the regulator, regulated operons, and TFBS. In addition, the web scraping tool extracts information regarding each regulon's regulatory family, effector, and metabolic pathway. Supplementary material I.9 fully describes the regulatory information available in the RegPrecise database.

RegulonDB - The RegulonDB database [67] is the most comprehensive resource of regulatory data for *E. coli*. This database comprises a state-of-the-art TRN at the genome-scale for *E. coli* K12. In this work, we have downloaded TSV format files containing version 10.6 of RegulonDB. The TSV format files comprise information for the following entities: effector; evidence; gene; publication; TF; sRNA; sigma factor; regulatory family; binding site. Additionally, *E. coli* TRN is divided into four different files named *genetic network*, *regulatory interaction*, *srna interaction*, and *tf gene interaction*. Each file contains many regulatory interactions between regulators and target genes. Supplementary material I.9 contains examples of the regulatory content in the RegulonDB database.

Literature - The TRNs of *E. coli*, *B. subtilis*, and *M. tuberculosis* were obtained from the works by Fang *et al* [207], Faria *et al* [106], and Turkarslan *et al* [47] in Excel Microsoft Office Open XML Format Spreadsheet File (XLSX) format, respectively. These networks contain numerous regulatory interactions, including the following properties: regulator locus tag; gene locus tag; regulatory effect; metabolite (whenever available);

and regulatory mechanism (e.g., TF and sigma factor). Supplementary material I.9 fully describes the regulatory data obtained from the published literature.

4.2.6 Data transformation sub-system

External data sources may have different notations and namespaces for the same entities and properties. Therefore, the transformation sub-system should resolve all third-party terms for a single entity or property in the CDS.

In detail, the transform tool applies a transformation function to the extracted data (definition 4.2.5). This function is responsible for mapping raw unstructured objects and relevant properties to graph objects (check definition 4.2.1). Likewise, the transform function is also responsible for resolving the namespace of these raw unstructured objects.

Theorem 4.2.5. ***Object transformation***

$f : o_r \longrightarrow o_a$, *such that* o_r *is a raw and unstructured object in the extracted data and* $o_a = (id_a, n_a, attr_a)$ *is a graph object to be integrated into the CDS*

The transform tool identifies relationships in the extracted data, creating graph edges between transformed objects (check definition 4.2.2). In addition, this tool creates relationships between data sources and transformed objects, tracking the source of the regulatory data in the CDS. Table 5 summarizes the transformed objects by data source, while the supplementary material I.9 provides detailed transformation functions for each external data source.

A particular transform function resolves regulatory interactions extracted from CollecTF and RegPrecise. These databases contain multiple regulatory interactions between regulators and operons. However, ProTReND's database model only supports regulatory interactions between regulators and genes. Therefore, the transform functions of CollecTF and RegPrecise split regulatory interactions by the genes associated with the regulated operon, as gene regulation often occurs at the operon level in prokaryotes [20]. Besides, these resources do not provide details regarding the type of regulatory interaction or intra-specific gene regulation.

The transform tool contains several data processing operations, such as handling missing and removing duplicate data, and manipulating data types. Furthermore, the transform tool includes some standard operations:

- Removing white space from locus tags and names;
- Removing non-alphanumeric characters from locus tags and names;
- Removing non-nucleotide characters from TFBS data;
- Removing poorly annotated objects;

Table 5: Transformation of the regulatory data by source. Summary of the transformations performed in the data transformation sub-system according to the source of regulatory data. A bullet point marks the transformation of raw unstructured objects extracted from a specific resource into graph objects belonging to a given namespace.

Entity	**Abasy** [76]	**CollecTF** [73]	**Coryne-RegNet** [71]	**DBTBS** [69]	**ODB** [78]	**RegPrecise** [74]	**RegulonDB** [67]	**Literature** [47], [106], [207]
Effector						●	●	●
Evidence		●	●	●			●	
Gene	●	●	●	●	●	●	●	●
Operon					●			
Organism	●	●	●	●	●	●	●	●
Pathway						●		
Publication		●	●	●	●	●	●	
Regulator	●	●	●	●		●	●	●
Regulatory Family				●		●	●	
Regulatory Interaction	●	●	●	●		●	●	●
TFBS		●	●	●		●	●	

- Removing incorrect or less relevant properties.

The ProTReND repository comprises a custom transformation tool for each extracted data source. Each component is responsible for applying the correct transformation function and cleaning operations to the raw regulatory data. For that, we have used the pandas Python package, which is a tool for data manipulation. Finally, the transformation tool stores all transformed objects into JSON format files in the data lake sub-system.

In addition, the transform tool includes two custom components named *motif* and *trimmer*. While the former collects unaligned TFBS in the CDS to assemble binding site motifs, the latter is responsible for trimming orphan objects found in the CDS.

4.2.7 Knowledge expansion sub-system

A preliminary analysis of the extracted data revealed that most objects missed relevant properties to perform the data integration routines. The purpose of the knowledge expansion sub-system is to retrieve standardized nomenclature, identifiers, and functional annotations for the main entities. In addition, these tools have contributed to the enrichment and contextualization of the extracted data.

Effector - Most effectors retrieved in the data extraction sub-system consist of metabolic compounds. Thus, effector objects have been completed with metabolic information from the KEGG database [81]. The effector annotation tool uses the KEGG list of chemical compounds retrieved with the database API. The name of each effector was sought in this list of chemical compounds to obtain the set of similar KEGG compounds. The Whoosh Python package, a fast text search engine, was used to find KEGG compounds matching the effectors' names. Ultimately, KEGG compound data was stored in the data lake and loaded into the CDS.

Pathway - Some resources included the biological pathways associated with the regulatory proteins. Thus, pathway objects have been extracted, transformed, and associated with regulator objects. In addition, pathway objects have been completed with additional metabolic information from the KEGG database. The pathway annotation tool retrieves the list of all KEGG metabolic pathways and searches for the most similar ones using the pathway's name. As with the effector annotation tool, we have used the Whoosh search engine to find close KEGG pathways. KEGG pathway data is stored in the data lake and loaded into the CDS.

Publication - Most resources comprised publication sets linked to regulatory interactions or regulators. The publication tool can fetch the citation record from the PubMed database using the PMID and retrieve the following information: Digital Object Identifier (DOI); title; author; year of publication. This tool is based on the Biopython Python package [208] to automatically access the NCBI Entrez API [209] and download the mentioned information. Ultimately, publication data has been stored in the data lake and loaded into the CDS.

Organism - Organisms have different nomenclatures, identifiers, and taxonomy classifications across different data sources. For instance, some databases only contain the species name, while others include an internal identifier but no external references. The NCBI taxonomy database allows retrieving the standardized name and universal NCBI taxonomy identifier for a given organism. Hence, we have developed an organism annotation tool based on the Biopython package and Entrez API to automatically fetch taxonomy records from the NCBI taxonomy database using the species name. The taxonomy records contain the NCBI assembly, GenBank, and RefSeq genome accessions of the organism's representative genome. Furthermore, we have collected NCBI File Transfer Protocol (FTP) addresses associated with the GenBank and RefSeq genome accessions.

The organism data retrieved by the organism annotation tool is added to each organism object and saved in the data lake. More importantly, these annotations are essential to integrate organisms into the CDS. However, some species names returned empty results from the NCBI taxonomy database, as these were obsolete or poorly annotated in the extracted resources. Thus, we have manually collected additional data for these organisms.

GenBank genomes - A database of regulatory data should encompass genomics and proteomics data whenever possible. Integrating nucleotide and amino acid sequences with regulators and genes can boost future analysis and TRN inference tools. More importantly, mapping genes in the organism genome is mandatory to assemble TRNs at the genome scale. Therefore, the knowledge expansion sub-system comprises a database of genome sequences retrieved from the GenBank database. The representative genome sequence of a given organism was obtained using the NCBI FTP address linked to the organism's GenBank accession. The downloaded GenBank files were parsed with the Biopython Python package to retrieve the following gene information: locus tag, name, synonyms, UniProt accession, GenBank accession, nucleotide sequence, gene start, end, and strand. In addition, amino acid sequences were retrieved by translating nucleotide sequences using Biopython. For each organism in the CDS, a JSON file containing the relevant genome sequence data was stored in the data lake and used later in the gene annotation tool.

Gene - Regulators and genes obtained from external resources often miss persistent non-ambiguous identifiers and external references. For instance, the DBTBS database only has gene names for regulators and genes. Although CollecTF contains name and UniProt accession for all regulators, several UniProt accessions are incorrectly assigned to proteins of other organisms. Interestingly, only RegulonDB and Abasy contain proper references between genes locus tags and references to genomics databases like NCBI gene and GenBank. The remaining databases solely contain genes' locus tags and names, which can hinder a standard identification of regulators and gene objects in the CDS.

The standard representation of these objects is highly relevant in the data integration system. Regulators and genes should be combined with functional, genomics, and proteomics information, namely locus tag, name, synonyms, function, description, nucleotide sequence, amino acid sequence, DNA strand, and both start and stop position in the representative genome. In addition, regulators and genes should contain external references, such as NCBI gene identifier, NCBI protein identifier, GenBank accession, RefSeq accession, and UniProt accession. Otherwise, the identification of genes using only one reference can be faulty.

The gene annotation tool can retrieve relevant information from genomics and proteomics databases, improving the standard representation of regulators and genes in the CDS. Gene annotation comprehends the following steps:

1. If the UniProt accession of a given gene or regulator is available, its protein record is obtained from UniProtKB. Otherwise, a combined query is submitted to UniProtKB using the organism's NCBI taxonomy identifier and the gene locus tag or name. In this case, we have selected the first protein record matching the gene locus tag and organism identifier submitted in the query. The UniProtKB protein record allows obtaining the locus tag, name, synonyms, function, description, and amino acid sequence for a given gene or regulator. UniProt RESTful API is used to query and download protein records from the UniProtKB database.

2. If the NCBI protein identifier, GenBank accession or RefSeq accession is available, the corresponding protein record is obtained from NCBI protein database. Otherwise, a combined query is submitted to this database using the organism's NCBI taxonomy identifier and the gene locus tag or name. In this case, we have selected the first record matching the gene locus tag submitted in the query. The NCBI protein record allows obtaining the locus tag, synonyms, amino acid sequence, GenBank, and RefSeq accessions for a given gene or regulator. The Biopython package and Entrez API are used to query and download records from NCBI protein database.

3. If the NCBI gene identifier is available, its gene record is obtained from the NCBI gene database. Otherwise, a combined query is submitted to this database using the organism's NCBI taxonomy identifier and the gene locus tag or name. In this case, we select the first record matching the gene locus tag submitted in the query. The NCBI gene record contains the locus tag, name, synonyms, function, and genomic positions such as strand, start and stop. As with the NCBI protein database, Biopython and Entrez API are used to query and download records from the NCBI gene database.

4. The UniProt accession found for a given gene or regulator is used in UniProt's ID mapping tool. This tool allows mapping UniProt accessions to identifiers of other databases, such as the NCBI gene, NCBI protein, GenBank, and RefSeq. The purpose is to complete the first steps, as missing identifiers and accessions were found.

5. The information retrieved from UniProtKB, NCBI protein and NCBI gene databases is merged into a single annotation for a given gene or regulator. Overlapping or inconsistent information is merged according to the order of the steps. That is, data obtained from UniProtKB database takes precedence over NCBI protein and NCBI gene databases, as UniProt accessions are stable and persistent throughout time [98]. Although we have obtained several outdated or migrated records from the NCBI protein and NCBI gene databases, these records have still been used to perform the gene annotation.

6. Finally, the GenBank genomes database was used to standardize the locus tag, name, nucleotide sequence, amino acid sequence, and gene coordinates. In detail, we have used the gene's locus tag, synonyms, UniProt accession, and GenBank accession obtained so far to find related information in the genomes database. As a result, matching genes were updated with the locus tag, name, nucleotide sequence, amino acid sequence, and gene coordinates of the GenBank genomes database.

Motif - Motif search tools use PWMs or Position-Specific Scoring Matrix (PSSM)s inferred from aligned fixed-length TFBS data to discover novel binding sites in target genomes. However, most databases contain unaligned variable-length binding sites associated with regulators. To ease motif searches using ProTReND's TFBS data, we have included a tool to perform binding site alignment in the knowledge expansion sub-system.

The motif annotation tool is based on the LASAGNA package, an algorithm for TFBS alignment [210]. The set of binding sites associated with a regulator is compiled from the CDS and submitted to LASAGNA. This algorithm performs a length-aware binding site alignment and returns aligned fixed-length sequences. LASAGNA can add gaps to each sequence to perform the alignment successfully. Then, the motif annotation tool can infer the consensus sequence, PWM, PSSM and motif logo using the aligned TFBS. All generated motifs have been stored in the data lake and loaded into the CDS. Motifs stored in the CDS only contain the aligned binding site sequences, as the remaining information can be easily generated from the binding sequences.

The knowledge expansion sub-system comprehends a series of independent modules and components in the ProTReND repository. The *annotation* component is responsible for performing the above annotation routines and merging the collected data into the transformed objects. The component named *bioapis* contains all tools to access the KEGG, NCBI and UniProt databases. This component also implements a caching system to improve the performance of the ETL tool, as fetching information to external resources via API is time-consuming. The cache system uses DiskCache, a disk file-backed cache software built in Python. All queries performed to KEGG, NCBI and UniProt databases are hashed using the query arguments as keys, and stored in SQLite databases. Cache databases are stored in the data lake, allowing their portability and versioning. Finally, *binding site alignment* and *descriptors* components implement TFBS alignment and motif reconstruction strategies.

4.2.8 Data integration sub-system

The same object can have slightly different terms and annotations across multiple data sources. Transformation and knowledge expansion tools help reduce this data heterogeneity. Nevertheless, an integration tool is still mandatory to merge duplicated entries and perform data normalization. The purpose of an integration tool is to apply a set of rules to transformed objects enabling a centralized and unified view of the heterogeneous regulatory data compiled so far in the ETL tool.

Overall, transformation and integration sub-systems comprise the cornerstone of the data integration system and impose data constraints in the CDS. However, the transform tool has not applied any integration rule per se, because the namespace resolution and object annotation do not prevent obtaining duplicated objects. For instance, the data lake contains many duplicated objects extracted from different data sources. The following paragraphs fully describe the integration rules applied to all entities in the CDS based on their primary properties.

Effector, evidence, pathway, and regulatory family - The property *name* is the primary key to formulating a generic integration rule for effectors, evidence, pathways, and regulatory families due to the scarcity of information obtained for these entities. Definition 4.2.6 is used to merge data from different sources regarding objects of the mentioned entities. Therefore, the integration process removed the effector, evidence, pathway, and regulatory family objects missing the name property.

Theorem 4.2.6. ***Definition of equivalent objects by name***

Let $o_a = (id_a, n_a, "name_a", ...) \in V$ *and* $o_b = (id_b, n_b, "name_b", ...) \in V$*, then* $o_a \equiv o_b$ *if* $n_a = n_b \wedge n_a, n_b \in ("Effector", "Evidence", "Pathway", "RegulatoryFamily")$ *and* $"name_a" = "name_b"$*.*

Organism - *Name* and *NCBI taxonomy identifier* properties are the primary keys to setting organisms' integration rule in the CDS. Following definition 4.2.7, organisms are equivalent across data sources if the name and NCBI taxonomy identifier are identical.

Theorem 4.2.7. ***Definition of equivalent organisms***

Let $o_a = (id_a, n_a, "name_a", "ncbiTaxonomy_a", ...) \in V$ *and* $o_b = (id_b, n_b, "name_b", "ncbiTaxonomy_b", ...) \in V$*, then* $o_a \equiv o_b$ *if* $n_a = "Organism" \wedge n_b = "Organism"$, $"name_a" = "name_b"$*, and* $"ncbiTaxonomy_a" = "ncbiTaxonomy_b"$*.*

Regulator and gene - *Locus tag* and *UniProt accession* properties are the primary keys to set the integration rule for regulators and genes. Following definition 4.2.8, regulators and genes are equivalent across data sources if they have the same locus tag and UniProt accession. The locus tag is available across most data sources and standardized in the knowledge expansion sub-system using the GenBank genomes database. Furthermore, the integration tool removes regulators and genes missing the locus tag property. The UniProt accession is also available in most regulators and genes. In addition, the UniProt accession is a persistent identifier in the UniProtKB database.

Theorem 4.2.8. ***Definition of equivalent regulators and genes***

Let $o_a = (id_a, n_a, "locusTag_a", "uniprotAccession_a", ...) \in V$ *and* $o_b = (id_b, n_b, "locusTag_b", "uniprotAccessio$
$\in V$*, then* $o_a \equiv o_b$ *if* $n_a = n_b \wedge n_a, n_b \in ("Regulator", "Gene")$*,* $"locusTag_a" = "locusTag_b"$*, and* $"uniprotAccession_a" = "uniprotAccession_b"$*.*

Regulators and genes have a second data constraint based on the assumption that one regulator or gene can only be associated with a single organism. Hence, regulatory data is normalized at the genome scale, as regulators and genes must be distinct across organisms.

Operon - The *ODB identifier* is available to all operons retrieved from ODB. Hence, the ODB identifier is the primary key for formulating the integration rule of operons (definition 4.2.9). Nevertheless, the integration of operons in ProTReND's repository was limited to operons having genes integrated into the CDS. An operon is integrated into ProTReND's database if one of the operon genes is available in the CDS at the time of the integration. As with regulators and genes, operons must be associated with a single organism.

Theorem 4.2.9. ***Definition of equivalent operons***

Let $o_a = (id_a, n_a, "odb_identifier_a", ...) \in V$ *and* $o_b = (id_b, n_b, "odb_identifier_b", ...) \in V$*, then* $o_a \equiv o_b$ *if* $n_a = "Operon" \wedge n_b = "Operon"$ *and* $"odb\_identifier_a" = "odb\_identifier_b"$*.*

Publication - The *PMID* property is the primary key to setting the integration rule of publications in the CDS (definition 4.2.10). The citation identifier is retrieved from the PubMed database and assigned to all publications in the publication annotation tool. The integration process removes all publication objects missing a PMID.

Theorem 4.2.10. ***Definition of equivalent publications***

Let $o_a = (id_a, n_a, "pmid_a", ...) \in V$ *and* $o_b = (id_b, n_b, "pmid_b", ...) \in V$*, then* $o_a \equiv o_b$ *if* $n_a = "Publication" \wedge n_b = "Publication"$ *and* $"pmid_a" = "pmid_b"$*.*

Regulatory interaction - A star shape schema is often used to model data in a data warehouse [204], [211]. This schema contains a fact table at the center to power data analyses. The fact table should not store data directly, but rather foreign keys of the multiple dimensions in the data warehouse. Dimension tables should be related to the fact table and store the relevant properties of each dimension.

Regulatory interactions are the primary event in ProTReND's repository. Therefore, the regulatory interaction entity in the CDS is ProTReND's facts table, and all objects having the *RegulatoryInteraction* namespace are nodes or rows of this fact table. A regulatory interaction object contains the organism, regulator, gene, TFBS, and effector foreign keys.

Equivalent regulatory interactions have been detected across multiple data sources using a combination of all foreign keys and the regulatory effect. Thus, regulatory interactions are equivalent if the combination of the related organism, regulator, gene, TFBS, effector, and regulatory effect are identical (definition 4.2.11). A composed hashable key has been added to all regulatory interactions to ease their integration. This key is composed of all foreign keys mentioned above plus the regulatory effect of the regulatory interaction.

Figure 21 provides an example of the integration process performed in regulatory interactions. Note that both TFBS and effector foreign keys are not required to create a regulatory interaction object, as most resources of regulatory data miss information for these dimensions. Consequently, regulatory interactions having the same organism, regulator, gene, and regulatory effect might still be different objects in the CDS due to the presence or absence of TFBS and effector foreign keys.

Theorem 4.2.11. ***Definition of equivalent regulatory interactions***

Let $o_a = (id_a, n_a, "organism_a", "regulator_a", "gene_a", "tfbs_a", "effector_a", "regulatory\_effect_a") \in V$ *and* $o_b = (id_b, n_b, "organism_b", "regulator_b", "gene_b", "tfbs_b", "effector_b", "regulatory\_effect_b") \in V$*, then* $o_a \equiv o_b$ *if the following conditions are satisfied:*

- $n_a = "RegulatoryInteraction" \wedge n_b = "RegulatoryInteraction"$
- $"organism_a" = "organism_b"$
- $"regulator_a" = "regulator_b"$
- $"gene_a" = "gene_b"$
- $"tfbs_a" = "tfbs_b"$
- $"effector_a" = "effector_b"$
- $"regulatory_effect_a" = "regulatory_effect_b"$

Source	Organism	Regulator	Gene	TFBS	Effector	Regulatory Effect	**Interaction**
RegPrecise	PRT.ORG.0000001	PRT.REG.0000001	PRT.GEN.0000001	PRT.TBS.0000001	PRT. EFC.0000001	Repression	**PRT.RIN. 0000001**
RegPrecise	PRT.ORG.0000001	PRT.REG.0000002	PRT.GEN.0000002			Activation	**PRT.RIN. 0000002**
RegulonDB	PRT.ORG.0000001	PRT.REG.0000002	PRT.GEN.0000002			Activation	**PRT.RIN. 0000002**

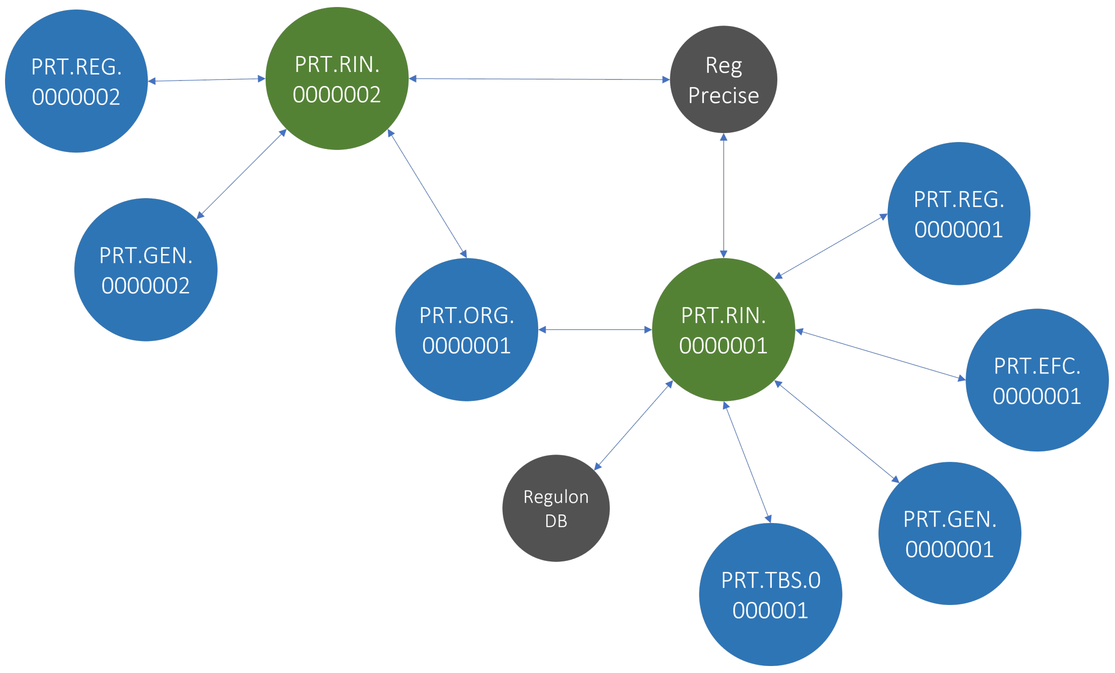


Figure 21: Example of the integration of regulatory interactions in the CDS. The collection of data from different sources can lead to duplicated data. Moreover, regulatory interactions found in external sources often miss normalization. In ProTReND, regulatory interactions are normalized based on the organism, regulator, gene, TFBS, effector and regulatory effect. Therefore, the regulatory interactions fact table can be used to visualize the relationships among the primary dimensions of the CDS.

TFBS - Structural and genomics information compose the integration rule of TFBSs in the CDS (definition 4.2.12). Binding site data such as the nucleotide sequence, strand, and both start and stop positions can be used to establish equivalent TFBSs. Nevertheless, there is still a remote possibility that two binding sites from different organisms may contain the same structural and genomic information. Therefore, TFBS objects contain the organism's foreign key. TFBS objects include a composed hashable key comprehending the organism's foreign key plus the structural information of the binding site. As for the TFBS nucleotide sequence, the organism foreign key is mandatory in TFBS objects. Thus, the integration process removes TFBS objects missing these two properties while merging those that share the same two properties.

Theorem 4.2.12. ***Definition of equivalent TFBSs***

Let $o_a = (id_a, n_a, "organism_a", "sequence_a", "strand_a", "start_a", "stop_a", ...) \in V$ *and* $o_b =$

$(id_b, n_b, "organism_b", "sequence_b", "strand_b", "start_b", "stop_b", ...) \in V$, then $o_a \equiv o_b$ if the following conditions are satisfied:

- $n_a = "TFBS" \land n_b = "TFBS"$
- $"organism_a" = "organism_b"$
- $"sequence_a" = "sequence_b"$
- $"strand_a" = "strand_b"$
- $"start_a" = "start_b"$
- $"stop_a" = "stop_b"$

Motif - The CDS can only store one motif object per regulator. Following integration rule 4.2.13, two motif objects are equivalent if their regulator foreign keys are also different.

Theorem 4.2.13. ***Definition of equivalent motifs***

Let $o_a = (id_a, n_a, "regulator_a", ...) \in V$ and $o_b = (id_b, n_b, "regulator_b", ...) \in V$, then $o_a \equiv o_b$ if $n_a = "Motif" \land n_b = "Motif"$ and $"regulator_a" = "regulator_b"$.

Source - Source objects are created manually in the CDS as new data sources are added to the ETL tool. Thus, the data integration system does not implement any integration rule for source objects.

The transform component of the ETL tool comprehends the integration sub-system. The transformer of each data source is responsible for applying a transform function to the extracted data and using the knowledge expansion sub-system to improve the annotation of the transformed objects. Then, each transformer uses the generic integration tool to filter objects that must be integrated and loaded into the CDS.

The integration tool first standardizes the primary keys used in the integration rules of each entity as follows: text format conversion, lowercase format conversion, and removal of leading and trailing spaces. Then, the entity integration rules determine whether an object is integrated into the CDS. For that, the integration tool takes a snapshot of ProTReND's database and compares incoming objects with the objects present in the database. As a result, objects are marked to be created or updated in the CDS. In a worst-case scenario, a given object can be removed from the integration process if it does not meet the integration rules defined earlier. Finally, all integrated objects are stored in the data lake to be loaded into the CDS later.

The integration tool assigns a single and universal identifier to the integrated objects. ProTReND's identifier is a property common to all entities and used as the primary key in the CDS. ProTReND's identifiers are generated automatically in sequential order for each new object. On the other hand, objects marked to be updated do not take a new identifier, as only non-primary properties will be updated. Figure

22 portrays an example of a ProTReND identifier containing the immutable *PRT* initials, a string entity-based code, and a numeric increment-based code.

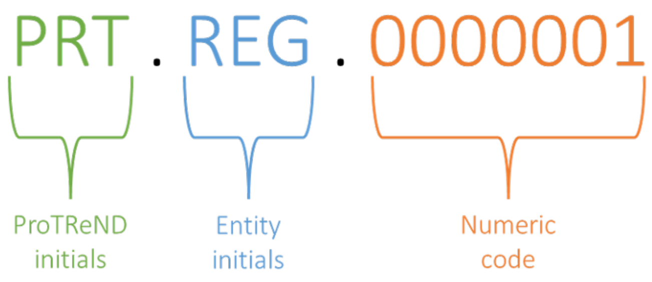


Figure 22: Example of a ProTReND identifier for a regulatory object. The ProTReND identifier contains the immutable ProTReND initials, a string entity-based code (regulator in this example), and a numeric increment-based code.

Once all objects are fully integrated into the CDS, the integration tool standardizes objects' relationships. Relationships are created in the transformation sub-system linking different objects by their primary keys. However, objects can be removed from the integration system, making some relationships obsolete. Furthermore, the data loading sub-system can only create edges in the CDS using ProTReND identifiers. Hence, the integration tool inspects all connections to find corresponding ProTReND identifiers. This lookup procedure is performed based on the objects' primary keys, which have been stored with the connections in the transformation sub-system.

4.2.9 Data loading sub-system

Hitherto, the ETL tool is mainly composed of a set of custom-tailored tools, as the extraction, transformation, and integration of heterogeneous data require different procedures. For instance, some sources covered in this repository are organism-specific, while others contain regulatory data for many prokaryotes. Hence, the tools implemented to deal with these dimensions must be fundamentally different.

On the other hand, the ProTReND repository has a single data loading tool. The integration tool has already performed iterative findings of duplicated objects according to the entities' primary keys, so the loading tool does not have to deal with data normalization and integration. Moreover, integrated objects are available as structured and standardized objects in the data lake.

The data loading sub-system fetches integrated objects stored in the data lake automatically. Then, this tool uses the CDS data model to trim irrelevant properties and add two timestamps. One stands for the time of object creation, while the other corresponds to the time of the last update. Finally, according to the instructions left by the integration tool, the loading tool automatically creates or updates nodes in the CDS with the related properties.

Regarding relationships, the data loading sub-system automatically reads files from the data lake containing edge information. In detail, the data lake contains JSON files that store relationships for many objects. These files have been generated by the integration tool and contain the ProTReND identifier of the source and destination node. Thus, the loading tool is responsible for finding both objects in the CDS and creating the relationship between them with the corresponding properties.

Objects are transformed, integrated, and loaded independently on the CDS. Alternatively, the loading tool can only create relationships if the source and target objects are available on the CDS. Hence, objects can be created without relationships unless this violates their integration rules (mandatory foreign keys). Finally, the data loading sub-system performs a final trimming procedure to circumvent orphan objects (objects without relationships) in the CDS.

4.2.10 The first version of ProTReND's database

The ETL pipeline comprises several steps to populate the CDS. Although each step is autonomous, the order by which each data source is integrated into the CDS can yield slightly different results. Objects can be updated with distinct values, as these are not always the same across databases. According to the integration sub-system, a new object is not created if a similar object is already available in the CDS. Instead, the object in the CDS is updated with the most recent information.

Non-organism-specific databases were firstly integrated into ProTReND's database in the following order: CollecTF; RegPrecise; Abasy; CoryneRegNet. A preliminary analysis found that these databases might contain less detail about a single organism than organism-specific resources. Then, ETL tool integrated the published TRNs, DBTBS, and RegulonDB to complete the objects related to *E. coli*, *B. subtilis*, and *M. tuberculosis*. ODB was the last database to be integrated into the repository. Then, the *Motif* transformer listed all regulators and related TFBS to generate binding motifs. Finally, the ETL ran the trimming procedure to clean orphan objects.

Figure 23 depicts the ETL pipeline execution order used to populate ProTReND's database.

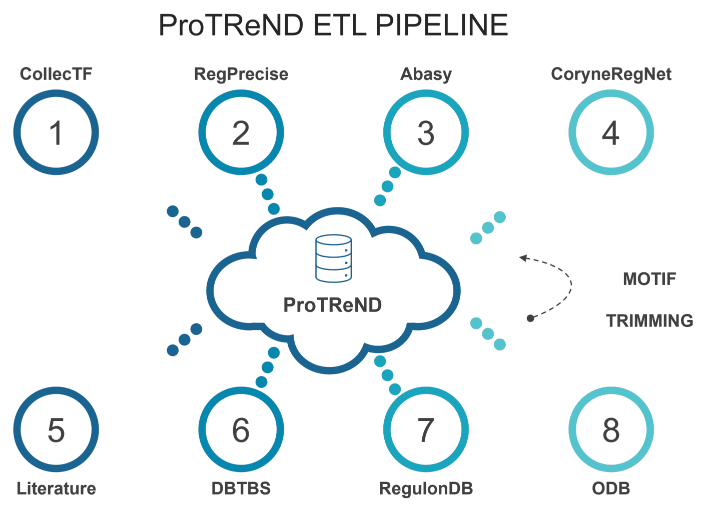


Figure 23: ETL pipeline used to populate ProTReND's database. Non-organism-specific databases, namely CollecTF, RegPrecise, Abasy, and CoryneRegNet, were integrated first. Then, published TRNs and organism-specific database DBTBS and RegulonDB were integrated into the repository. ODB is the last external resource to be integrated into the database. The last steps include generating motifs and a trimming procedure.

4.3 Data integration results

4.3.1 Overview of the data integration results

ProTReND is vastly composed of regulatory interactions. According to figure 24, the regulatory interaction facts table accounts for around 47% of all objects in the CDS. Genes, TFBSs, and operons also make up a significant share of all objects stored in the database, attaining around 47% of all objects. Regulators and motifs are equally represented on the CDS, with around 3% of all objects, indicating that most regulators must have a binding motif. Finally, organisms and other objects account for nearly 1% of all objects in the CDS. The distribution of objects according to their namespace mirrors the graph store model design implemented in the CDS. This graph store model is based on a facts table for regulatory interactions associated with the primary dimensions: organism, regulator, gene, and TFBS.

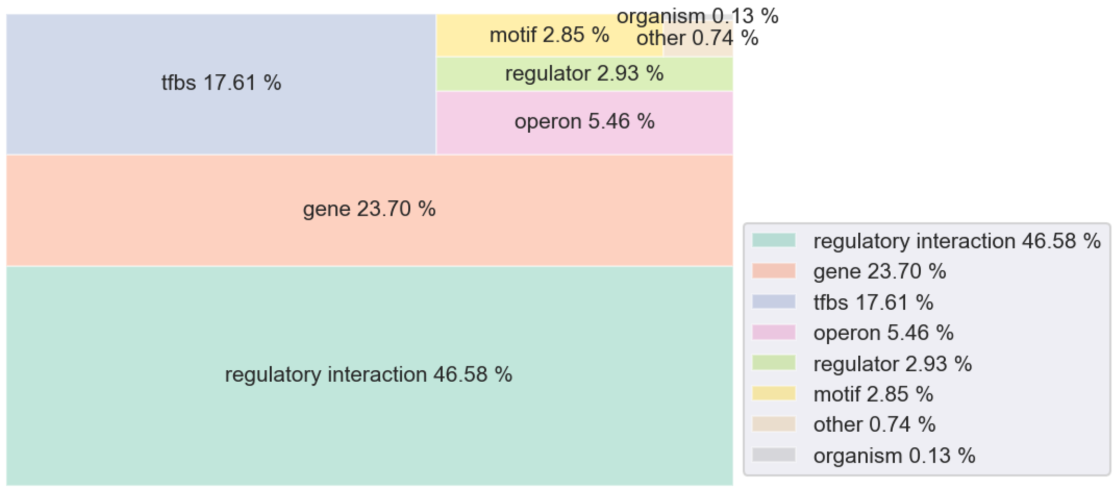


Figure 24: Distribution of objects stored in the ProTReND database by their entity. The treemap shows the relative frequency associated with the main ProTReND entities, namely Regulatory Interaction, Gene, TFBS, operons, regulators, motifs, and organisms. Other entities such as effectors, pathways, regulatory families, evidence, and publications are included in the group named "other".

4.3.2 Integration report

The ETL pipeline reports the results obtained in the extract, transform, integrate and load sub-systems. The reporting tool collects objects at each step for a given regulatory resource integrated into the CDS. Table 6 contains the integration results obtained with the ETL tool.

According to figure 25, the number of genes and regulators extracted from RegPrecise has dropped significantly in the transform sub-system. This decline might be associated with the fact that, due to the ambiguity and lack of data at the genome scale, the transformation tool ignored sRNAs from RegPrecise. Consequently, genes associated with these sRNAs have also been ignored. The sRNAs extracted from RegPrecise are identified with the regulatory family name and shared among several organisms. Besides, the knowledge expansion sub-system attained poor results when annotating sRNAs. Another significant decline is observed in genes, operons, and publications extracted from ODB (table 6). The transform tool for ODB removes operons missing connections with the genes available in the CDS.

The extract sub-system often includes duplicated data for some entities. For example, figure 25 reports a substantial decrease of TFBS in the DBTBS and regulators in the CollecTF databases due to the redundancy of these objects in the extract sub-system.

Regarding the integration sub-system, the number of integrated objects drops significantly in the latest data sources. Figure 25 indicates that multiple organisms, genes, and regulators transformed from Abasy, CoryneRegNet, literature, DBTBS, and RegulonDB already existed in the CDS, as the number of new objects registered in the integration sub-system is lower than in the transform sub-system. On the other hand, CollecTF and RegPrecise were the first resources to be integrated, thus attaining the same numbers in the transform and integration sub-system. However, both TFBSs and regulatory interactions

Table 6: Integration results reported by the ETL pipeline used to assemble ProTReND database. Databases of regulatory data and TRNs available in the literature are listed by the extract, transform, integrate and load sub-system. Each step reports the number of objects associated with a database entity for a regulatory resource. Regulatory resources missing objects of a given entity are marked with a hyphen (-). The number of objects observed in the integration sub-system represents the number of new objects to be integrated into the CDS.

Source	**System**	**Effector**	**Evidence**	**Gene**	**Operon**	**Organism**	**Pathway**	**Publication**	**Regulator**	**R.Family**	**R.Interaction**	**TFBS**
Abasy	extract	-	-	11662	-	9	-	-	707	-	11332	-
	transform	-	-	11543	-	9	-	-	707	-	11332	-
	integrate	-	-	9423	-	3	-	-	526	-	11329	-
	load	-	-	9423	-	3	-	-	526	-	11329	-
CollecTF	extract	-	56	1980	-	153	-	526	342	-	2539	3649
	transform	-	56	1976	-	153	-	526	236	-	2539	3579
	integrate	-	56	1976	-	153	-	526	230	-	2539	3579
	load	-	56	1976	-	153	-	526	230	-	2539	3579
CoryneRegNet	extract	-	3	3921	-	4	-	465	539	-	9051	4841
	transform	-	3	3881	-	4	-	465	538	-	9051	4479
	integrate	-	3	419	-	0	-	441	168	-	8844	4479
	load	-	3	419	-	0	-	441	168	-	8844	4479
DBTBS	extract	-	-	682	-	1	-	819	109	2	1163	1374
	transform	-	-	682	-	1	-	819	109	2	1163	965
	integrate	-	-	20	-	0	-	462	0	2	1000	965
	load	-	-	20	-	0	-	462	0	2	1000	965
Literature	extract	140	-	4859	-	3	-	-	530	-	12057	-
	transform	140	-	4859	-	3	-	-	530	-	12057	-
	integrate	90	-	649	-	0	-	-	29	-	9549	-
	load	90	-	649	-	0	-	-	29	-	9549	-
ODB	extract	-	-	3232959	6135320	-	-	485	-	-	-	-
	transform	-	-	12156	22905	-	-	137	-	-	-	-
	integrate	-	-	492	22905	-	-	66	-	-	-	-
	load	-	-	492	22905	-	-	66	-	-	-	-
RegPrecise	extract	308	-	116716	-	524	287	311	15432	171	155893	62483
	transform	308	-	89245	-	524	287	311	11605	171	155893	61965
	integrate	308	-	88477	-	460	287	299	11518	171	155893	61965
	load	308	-	88477	-	460	287	299	11518	171	155893	61965
RegulonDB	extract	137	127	4168	-	1	-	16730	199	52	11384	6124
	transform	137	127	4085	-	1	-	16730	199	52	11384	6124
	integrate	103	123	1565	-	0	-	16628	7	30	6540	6117
	load	103	123	1565	-	0	-	16628	7	30	6540	6117

are notable exceptions to the decline in the integration sub-system. Integration rules created for these objects are based on data normalization involving properties that usually do not match across databases.

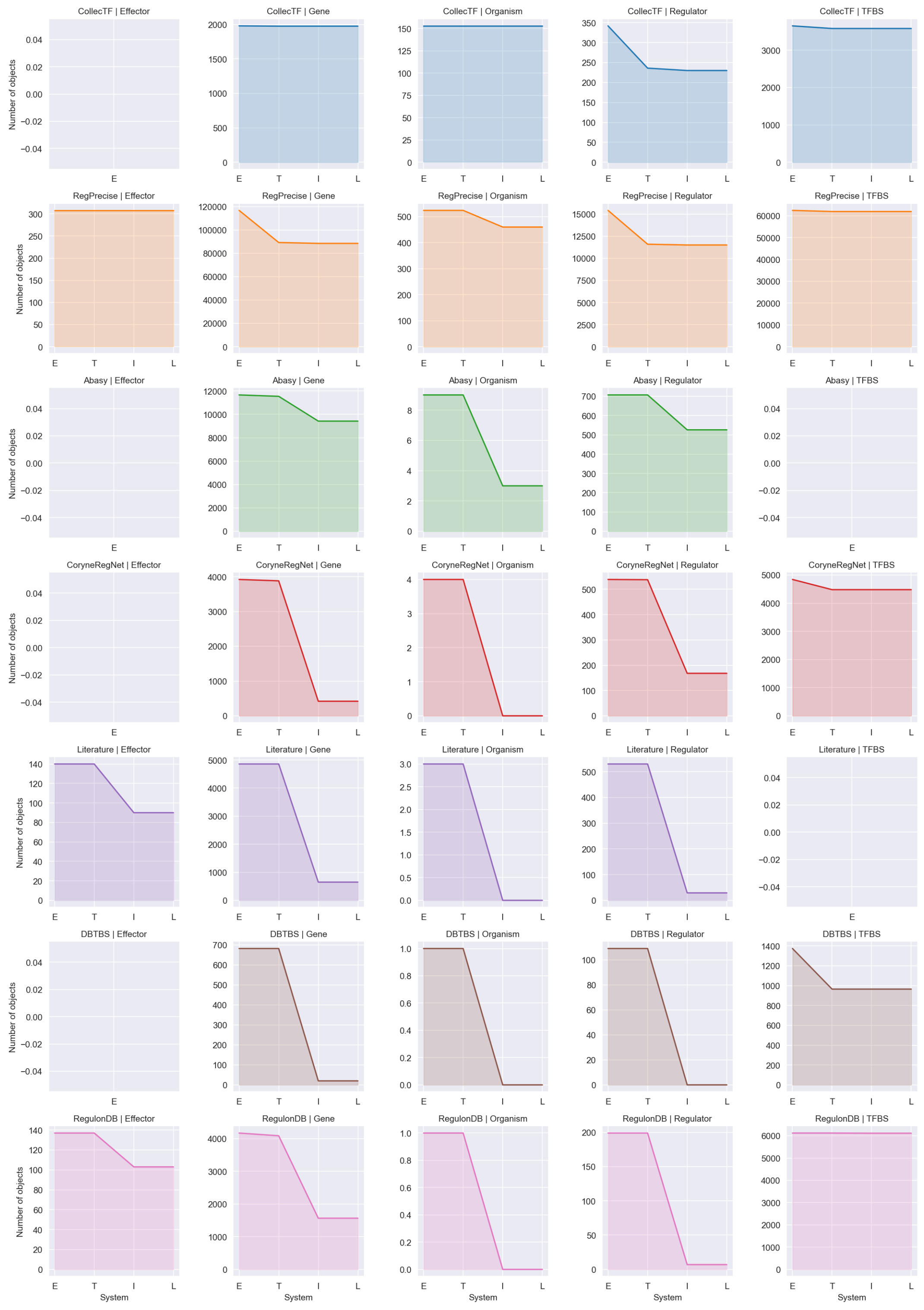


Figure 25: Integration results of ProTReND's ETL pipeline summarized by the main entities and sources. Databases of regulatory data and TRNs available in the literature are separated by rows. Effectors, genes, organisms, regulators, and TFBS are divided into columns. Each chart depicts the number of objects streaming from the extract to the load sub-systems. Regulatory resources missing objects of a given entity are marked with a blank chart. The area under the line represents the number of objects observed in the following sub-systems: extract (E), transform (T), integration (I), and load (L).

In ProTReND, regulatory interactions must include foreign keys for organisms, regulators, genes, TFBS, and effectors. Whereas several regulatory resources provide regulatory interactions that follow this definition, other resources include non-intuitive and indirect associations between these entities in multiple files. To avoid reporting raw regulatory interactions from some sources and transformed objects from others, we have decided that the number of regulatory interactions is determined in the transform sub-system.

The last step of the ETL pipeline consists of a trimming procedure to remove objects missing relationships and other errors. For instance, regulators missing relationships with genes or vice-versa are removed from the database during the trimming procedure. Likewise, organisms missing links to regulators and genes are removed from the CDS. Figure 26 reports the relative frequency of removed objects in this procedure. The relative frequency is determined by the number of objects before the trimming procedure divided by the number of objects available in the CDS. Numerous publications and evidence were not associated with regulatory interactions and regulators and were thus removed from the database. These objects, mainly obtained from RegulonDB, were associated with other entities not included in ProTReND, such as promoters and signals. A few organisms, effectors, and regulatory families were also removed from the CDS due to missing relationships with regulator objects.

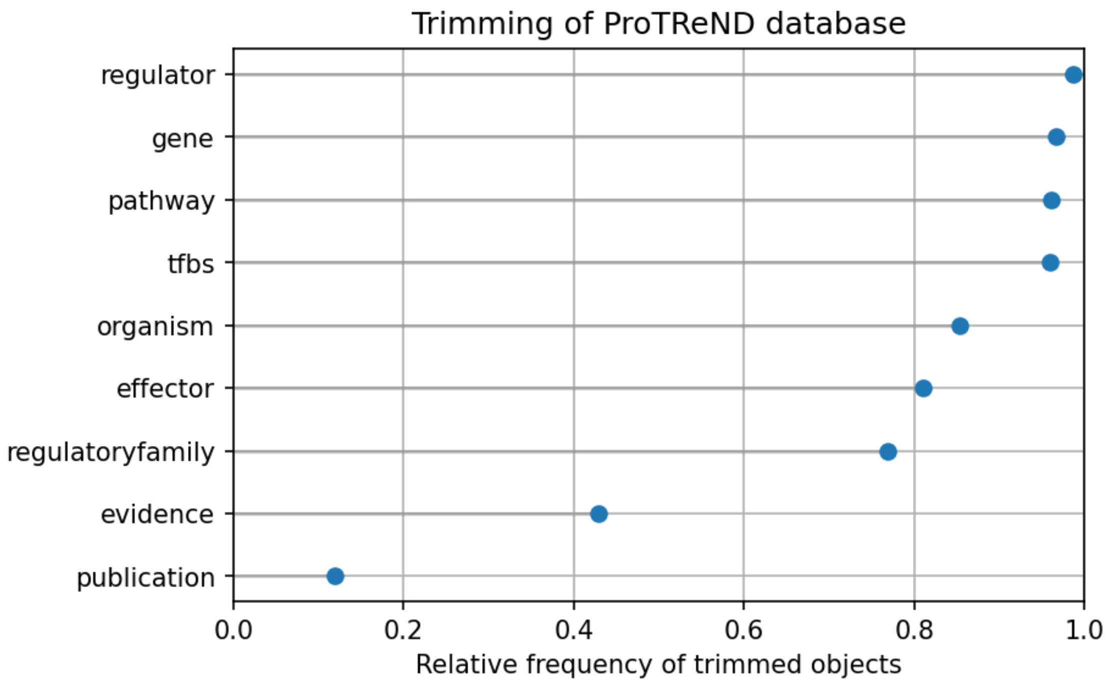


Figure 26: Summary of the trimming procedure performed in the ProTReND database. The relative frequency of objects removed during the trimming procedure is depicted for each entity. The relative frequency is determined by the number of objects before the trimming procedure divided by the number of objects available in the CDS.

The current version of ProTReND comprehends seven resources of regulatory data and three TRNs retrieved from the literature. Figure 27 contains an analysis of the relationships between objects and regulatory sources available in ProTReND's database. This analysis is based on the number of relationships between objects and sources for each entity. The relative frequency of one source is determined by the number of relationships of this source divided by the number of all source relationships.

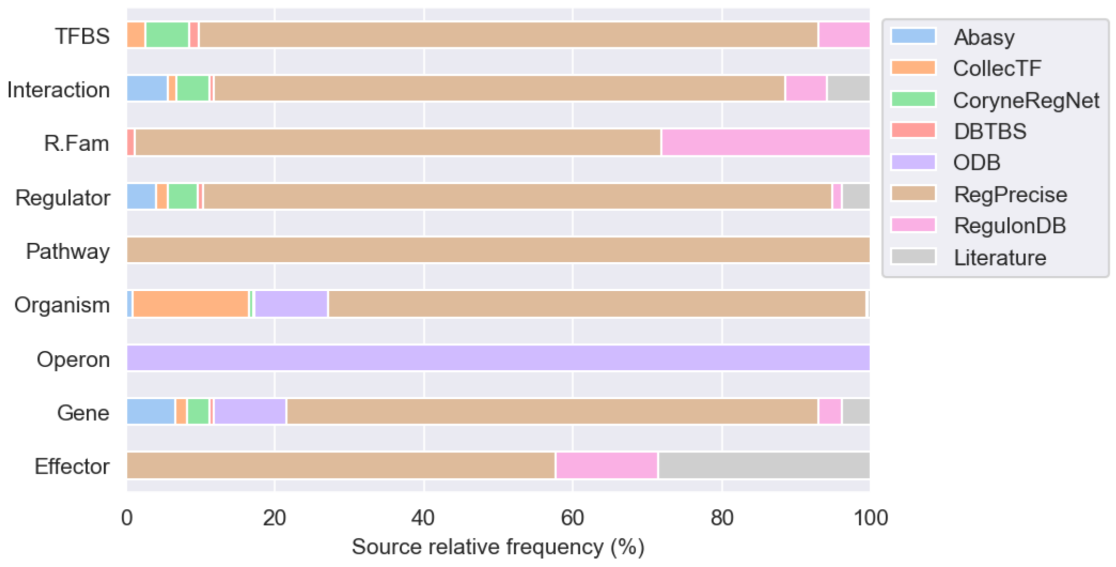


Figure 27: Analysis of the regulatory data available in ProTReND by the data source. The frequency of relationships between objects and regulatory sources was surveyed in ProTReND database. For a database entity, the relative frequency was determined as the number of relationships of a single source divided by the total of source-like relationships. The data sources' relative frequency has been stacked into a horizontal bar for each entity.

According to figure 27, RegPrecise is the primary source of regulatory data in ProTReND accounting for numerous organisms, TFBSs, regulatory interactions, regulators, and genes, among others. Abasy database is also linked to a significant share of regulators, genes, and regulatory interactions. Likewise, many regulators, genes, and regulatory interactions have been extracted from CoryneRegNet, also linked to a significant number of TFBS. However, Abasy and CoryneRegNet only contain data for 9 organisms in total. In contrast to the large number of organisms directly linked to CollecTF, this database has provided fewer interactions than RegulonDB (an organism-specific database) and literature. DBTBS is a notable exception, as a small volume of regulatory data has been extracted for *B. subtilis* in this database. It is worth noting that one object can be linked to several regulatory sources. Thus, the relative frequency mentioned above cannot be interpreted as the percentage of objects that belong to a given source. Figure 28 highlights the out-degree (Kout) of effectors, genes, organisms, regulators, regulatory families, regulatory interactions, and TFBSs by source. The out-degree of an object stands for the number of outgoing relationships with other objects. In this case, the out-degree is determined by the relationships associated with source nodes.

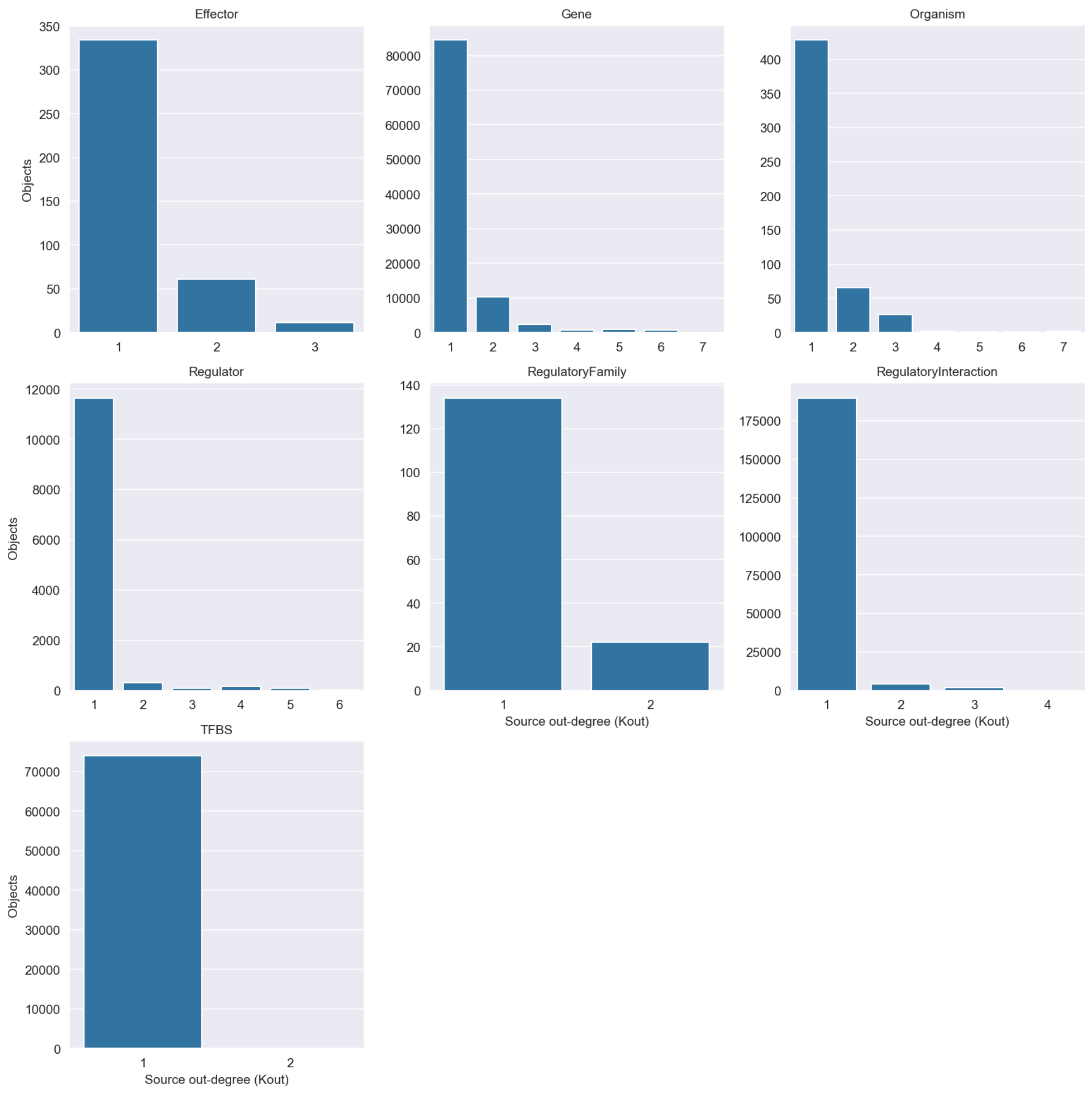


Figure 28: Out-degree (Kout) frequency by the data source in ProTReND database. The source out-degree (Kout) of effectors, genes, organisms, regulators, regulatory families, regulatory interactions, and TFBSs is calculated by the number of outgoing relationships towards objects of the source namespace.

Most objects are linked to a single resource of regulatory information (figure 28). This observation goes in line with the fact that most data has been extracted from RegPrecise, while other databases contributed with significantly less data. Nevertheless, objects obtained from the remaining databases are also linked to RegPrecise, as some effectors, genes, organisms, regulators, and regulatory families have out-degrees superior to one. Regulatory interactions and TFBSs have fewer integrations across the data sources, as these objects systematically rank out-degrees of one due to complex integration rules.

The knowledge expansion sub-system was designed to enhance the annotation of organisms, regulators, genes, effectors, pathways, and publications often missing relevant data to assemble TRNs at the genome scale. Figure 29 shows that the knowledge expansion sub-system has significantly improved the functional annotation of regulators and genes. In addition, this system retrieved genomics and proteomics data often missing in the extracted regulatory sources. Around 90% of regulators and genes comprehend genomic coordinates, gene sequence, protein sequence, and external references to genomics and proteomics databases, except RefSeq accession, which is present in around 75% of these objects.

Furthermore, all organisms integrated into the CDS have been associated with the corresponding NCBI taxonomy identifier, and 99% of the prokaryotes are linked to the GenBank, RefSeq and NCBI assembly databases. Likewise, all publications in ProTReND database are linked to the PubMed database.

According to figure 29, around 85% of effectors have been linked to at least one KEGG compound. Nevertheless, most pathways integrated into the database are not associated with any KEGG pathway.

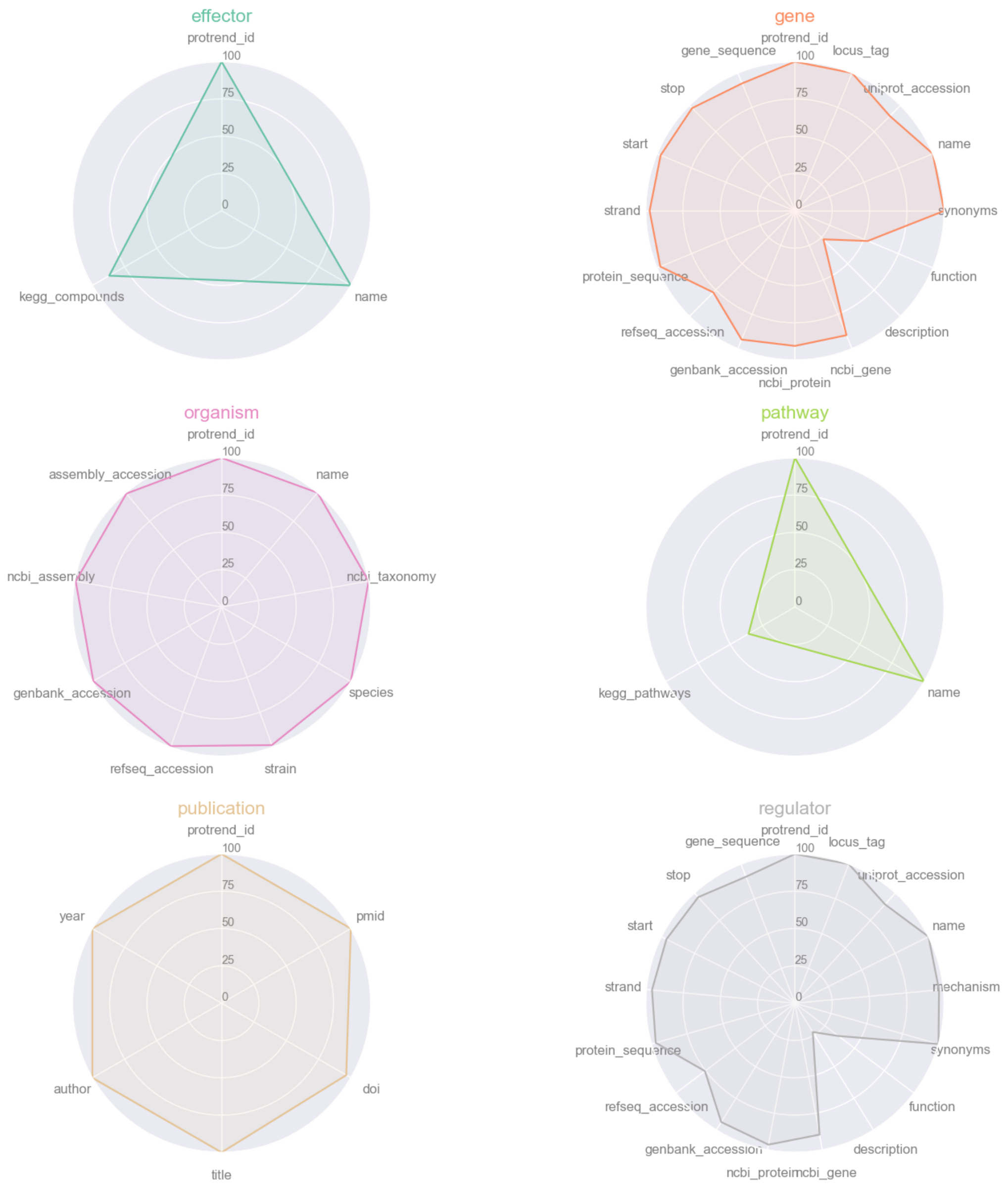


Figure 29: Results of the analysis of the knowledge expansion sub-system in the ProTReND repository. Properties' relative frequency is determined by the number of objects having the property divided by the total of objects in the CDS.

4.3.3 CDS topology

The analysis of the CDS topology consists of a survey of the TRNs compiled in the data integration system. Figure 30 contains the organism density distribution as a function of the number of regulators, genes, TFBSs, and regulatory interactions. The density distribution, based on the organism-entity out-degree (Kout), indicates how frequently organisms are associated with regulators, genes, TFBSs, and regulatory interactions. For example, organism *PRT.ORG.0000012* (*B. subtilis* 168) has a regulator out-degree of 246, as this organism is associated with 246 distinct regulators. Nevertheless, the probability obtained by the density function for a regulator out-degree of 246 can be very low, as few organisms can be associated with 246 (or similar) distinct regulators. In fact, figure 30 contains logarithmized out-degrees ($lnKout$), as a few organisms are associated with numerous regulators, genes, TFBSs, and regulatory interactions. Besides, the range of the regulators' out-degree is significantly inferior to the range of the genes, TFBSs, and regulatory interactions out-degree.

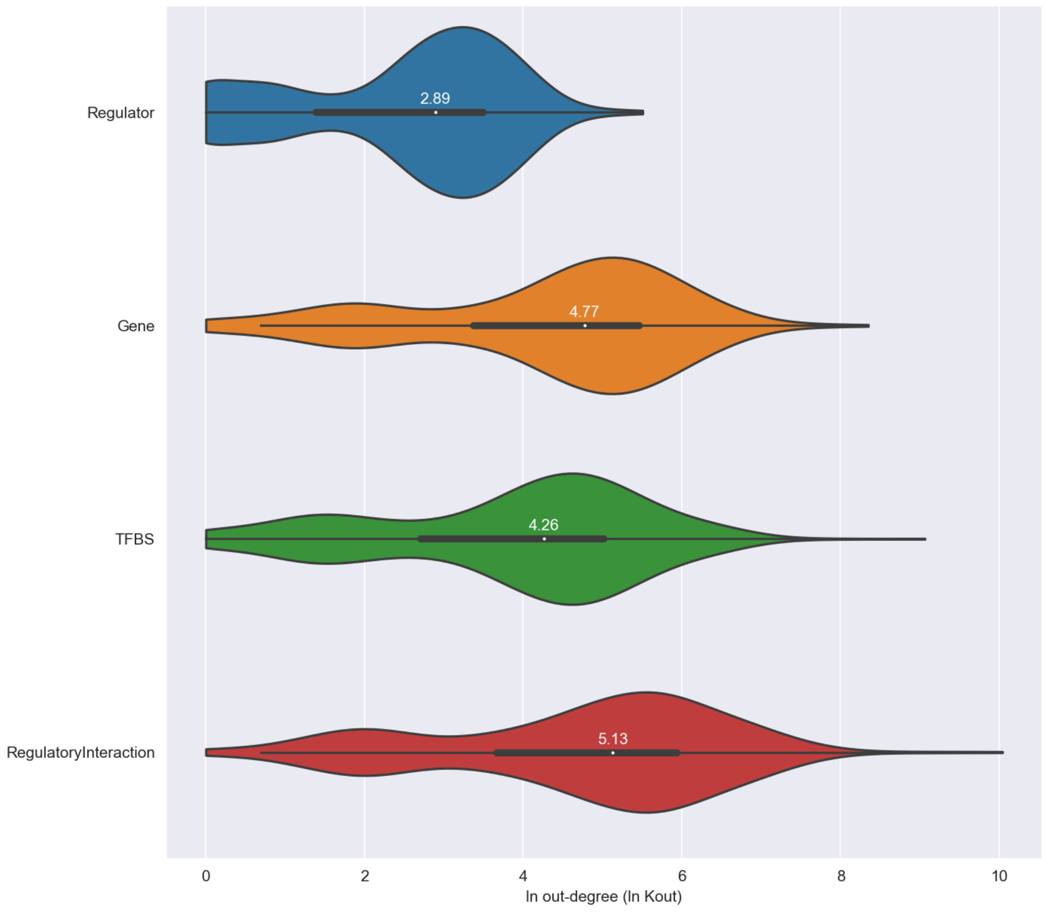


Figure 30: Density distribution of regulators, genes, TFBSs and regulatory interactions per organism. A probability density function estimates the density distribution based on the organism-entity out-degree (Kout). Due to the different range of values among entities, out-degrees were logarithmized ($lnKout$). Moreover, a small share of organisms is associated with numerous regulators, genes, TFBSs, and regulatory interactions, following an exponential distribution. The median out-degree (white dot) between organisms and entities is plotted together with the interquartile range (thick black horizontal bar). The density estimation corresponds to the area of the violin plot.

According to figure 30, the median out-degree between organisms and regulators is significantly inferior to the median of out-degrees for genes, TFBS, and regulatory interactions. Organisms are more frequently associated with about $\ln 18$ (2.89) regulators, $\ln 118$ (4.77) genes, $\ln 71$ (4.26) TFBS, and $\ln 169$ (5.13) regulatory interactions. The density distribution also shows a small number of organisms associated with fewer regulators, genes, TFBS, and regulatory interactions. On the other hand, small density values estimated in higher out-degrees indicate that a small number of organisms have out-degrees superior to the median values. Organism-specific databases or published TRNs have likely increased the volume of regulatory interactions in these organisms. Figure 31 contains the top 20 organisms in terms of out-degree for regulators, genes, TFBSs, and regulatory interactions, showing the composition of the most well-represented TRNs in the ProTReND database.

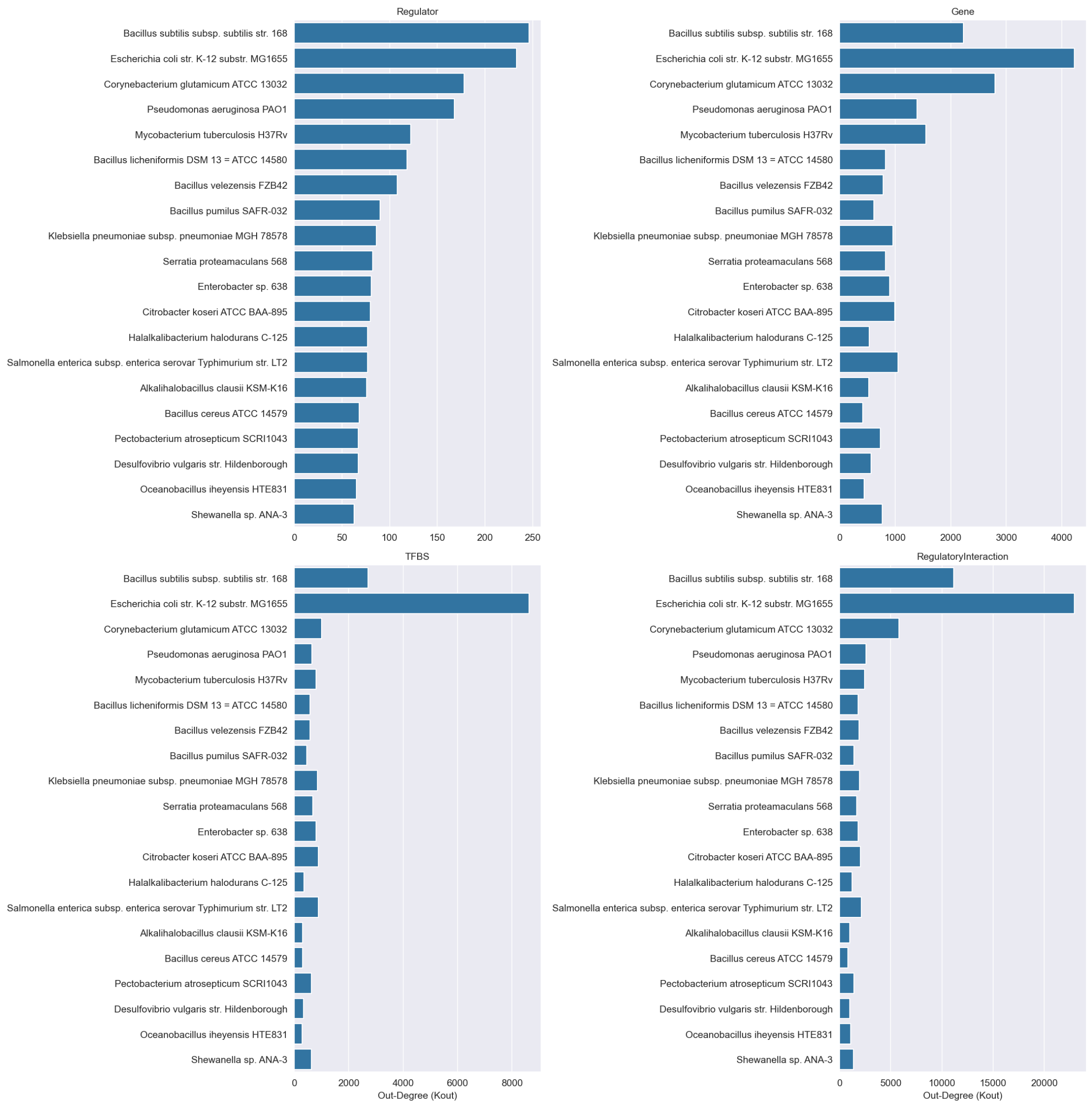


Figure 31: Organisms having the largest out-degrees regarding regulators, genes, TFBSs and regulatory interactions. Top 20 organisms ranking the highest out-degrees (Kout) for regulators, genes, TFBSs, and regulatory interactions. The organisms are labeled by their scientific name.

B. subtilis 168, *E. coli* K-12 and *M. tuberculosis* H37Rv are well documented in the organism-specific databases and published TRNs. Consequently, these bacteria have high out-degrees for regulators, genes, TFBS, and regulatory interactions. In detail, *B. subtilis* 168 is associated with more regulators than *E. coli* K-12, though the lactic acid bacterium has fewer relationships with genes and TFBSs. Hence, *E. coli* K-12

TRN has significantly more regulatory interactions than the network of *B. subtilis* 168. Interestingly, *C. glutamicum* ATCC 13032 and *P. aeruginosa* PAO1 are also highly represented in ProTReND. The former bacterium is well documented with regulators, genes, TFBSs, and regulatory interactions in CoryneRegNet, while Abasy contains an extensive TRN for *P. aeruginosa* PAO1.

The regulatory data associated with the remaining organisms are mainly retrieved from RegPrecise and CollecTF. For instance, the regulators associated with the remaining 15 organisms were extracted exclusively from these two data sources. Overall, the out-degree distribution in the remaining 15 organisms is homogeneous, having fewer genes, TFBS, and regulatory interactions than the top 5 bacteria.

Regarding the regulators' out-degree distribution, figure 32 contains the logarithmized out-degrees ($lnKout$) between regulators and genes, TFBSs, and regulatory interactions. As before, a small share of regulators is associated with multiple genes, TFBSs, and regulatory interactions, thus following an exponential distribution. The natural logarithm of the regulator-entity out-degree allows standardizing large values that can hinder the estimation of the density function.

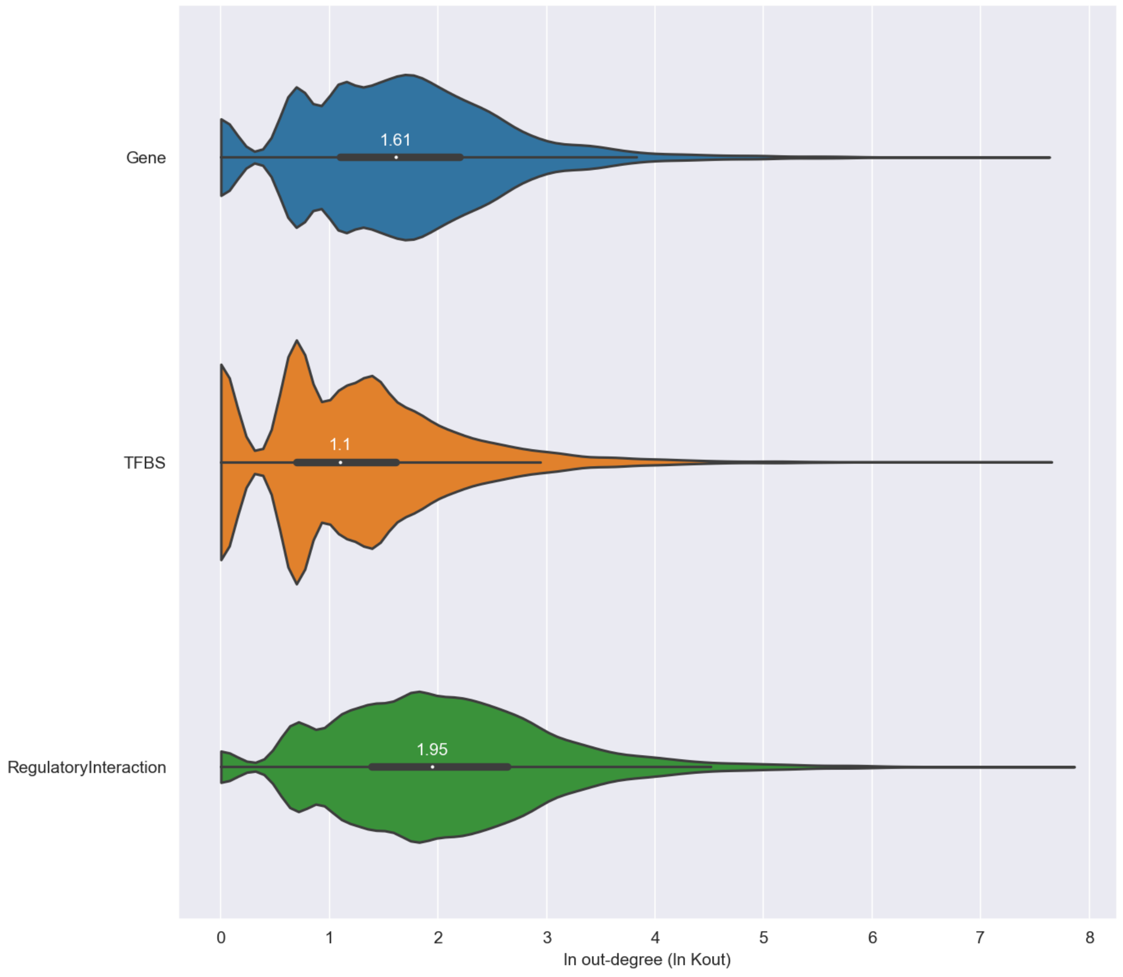


Figure 32: Density distribution of genes, TFBSs and regulatory interactions per regulator. A probability density function estimates the density distribution based on the regulator-entity out-degree (Kout). Out-degrees were logarithmized ($lnKout$) as a small share of organisms is associated with numerous regulators, genes, TFBSs, and regulatory interactions, following an exponential distribution. The median out-degree (white dot) between organisms and entities is plotted together with the interquartile range (thick black horizontal bar). The density estimation corresponds to the area of the violin plot.

Figure 32 shows that median out-degrees are similar across genes, TFBSs, and regulatory interactions. Regulators are more frequently associated with around five ($\ln 1.61$) genes, three ($\ln 1.1$) genes, and seven ($\ln 1.95$) regulatory interactions. Regulators are associated with fewer genes, TFBS, and regulatory interaction, as the density probabilities are higher for smaller out-degrees. Nevertheless, regulators can be related to a large number of interactions. For example, global transcription factors *PRT.REG.0000202* (B0683/fur regulator in *E. coli*), *PRT.REG.0000198* (B3357/crp regulator in *E. coli*), and *PRT.REG.0011761* (BSU_25200/sigA sigma factor in *B. subtilis*) are associated with more than 500 genes each.

The most well-represented regulatory families in ProTReND are presented in figure 33. Most families depicted in this figure are well documented in the literature, RegPrecise, and RegulonDB. For instance, the

GntR [212] and LacI [213] families attained high out-degrees, being the most prevalent families associated with regulators stored in the CDS. Likewise, TetR [214], Fur [215], and MerR [216] families are also broadly available in the CDS, having relationships with around 600 regulators each.

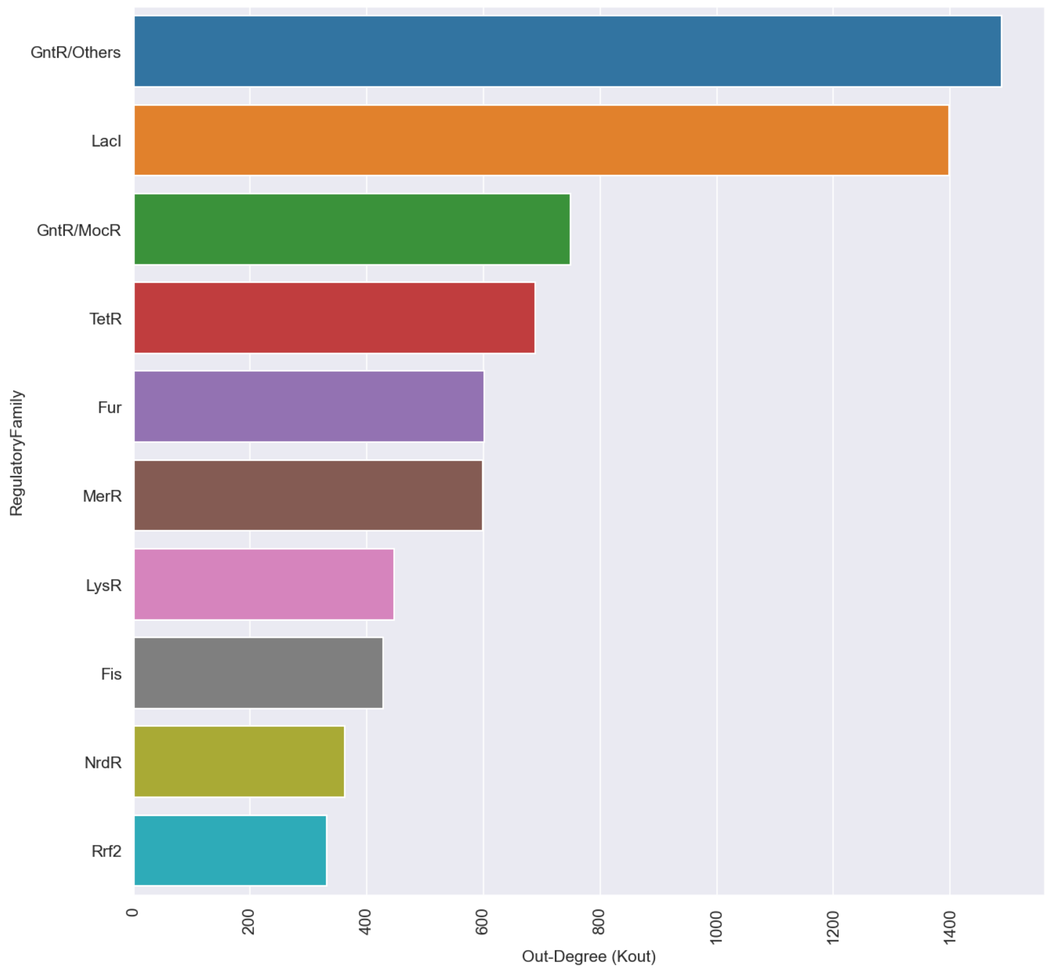


Figure 33: Regulatory families having the largest out-degree regarding regulators. Top 10 regulatory families with the highest out-degrees (Kout) for regulators.

4.4 ProTReND web application

4.4.1 A user-friendly hub of regulatory interactions

ProTReND web application is designed to be a user-friendly tool to visualize regulatory interactions available in ProTReND's database. Besides visualizing organisms, regulators, genes, and interactions, one can download manually or programmatically all regulatory data compiled within ProTReND. The web application allows users to contribute to ProTReND's database using the user-friendly community's graphical interface. The next sections will demonstrate the most relevant features of ProTReND's web application. ProTReND is available at protrend.bio.uminho.pt.

4.4.2 Implementation

ProTReND web application is implemented using the Django framework in Python. The architecture of this web application is available in figure 34. The web application follows the model-template-views pattern adopted by Django, which encourages the implementation of object-oriented models for handling CRUD operations in databases. Hence, the Django framework is highly compatible with the graph store model adopted in the CDS. Furthermore, the django-neomodel plugin, an extension of neomodel for Django, was used to link ProTReND's database to the development of the web application. As a result, the web application is based on the data warehouse reconstructed in the data integration system.

The Django REST framework plugin was used to implement ProTReND's RESTful API. This plugin allows serializing database objects into several formats, such as JSON, Extensible Markup Language (XML), CSV, and XLSX, and provides a browsable interface for the API.

The web application contains a powerful search engine created with the Whoosh Python package. Organisms, regulators, genes, interactions, effectors, pathways, and regulatory families have been retrieved from the database and indexed as documents to create the search index.

The user-friendly community application has been implemented with the django-material plugin. This plugin creates a web-based single-page application supporting CRUD operations in a relational database. Each community object contains a list and detail view based on table-like and form-like layouts. These views were created to visualize, create, update and delete a single table entry in the relational database. Hence, a new database has been designed to store user contributions in the community web application.

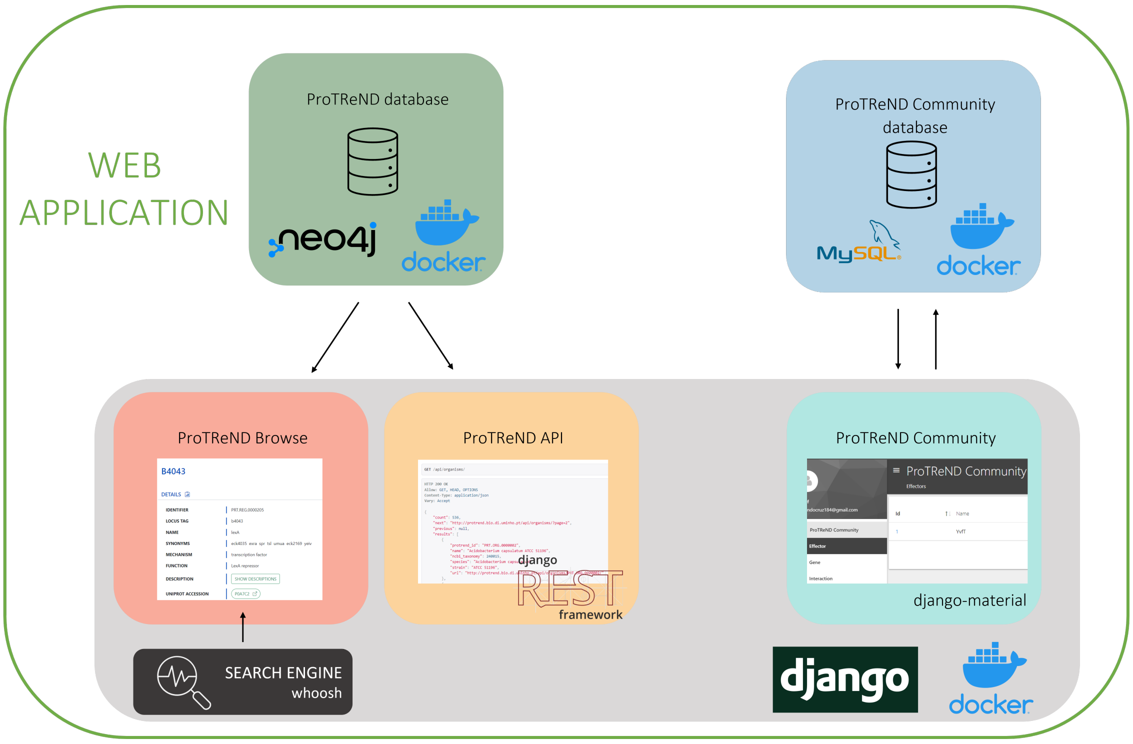


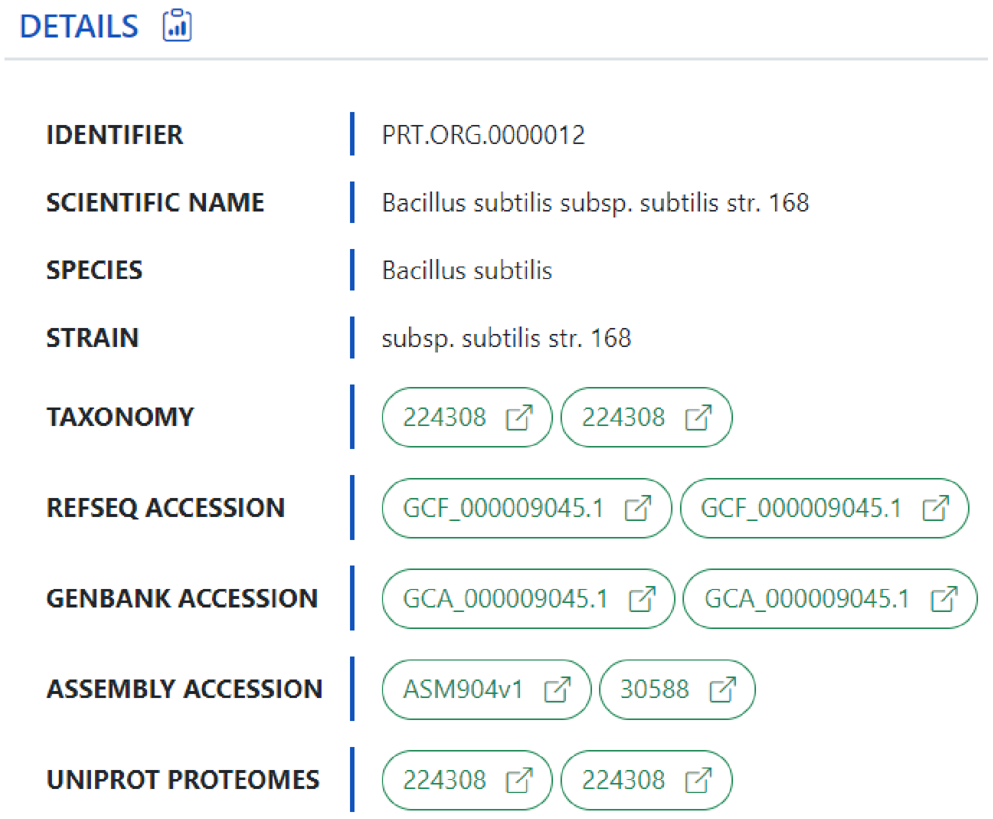
Figure 34: Architecture of ProTReND web application. The web application retrieves data from the CDS using the Django framework. Regulatory data can also be automatically retrieved using ProTReND API implemented with the Django REST framework. Users can contribute to ProTReND in the community application implemented with the django-material plugin. This user interface performs CRUD operations to a relational database implemented with MySQL DBMS. The web application has been containerized into several autonomous sub-applications.

An in-house server hosts the ProTReND web application at protrend.bio.uminho.pt containerized with Docker.

4.4.3 Browse regulatory interactions

Users can browse the list of organisms available in ProTReND. In the organism's view, one can find the ProTReND identifier, scientific name, and NCBI taxonomy identifier and the link to the detailed view of each organism. This view also contains statistics for the organism-regulator distribution, most frequent organisms by regulator number, among others.

The detailed view of a given organism includes relevant taxonomy information, such as scientific name, species, strain, and NCBI taxonomy identifier. Figure 35 shows external references that link the organism's genome and proteome in the following databases: RefSeq, GenBank, NCBI Assembly, and UniProt. The statistics of the prokaryote's TRN are also available on the organism web page comprising, for instance, regulator-gene distributions. The organism view lists all resources of regulatory information used to compile the prokaryote TRN. The source of the regulatory data is cross-referenced to the organism's detailed web page in the resource whenever possible.

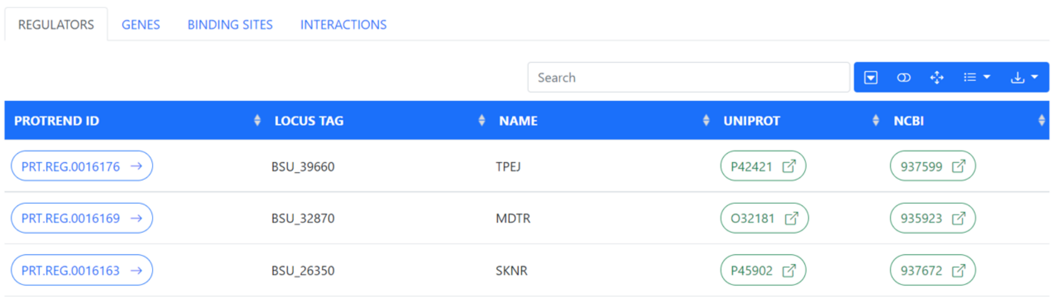


DETAILS

IDENTIFIER	PRT.ORG.0000012
SCIENTIFIC NAME	Bacillus subtilis subsp. subtilis str. 168
SPECIES	Bacillus subtilis
STRAIN	subsp. subtilis str. 168
TAXONOMY	224308 224308
REFSEQ ACCESSION	GCF_000009045.1 GCF_000009045.1
GENBANK ACCESSION	GCA_000009045.1 GCA_000009045.1
ASSEMBLY ACCESSION	ASM904v1 30588
UNIPROT PROTEOMES	224308 224308

Figure 35: Example of the information available for an organism in ProTReND. The detailed web page of an organism (e.g., *B. subtilis* subsp. *subtilis* str. 168) contains relevant taxonomy information, such as scientific name, species, strain, and NCBI taxonomy identifier together with external references.

TRNs are the main focus of ProTReND web application. The network section contains data tables to visualize and search a given organism's TRN. As shown in figure 36, these data tables contain information regarding regulators, genes, TFBSs, and regulatory interactions associated with a given organism. More importantly, regulators, genes, and interactions listed in the network section contain links to their detailed web pages or external references (UniProt and NCBI).



PROTREND ID	LOCUS TAG	NAME	UNIPROT	NCBI
PRT.REG.0016176	BSU_39660	TPEJ	P42421	937599
PRT.REG.0016169	BSU_32870	MDTR	O32181	935923
PRT.REG.0016163	BSU_26350	SKNR	P45902	937672

Figure 36: Example of the organism network section in ProTReND. The detailed web page of an organism (e.g., *B. subtilis* subsp. *subtilis* str. 168) includes data tables listing regulators, genes, TFBS, and regulatory interactions available in the TRN.

The detailed organism view also contains tools to ease TRN visualization. For example, users can select up to 15 regulators to visualize corresponding regulatory interactions in the graph network and HEB tools. Figure 37 depicts the regulatory interactions associated with the regulators *PRT.REG.0016169* and *PRT.REG.0016176* in the graph network and HEB tools.

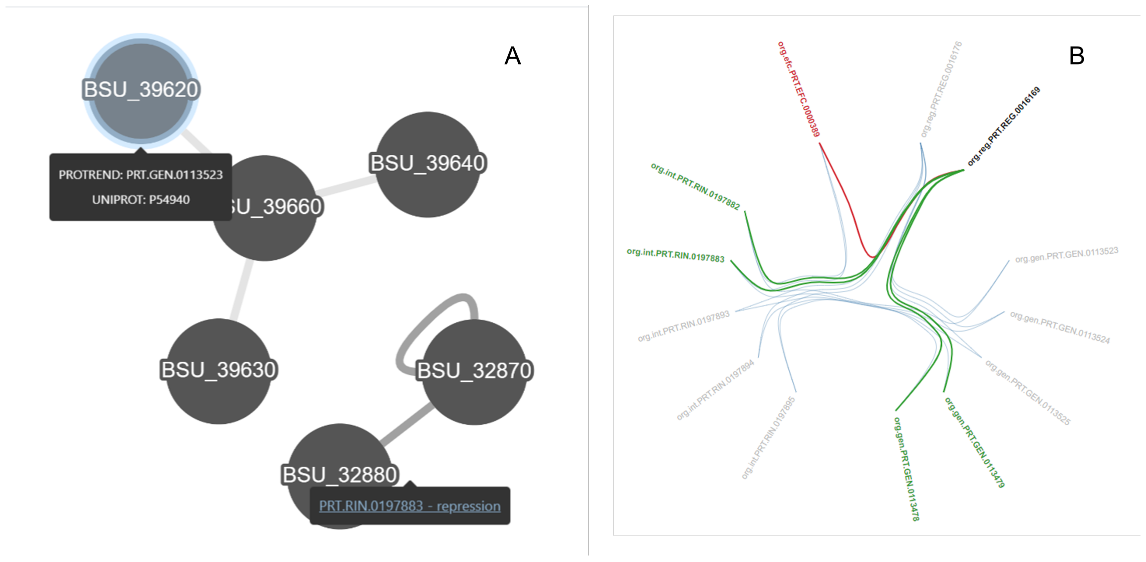
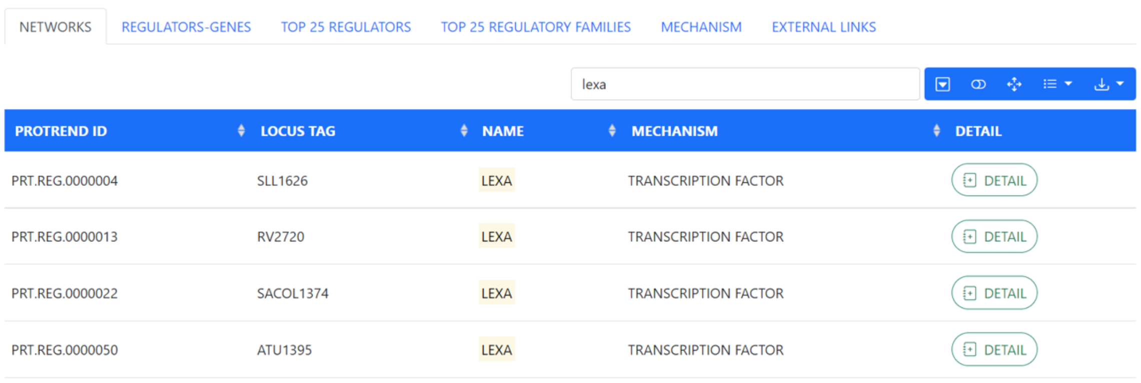


Figure 37: Example of the graph network and HEB visualizations for an organism in ProTReND. The detailed web page of an organism (e.g., *B. subtilis* subsp. *subtilis* str. 168) includes graph network (A) and HEB (B) tools to ease the visualization of regulatory interactions associated with regulators (e.g., *PRT.REG.0016169* and *PRT.REG.0016176*).

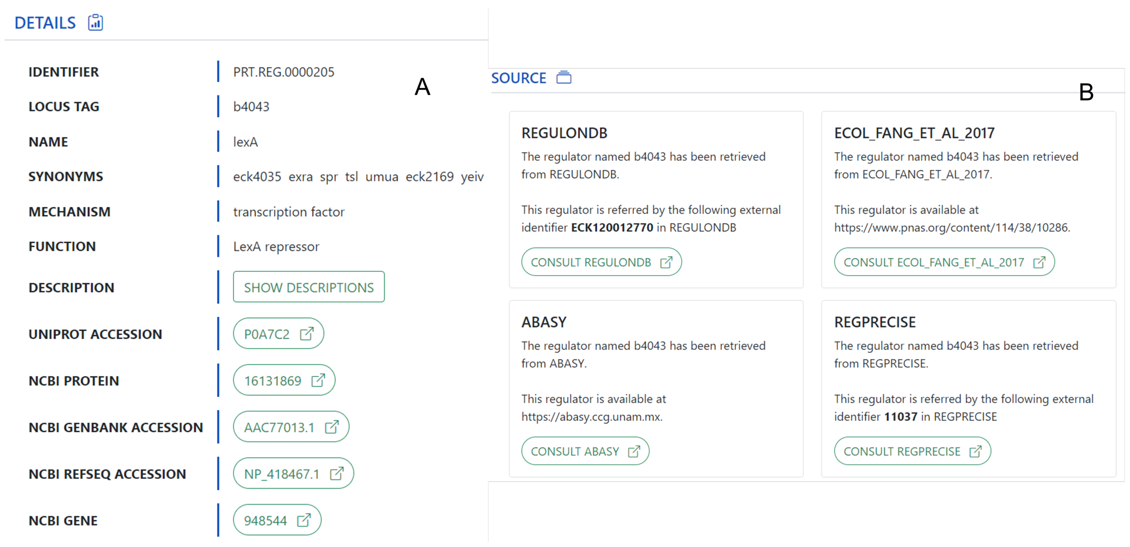
The regulators' view includes a data table to browse and search the list of regulators by their unique identifier, locus tag, and name. One can find the link to the detailed view of a given regulator in this table. For example, figure 38 highlights a search for all *lexa* regulators. In addition to the list of regulators, this view also contains several statistics for the regulator-gene distribution, most frequent regulators by gene number, among others.



PROTREND ID	LOCUS TAG	NAME	MECHANISM	DETAIL
PRT.REG.0000004	SLL1626	LEXA	TRANSCRIPTION FACTOR	DETAIL
PRT.REG.0000013	RV2720	LEXA	TRANSCRIPTION FACTOR	DETAIL
PRT.REG.0000022	SACOL1374	LEXA	TRANSCRIPTION FACTOR	DETAIL
PRT.REG.0000050	ATU1395	LEXA	TRANSCRIPTION FACTOR	DETAIL

Figure 38: Example of the regulators' data table in ProTReND. The Regulators web page allows browsing and searching all regulators (e.g., *lexa* regulators) in ProTReND.

Users can access the regulator's detailed web page to obtain information about functional annotation, genomics, proteomics, statistics, sources, organisms, regulatory families, metabolic pathways, binding motifs, and sub-networks. Figure 39(A) depicts the web page of regulator *b4043* of *E. coli*, including ProTReND identifier, locus tag, name, synonyms, mechanism, function, and description. The detail section also contains external references found for the regulator in NCBI's and UniProt's genomics and proteomics databases. Genomics and proteomics data can also be downloaded in the FAST-All (FASTA) and GenBank file formats. This view also includes statistics regarding the sub-network directly associated with the regulator. As with the organism web page, the source of the regulatory data is cross-referenced to the regulator's detailed web page in the resource whenever possible (Figure 39(B)). One can also access the regulatory family and metabolic pathways associated with the regulator.



IDENTIFIER	PRT.REG.0000205
LOCUS TAG	b4043
NAME	lexA
SYNONYMS	eck4035 exra spr tsl umua eck2169 yeiv
MECHANISM	transcription factor
FUNCTION	LexA repressor
DESCRIPTION	SHOW DESCRIPTIONS
UNIPROT ACCESSION	P0A7C2
NCBI PROTEIN	16131869
NCBI GENBANK ACCESSION	AAC77013.1
NCBI REFSEQ ACCESSION	NP_418467.1
NCBI GENE	948544

REGULONDB
The regulator named b4043 has been retrieved from REGULONDB.
This regulator is referred by the following external identifier **ECK120012770** in REGULONDB
CONSULT REGULONDB

ECOL_FANG_ET_AL_2017
The regulator named b4043 has been retrieved from ECOL_FANG_ET_AL_2017.
This regulator is available at https://www.pnas.org/content/114/38/10286.
CONSULT ECOL_FANG_ET_AL_2017

ABASY
The regulator named b4043 has been retrieved from ABASY.
This regulator is available at https://abasy.ccg.unam.mx.
CONSULT ABASY

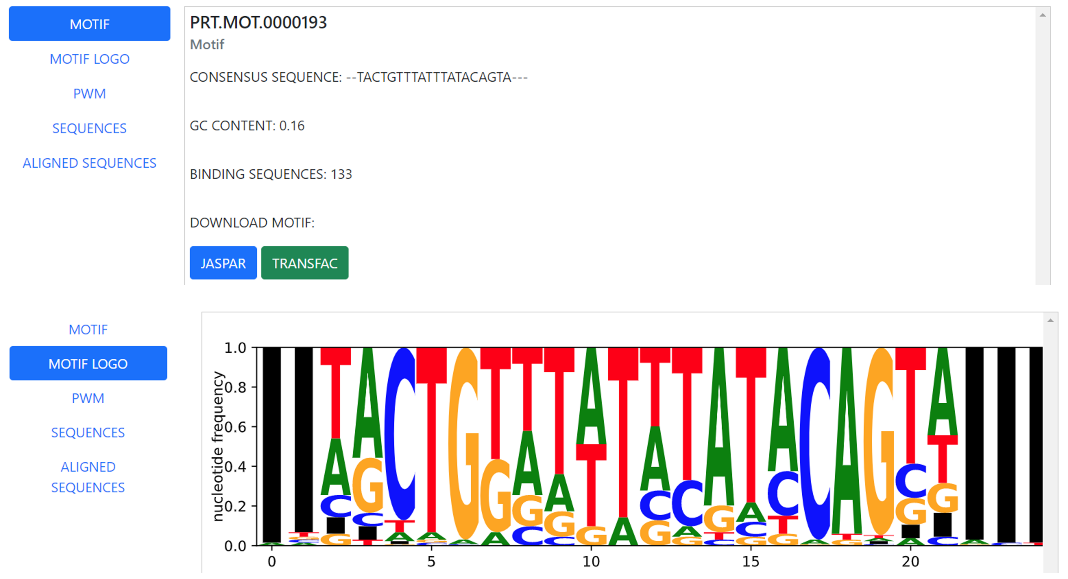
REGPRECISE
The regulator named b4043 has been retrieved from REGPRECISE.
This regulator is referred by the following external identifier **11037** in REGPRECISE
CONSULT REGPRECISE

Figure 39: Example of the information available for a regulator in ProTReND. The detailed web page of a regulator (e.g., *b4043* in *E. coli*) contains relevant information, such as locus tag, name, synonyms, mechanism, function, and description together with external references (A). This view also contains cross-references to the resources of regulatory data found to be associated with the regulator (B).

The binding site motif section contains information about the regulator's motif. As described in figure 40, one can visualize the motif's consensus sequence, logo, PWM, binding sites, and aligned binding sites. Besides, it is possible to download the motif in the JASPAR and TRANScription FACtor database (TRANSFAC) format.

The regulator's network section allows browsing and searching the sub-TRN for effectors, genes, TFBS, and regulatory interactions. This section is similar to the organism's network section, containing several data tables listing the records associated with the regulator's sub-network.

The detailed gene view is similar to the regulator web page, as it contains detailed information about functional annotation, genomics, proteomics, sources, and organism. In addition, the gene web page lists all operons associated with the respective gene. As with the regulator web page, genomics and

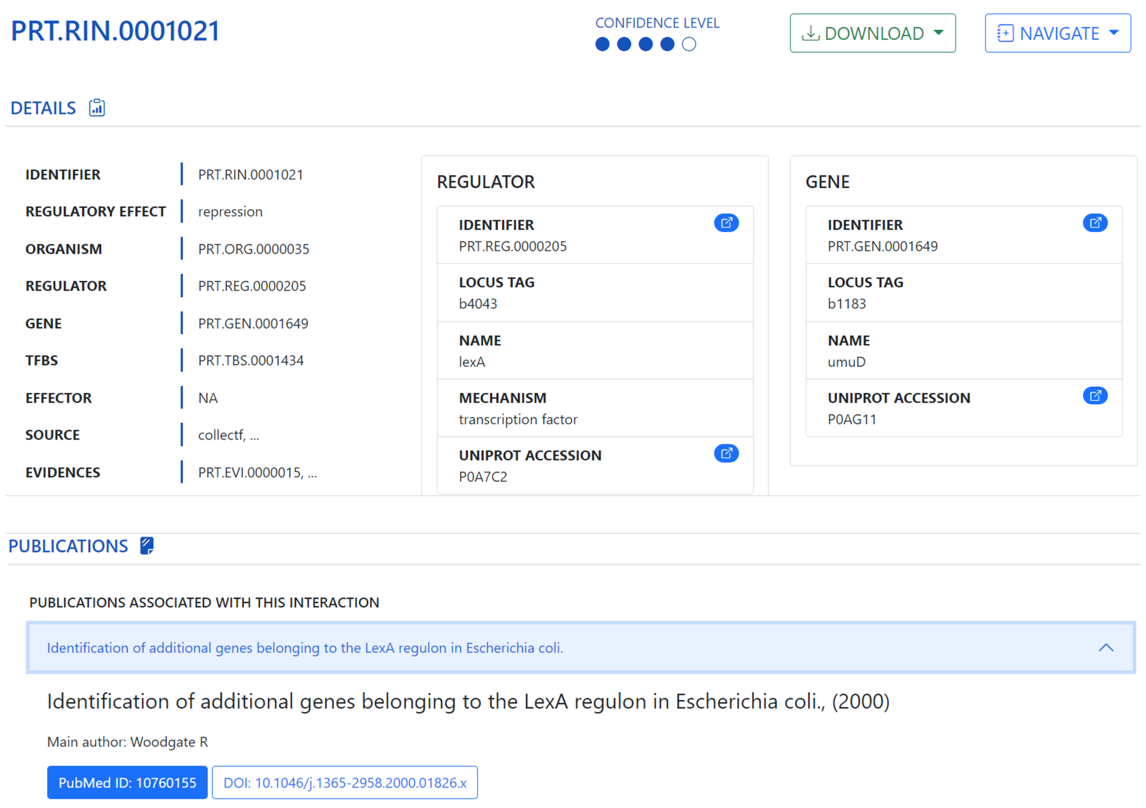


PRT.MOT.0000193
Motif
CONSENSUS SEQUENCE: --TACTGTTTATTTATACAGTA---
GC CONTENT: 0.16
BINDING SEQUENCES: 133
DOWNLOAD MOTIF:

Figure 40: Example of the information available for a regulator's motif in ProTReND. The detailed web page of a regulator (e.g., *b4043* in *E. coli*) contains a binding site motif section comprising relevant information, such as consensus sequence, logo, PWM, binding sites, and aligned binding sites.

proteomics data can be downloaded in FASTA and GenBank file formats or consulted in NCBI's and UniProt's databases.

Users can access regulatory interactions by browsing either organisms' or regulators' web pages. Nevertheless, one can also access the details of a given regulatory interaction by opening the detailed interaction view. For example, figure 41 highlights the regulatory interaction *PRT.RIN.0001021* between regulator *b4043* and gene *b1183*. Additionally, if available, a regulatory interaction web page shows effector and TFBS data. Finally, publications associated with the regulatory interaction can be accessed at the journal's website or PubMed using the PMID or DOI.


PRT.RIN.0001021

IDENTIFIER	PRT.RIN.0001021
REGULATORY EFFECT	repression
ORGANISM	PRT.ORG.0000035
REGULATOR	PRT.REG.0000205
GENE	PRT.GEN.0001649
TFBS	PRT.TBS.0001434
EFFECTOR	NA
SOURCE	collectf, ...
EVIDENCES	PRT.EVI.0000015, ...

REGULATOR

IDENTIFIER
PRT.REG.0000205

LOCUS TAG
b4043

NAME
lexA

MECHANISM
transcription factor

UNIPROT ACCESSION
P0A7C2

GENE

IDENTIFIER
PRT.GEN.0001649

LOCUS TAG
b1183

NAME
umuD

UNIPROT ACCESSION
P0AG11

PUBLICATIONS ASSOCIATED WITH THIS INTERACTION

Identification of additional genes belonging to the LexA regulon in Escherichia coli.

Identification of additional genes belonging to the LexA regulon in Escherichia coli., (2000)

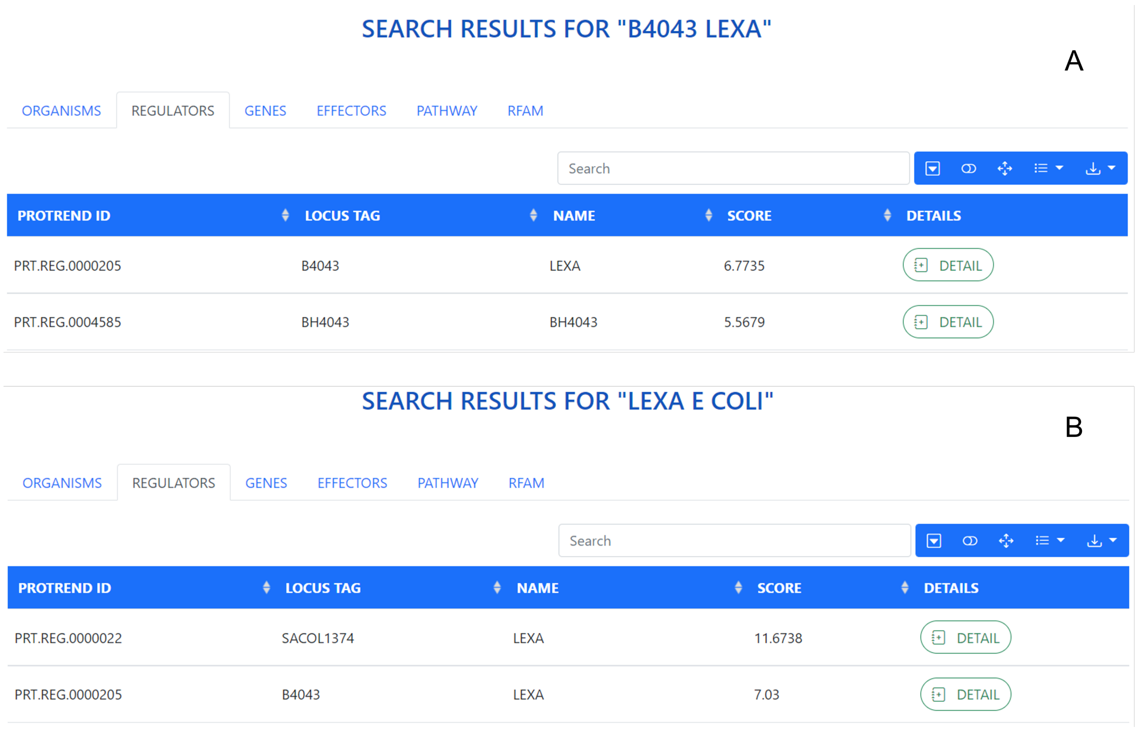
Main author: Woodgate R

PubMed ID: 10760155 DOI: 10.1046/j.1365-2958.2000.01826.x

Figure 41: Example of the information available for a regulatory interaction in ProTReND. The detailed web page of a regulatory interaction (e.g., *PRT.RIN.0001021*) highlights regulator and gene data together with the list of relevant publications.

ProTReND also provides the confidence level for organisms, regulators, genes, and regulatory interactions. The detailed web page of each record contains a ranked scale that varies from ”no evidence” to ”manual revision” levels. The confidence level indicates the status of the information associated with the record. For instance, figure 41 indicates that regulatory interaction *PRT.RIN.0001021* has been confirmed in the literature but has not yet been reviewed by our team or contributor.

The search engine available in ProTReND web application can be used to find organisms, regulators, genes, effectors, pathways, and regulatory families. The search engine allows simple queries, such as ProTReND identifiers, locus tags, names, and accessions (UniProt and NCBI). Nevertheless, the search engine also supports complex queries. For instance, one can search the combination of multiple attributes of the same entity, such as the search of regulator *PRT.REG.0000205* using the locus tag *b4043* and name *lexa* (Figure 42 (A)). Likewise, the search engine also supports queries that combine multiple attributes of different entities, such as the search for regulator *PRT.REG.0000205* using regulator name *lexa* and organism name *E. coli* (Figure 42 (B)). The search engine accepts vague terms allowing misspellings and similar terms. As vague term queries often yield many results, the search results are compiled into data tables according to the returned records and ordered by the search score. The Whoosh package calculates the score according to the object’s relevance to the search query.



SEARCH RESULTS FOR "B4043 LEXA"

PROTREND ID	LOCUS TAG	NAME	SCORE	DETAILS
PRT.REG.0000205	B4043	LEXA	6.7735	DETAIL
PRT.REG.0004585	BH4043	BH4043	5.5679	DETAIL

SEARCH RESULTS FOR "LEXA E COLI"

PROTREND ID	LOCUS TAG	NAME	SCORE	DETAILS
PRT.REG.0000022	SACOL1374	LEXA	11.6738	DETAIL
PRT.REG.0000205	B4043	LEXA	7.03	DETAIL

Figure 42: Example of the search engine in ProTReND. Example of the search results obtained for a simple (A) (e.g., regulator locus tag *b4043* and name *lexa*) and complex (B) (e.g., regulator name *lexa* and organism name *E. coli*) queries.

4.4.4 Data access

All regulatory data included in ProTReND can be downloaded on the web pages mentioned above or in the browsable API. The web pages of a given organism, regulator, gene, and interaction contain direct links to manually download the object in the JSON, XML, CSV and XLSX format. For instance, one can download the TRN of a given organism by accessing the organism's web page. Additionally, the download button can redirect users to the record information in the browsable API. Genomics information can be downloaded manually from regulator and gene pages in the FASTA and GenBank format files. Likewise, the motif associated with a given regulator can be downloaded manually from the regulator's page in the JASPAR and TRANSFAC formats.

Tables containing all organisms and regulators can also be manually downloaded in the JSON, XML, CSV and XLSX formats. However, the exported files will only contain the information presented in each data table. Likewise, tables comprising, for instance, organism networks, regulators sub-networks, or results of the search engine can be downloaded in the same formats.

The RESTful API available at protrend.bio.di.uminho.pt/api is an important feature of ProTReND's web application. This tool allows users to obtain ProTReND's data by performing programmed requests. The web API is divided into list and detail views for each entity. Each view supports the following application formats: JSON; XML; CSV; XLSX. Both Swagger and redoc documentation is available for ProTReND's API. One can access the examples of the API documentation to optimize the programmatic access to the database. Finally, a browsable API is also available at ProTReND, so one can visualize the format of the list and detail views.

4.4.5 Contributing to ProTReND

ProTReND community is a single-page application allowing users to contribute to ProTReND's database. To use ProTReND community, one must first register as a user. The user management system allows registering and logging users based on e-mail authentication. Registered users can use the community's user-friendly interface to perform one of the following contributions:

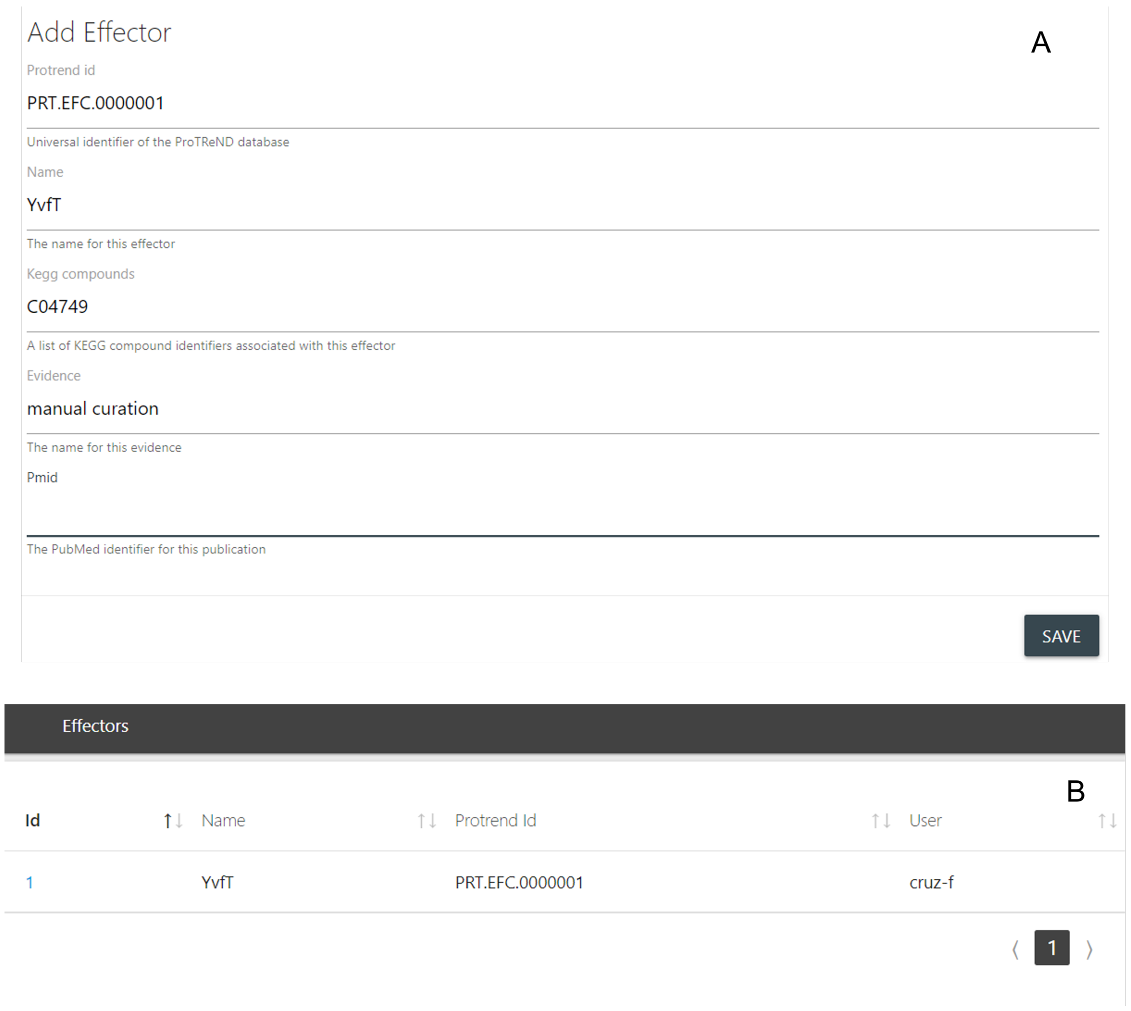
- Suggest new objects to ProTReND database;
- Suggest updates to existing objects in the ProTReND database;
- Suggest the removal of incorrect objects in the ProTReND database;
- Access the status of a given object in the ProTReND database.

Users' contributions are not directly added to ProTReND's database. The community application is designed asynchronously. Thus, the current state of the CDS does not yet include any user contribution. Alternatively, the community application is considered another resource of regulatory data that can be integrated into the CDS using the data integration system mentioned above. Thus, the community application is based on a new relational database implemented with the MySQL DBMS. The following tables/entities are available in ProTReND community:

- Effector
- Gene
- Interaction
- Organism
- Regulator
- TFBS
- Community user

All CRUD operations executed by a user are only stored in the community's relational database. Then, these contributions must be manually revised by our team and integrated into the CDS.

Users can contribute with new effectors, genes, interactions, organisms, regulators, and TFBS. The community interface contains customized forms having specific fields and validation according to the entity (Figure 43 (A)). One should fill the object form with relevant information and save the changes to submit a contribution. Once a contribution is made, it is listed in the corresponding data table of the community interface (Figure 43 (B)). One can create a new interaction to establish relationships among new objects in the community interface.



Add Effector

Protrend id	PRT.EFC.0000001
	Universal identifier of the ProTReND database
Name	YvfT
	The name for this effector
Kegg compounds	C04749
	A list of KEGG compound identifiers associated with this effector
Evidence	manual curation
	The name for this evidence
Pmid	
	The PubMed identifier for this publication

Effectors

Id	Name	Protrend Id	User
1	YvfT	PRT.EFC.0000001	cruz-f

Figure 43: Example of ProTReND community application. Example of the effector form that can be used to create a new effector object (A). Example of all effectors added to the ProTReND community application (B).

5

Definition and analysis of integrated metabolic-regulatory models

The work presented in this chapter was partially published in the following publication:

- *Pereira, V., Cruz, F., Rocha, M. (2021). MEWpy: a computational strain optimization workbench in Python, Bioinformatics, Volume 37, Issue 16, Pages 2494–2496.*

5.1 Introduction

The analysis of integrated genome-scale models of metabolism and regulation has been growing in recent years. Methods based on the switch approach, namely rFBA [144] and SR-FBA [145], limit the flux space of solutions with discrete constraints inferred from the state of metabolic and regulatory genes. On the other hand, valve-based techniques, namely PROM [146], IDREAM [150], CoRegFlux [148], and TRFBA [149], are not so deterministic, putting forward continuous constraints derived from gene expression data.

Despite the growth of methods to analyze integrated models, most tools are not easy to use out of their scope. These tools tend not to scale well with other case studies, apart from the ones in the original publications. TIGER [152] and FlexFlux [151] toolboxes ease the utilization of modeling techniques such as the mentioned above. Nevertheless, TIGER is available in the MATLAB proprietary language, while FlexFlux is a command-line-based tool with a simple GUI offering limited capabilities. In the latter, model inspection, analysis, and simulation are limited to a few command-line functions. OptFlux [217], a user-friendly tool for constraint-based modeling, allows analyzing integrated models using rFBA or SR-FBA. Still, it lacks the most recent methods based on the valve approach.

With this in mind, we have implemented several tools to define, simulate, analyze and optimize integrated models of metabolism and regulation in MEWpy. MEWpy is a metabolic engineering workbench implemented in the Python programming language. This package offers several methods to perform phenotype simulation and strain optimization using constraint-based models, as well as more recent paradigms. In addition, MEWpy allows the prediction of phenotypes using metabolic and regulatory-derived constraints, a feature added in this work.

5.2 MEWpy: a strain optimization workbench in Python

MEWpy offers a metabolic modeling framework implemented in Python. This framework includes phenotype simulation and strain optimization algorithms based on GECKO models [218], OptORF [219], and OptRAM [185], circumventing some limitations of other open-source Computational Strain Optimization (CSO) frameworks, such as OptFlux [217] and CAMEO [220]. The architecture of MEWpy comprehends several components, namely model definition, problem representation, phenotype prediction, and optimization, as described in figure 44.

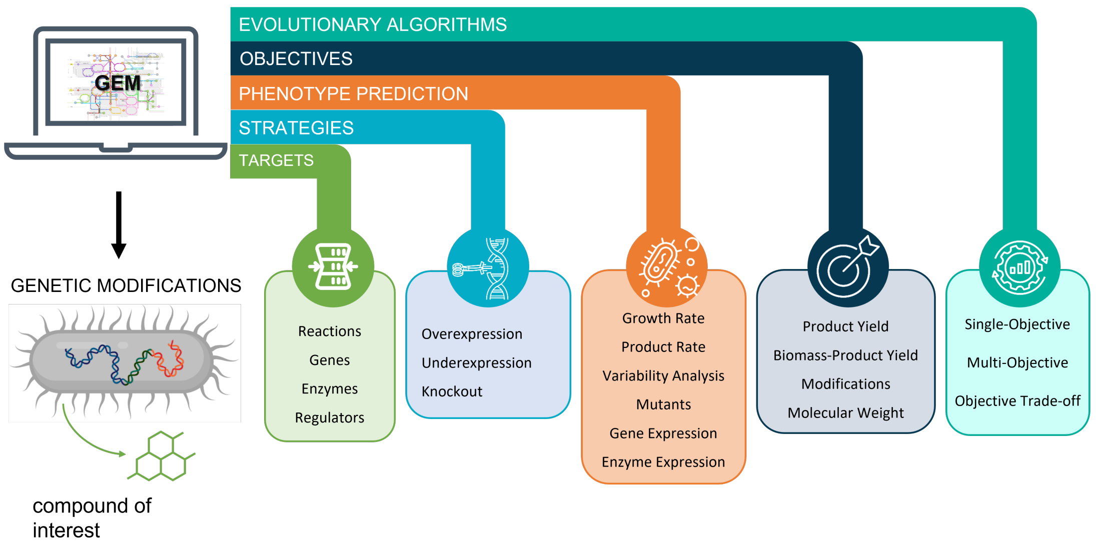


Figure 44: Overview of MEWpy framework. The MEWpy framework comprehends several components, such as problem representation (targets and strategies), phenotype prediction, and problem optimization (objectives and EAs). The evolutionary computation engine is responsible for suggesting modifications to the targets using a given strategy. These modifications should favor a desired metabolic engineering goal (e.g., compound production) encoded in the objective functions.

MEWpy's problem representation comprehends the following items:

- constraint-based modeling framework (metabolic, enzymes, or regulatory constraints);
- modification targets (reactions, genes, enzymes, or regulators);
- modification strategies (deletion or over/under expression).

MEWpy supports COBRApy [130] and Reframed (https://github.com/cdanielmachado/reframed) constraint-based modeling frameworks for model representation and phenotype simulation. These frameworks comprehend useful representations of genes, metabolites, and reactions available in a GEM model. Furthermore, MEWpy uses the phenotype prediction methods available in COBRApy and Reframed to evaluate different phenotypes resulting from over-expression, under-expression, and deletion of genes and reactions. In detail, MEWpy can convert gene modifications into additional flux constraints using the GPR rules available in the model. Regarding reaction modifications, flux constraints are added directly to the problem by altering reactions' bounds.

In addition to the constraint-based modeling frameworks, MEWpy supports the representation of enzymes using GECKO [218] and sMOMENT [221] models. MEWpy can convert enzymatic expression data into additional constraints leading to accurate phenotype predictions. Thus, one can also perform strain optimization based on enzyme modifications, namely over-expression, under-expression, and deletion of enzymes.

Lastly, the integration of gene regulation in GEM models was seamlessly implemented in MEWpy's ecosystem, in the context of this thesis. The GERM modeling framework comprehends several tools to represent and simulate regulatory and metabolic models. Also, one can perform strain optimization based on regulatory constraints. In detail, MEWpy includes the OptORF strategy that allows the deletion of regulators. Alternatively, OptRAM's implementation considers the over-expression and under-expression of regulators.

The phenotype prediction component includes several methods to predict the behavior of a given strain in several conditions. For example, MEWpy can predict mutant phenotypes encompassing several modifications described above. The different phenotype prediction methods are seamlessly provided by COBRApy or Reframed. Users can use a unified interface to perform the following simulation methods: FBA, pFBA, Minimization of Metabolic Adjustment (MOMA), linear Minimization of Metabolic Adjustment (lMOMA), Regulatory on/off Minimization of Metabolic flux changes (ROOM), and FVA. Alternatively, one can use the *mewpy.germ* module described in this chapter, for phenotype prediction based on integrated metabolic-regulatory models, as it is fully integrated with the phenotype prediction interface.

The optimization component includes methods based on EAs to find modifications favoring a given phenotype. In detail, an EA iteratively generates and combines modifications (population of solutions) using one of the abovementioned strategies. At each generation, the EA selects solutions (sets of modifications) with higher probabilities associated with the fittest candidates and recombines those to create new ones. Solutions are assessed using one or more optimization objectives designed for the metabolic engineering goal. Notably, MEWpy can use Multi-objective Evolutionary Algorithm (MOEA)s to solve strain engineering problems composed of multiple optimization objectives, such as weighted yield, Biomass-Product Coupled Yield (BPCY), product yield, and the number of modifications. MOEAs can attain several modifications favoring the trade-off between different objectives in a single run. MEWpy provides a simple EA interface to handle single (e.g., Genetic Algorithm (GA) and Simulated Annealing (SA)) and multiple objective problems (e.g., Strength Pareto Evolutionary Algorithm (SPEA2), Non-sorting Genetic Algorithm (NSGA), and Hypervolume Estimation algorithm (HypE)) via the Inspyred [222] and JMetalPy [223] engines.

5.3 A framework for integrated models in MEWpy

5.3.1 Overview of the genome-scale regulatory-metabolic model framework

The *mewpy.germ* module supports the definition, simulation, and analysis of integrated models of regulation and metabolism, named GERM models hereafter. For example, the *mewpy.germ* module includes phenotype prediction methods, such as rFBA, SR-FBA, PROM, and CoRegFlux. Figure 45 provides an overview of the *mewpy.germ* module.

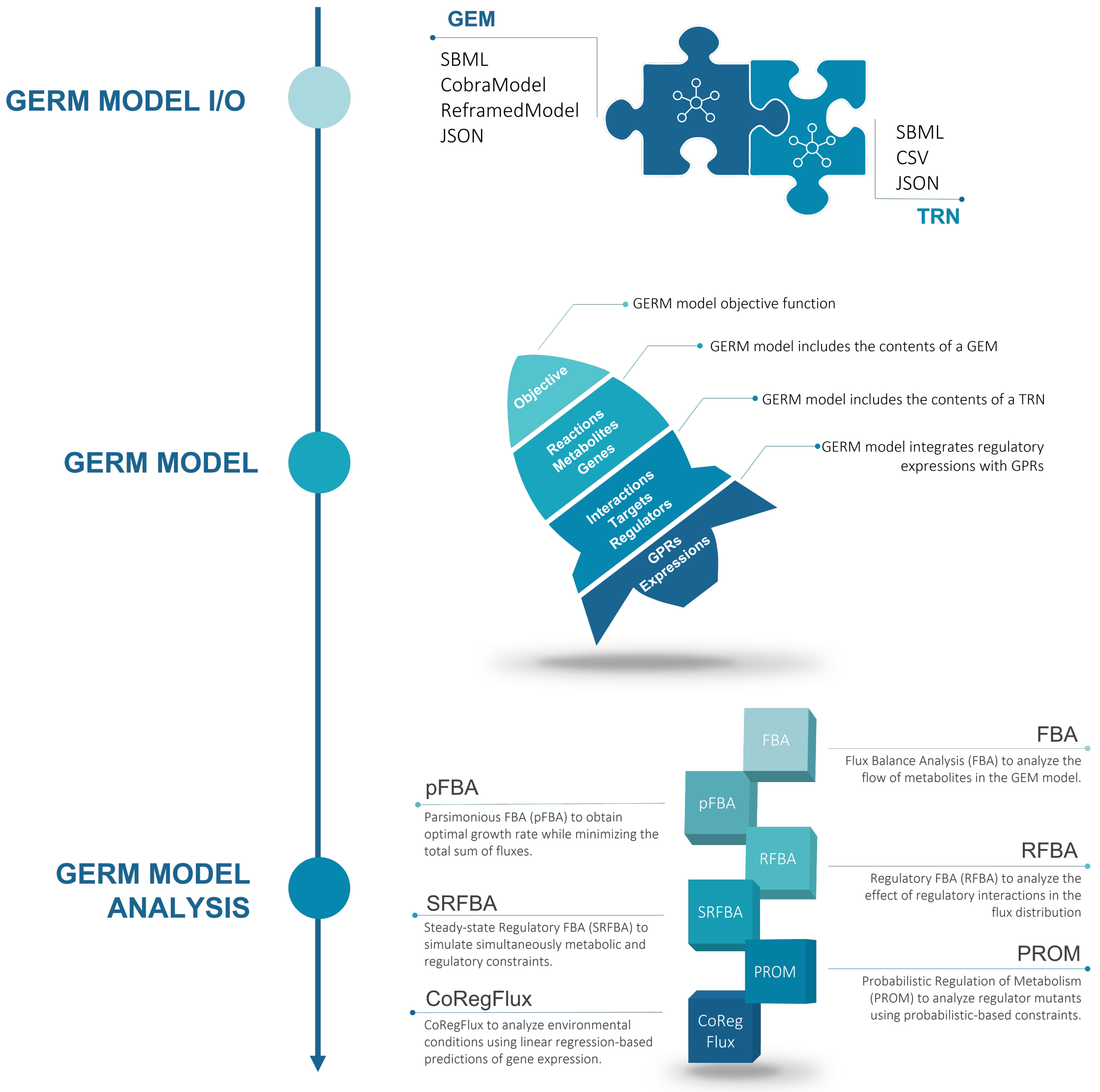

GERM MODEL I/O
GEM
SBML
CobraModel
ReframedModel
JSON
SBML
CSV
JSON
TRN
GERM MODEL
GERM model objective function
GERM model includes the contents of a GEM
GERM model includes the contents of a TRN
GERM model integrates regulatory expressions with GPRs
Objective
Reactions Metabolites Genes
Interactions Targets Regulators
GPRs Expressions
GERM MODEL ANALYSIS
FBA
Flux Balance Analysis (FBA) to analyze the flow of metabolites in the GEM model.
pFBA
Parsimonious FBA (pFBA) to obtain optimal growth rate while minimizing the total sum of fluxes.
RFBA
Regulatory FBA (RFBA) to analyze the effect of regulatory interactions in the flux distribution
SRFBA
Steady-state Regulatory FBA (SRFBA) to simulate simultaneously metabolic and regulatory constraints.
PROM
Probabilistic Regulation of Metabolism (PROM) to analyze regulator mutants using probabilistic-based constraints.
CoRegFlux
CoRegFlux to analyze environmental conditions using linear regression-based predictions of gene expression.
CoReg Flux

Figure 45: Overview of the GERM model framework in MEWpy. Users can load GERM models from separate files, such as SBML and CSV. The model input/output tools can generate a GERM model containing metabolic and regulatory contents. The analysis of GERM models is based on several phenotype predictions methods, such as FBA, pFBA, rFBA, SR-FBA, PROM, and CoRegFlux.

A GERM model can represent the integration of a TRN into a GEM model. Accordingly, the GERM model comprehends metabolic variables, such as reactions, metabolites, and genes, as well as regulatory variables, such as interactions, targets, and regulators. Furthermore, the GERM model integrates GPR rules in the GEM model with Boolean algebra expressions defining regulatory interactions in TRNs.

The *mewpy.germ* input/output tools can read GEM models available in the SBML file format and TRNs in the SBML and CSV file formats. In addition, one can load a GERM model using JSON and other structures.

The following methods are available in the *mewpy.germ* module to predict the phenotypic behavior of an organism in a wide range of metabolic, regulatory, and genetic conditions:

- FBA - Computes a flux distribution that achieves the optimal growth rate in metabolic models;
- pFBA - Computes a flux distribution that achieves the optimal growth rate while minimizing the total sum of fluxes;
- rFBA - Computes a flux distribution that achieves the optimal growth rate though limited by the effect of regulatory interactions in the flux distribution;
- SR-FBA - Computes a flux and regulatory distribution simultaneously to attain the optimal growth rate;
- PROM - Analysis of regulatory mutants using a continuous approach to limit the flux distribution;
- CoRegFlux - Computes a flux distribution that achieves the optimal growth rate based on a continuous approach to limit the flux of reactions using gene expression.

5.3.2 Genome-scale regulatory-metabolic model

The *mewpy.germ* module implements a framework that supports inspecting and manipulating metabolic and regulatory models. On the one hand, this framework includes a set of structures designed to represent the main features of a GEM model. On the other hand, the *mewpy.germ* module provides a simple interface to work with TRNs. In detail, the *Model* ecosystem is the main interface to access and manipulate integrated models. Custom Python objects represent the variables available in the TRN and GEM model that live together in the *Model* class (*mewpy.germ.models*). Figure 46 shows the software architecture responsible for representing integrated models of metabolism and regulation.

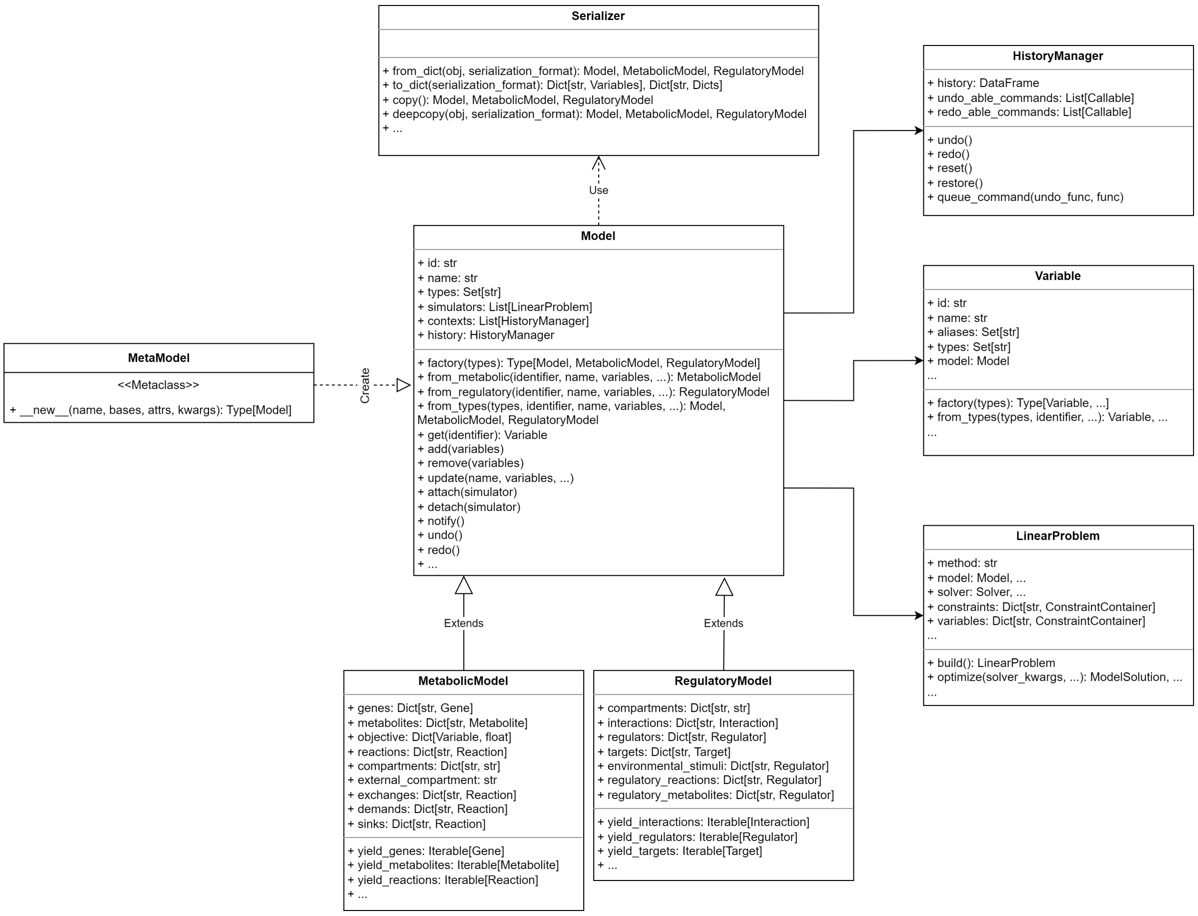

Serializer
+ from_dict(obj, serialization_format): Model, MetabolicModel, RegulatoryModel
+ to_dict(serialization_format): Dict[str, Variables], Dict[str, Dicts]
+ copy(): Model, MetabolicModel, RegulatoryModel
+ deepcopy(obj, serialization_format): Model, MetabolicModel, RegulatoryModel
+ ...
Use
HistoryManager
+ history: DataFrame
+ undo_able_commands: List[Callable]
+ redo_able_commands: List[Callable]
+ undo()
+ redo()
+ reset()
+ restore()
+ queue_command(undo_func, func)
Model
+ id: str
+ name: str
+ types: Set[str]
+ simulators: List[LinearProblem]
+ contexts: List[HistoryManager]
+ history: HistoryManager
+ factory(types): Type[Model, MetabolicModel, RegulatoryModel]
+ from_metabolic(identifier, name, variables, ...): MetabolicModel
+ from_regulatory(identifier, name, variables, ...): RegulatoryModel
+ from_types(types, identifier, name, variables, ...): Model, MetabolicModel, RegulatoryModel
+ get(identifier): Variable
+ add(variables)
+ remove(variables)
+ update(name, variables, ...)
+ attach(simulator)
+ detach(simulator)
+ notify()
+ undo()
+ redo()
+ ...
Variable
+ id: str
+ name: str
+ aliases: Set[str]
+ types: Set[str]
+ model: Model
...
+ factory(types): Type[Variable, ...]
+ from_types(types, identifier, ...): Variable, ...
...
MetaModel
<<Metaclass>>
+ __new__(name, bases, attrs, kwargs): Type[Model]
Create
LinearProblem
+ method: str
+ model: Model, ...
+ solver: Solver, ...
+ constraints: Dict[str, ConstraintContainer]
+ variables: Dict[str, ConstraintContainer]
...
+ build(): LinearProblem
+ optimize(solver_kwargs, ...): ModelSolution, ...
...
Extends
Extends
MetabolicModel
+ genes: Dict[str, Gene]
+ metabolites: Dict[str, Metabolite]
+ objective: Dict[Variable, float]
+ reactions: Dict[str, Reaction]
+ compartments: Dict[str, str]
+ external_compartment: str
+ exchanges: Dict[str, Reaction]
+ demands: Dict[str, Reaction]
+ sinks: Dict[str, Reaction]
+ yield_genes: Iterable[Gene]
+ yield_metabolites: Iterable[Metabolite]
+ yield_reactions: Iterable[Reaction]
+ ...
RegulatoryModel
+ compartments: Dict[str, str]
+ interactions: Dict[str, Interaction]
+ regulators: Dict[str, Regulator]
+ targets: Dict[str, Target]
+ environmental_stimuli: Dict[str, Regulator]
+ regulatory_reactions: Dict[str, Regulator]
+ regulatory_metabolites: Dict[str, Regulator]
+ yield_interactions: Iterable[Interaction]
+ yield_regulators: Iterable[Regulator]
+ yield_targets: Iterable[Target]
+ ...

Figure 46: Architecture of a GERM model. The representation of *Model*, *MetabolicModel* and *RegulatoryModel* using UML. In addition, a *Model* object can be associated with an *HistoryManager*, a set of *Variable*s, and *LinearProblem*s.

The *Model* class defines simple and generic attributes, such as *id* and *name*, but misses most of the features available in TRNs and GEM models. In turn, *MetabolicModel* and *RegulatoryModel* classes extend the *Model* parent class to implement specific representations of metabolic and regulatory objects, respectively.

A *Model* object follows a dictionary-like interface that implements the *get*, *add*, *remove*, and *update* methods. In detail, the *add* method can store variables in the model containers, namely dictionary-like attributes indexing variables objects by their identifiers (dictionary keys). For example, a *MetabolicModel* instance contains variables of type *Gene* in the genes' container. The *remove* method is responsible for removing variables from the *Model* containers. Lastly, the *get* method retrieves the variable object using its identifier. *Model* containers implement Python's *property* descriptor protocol, so getting and setting routines can use the abovementioned methods.

The *Serializer* class implements serialization routines for the *Model* framework. All model objects can be serialized and deserialized into dictionaries, JSON, and pickle files. The *Serializer* is a mix-in class that both *Model* and *Variable* can use. The *Serializer* class converts relevant *Model* attributes to native-serializable Python objects.

Model instances can be associated with several phenotype prediction methods in the *simulators*' attribute. In detail, the synergy model-simulator follows the *observer* pattern. The *attach* method links *LinearProblem* objects to the model instance and the *notify* method warns simulators about the latest model changes. That is, multiple *LinearProblem* instances subscribe to a single *Model* instance. Then, the *Model* instance is responsible for notifying the subscribers about the latest news. In this observer pattern, *LinearProblem* objects must update variables and constraints according to the latest state of the *Model* instance.

Model instances support reversible and temporary changes using the *HistoryManager* class. This class implements a command pattern to queue redoable and undoable instructions about the operations performed in a single *Model* instance. The *undo*, *redo*, *reset* and *restore* methods execute the queued commands according to the related resolution order.

The *Model* factory pattern can generate dynamic model types. For example, the *factory* constructor can create an integrated *MetabolicRegulatoryModel* class derived from the *MetabolicModel* and *RegulatoryModel* classes. Accordingly, one can dynamically create an object of the *MetabolicRegulatoryModel* class using the *from_types* polymorphic constructor.

The *MetabolicModel* class extends the parent *Model* class to implement the following attributes:

- *genes*: The *genes*' container stores variables of type gene;
- *metabolites*: The *metabolites*' container stores variables of type metabolite;
- *reactions*: The *reactions*' container stores variables of type reaction;
- *objective*: The *objective*'s container stores variables that define the objective function. This container keeps variables as keys and coefficients as the corresponding values;
- *gprs*: The *GPRs*' container stores the Boolean algebra expressions associated with reactions;
- *compartments*: The *compartments*' container stores model compartments;
- *external_compartment*: The *external_compartment* attribute consists of the compartment most well represented in the boundary reactions (unbalanced reactions having reactants or products);
- *exchanges*: The *exchanges*' container stores variables of type reaction. Exchange reactions are boundary reactions (unbalanced reactions having reactants or products) associated with the external compartment;
- *demands*: The *demands*' container stores variables of type reaction. Demand reactions are irreversible boundary reactions not associated with the external compartment;

- *sinks*: The *sinks*' container stores variables of type reaction. Sink reactions are reversible boundary reactions not associated with the external compartment.

The *RegulatoryModel* class extends the parent *Model* class to implement the following attributes:

- *interactions*: The *interactions*' container stores variables of type interaction;
- *targets*: The *targets*' container stores variables of type target;
- *regulators*: The *regulators*' container stores variables of the type regulator;
- *environmental_stimuli*: The *environmental_stimuli*' container stores variables of type regulator; Environmental stimuli (also named effectors) are regulators not regulated by other regulators;
- *compartments*: The *compartments*' container stores model compartments;
- *regulatory_reactions*: The *regulatory_reactions*' container stores variables of type regulator and reaction;
- *regulatory_metabolites*: The *regulatory_metabolites*' container stores variables of the type regulator and metabolite;

5.3.3 Variables of genome-scale regulatory-metabolic models

The *mewpy.germ.variables* sub-module includes several structures representing variables usually found in TRNs and GEM models. These structures provide a clear and straightforward interface to inspect and manipulate the main features of reactions, and interactions, among other variables. Figure 47 shows the software architecture used to represent metabolic and regulatory variables, namely genes, metabolites, reactions, interactions, targets, and regulators.

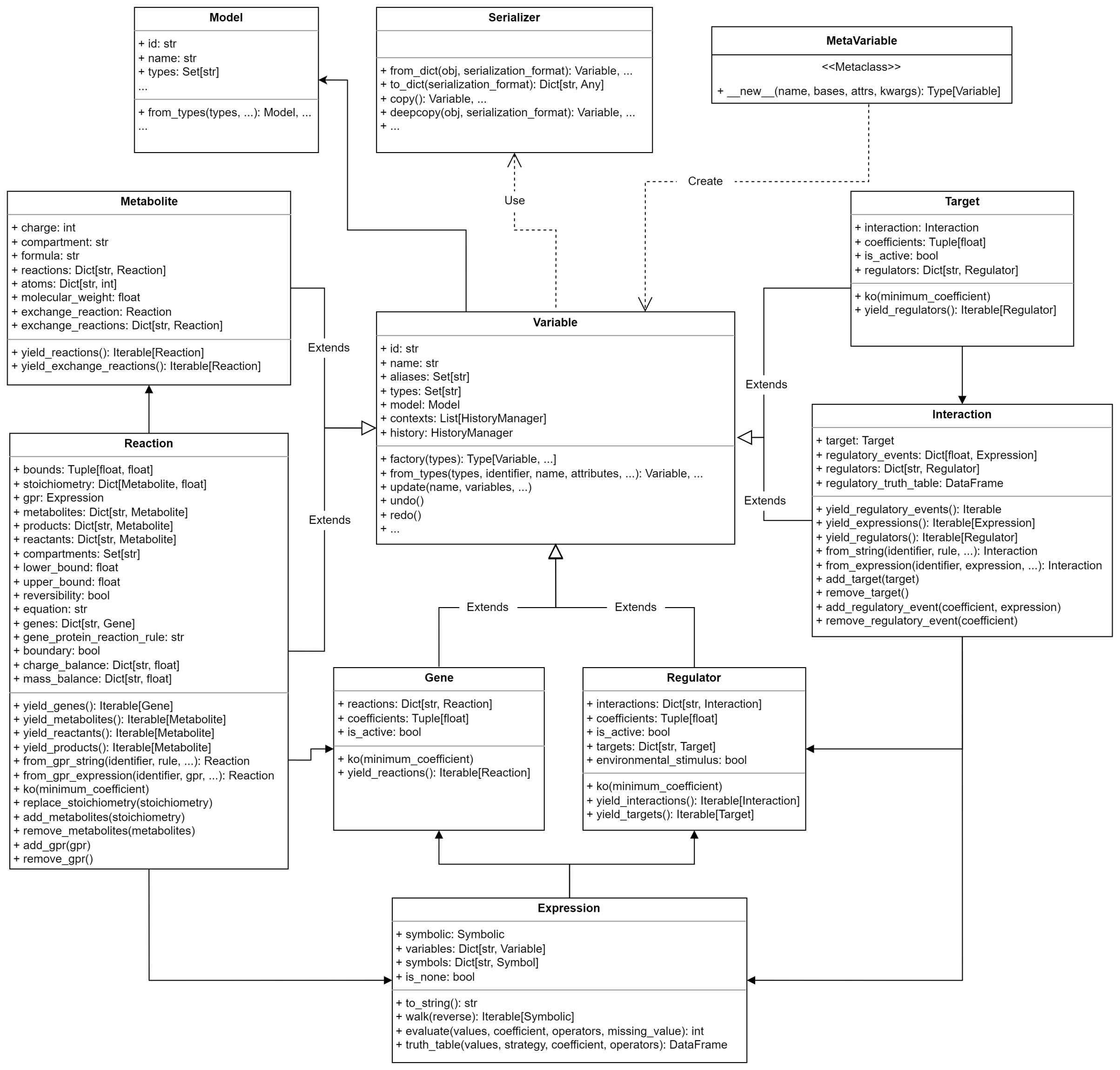

Model
+ id: str
+ name: str
+ types: Set[str]
...
+ from_types(types, ...): Model, ...
...
Serializer
+ from_dict(obj, serialization_format): Variable, ...
+ to_dict(serialization_format): Dict[str, Any]
+ copy(): Variable, ...
+ deepcopy(obj, serialization_format): Variable, ...
+ ...
MetaVariable
<<Metaclass>>
+ __new__(name, bases, attrs, kwargs): Type[Variable]
Create
Use
Metabolite
+ charge: int
+ compartment: str
+ formula: str
+ reactions: Dict[str, Reaction]
+ atoms: Dict[str, int]
+ molecular_weight: float
+ exchange_reaction: Reaction
+ exchange_reactions: Dict[str, Reaction]
+ yield_reactions(): Iterable[Reaction]
+ yield_exchange_reactions(): Iterable[Reaction]
Target
+ interaction: Interaction
+ coefficients: Tuple[float]
+ is_active: bool
+ regulators: Dict[str, Regulator]
+ ko(minimum_coefficient)
+ yield_regulators(): Iterable[Regulator]
Extends
Variable
+ id: str
+ name: str
+ aliases: Set[str]
+ types: Set[str]
+ model: Model
+ contexts: List[HistoryManager]
+ history: HistoryManager
+ factory(types): Type[Variable, ...]
+ from_types(types, identifier, name, attributes, ...): Variable, ...
+ update(name, variables, ...)
+ undo()
+ redo()
+ ...
Extends
Interaction
+ target: Target
+ regulatory_events: Dict[float, Expression]
+ regulators: Dict[str, Regulator]
+ regulatory_truth_table: DataFrame
+ yield_regulatory_events(): Iterable
+ yield_expressions(): Iterable[Expression]
+ yield_regulators(): Iterable[Regulator]
+ from_string(identifier, rule, ...): Interaction
+ from_expression(identifier, expression, ...): Interaction
+ add_target(target)
+ remove_target()
+ add_regulatory_event(coefficient, expression)
+ remove_regulatory_event(coefficient)
Reaction
+ bounds: Tuple[float, float]
+ stoichiometry: Dict[Metabolite, float]
+ gpr: Expression
+ metabolites: Dict[str, Metabolite]
+ products: Dict[str, Metabolite]
+ reactants: Dict[str, Metabolite]
+ compartments: Set[str]
+ lower_bound: float
+ upper_bound: float
+ reversibility: bool
+ equation: str
+ genes: Dict[str, Gene]
+ gene_protein_reaction_rule: str
+ boundary: bool
+ charge_balance: Dict[str, float]
+ mass_balance: Dict[str, float]
+ yield_genes(): Iterable[Gene]
+ yield_metabolites(): Iterable[Metabolite]
+ yield_reactants(): Iterable[Metabolite]
+ yield_products(): Iterable[Metabolite]
+ from_gpr_string(identifier, rule, ...): Reaction
+ from_gpr_expression(identifier, gpr, ...): Reaction
+ ko(minimum_coefficient)
+ replace_stoichiometry(stoichiometry)
+ add_metabolites(stoichiometry)
+ remove_metabolites(metabolites)
+ add_gpr(gpr)
+ remove_gpr()
Extends
Extends
Extends
Extends
Gene
+ reactions: Dict[str, Reaction]
+ coefficients: Tuple[float]
+ is_active: bool
+ ko(minimum_coefficient)
+ yield_reactions(): Iterable[Reaction]
Regulator
+ interactions: Dict[str, Interaction]
+ coefficients: Tuple[float]
+ is_active: bool
+ targets: Dict[str, Target]
+ environmental_stimulus: bool
+ ko(minimum_coefficient)
+ yield_interactions(): Iterable[Interaction]
+ yield_targets(): Iterable[Target]
Expression
+ symbolic: Symbolic
+ variables: Dict[str, Variable]
+ symbols: Dict[str, Symbol]
+ is_none: bool
+ to_string(): str
+ walk(reverse): Iterable[Symbolic]
+ evaluate(values, coefficient, operators, missing_value): int
+ truth_table(values, strategy, coefficient, operators): DataFrame

Figure 47: Architecture of the GERM model variables. The representation of *Variable*, *Metabolite*, *Reaction*, *Gene*, *Regulator*, *Interaction* and *Target* using UML. In addition, a *Variable* object is often associated with other variables, a *Model* object, and *Expressions*.

The *Variable* class contains generic attributes widely spread in metabolic and regulatory variables, namely *id*, *name*, and *aliases*. In addition, the *Variable* interface includes a generic *update* method to change the value of a given attribute. Variables can be associated with a single *Model* instance in the *model* attribute. Hence, the *Variable* class supports temporary and reversible changes using the model's *HistoryManager* implementation. As with the *Model* class, the *Variable* structure uses the serialization routines implemented in the *Serializer* mix-in.

Gene, *Target*, and *Regulator* sub-classes implement abstractions of genes, targets, and regulators available in TRNs and GEM models. These sub-classes are very similar, sharing standard features, such as *coefficients*, *is_active*, and *ko*. On the other hand, a gene is often associated with several reactions, while regulators are usually related to interactions and targets. Finally, a *Target* instance should be associated with a single regulatory interaction and related regulators.

The *Metabolite* sub-class represents standard features of chemical compounds, namely charge, formula, atoms, and molecular weight. Moreover, this sub-class contains the *compartment* attribute, as metabolites are always associated with a cellular compartment in GEM models. Finally, *Metabolite* instances are related to several reactions and sometimes exchange reactions.

The *Interaction* sub-class represents regulatory rules (Boolean algebra expressions) involving a single *Target* instance with many *Regulator* instances. One can use the *add_target* and *remove_target* interfaces to change the *Target* instance and the *add_regulatory_event* and *remove_regulatory_event* methods to change the regulatory rules.

In detail, the *regulatory_events* attribute implements a Python dictionary holding pairs of target coefficients and *Expression* instances. An *Expression* instance represents the Boolean algebra expression encoding the regulatory rule of a given target gene. The *mewpy.germ.algebra* sub-module can parse regulatory rules in the string format to *Symbolic* objects using the following operators: *AND* (&), *OR* (|), *Not* ( ), *Greater* (>), *GreaterEqual* (≥), *Less* (<), and *LessEqual* (≤). Regulatory operands, on the other hand, are converted into *Symbol* instances that can be associated with *Regulator* instances. Hence, *Expression* instances can be evaluated using the regulators' coefficients.

The *Reaction* class includes the main attributes of reactions in constraint-based modeling, namely bounds (tuple of lower and upper bounds), stoichiometry (metabolites-coefficient dictionary), and GPR rule (*Expression* object). In addition, *Reaction* instances have simple methods to add or remove *Metabolite* and *Expression* instances to the *stoichiometry* and *gpr* attributes, respectively. Finally, one can inspect many helpful attributes of a reaction object, such as *products*, *reactants*, *compartments*, *reversibility*, *equation*, *genes*, gene_protein_*reaction*_rule, *boundary*, *mass_balance*, and *charge_balance*.

The *Variable* factory pattern can be used to create dynamic variable types. For example, the integrated *TargetRegulator* class can be derived from the *Target* and *Regulator* classes using the *factory* method. Accordingly, users can use the *from_types*' polymorphic constructor to obtain integrated variables that include the abovementioned representations and methods. Integrated classes reduce the burden of maintaining duplicated objects in a single *Model* instance. Besides, the *Variable* factory pattern circumvents code duplication and the implementation of complex sub-classes using inheritance.

5.3.4 Input/output tools for genome-scale regulatory-metabolic models

The *mewpy.io* module comprises file parsers implemented to extract relevant information of TRNs and GEM models. GEM models are widely available in the SBML file format, while TRNs are often encoded into CSV files or SBML files using the SBML-qual plugin.

The software architecture implemented in the *mewpy.io* module follows the builder design pattern. This architecture's main advantage is modularity: a single parser can build a model from a TRN or GEM file, while multiple parsers can read an integrated model simultaneously. Figure 48 highlights the builder architecture, including the *Director* and *Builder* classes that can parse relevant information from different files to assemble parts of an integrated model.

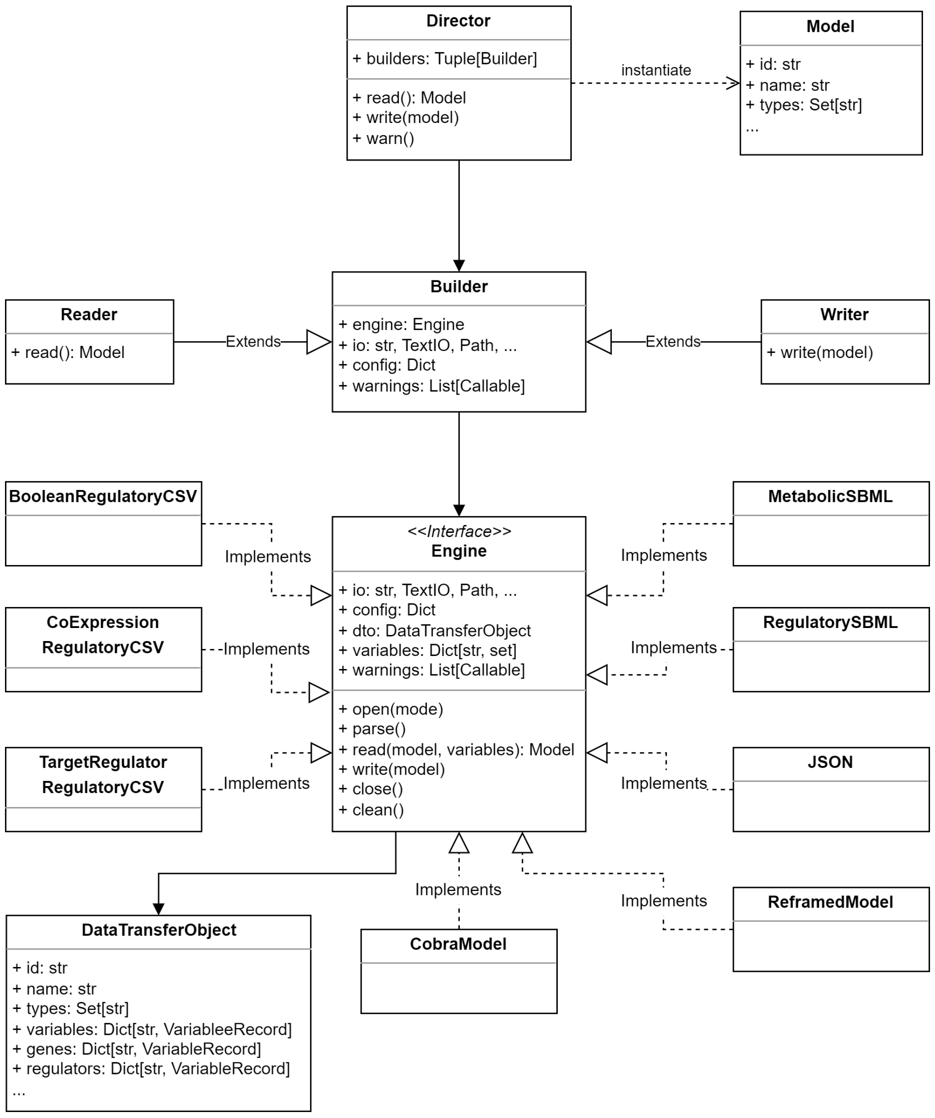


Figure 48: Architecture of *mewpy.io* tools. The representation of the builder pattern implemented in the *mewpy.io* module using UML. The *Director* class generates a *Model* instance by orchestrating several *Reader* objects. In turn, the *Reader* class encapsulates an *Engine* object that is responsible for reading/writing GERM model files (e.g., SBML, CSV, and JSON).

The *Director* class is initiated with one or more steps (*builders*) and defines how to execute them. On the other hand, *Builder* instances take the *engine*, *io*, and *config* attributes. In addition, the *Reader* and *Writer* sub-classes extend the base *Builder* representation to implement the *read* and *write* methods. Although the *read* and *write* methods could have been implemented in the *Builder* class directly, these sub-classes present a familiar interface to users.

The *Builder* is mainly a wrapper of an *Engine* instance. The *Engine* class is an interface (abstract) class having the following methods: *open*, *parse*, *read*, *write*, *close*, and *clean*. Hence, the *Engine* sub-classes must implement in these methods file-specific routines to parse and handle input/output resources. For example, the *MetabolicSBML* sub-class implements the tools to parse metabolic information of an SBML file into a *Model* instance. Alternatively, an *Engine* instance can also write metabolic or regulatory contents to a specific file type.

Each *Engine* instance comprehends two main steps, parsing and reading, that must be called in two different stages by the *Director* instance. Engines' *parse* method collects information into a *DataTransfer-Object* object that can hold raw data before model building takes place. This object is shared with other *Engine* instances to ease the integration of metabolic and regulatory information into the same *Model* instance. Variables having the same identifier across different files are stored in the same record within the *DataTransferObject* object. Variables' types are also saved in the *DataTransferObject*. After calling the *parse* method of all *Engine* instances, the *Director* calls the *read* method next. This method re-collects raw integrated data from the *DataTransferObject* object to build a specific part of the *Model* instance.

Table 7 summarizes all engines in the *mewpy.io* module according to the associated file format and extracted information.

Finally, the *mewpy.io* module includes several utility functions to read and write integrated models, such as the *read_model*, *read_csv*, *read_json*, *read_sbml*, and *write_model*. These functions provide a simple interface to read and write models.

5.3.5 Phenotype prediction using genome-scale regulatory-metabolic models

The analysis of integrated models of metabolism and regulation includes several phenotype prediction methods, available in the *mewpy.germ.analysis* sub-module. One can use constraint-based modeling approaches such as FBA and pFBA. More importantly, this sub-module includes methods to predict phenotypes using models resulting from the integration of TRNs into GEM models, using both discrete- (rFBA and SR-FBA) and continuous-based (PROM and CoRegFlux) methods. Figure 49 outlines the software architecture of the *mewpy.germ.analysis* sub-module.

Table 7: Engines available in the *mewpy.io* module. The GERM model input/output tools are described in terms of the file format and data type handled by the engine. In addition, a brief description of the engine implementation is also provided.

Engine	**File format**	**Data**	**Description**
BooleanRegulatoryCSV	CSV or TXT	regulatory interactions divided into two columns: target, Boolean algebra expression with regulators	parser for a TRN encoded into a CSV using pandas package and *mewpy.germ.algebra* module.
CoExpressionRegulatoryCSV	CSV or TXT	regulatory interactions divided into three columns: target, co-activators, co-repressors	parser for a TRN encoded into a CSV using pandas package and *mewpy.germ.algebra* module.
TargetRegulatorRegulatoryCSV	CSV or TXT	regulatory interactions divided into two columns: target and regulator	parser for a TRN encoded into a CSV using pandas package and *mewpy.germ.algebra* module.
MetabolicSBML	SBML or XML	genes, metabolites and reactions (w/ GPRs)	parser for a GEM model encoded into a SBML using python-libsbml package.
RegulatorySBML	SBML or XML	Targets, regulators, and regulatory interactions encoded into the SBML-qual plugin	parser for a TRN encoded into the SBML-qual plugin using python-libsbml package.
CobraModel	Python object	genes, metabolites and reactions (w/ GPRs)	COBRApy's *Model* object parser.
ReframedModel	Python object	genes, metabolites and reactions (w/ GPRs)	Reframed's *CBModel* object parser.
JSON	JSON	GERM model variables	JSON file created by *mewpy.germ.models* or *mewpy.germ.variables* modules.

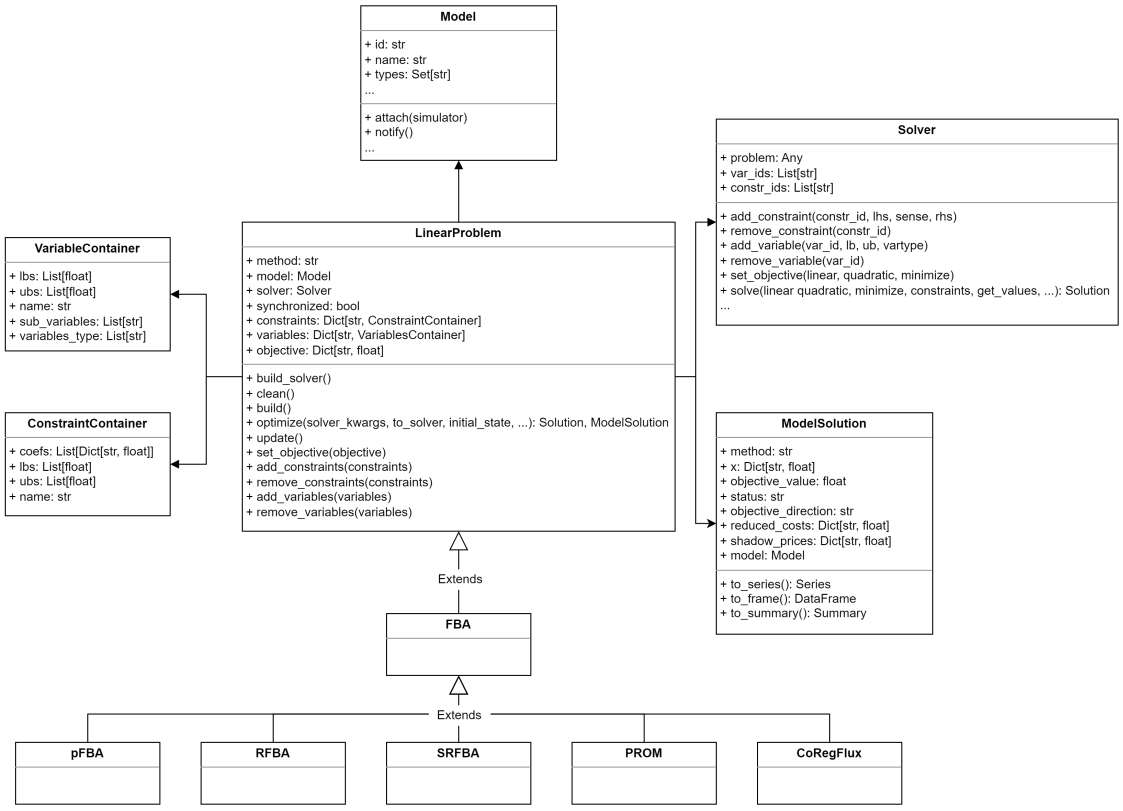

Model
+ id: str
+ name: str
+ types: Set[str]
...
+ attach(simulator)
+ notify()
...
Solver
+ problem: Any
+ var_ids: List[str]
+ constr_ids: List[str]
+ add_constraint(constr_id, lhs, sense, rhs)
+ remove_constraint(constr_id)
+ add_variable(var_id, lb, ub, vartype)
+ remove_variable(var_id)
+ set_objective(linear, quadratic, minimize)
+ solve(linear quadratic, minimize, constraints, get_values, ...): Solution
...
VariableContainer
+ lbs: List[float]
+ ubs: List[float]
+ name: str
+ sub_variables: List[str]
+ variables_type: List[str]
LinearProblem
+ method: str
+ model: Model
+ solver: Solver
+ synchronized: bool
+ constraints: Dict[str, ConstraintContainer]
+ variables: Dict[str, VariablesContainer]
+ objective: Dict[str, float]
+ build_solver()
+ clean()
+ build()
+ optimize(solver_kwargs, to_solver, initial_state, ...): Solution, ModelSolution
+ update()
+ set_objective(objective)
+ add_constraints(constraints)
+ remove_constraints(constraints)
+ add_variables(variables)
+ remove_variables(variables)
ConstraintContainer
+ coefs: List[Dict[str, float]]
+ lbs: List[float]
+ ubs: List[float]
+ name: str
ModelSolution
+ method: str
+ x: Dict[str, float]
+ objective_value: float
+ status: str
+ objective_direction: str
+ reduced_costs: Dict[str, float]
+ shadow_prices: Dict[str, float]
+ model: Model
+ to_series(): Series
+ to_frame(): DataFrame
+ to_summary(): Summary
Extends
FBA
Extends
pFBA
RFBA
SRFBA
PROM
CoRegFlux

Figure 49: Architecture of the GERM model analysis tools using UML. *LinearProblem* is the base class for all phenotype prediction methods implemented in the *mewpy.germ.analysis* sub-module. This class represents a LP containing linear variables and constraints inferred from a *Model* object. All phenotype prediction methods are associated with a *Solver* instance and generate a *ModelSolution* upon optimization.

The *LinearProblem* class is the base representation for linear programming and MILP problems, including the *variables*, *constraints*, and *objective* attributes. Regarding the *LinearProblem* interface, the *LinearProblem* class includes the *add_constraints*, *remove_constraints*, *add_variables*, *remove_variables*, and *set_objective* methods to operate linear variables and constraints.

Linear variables and constraints are represented in the *VariableContainer* and *ConstraintContainer* classes, respectively. These container-like classes can group mid-term variables and constraints generated by some phenotype prediction methods. For instance, SR-FBA creates mid-term variables and constraints for operators in Boolean algebra expressions. The mid-term variables and constraints are stored in the same *VariableContainer* and *ConstraintContainer* instances, easing their traceability and operation.

The *LinearProblem* class retrieves linear variables and constraints from a *Model* object. Therefore, sub-classes of *LinearProblem* must implement the *build* method responsible for converting model variables into linear constraints and variables. Then, the *build_solver* method populates a *Solver* (*mewpy.solvers*) instance with the resulting objective, variables, and constraints. This method uses the *Solver* class interface to create a linear problem (*problem*) in one of the solvers supported by MEWpy (CPLEX, GUROBI, and OptLang).

After building the linear problem, one can use the *optimize* method to solve the problem. Internally, the optimization is performed using the *solve* method of the *Solver* class. Hence, the signature of the *optimize* method includes a dictionary of solver-specific arguments, such as the *objective*, *constraints*, *get_values*, among others. For example, one can temporarily change the objective function or bounds using the solver-specific arguments. Seldom, phenotype prediction methods implement different versions of the *optimize* method having additional arguments, such as the *initial_state*. The optimization returns a *ModelSolution* instance, including the status, objective, and solution values. Furthermore, the *ModelSolution* object provides user-friendly methods (e.g., *to_frame*, *to_series*, and *to_summary*) to visualize the solution.

LinearProblem instances are asynchronous by default, meaning constraints are not updated each time the model changes. Therefore, to synchronize a *LinearProblem*, one must re-run the *build* method to retrieve the latest state. On the other hand, a *Model* instance can notify *LinearProblem* instances about the latest changes using the observer pattern. In this case, variables and constraints are updated before optimization.

The *FBA* class is an extension of the *LinearProblem* class and implements a mathematical approach to represent and analyze metabolic reactions available in a metabolic network [126]. According to the FBA formulation, the *build* method infers steady-state mass balance constraints using the model metabolites and reactions. The following techniques extend the *FBA* class to reuse these mass balance constraints.

The *pFBA* parent class implements a phenotype prediction method to find the optimal distribution of the flow of metabolites that maximizes biomass production, while minimizing the total sum of fluxes in the metabolic network [127]. Accordingly, the *build* method reformulates the objective function derived from the FBA implementation.

rFBA inherits the *build* method of the *FBA* class, as both techniques share the same linear problem formulation. However, the rFBA method differs from FBA by accounting for the regulatory state of the model in the flux distribution [144]. The rFBA method is detailed in the 2.9 section. First, *rFBA*'s *optimize* method performs a synchronous evaluation of the regulatory network using an initial regulatory state (effectors' coefficients). This simulation allows inferring the coefficients of all regulators in the regulatory networks. In the second step, the *optimize* method performs a second evaluation of all regulatory interactions in the regulatory model to determine the metabolic state (coefficients of target/metabolic genes). GPR rules available in the metabolic model are evaluated next with the resulting metabolic state and translated into a set of constraints limiting the flux of the model reactions.

rFBA can infer the initial regulatory state as follows:

- If the regulator is a regulatory metabolite, rFBA uses the absolute value of the lower bound of the exchange reaction;

- If the regulator is a regulatory reaction, rFBA uses the absolute value of its upper bound;

- Otherwise, rFBA uses the maximum coefficient (often one) of a regulator.

Nevertheless, some initial states might lead to the deletion of essential genes and reactions and yield infeasible solutions. To mitigate conflicts in the initial state of the regulatory network, one can use the *mewpy.germ.analysis.find_conflicts* function. This method can find regulatory states that lead to deletions of essential genes and reactions, thus hindering optimal growth.

SR-FBA is a phenotype prediction method integrating simultaneously regulatory and mass balance constraints [145]. This method is detailed in 2.9 section. The *SR-FBA* class represents a set of mixed-integer linear constraints retrieved from the integrated model. In detail, the *build* method creates mid-term linear variables and constraints according to the operators and operands of each Boolean algebra expression available in the model (regulatory events and GPR rules). *SR-FBA*'s optimization uses MILP to find an optimal flux distribution compatible with regulatory and metabolic constraints.

The *PROM* class implements a probabilistic-based technique to predict the phenotype of regulatory mutants. According to PROM's description in section 2.9, this method uses the probability of a metabolic gene being active to limit the flux of model reactions [146]. The probability of a metabolic gene being transcribed is calculated for the number of samples in which the gene is active, while the related regulator is inactive. Metabolic gene probability can be calculated using the *mewpy.germ.analysis. target_regulator_interaction_probability* function. Finally, reaction bounds are proportional to the probability of the related genes.

The CoRegFlux technique uses multiple linear regression models to predict the levels of expression of metabolic genes as a function of the regulators' co-expression (co-activators and co-repressors). According to CoRegFlux's description in section 2.9, the predicted levels of expression are translated into limiting flux bounds [148]. First, CoRegFlux uses gene expression data and regulators' influence scores (inferred with CoRegNet [115]) to train linear regression models. Linear regression models can be trained using the *mewpy.germ.analysis.pre- dict_gene_expression*. Then, these models are used to make predictions in another gene expression dataset. Finally, the *optimize* method uses a continuous approach to limit the flux distribution according to the soft plus activation of the predicted expression.

The *mewpy.germ.analysis* module also includes several functions to analyze integrated models of metabolism and regulation. For example, one can perform an FVA analysis (*mewpy.germ.analysis.fva* or *mewpy.germ.analysis.ifva*) using FBA, rFBA, or SR-FBA techniques. Likewise, the analysis module comprises functions to perform fast analysis of reactions, genes, and regulators perturbations.

MEWpy's phenotype prediction component uses external packages, such as COBRApy and Reframed, to evaluate different phenotypes within the CSO framework. Nevertheless, this module provides a common interface to seamlessly integrate other modeling frameworks. In detail, the *Simulator* class implemented in *mewpy.simulation* determines an interface for phenotype simulators. For example, the *Simulation* class available in *mewpy.simulation.cobra* sub-module implements a phenotype simulator for the COBRApy modeling framework. In this sub-class of *Simulator*, one has access to simple representations of genes, metabolites, and reactions available in the *Model* object of COBRApy. Moreover, simulation methods available in the COBRApy package are seamlessly provided in the *simulate* method of the *Simulation* class.

The GERM model framework is integrated into MEWpy's phenotype prediction module, so users can access a common interface to perform phenotype prediction using GERM, COBRApy, and Reframed models. Moreover, implementing the GERM model simulator allows using this modeling framework in the strain optimization routines. In detail, the *Simulation* sub-class available in the sub-module *mewpy.simulation.germ* implements a GERM model simulator by extending the *Simulator* class. This sub-class implements a *simulate* method that uses the FBA and pFBA simulation methods implemented in the *mewpy.germ.analysis* sub-module. Furthermore, one can initiate the GERM model simulator using the *mewpy.simulation.get_simulator* function. This function accepts a GERM model object returning the corresponding simulator.

5.3.6 Strain optimization using genome-scale regulatory-metabolic models

MEWpy's optimization component is based on evolutionary computation to solve an optimization problem. Optimization problems encompass the modeling framework, the objective functions, the modification targets, and strategies. Whereas optimization problems can include a combination of independent objective functions, targets and strategies are sometimes limited to the modeling framework. Yet, *mewpy.problems* module includes several tools implementing problems for over-expression, under-expression, and deletion of variables in the modeling framework. Table 8 summarizes the optimization problems available in MEWpy according to the modification targets, strategies, and modeling framework.

Table 8: Optimization problems available in the *mewpy.problems* module. MEWpy's optimization problems are described in terms of modeling framework and both modification strategy and target.

Optimization problem	**Target**	**Strategy**	**Modeling framework**
RKOProblem	Reactions	Deletion	COBRApy, Reframed, and GERM models
ROUProblem	Reactions	Over and under expression	COBRApy, Reframed, and GERM models
GKOProblem	Genes	Deletion	COBRApy, Reframed, and GERM models
GOUProblem	Genes	Over and under expression	COBRApy, Reframed, and GERM models
GeckoKOProblem	Enzymes	Deletion	GECKO or sMOMENT models implemented with COBRApy or Reframed
GeckoOUProblem	Enzymes	Over and under expression	GECKO or sMOMENT models implemented with COBRApy or Reframed
OptORFProblem	Regulators	Deletion	GERM models
OptRamProblem	Regulators	Over and under expression	IDREAM integrated network based implementation

As the GERM model framework is integrated into MEWpy's phenotype prediction module, users can create optimization problems that encompass the over-expression, under-expression, and deletion of reactions and genes available in a GERM model. Nevertheless, the GERM model framework is still missing a few phenotype prediction methods, namely MOMA, lMOMA, and ROOM.

OptORF is an algorithm for strain optimization that suggests perturbations to genes and regulators simultaneously [219]. This optimization framework proposes evolutionary strain designs using an integrated regulatory and metabolic model. In OptORF's original formulation, GPR rules are translated directly into linear constraints using a Boolean algebra approach. Consequently, reactions are blocked by deleting the associated genes in the linear problem. Likewise, integrating TRNs into the metabolic network is also implemented using a Boolean algebra approach. Metabolic genes can be switched on or off according to the linear relationship of the associated regulators. These linear relationships are directly added to the linear problem through binary variables and constraints obtained from the regulatory network.

In the context of this work, the *OptORFProblem* class implements an adapted version of the original OptORF problem published by Kim and Reed [219]. In this OptORF-adapted version, GPR rules and regulatory interactions are not converted into linear constraints using the Boolean algebra approach. Instead, both regulatory and metabolic networks are evaluated using the binary values suggested for regulators and genes. Then, reactions are removed according to the state of metabolic genes. In detail, the *OptORFProblem* class extends the *AbstractKOProblem* abstract interface, which is the base class for the deletion modification strategy. The adapted version of OptORF is based on the synchronous evaluation of the regulatory network represented in a GERM model. In the first step, the initial regulatory state (effectors' coefficients) and the candidate state (regulator deletions) are combined to evaluate the regulatory interactions in the GERM model. This simulation determines the coefficients of regulators and target genes. In the second step, the *OptORFProblem* class solves the metabolic state by evaluating the Boolean GPR rules available in the metabolic model. Finally, the resulting metabolic state is translated into a set of constraints limiting the flux of the model reactions.

The *OptORFProblem* class is seamlessly integrated into the evolutionary computation framework of MEWpy. One can run OptORF's optimization task using the *EA* class with several objective functions. At each generation, the *EA* class suggests new candidate regulatory mutants that favor the desired metabolic engineering goal.

OptRAM is an alternative strain optimization method that suggests deletion, over and under expression modifications to metabolic genes and regulators simultaneously [185]. This optimization framework does not require a regulatory network, as it is based on the IDREAM computational tool. IDREAM is an integrated framework that can infer regulatory networks using gene expression data. The expression levels of metabolic genes are inferred using the expression of corresponding regulators. Then, OptRAM can translate the expression levels of metabolic genes into flux constraints using the GPR rules of the GEM model. As regards the strain optimization engine, OptRAM uses simulated annealing to suggest a series of perturbations to the expression of regulators towards the overproduction of the compound of interest. OptRAM has been previously implemented in MEWpy beyond the context of this work. The implementation of OptRAM in MEWpy can use MOEAs to combine different optimization objectives.

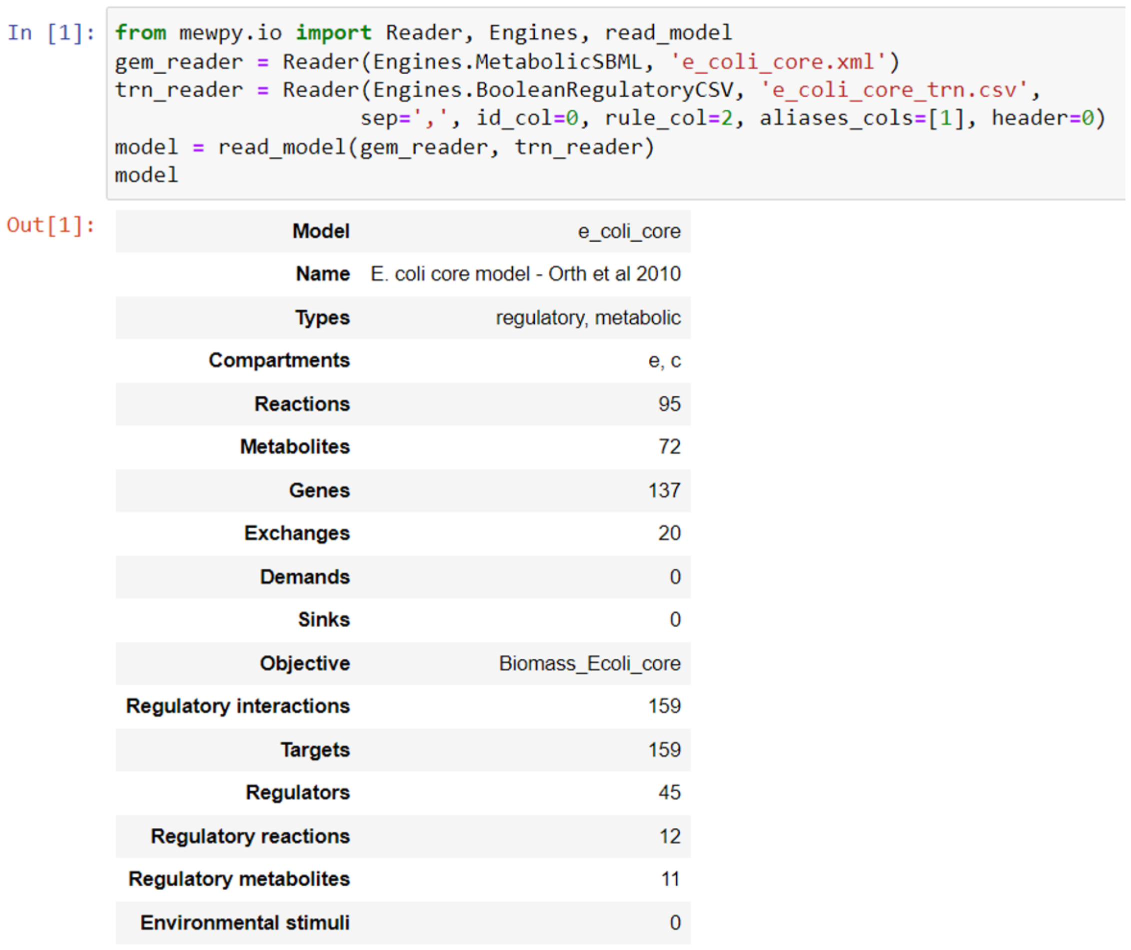
5.4 Working with genome-scale regulatory-metabolic models in MEWpy

5.4.1 Model workflow

The following section comprises a detailed example of working with an integrated model of metabolism and regulation in MEWpy. In this example, the integrated *E. coli* core GERM model published by Orth *et al.* [224] will be used to showcase the capabilities of the GERM model framework. Furthermore, MEWpy documentation also includes a more extended version of the examples described in this sub-chapter at MEWpy examples repository and MEWpy documentation.

The integrated *E. coli* core GERM model has been retrieved from the original publication [224] and is available in two separate files: GEM model in the SBML file format (MEWpy examples repository) and TRN in the CSV file format (MEWpy examples repository).

The function *mewpy.io.read_model* was used to read the *E. coli* integrated model. Figure 50 shows the *read_model* signature, including a GEM model reader using the *MetabolicSBML* engine and the TRN reader using the *BooleanRegulatoryCSV* engine. Note that both readers have been initiated with specific arguments, such as *filename, sep, id_col, and rule_col*, among others.



```
In [1]: from mewpy.io import Reader, Engines, read_model
        gem_reader = Reader(Engines.MetabolicSBML, 'e_coli_core.xml')
        trn_reader = Reader(Engines.BooleanRegulatoryCSV, 'e_coli_core_trn.csv',
                            sep=',', id_col=0, rule_col=2, aliases_cols=[1], header=0)
        model = read_model(gem_reader, trn_reader)
        model
```

Out[1]:

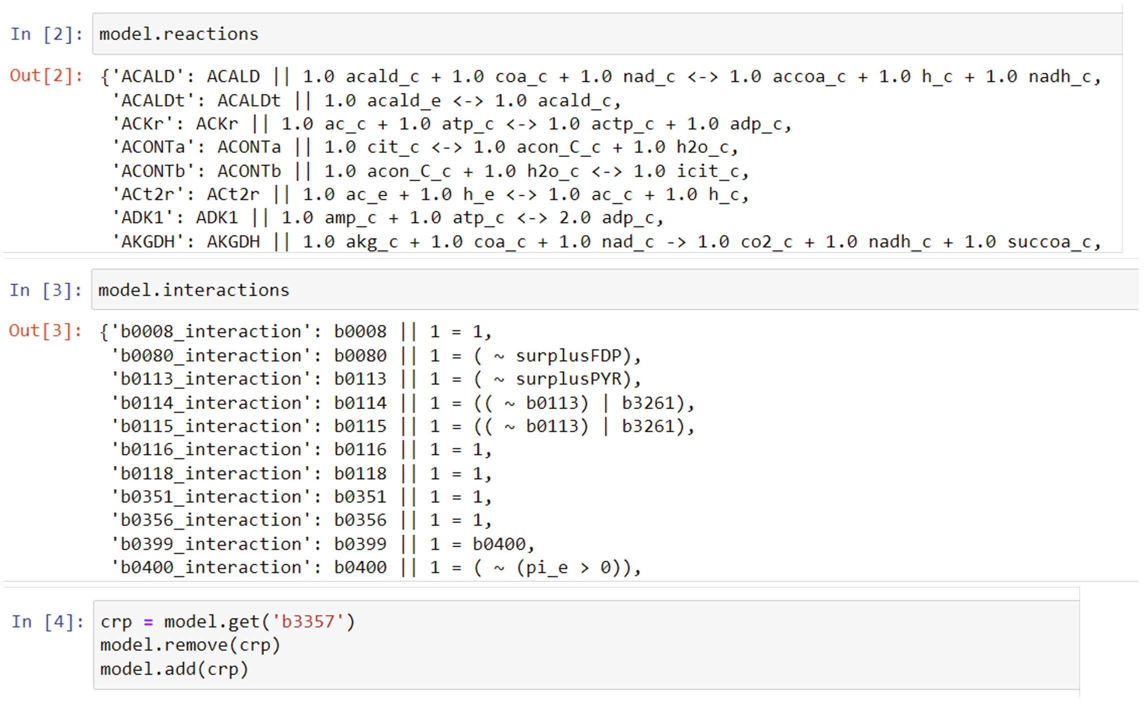
Model	e_coli_core
Name	E. coli core model - Orth et al 2010
Types	regulatory, metabolic
Compartments	e, c
Reactions	95
Metabolites	72
Genes	137
Exchanges	20
Demands	0
Sinks	0
Objective	Biomass_Ecoli_core
Regulatory interactions	159
Targets	159
Regulators	45
Regulatory reactions	12
Regulatory metabolites	11
Environmental stimuli	0

Figure 50: GERM model loading with the *mewpy.io* tools. A GERM model can be loaded from two separate files using the GEM model reader (*MetabolicSBML*) and TRN reader (*BooleanRegulatoryCSV*).

In the Jupyter Notebook cell, one can visualize the contents of the integrated *E. coli* core GERM model. The *Model* representation includes the identifier, name, types, compartments, and the total metabolic and regulatory variables available in the model.

Users can inspect metabolic and regulatory containers in Jupyter Notebook. Figure 51 shows a short version of the reactions and interactions available in the integrated *E. coli* core GERM model. The model containers are standard Python dictionaries storing the model variables.

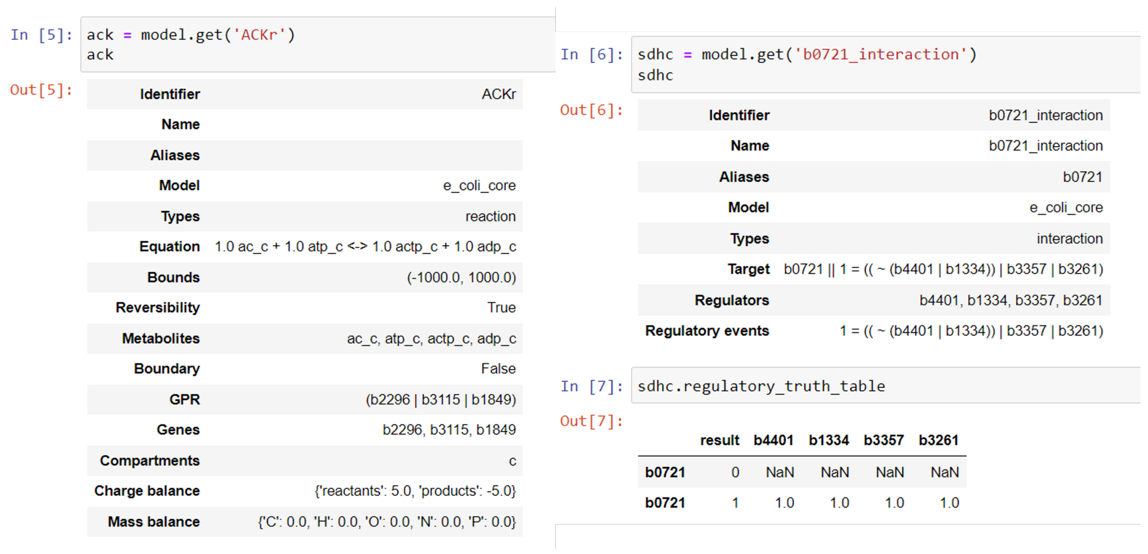
A GERM model supports the inspection and manipulation of metabolic and regulatory variables using the *get*, *add*, *update*, and *remove* methods. For example, figure 51 shows how to access, add, and remove the Crp regulator (b3357) of the *E. coli* model. In addition, a GERM model supports temporary changes using the *with model* context manager.



```
In [2]: model.reactions

Out[2]: {'ACALD': ACALD || 1.0 acald_c + 1.0 coa_c + 1.0 nad_c <-> 1.0 accoa_c + 1.0 h_c + 1.0 nadh_c,
         'ACALDt': ACALDt || 1.0 acald_e <-> 1.0 acald_c,
         'ACKr': ACKr || 1.0 ac_c + 1.0 atp_c <-> 1.0 actp_c + 1.0 adp_c,
         'ACONTa': ACONTa || 1.0 cit_c <-> 1.0 acon_C_c + 1.0 h2o_c,
         'ACONTb': ACONTb || 1.0 acon_C_c + 1.0 h2o_c <-> 1.0 icit_c,
         'ACt2r': ACt2r || 1.0 ac_e + 1.0 h_e <-> 1.0 ac_c + 1.0 h_c,
         'ADK1': ADK1 || 1.0 amp_c + 1.0 atp_c <-> 2.0 adp_c,
         'AKGDH': AKGDH || 1.0 akg_c + 1.0 coa_c + 1.0 nad_c -> 1.0 co2_c + 1.0 nadh_c + 1.0 succoa_c,

In [3]: model.interactions

Out[3]: {'b0008_interaction': b0008 || 1 = 1,
         'b0080_interaction': b0080 || 1 = ( ~ surplusFDP),
         'b0113_interaction': b0113 || 1 = ( ~ surplusPYR),
         'b0114_interaction': b0114 || 1 = (( ~ b0113) | b3261),
         'b0115_interaction': b0115 || 1 = (( ~ b0113) | b3261),
         'b0116_interaction': b0116 || 1 = 1,
         'b0118_interaction': b0118 || 1 = 1,
         'b0351_interaction': b0351 || 1 = 1,
         'b0356_interaction': b0356 || 1 = 1,
         'b0399_interaction': b0399 || 1 = b0400,
         'b0400_interaction': b0400 || 1 = ( ~ (pi_e > 0)),

In [4]: crp = model.get('b3357')
        model.remove(crp)
        model.add(crp)
```

Figure 51: GERM model manipulation using the *Model* interface. Inspection and manipulation of a GERM model is available using the *Model* containers (*model.interactions* and *model.reactions*) and operations (*model.get*, *model.add*, and *model.remove*).

The integrated *E. coli* core GERM model includes various variables with different attributes. For example, figure 52 shows the characteristics of the *ACKr* reaction and *b0721* regulatory interaction.



```
In [5]: ack = model.get('ACKr')
        ack
Out[5]:
```

Identifier	ACKr
Name	
Aliases	
Model	e_coli_core
Types	reaction
Equation	1.0 ac_c + 1.0 atp_c <-> 1.0 actp_c + 1.0 adp_c
Bounds	(-1000.0, 1000.0)
Reversibility	True
Metabolites	ac_c, atp_c, actp_c, adp_c
Boundary	False
GPR	(b2296 | b3115 | b1849)
Genes	b2296, b3115, b1849
Compartments	c
Charge balance	{'reactants': 5.0, 'products': -5.0}
Mass balance	{'C': 0.0, 'H': 0.0, 'O': 0.0, 'N': 0.0, 'P': 0.0}

```
In [6]: sdhc = model.get('b0721_interaction')
        sdhc
Out[6]:
```

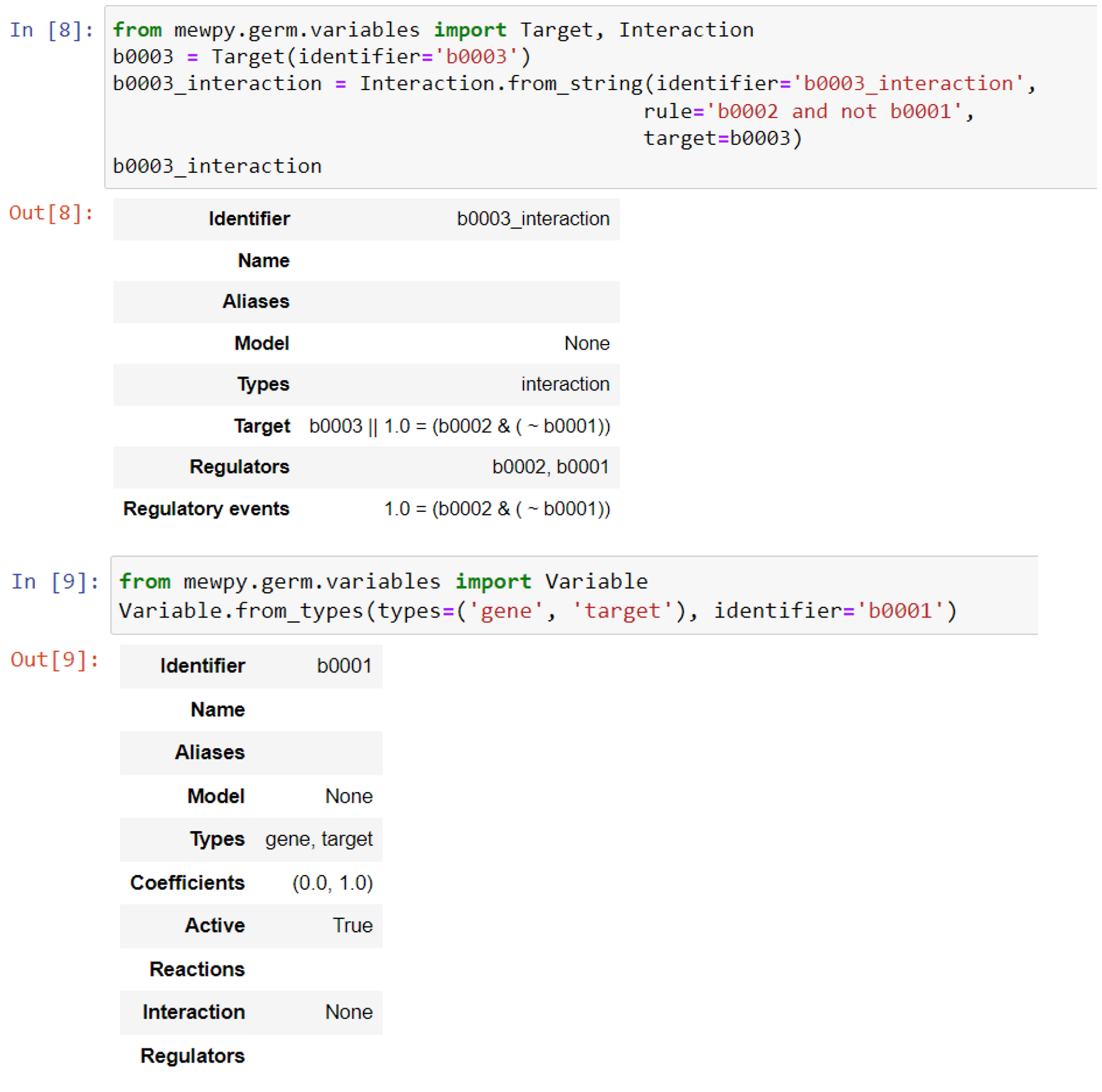
Identifier	b0721_interaction
Name	b0721_interaction
Aliases	b0721
Model	e_coli_core
Types	interaction
Target	b0721 || 1 = ((~ (b4401 | b1334)) | b3357 | b3261)
Regulators	b4401, b1334, b3357, b3261
Regulatory events	1 = ((~ (b4401 | b1334)) | b3357 | b3261)

```
In [7]: sdhc.regulatory_truth_table
Out[7]:
```

	result	b4401	b1334	b3357	b3261
b0721	0	NaN	NaN	NaN	NaN
b0721	1	1.0	1.0	1.0	1.0

Figure 52: Manipulation of GERM model variables using the *Variable* interface. Inspection and manipulation of the *ACKr* reaction and *b0721* regulatory interaction using the *Variable* framework.

Variables can be created using the *Reaction*, *Metabolite*, *Gene*, *Interaction*, *Target*, and *Regulator* classes (Figure 53). Alternatively, integrated models often include multi-type variables comprising different groups of attributes. For instance, metabolic genes tend to be the target variables of a TRN. Figure 53 shows the *b0001* variable of type gene and target. This variable contains the attributes represented in the *Gene* and *Target* classes. Figure 53 also highlights how to create multi-type variables using the *from_types* polymorphic constructor of the *Variable* class. Integrating metabolic and regulatory attributes into the same object provides a simple interface for reconstructing and analyzing integrated models.



```
In [8]: from mewpy.germ.variables import Target, Interaction
        b0003 = Target(identifier='b0003')
        b0003_interaction = Interaction.from_string(identifier='b0003_interaction',
                                                    rule='b0002 and not b0001',
                                                    target=b0003)

        b0003_interaction
```

```
Out[8]:
          Identifier                           b0003_interaction
                Name
             Aliases
               Model                                        None
               Types                                 interaction
              Target    b0003 || 1.0 = (b0002 & ( ~ b0001))
          Regulators                                b0002, b0001
   Regulatory events           1.0 = (b0002 & ( ~ b0001))
```

```
In [9]: from mewpy.germ.variables import Variable
        Variable.from_types(types=('gene', 'target'), identifier='b0001')
```

```
Out[9]:
    Identifier          b0001
          Name
       Aliases
         Model           None
         Types   gene, target
  Coefficients     (0.0, 1.0)
        Active           True
     Reactions
   Interaction           None
    Regulators
```

Figure 53: Assembly of single- and multi-type GERM model variables using the *Variable* interface. The *b0003* interaction can be created from a Boolean regulatory rule associated with the single-type *b0003* target variable. Alternatively, one can create integrated multi-type variables, such as the *b0001* metabolic gene and regulatory target.

5.4.2 Model analysis workflow

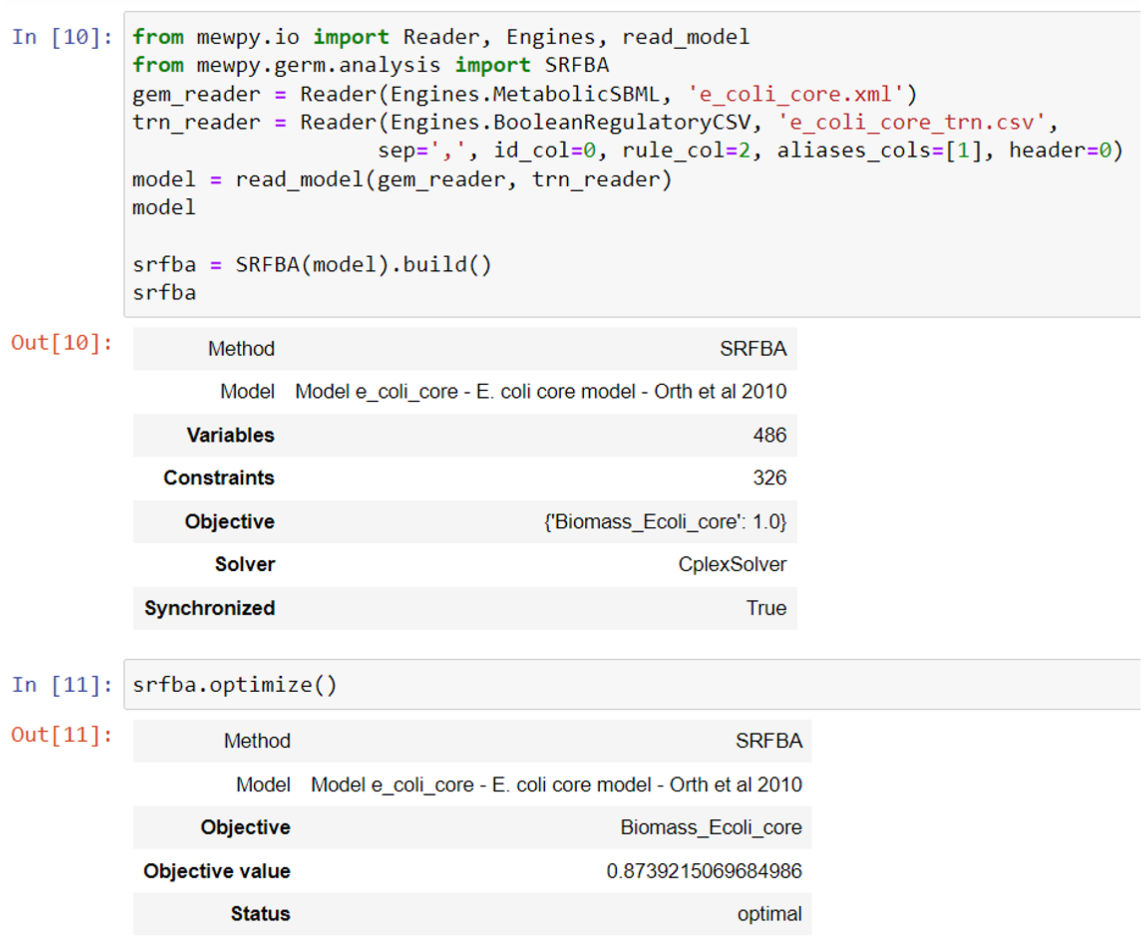
The following section comprises several examples to analyze integrated models of metabolism and regulation in MEWpy. Table 9 includes the case studies used to showcase the capabilities of the *mewpy.germ.analysis* framework. Furthermore, MEWpy's documentation comprehends a more extended version of the examples described in this sub-chapter at MEWpy examples repository and MEWpy documentation.

Figure 54 shows the main attributes of an SR-FBA instance created with the integrated *E. coli* core GERM model and the *ModelSolution* object obtained after the phenotype prediction. In addition, all methods have a fast one-line function to run the phenotype prediction. These functions only return the objective

Table 9: Case studies used to showcase the GERM model framework. Several GERM models are described in terms of identifier, organism, GEM model, *trn*, content, and availability.

Model	**Organism**	**GEM model**	**TRN**	**Content**	**Gene expression data**	**Availability**
E. coli core model	*Escherichia coli* K-12 MG1655	*E. coli* core GEM model [224]	*E. coli* core TRN [224]	137 metabolic genes, 95 reactions, and 159 regulatory interactions	-	Supplementatry material I.9
iMC1010 [144]	*Escherichia coli* K-12 MG1655	*E. coli* iJR904 GEM model [225]	*E. coli* iMC1010 TRN [144]	904 metabolic genes, 931 reactions, and 1010 regulatory interactions	-	Supplementatry material I.10
iNJ661 [146]	*Mycobacterium tuberculosis* H37Rv	*M. tuberculosis* iNJ661 GEM model [226]	*M. tuberculosis* H37Rv TRN published by Balazsi *et al.* [227]	691 metabolic genes, 1028 reactions, and 2018 regulatory interactions	Gene expression dataset with 437 experiments (regulator deletions) [146]	Supplementatry material I.11
iMM904 [148]	*Saccharomyces cerevisiae* S288C	*S. cerevisiae* iMM904 GEM model [228]	*S. cerevisiae* S288C TRN inferred by CoRegNet [115]	904 metabolic genes, 1557 reactions, and 3748 regulatory interactions	Gene expression dataset of 247 experiments [84], influence scores inferred by CoRegNet [115], and another gene expression dataset of 12-time points [229]	Supplementatry material I.12

value of the solution. Phenotype prediction methods can be attached to GERM models using the argument *attach*. This option allows synchronizing simulation objects with model changes before optimization. Besides, one can connect many simulation methods to a single *Model* instance.



```
In [10]: from mewpy.io import Reader, Engines, read_model
         from mewpy.germ.analysis import SRFBA
         gem_reader = Reader(Engines.MetabolicSBML, 'e_coli_core.xml')
         trn_reader = Reader(Engines.BooleanRegulatoryCSV, 'e_coli_core_trn.csv',
                             sep=',', id_col=0, rule_col=2, aliases_cols=[1], header=0)
         model = read_model(gem_reader, trn_reader)
         model

         srfba = SRFBA(model).build()
         srfba
```

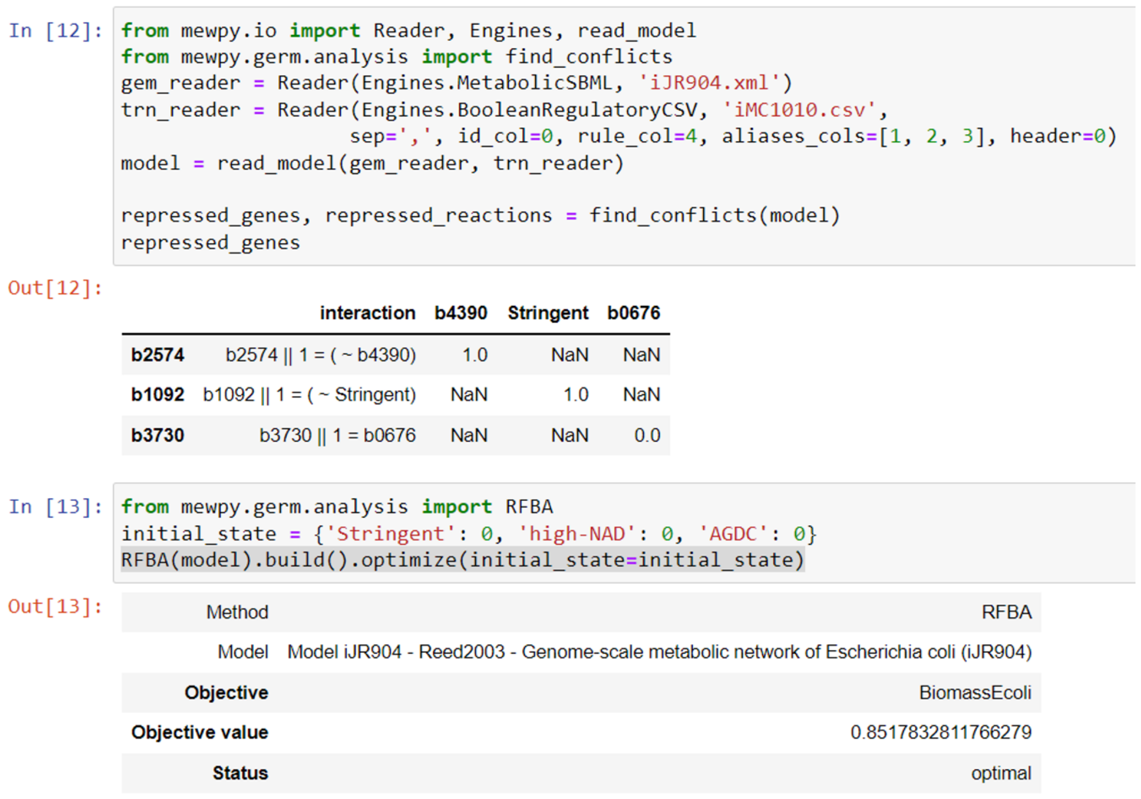
```
Out[10]:
            Method                                                SRFBA
             Model   Model e_coli_core - E. coli core model - Orth et al 2010
         Variables                                                  486
       Constraints                                                  326
         Objective                               {'Biomass_Ecoli_core': 1.0}
            Solver                                          CplexSolver
      Synchronized                                                 True
```

```
In [11]: srfba.optimize()
```

```
Out[11]:
              Method                                                SRFBA
               Model   Model e_coli_core - E. coli core model - Orth et al 2010
           Objective                                    Biomass_Ecoli_core
     Objective value                                    0.8739215069684986
              Status                                               optimal
```

Figure 54: Phenotype prediction workflow using the GERM model framework. SR-FBA analysis of the integrated *E. coli* core GERM model. Users can load GERM models using MEWpy input/output tools. Then, an integrated phenotype prediction can be performed using the *build-optimize* workflow implemented in all phenotype prediction methods.

rFBA and *SR-FBA* techniques follow a discrete approach to limit the flux space of solutions using the state of metabolic and regulatory genes. rFBA accepts an initial regulatory state to evaluate the model regulatory interactions. The *mewpy.germ.analysis.find_conflicts* method assists the design of an initial state that leads to feasible solutions and optimal growth rates. Figure 55 shows how *find_conflicts* function can be used to find conflicting regulatory states in the integrated iMC1010 GERM model. This method suggests that three essential genes (*b2574*, *b1092*, and *b3730*) are repressed by three regulators (*b4390*, *Stringent*, and *b0676*). However, some regulators do not hinder growth directly. For example, figure 55 shows that regulator-target *b4390* is only active with *high-NAD* environmental stimuli. Figure 55 shows an *rFBA* analysis of the iMC1010 GERM model. In this case, the initial regulatory state suggested by the *find_conflicts* method has been used to obtain a wild-type growth rate of 0.8518 h^{-1}.



```
In [12]: from mewpy.io import Reader, Engines, read_model
         from mewpy.germ.analysis import find_conflicts
         gem_reader = Reader(Engines.MetabolicSBML, 'iJR904.xml')
         trn_reader = Reader(Engines.BooleanRegulatoryCSV, 'iMC1010.csv',
                             sep=',', id_col=0, rule_col=4, aliases_cols=[1, 2, 3], header=0)
         model = read_model(gem_reader, trn_reader)

         repressed_genes, repressed_reactions = find_conflicts(model)
         repressed_genes
```

Out[12]:

	interaction	b4390	Stringent	b0676
b2574	b2574 || 1 = (~ b4390)	1.0	NaN	NaN
b1092	b1092 || 1 = (~ Stringent)	NaN	1.0	NaN
b3730	b3730 || 1 = b0676	NaN	NaN	0.0

```
In [13]: from mewpy.germ.analysis import RFBA
         initial_state = {'Stringent': 0, 'high-NAD': 0, 'AGDC': 0}
         RFBA(model).build().optimize(initial_state=initial_state)
```

Out[13]:

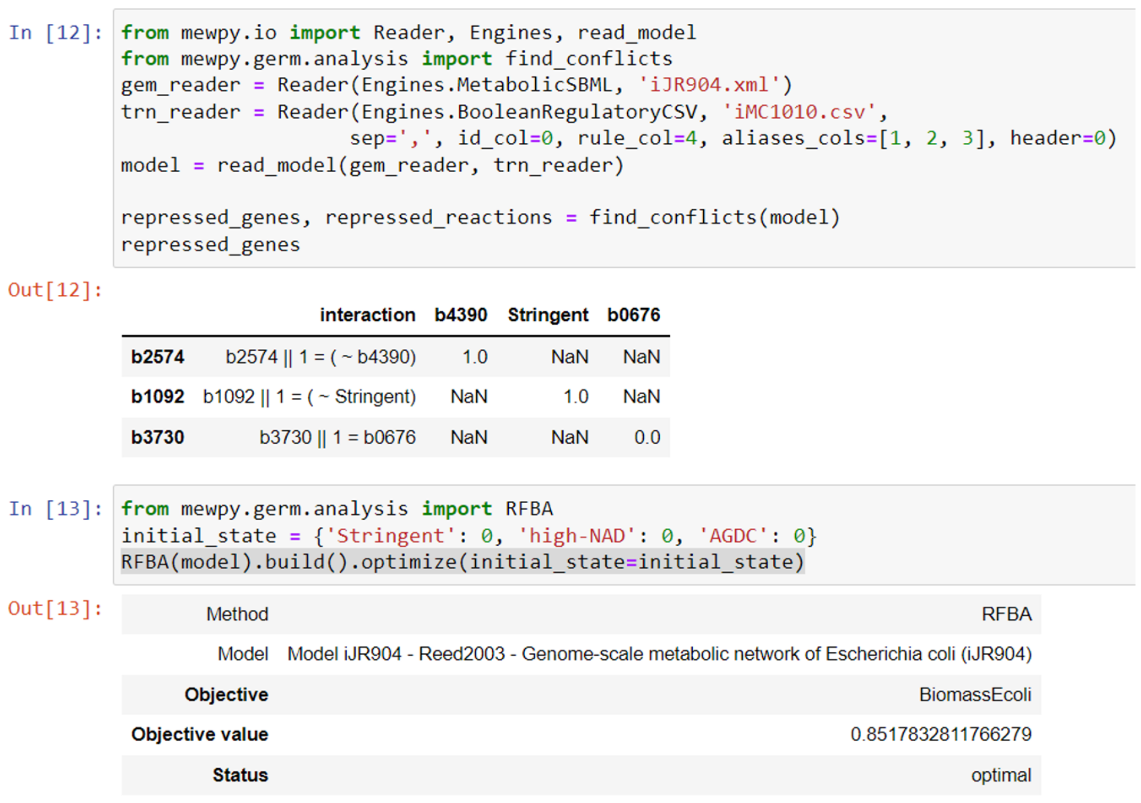
Method	RFBA
Model	Model iJR904 - Reed2003 - Genome-scale metabolic network of Escherichia coli (iJR904)
Objective	BiomassEcoli
Objective value	0.8517832811766279
Status	optimal

Figure 55: rFBA phenotype prediction workflow using the GERM model framework. rFBA analysis of the integrated iMC1010 GERM model of *E. coli*. Users can load GERM models using MEWpy input/output tools. Then, the initial regulatory state, a critical step in rFBA method, can be inferred with the *find_conflicts* function. Finally, an integrated phenotype prediction can be performed using the *build-optimize* workflow implemented in all phenotype prediction methods. The *optimize* method accepts rFBA's initial regulatory state and generates a phenotype prediction solution.

On the other hand, one can perform an *SR-FBA* analysis without an initial regulatory state. *SR-FBA* uses MILP to find a regulatory state with an optimal growth rate. Figure 56 shows an SR-FBA analysis of the integrated iMC1010 GERM model.

PROM and CoRegFlux techniques circumvent the discrete approach implemented in rFBA and SR-FBA by limiting the flux of reactions according to the model regulatory interactions and gene expression dataset.

PROM limits the flux of each reaction according to the probability of related genes being active while the related regulator is inactive. The *mewpy.omics* module includes several tools to perform the quantile pre-processing pipeline described by Chandrasekaran *et al.* [146]. Then, the *mewpy.germ.analysis.target_regulator_interaction_probability* function can be used to infer PROM's initial regulatory state. Figure 57 shows the inference of the target-regulator probability using the integrated iNJ661 GERM model and the gene expression dataset published by Chandrasekaran *et al.* [146].



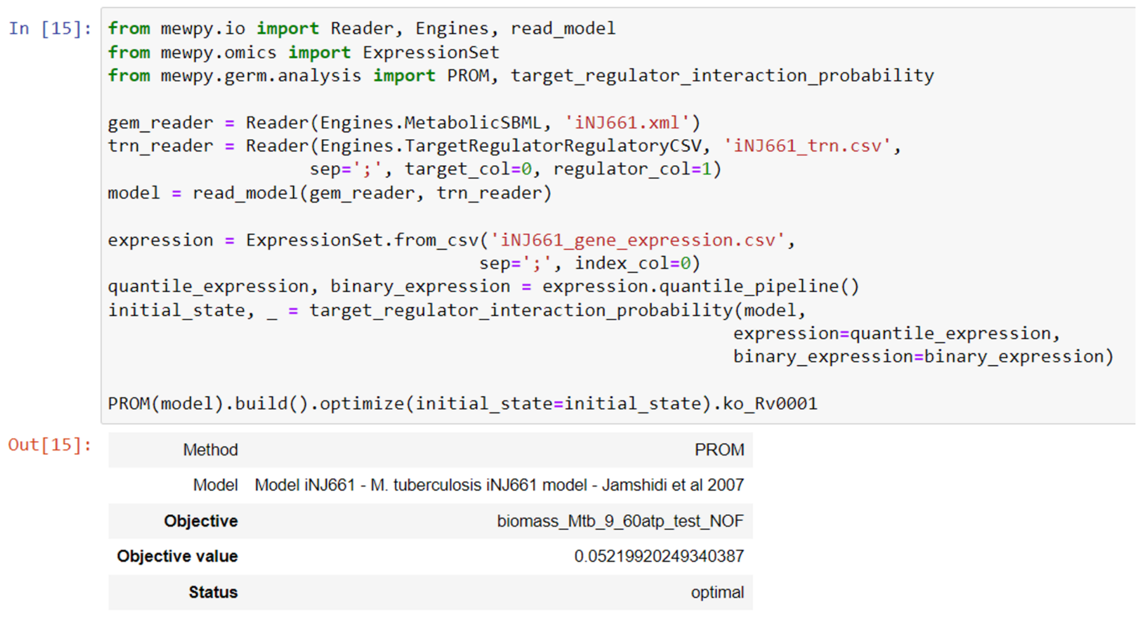
```
In [12]: from mewpy.io import Reader, Engines, read_model
         from mewpy.germ.analysis import find_conflicts
         gem_reader = Reader(Engines.MetabolicSBML, 'iJR904.xml')
         trn_reader = Reader(Engines.BooleanRegulatoryCSV, 'iMC1010.csv',
                             sep=',', id_col=0, rule_col=4, aliases_cols=[1, 2, 3], header=0)
         model = read_model(gem_reader, trn_reader)

         repressed_genes, repressed_reactions = find_conflicts(model)
         repressed_genes
Out[12]:
```

	interaction	b4390	Stringent	b0676
b2574	b2574 || 1 = (~ b4390)	1.0	NaN	NaN
b1092	b1092 || 1 = (~ Stringent)	NaN	1.0	NaN
b3730	b3730 || 1 = b0676	NaN	NaN	0.0

```
In [13]: from mewpy.germ.analysis import RFBA
         initial_state = {'Stringent': 0, 'high-NAD': 0, 'AGDC': 0}
         RFBA(model).build().optimize(initial_state=initial_state)
Out[13]:
```

Method	RFBA
Model	Model iJR904 - Reed2003 - Genome-scale metabolic network of Escherichia coli (iJR904)
Objective	BiomassEcoli
Objective value	0.8517832811766279
Status	optimal

Figure 56: SR-FBA phenotype prediction workflow using the GERM model framework. SR-FBA analysis of the integrated iMC1010 GERM model of *E. coli*. Users can load GERM models using MEWpy input/output tools. Then, an integrated phenotype prediction can be performed using the *build-optimize* workflow implemented in all phenotype prediction methods.



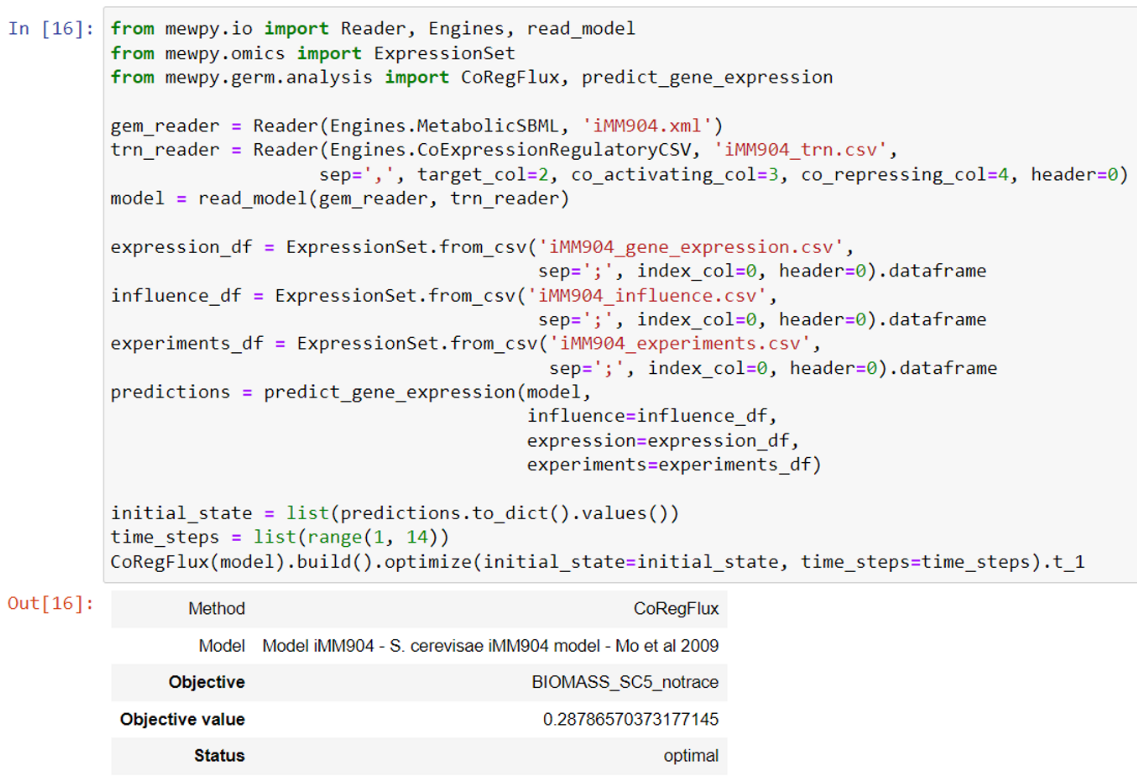
In [15]:
```
from mewpy.io import Reader, Engines, read_model
from mewpy.omics import ExpressionSet
from mewpy.germ.analysis import PROM, target_regulator_interaction_probability

gem_reader = Reader(Engines.MetabolicSBML, 'iNJ661.xml')
trn_reader = Reader(Engines.TargetRegulatorRegulatoryCSV, 'iNJ661_trn.csv',
                   sep=';', target_col=0, regulator_col=1)
model = read_model(gem_reader, trn_reader)

expression = ExpressionSet.from_csv('iNJ661_gene_expression.csv',
                                   sep=';', index_col=0)
quantile_expression, binary_expression = expression.quantile_pipeline()
initial_state, _ = target_regulator_interaction_probability(model,
                                                           expression=quantile_expression,
                                                           binary_expression=binary_expression)

PROM(model).build().optimize(initial_state=initial_state).ko_Rv0001
```

Out[15]:

Method	PROM
Model	Model iNJ661 - M. tuberculosis iNJ661 model - Jamshidi et al 2007
Objective	biomass_Mtb_9_60atp_test_NOF
Objective value	0.05219920249340387
Status	optimal

Figure 57: PROM phenotype prediction workflow using the GERM model framework. PROM analysis of the integrated iNJ661 GERM model of *M. tuberculosis*. Users can load GERM models using MEWpy input/output tools. The probability of related genes being active when a regulator is deleted can be inferred using the *quantile_pipeline* and *mewpy.analysis.target_regulator_interaction_probability* functions. Finally, an integrated phenotype prediction can be performed using the *build-optimize* workflow implemented in all phenotype prediction methods. The *optimize* method accepts PROM's initial regulatory state and generates a phenotype prediction solution for each regulator deletion.

Alternatively, CoRegFlux limits the flux of each reaction to the levels of expression of the related genes using a softplus activation function [148]. One can use the *mewpy.germ.analysis.predict_gene_expression* function to create multiple linear regression models capable of predicting the levels of expression of metabolic genes. Then, the predicted levels can be used in the CoRegFlux analysis. Figure 58 shows how to perform the CoRegFlux analysis in the integrated iMM904 GERM model for the first time-point, before glucose depletion, according to the gene expression dataset published by Brauer *et al.* [229].



```
In [16]: from mewpy.io import Reader, Engines, read_model
         from mewpy.omics import ExpressionSet
         from mewpy.germ.analysis import CoRegFlux, predict_gene_expression

         gem_reader = Reader(Engines.MetabolicSBML, 'iMM904.xml')
         trn_reader = Reader(Engines.CoExpressionRegulatoryCSV, 'iMM904_trn.csv',
                             sep=',', target_col=2, co_activating_col=3, co_repressing_col=4, header=0)
         model = read_model(gem_reader, trn_reader)

         expression_df = ExpressionSet.from_csv('iMM904_gene_expression.csv',
                                                sep=';', index_col=0, header=0).dataframe
         influence_df = ExpressionSet.from_csv('iMM904_influence.csv',
                                               sep=';', index_col=0, header=0).dataframe
         experiments_df = ExpressionSet.from_csv('iMM904_experiments.csv',
                                                 sep=';', index_col=0, header=0).dataframe
         predictions = predict_gene_expression(model,
                                               influence=influence_df,
                                               expression=expression_df,
                                               experiments=experiments_df)

         initial_state = list(predictions.to_dict().values())
         time_steps = list(range(1, 14))
         CoRegFlux(model).build().optimize(initial_state=initial_state, time_steps=time_steps).t_1
Out[16]:
```

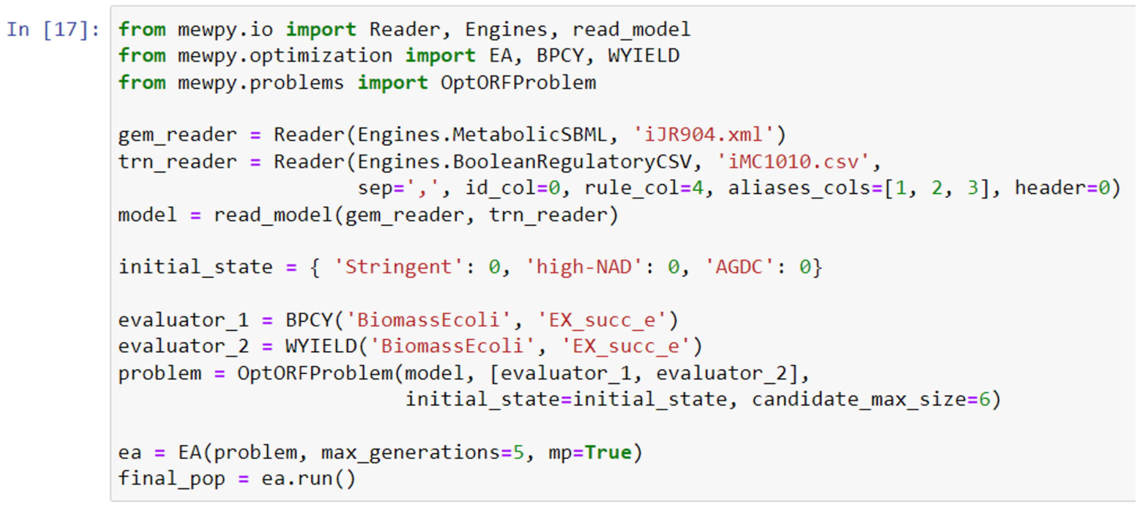
Method	CoRegFlux
Model	Model iMM904 - S. cerevisae iMM904 model - Mo et al 2009
Objective	BIOMASS_SC5_notrace
Objective value	0.28786570373177145
Status	optimal

Figure 58: CoRegFlux phenotype prediction workflow using the GERM model framework. CoRegFlux analysis of the integrated iMM904 GERM model of *S. cerevisiae*. Users can load GERM models using MEWpy input/output tools. Gene expression predictions can be inferred for each experiment using the *mewpy.analysis.predict_gene_expression* function. Finally, an integrated phenotype prediction can be performed using the *build-optimize* workflow implemented in all phenotype prediction methods. The *optimize* method accepts CoRegFlux's initial regulatory state and generates a phenotype prediction solution for each experiment.

5.4.3 Model optimization workflow

MEWpy provides a common interface to define an optimization problem. One can easily create optimization tasks with different modeling frameworks, modification targets, and strategies. The following sub-chapter describes how to create an optimization problem using the integrated iMC1010 GERM model published by Covert *et al.* [144]. MEWpy's documentation includes an extended version of the examples described in this sub-chapter at MEWpy examples repository and MEWpy documentation.

In this example, an OptORF problem encompasses the GERM model, two objective functions (e.g. *WYIELD* and *BPCY*), and the metabolic engineering goal, namely succinate production. Figure 59 shows how to read the iMC1010 GERM model from the GEM model and TRN files. Users can define an initial regulatory state that favors optimal growth or compound production. *WYIELD* and *BPCY* classes can be used to define objective functions towards succinate production. The *OptORFProblem* is initiated with the iMC1010 GERM model, *WYIELD* and *BPCY* objective functions, and initial regulatory state (figure 59). The evolutionary computation engine is then used to solve the optimization problem for a maximum of 10 generations using multi-processing.



```
In [17]: from mewpy.io import Reader, Engines, read_model
         from mewpy.optimization import EA, BPCY, WYIELD
         from mewpy.problems import OptORFProblem

         gem_reader = Reader(Engines.MetabolicSBML, 'iJR904.xml')
         trn_reader = Reader(Engines.BooleanRegulatoryCSV, 'iMC1010.csv',
                             sep=',', id_col=0, rule_col=4, aliases_cols=[1, 2, 3], header=0)
         model = read_model(gem_reader, trn_reader)

         initial_state = { 'Stringent': 0, 'high-NAD': 0, 'AGDC': 0}

         evaluator_1 = BPCY('BiomassEcoli', 'EX_succ_e')
         evaluator_2 = WYIELD('BiomassEcoli', 'EX_succ_e')
         problem = OptORFProblem(model, [evaluator_1, evaluator_2],
                                 initial_state=initial_state, candidate_max_size=6)

         ea = EA(problem, max_generations=5, mp=True)
         final_pop = ea.run()
```

Figure 59: Example of an *OptORF* optimization problem. An *OptORFProblem* can take a GERM model (e.g., iMC1010 GERM model), *WYIELD* and *BPCY* objective functions, and initial regulatory state. Then, the *EA* engine is responsible for finding a population favoring the metabolic engineering goal (e.g., succinate production).

6

Conclusion

6.1 Main contributions

The present work comprehends several tools to assist the reconstruction and analysis of regulatory and metabolic models at the genome-scale. The following tools and databases have been developed under the scope of this thesis:

- BioISO - A tool to assist the curation of GEM models;
- ProTReND - A database of prokaryotic TRNs;
- GERM model framework in MEWpy - Definition and analysis of GERM models.

Despite the variety of modeling frameworks focussed on GEM models and TRNs, the reconstruction and analysis of integrated GERM models has not been a common procedure for non-model organisms. The methodologies proposed throughout the years are difficult to adopt and often unavailable to the scientific community. Furthermore, the reconstruction of high-quality GEM models and TRNs can be affected by laborious tasks and missing data.

With this in mind, the outcomes of this thesis can be used to obtain new insights regarding the molecular mechanisms that control gene expression and metabolic processes. Additionally, all computational tools developed under the scope of this thesis are available to the scientific community. In addition, most of these tools provide a user-friendly interface to bypass barriers in reconstructing GEM models and TRNs. Lastly, a simple yet modular modeling framework, providing several methods to analyze integrated GERM models, has been delivered to the community. This modeling framework can leverage gene expression data and biological networks to improve phenotype predictions and extend the scope of GEM models and TRNs.

The first outcome of this work consists of BioISO, a user-friendly tool capable of performing guided searches of gaps in metabolic networks. This tool aims to help scientists without coding skills reconstruct high-quality GEM models, leveraging bottom-up reconstructions that require intensive manual curation and human intervention. Several state-of-the-art gap-finding tools have been compared with BioISO, which emerged as the only open-source tool ready to be used by the scientific community. Moreover, since BioISO does not perform gap-filling, it is not associated with this technique's significant drawbacks, such as poor usability, the need for additional data, and recommending biological artifacts due to the lack of evidence for the solutions. Moreover, BioISO has been validated with GEMs available in the literature. When debugging and validating the model for specific objective functions, such as growth maximization, BioISO seems better suited for reducing the search space for errors and gaps in metabolic networks than other tools.

The second outcome of this thesis is ProTReND, a state-of-the-art repository of genome-scale TRNs for many prokaryotes. This repository includes a robust data integration system unifying regulatory data available in several organism-specific and non-organism-specific databases and literature. The data integration system follows a modular ETL pipeline that can be extended with other resources of regulatory data. Furthermore, ProTReND's repository comprehends a knowledge expansion sub-system that has contributed greatly to improving regulatory data quality. The regulatory data integrated into ProTReND describes regulatory interactions between roughly 12300 regulators and 99500 genes for more than 500 organisms. Notably, this repository also includes binding site data, and effector information, among other elements of gene regulation in prokaryotes. All regulatory data is accessible at protrend.bio.di.uminho.pt. Besides being available through a web application, users can also obtain ProTReND's data programmatically at protrend.bio.di.uminho.pt/api. Lastly, ProTReND community is a single-page web application that allows users, after a short registration, to contribute to the database.

The GERM model framework implemented within MEWpy is the last output of this thesis. This modeling framework offers several tools to represent and analyze the integration of TRNs into GEM models. Users can read separately GEM models, TRNs, and GERM models from SBML, CSV, and JSON files. A GERM model is represented in MEWpy by a set of integrated metabolic and regulatory variables. One can use the framework to inspect and manipulate attributes of GERM models. For instance, this framework can parse Boolean algebra expressions into object-oriented regulatory interactions containing a target and several regulators. More importantly, the GERM model framework contains several phenotype prediction methods oriented to analyze the regulatory and metabolic layer. rFBA and SR-FBA are two discrete-based approaches to analyze the integration of regulatory interactions into metabolic reactions. Alternatively, the GERM model framework includes the PROM and CoRegFlux continuous-based approaches. The latter phenotype prediction methods are aimed at tailoring the flux space of solutions using the TRN and transcriptomics data.

6.2 Publications

The present thesis has contributed to several publications. The following have been submitted and accepted during the development of this work, some of which developed with the collaboration of other researchers in the host research group:

- Cruz, F., Lima, D., Faria, J. P., Rocha, M., Dias, O. (2020). Towards the Reconstruction of Integrated Genome-Scale Models of Metabolism and Gene Expression. In: Fdez-Riverola, F., Rocha, M., Mohamad, M., Zaki, N., Castellanos-Garzón, J. (eds) Practical Applications of Computational Biology and Bioinformatics, 13th International Conference. PACBB 2019. Advances in Intelligent Systems and Computing, vol 1005. Springer, Cham.

- Cruz, F., Faria, J. P., Rocha, M., Rocha, I., Dias, O. (2020). A review of methods for the reconstruction and analysis of integrated genome-scale models of metabolism and regulation. Biochemical Society Transactions.

- Lima, D., Cruz, F., Rocha, M., Dias, O. (2021). Reconciliation of Regulatory Data: The Regulatory Networks of Escherichia coli and Bacillus subtilis. In: Panuccio, G., Rocha, M., Fdez-Riverola, F., Mohamad, M., Casado-Vara, R. (eds) Practical Applications of Computational Biology & Bioinformatics, 14th International Conference (PACBB 2020). PACBB 2020. Advances in Intelligent Systems and Computing, vol 1240. Springer, Cham.

- Pereira, V., Cruz, F., Rocha, M. (2021). MEWpy: a computational strain optimization workbench in Python, Bioinformatics, Volume 37, Issue 16, Pages 2494–2496.

The following have been submitted (available as a pre-print) and are waiting for revision:

- Cruz, F., Capela, J., Ferreira, E. C., Rocha, M., Dias, O. BioISO: an objective-oriented application for assisting the curation of genome-scale metabolic models. Submitted. Pre-print available at: https://doi.org/10.1101/2021.03.07.434259

The following are currently under preparation for submission:

- Cruz, F., Lima, D., Rocha, M., Dias, O. ProTReND: a database of prokaryotic genome-scale TRNs. In preparation.

Finally, the following articles were also published during the doctoral program, but were not included in this thesis:

- Capela, J., Lagoa, D., Rodrigues, R., Cunha, E., Cruz, F., Barbosa, A., Bastos, J., Lima, D., Ferreira, E. C., Rocha, M., Dias, O. (2022). merlin, an improved framework for the reconstruction of high-quality genome-scale metabolic models. Nucleic Acids Research, Volume 50, Issue 11, Pages 6052–6066.

- Cruz, F., Lagoa, D., Mendes, J. Rocha, I., Ferreira, E. C., Rocha, M., Dias, O. (2019). SamPler – a novel method for selecting parameters for gene functional annotation routines. BMC Bioinformatics 20, 454.

- Oliveira, A., Cunha, E., Cruz, F., Ribeiro, J., Sequeira, J. C., Sampaio, M., Dias, O. (2022). Towards a Multivariate Analysis of Genome-Scale Metabolic Models Derived from the BiGG Models Database. In: Rocha, M., Fdez-Riverola, F., Mohamad, M.S., Casado-Vara, R. (eds) Practical Applications of Computational Biology & Bioinformatics, 15th International Conference (PACBB 2021). PACBB 2021. Lecture Notes in Networks and Systems, vol 325. Springer, Cham.

- Oliveira, A., Cunha, E., Cruz, F., Ribeiro, J., Sequeira, J. C., Sampaio, M., Sampaio, C., Dias, O. (2022). Systematic assessment of template-based genome-scale metabolic models created with the BiGG Integration Tool. Journal of Integrative Bioinformatics, Volume 19, Number 3, Pages 20220014.

6.3 Future work

The development of the above-mentioned tools also raised several questions and future improvements. In short, the accessibility and interoperability of the methodologies implemented in the scope of this work can be further improved. For instance, BioISO can be extended to guide the manual curation of integrated GERM models using the modeling framework implemented in MEWpy. In addition, the high-quality bottom-up reconstruction workflow suggested in BioISO can be added to the user-friendly GUI so users can visualize parsimonious solutions (when compared to other tools) for each dead-end metabolite. Furthermore, the GERM model framework can also include a set of tools to load TRNs available in the ProTReND database. The implementation of this feature would improve the interoperability between ProTReND and MEWpy, easing the integration of TRNs into GEM models.

ProTReND can also be expanded into a knowledgebase supporting the inference of novel TRNs. In detail, this database can serve as the base for several computational tools to reconstruct novel TRNs using the genomics and proteomics data associated with the TRNs. Likewise, ProTReND can be used to search for novel binding sites, thus assisting the reconstruction of novel regulatory networks. The integration of these tools with a GUI supporting the manual curation of the TRNs resulting into high-quality regulatory data for an organism of interest.

Lastly, the GERM model framework should be thoroughly assessed using known and novel case studies. Assessing the phenotype prediction and strain optimization methods implemented in this modeling framework could help further validate its resourcefulness for the scientific community. Furthermore, developing a GUI for the GERM model framework can make this tool accessible to a significant share of the scientific community.

Bibliography

[1] J. M. Lourenço, *The NOVAthesis LaTeX Template User's Manual*, NOVA University Lisbon, 2021.

[2] S. Mukherjee, D. Stamatis, J. Bertsch, *et al.*, "Genomes OnLine Database (GOLD) v.6: data updates and feature enhancements", *Nucleic Acids Research*, vol. 45, no. D1, pp. D446–D456, Jan. 2017.

[3] A. N. Gray, B.-M. Koo, A. L. Shiver, J. M. Peters, H. Osadnik, and C. A. Gross, "High-throughput bacterial functional genomics in the sequencing era", *Current Opinion in Microbiology*, vol. 27, pp. 86–95, Oct. 2015.

[4] J. Nielsen, "Systems Biology of Metabolism", *Annual Review of Biochemistry*, vol. 86, no. 1, pp. 245–275, Jun. 2017.

[5] J. S. Edwards and B. O. Palsson, "Systems properties of the Haemophilus influenzae Rd metabolic genotype", *Journal of Biological Chemistry*, vol. 274, no. 25, pp. 17 410–17 416, Jun. 1999.

[6] A. Bordbar and B. O. Palsson, "Using the reconstructed genome-scale human metabolic network to study physiology and pathology.", *Journal of internal medicine*, vol. 271, no. 2, pp. 131–41, Feb. 2012.

[7] C. Bro, B. Regenberg, J. Förster, and J. Nielsen, "In silico aided metabolic engineering of Saccharomyces cerevisiae for improved bioethanol production.", *Metabolic engineering*, vol. 8, no. 2, pp. 102–11, Mar. 2006.

[8] O. Dias, J. Saraiva, C. Faria, M. Ramirez, F. Pinto, and I. Rocha, "iDS372, a Phenotypically Reconciled Model for the Metabolism of Streptococcus pneumoniae Strain R6", *Frontiers in Microbiology*, vol. 10, no. JUN, p. 1283, Jun. 2019.

[9] O. Dias, R. Pereira, A. K. Gombert, E. C. Ferreira, and I. Rocha, "iOD907, the first genome-scale metabolic model for the milk yeast Kluyveromyces lactis", *Biotechnology Journal*, vol. 9, no. 6, pp. 776–790, Jun. 2014.

[10] C. S. Henry, J. F. Zinner, M. P. Cohoon, and R. L. Stevens, "iBsu1103: A new genome-scale metabolic model of Bacillus subtilis based on SEED annotations", *Genome Biology*, vol. 10, no. 6, Jun. 2009.

[11] J. D. Orth, T. M. Conrad, J. Na, *et al.*, "A comprehensive genome-scale reconstruction of Escherichia coli metabolism–2011.", *Molecular systems biology*, vol. 7, p. 535, Oct. 2011.

[12] O. Dias and I. Rocha, "Systems Biology in Fungi", in *Molecular Biology of Food and Water Borne Mycotoxigenic and Mycotic Fungi*, R. Paterson, Ed., Boca Raton: CRC Press, 2015, ch. 6, pp. 69–92.

[13] I. Thiele and B. Ø. Palsson, "A protocol for generating a high-quality genome-scale metabolic reconstruction", *Nature Protocols*, vol. 5, no. 1, pp. 93–121, Jan. 2010.

[14] D. Machado, S. Andrejev, M. Tramontano, and K. R. Patil, "Fast automated reconstruction of genome-scale metabolic models for microbial species and communities", *Nucleic Acids Research*, vol. 46, no. 15, pp. 7542–7553, Sep. 2018.

[15] J. P. Faria, R. Overbeek, F. Xia, M. Rocha, I. Rocha, and C. S. Henry, "Genome-scale bacterial transcriptional regulatory networks: reconstruction and integrated analysis with metabolic models", *Briefings in Bioinformatics*, vol. 15, no. 4, pp. 592–611, Jul. 2014.

[16] D. Machado and M. Herrgård, "Systematic Evaluation of Methods for Integration of Transcriptomic Data into Constraint-Based Models of Metabolism", *PLoS Computational Biology*, vol. 10, no. 4, C. D. Maranas, Ed., e1003580, Apr. 2014.

[17] D. Marbach, J. C. Costello, R. Küffner, *et al.*, "Wisdom of crowds for robust gene network inference", *Nature Methods*, vol. 9, no. 8, pp. 796–804, Aug. 2012.

[18] V. A. Huynh-Thu and G. Sanguinetti, "Gene Regulatory Network Inference: An Introductory Survey", in *Methods in Molecular Biology*, vol. 1883, Humana Press Inc., 2019, pp. 1–23.

[19] S. Barbosa, B. Niebel, S. Wolf, K. Mauch, and R. Takors, "A guide to gene regulatory network inference for obtaining predictive solutions: Underlying assumptions and fundamental biological and data constraints", *Biosystems*, vol. 174, pp. 37–48, Dec. 2018.

[20] D. L. Nelson and M. M. Cox, *Lehninger Principles of Biochemistry*, 6th edit. W.H. Freeman, 2008, p. 1328.

[21] H. Lodish, B. Arnold, Z. S Lawrence, M. Paul, B. David, and D. James, *Molecular Cell Biology*, 4th editio, W. H. Freeman, Ed. New York, 2000.

[22] C. Willson, D. Pebrin, M. Cohn, F. Jacob, and J. Monod, "Non-inducible mutants of the regulator gene in the "lactose" system of Escherichia coli", *Journal of Molecular Biology*, vol. 8, no. 4, pp. 582–592, Apr. 1964.

[23] F. Jacob and J. Monod, *Genetic regulatory mechanisms in the synthesis of proteins*, 1961.

[24] D. F. Browning and S. J. W. Busby, "Local and global regulation of transcription initiation in bacteria", *Nature Reviews Microbiology*, vol. 14, no. 10, pp. 638–650, Oct. 2016.

[25] K. Struhl, "Fundamentally Different Logic of Gene Regulation in Eukaryotes and Prokaryotes", *Cell*, vol. 98, no. 1, pp. 1–4, Jul. 1999.

[26] R. De Smet and K. Marchal, *Advantages and limitations of current network inference methods*, 2010.

[27] S. A. Teichmann and M. Babu, "Conservation of gene co-regulation in prokaryotes and eukaryotes", *Trends in Biotechnology*, vol. 20, no. 10, pp. 407–410, Oct. 2002.

[28] R. Bonneau, *Learning biological networks: From modules to dynamics*, 2008.

[29] A. Fadda, A. C. Fierro, K. Lemmens, P. Monsieurs, K. Engelen, and K. Marchal, "Inferring the transcriptional network of Bacillus subtilis", *Molecular BioSystems*, vol. 5, no. 12, pp. 1840–1852, 2009.

[30] B. K. Cho, K. Zengler, Y. Qiu, *et al.*, "The transcription unit architecture of the Escherichia coli genome", *Nature Biotechnology*, vol. 27, no. 11, pp. 1043–1049, Nov. 2009.

[31] I. Lozada-Chavez, S. C. Janga, and J. Collado-Vides, "Bacterial regulatory networks are extremely flexible in evolution", *Nucleic Acids Research*, vol. 34, no. 12, pp. 3434–3445, Jul. 2006.

[32] M. W. Covert, C. H. Schilling, and B. Palsson, "Regulation of Gene Expression in Flux Balance Models of Metabolism", *Journal of Theoretical Biology*, vol. 213, no. 1, pp. 73–88, Nov. 2001.

[33] C. H. Yeang and M. Vingron, "A joint model of regulatory and metabolic networks", *BMC Bioinformatics*, vol. 7, Jul. 2006.

[34] J. J. Tyson and B. Novák, "Functional Motifs in Biochemical Reaction Networks", *Annual Review of Physical Chemistry*, vol. 61, no. 1, pp. 219–240, Mar. 2010.

[35] T. M. Ramseier, S. Bledig, V. Michotey, R. Feghali, and M. H. Saier, "The global regulatory protein FruR modulates the direction of carbon flow in Escherichia coli", *Molecular Microbiology*, vol. 16, no. 6, pp. 1157–1169, 1995.

[36] D. Machado, M. J. Herrgård, and I. Rocha, "Modeling the Contribution of Allosteric Regulation for Flux Control in the Central Carbon Metabolism of E. coli", *Frontiers in Bioengineering and Biotechnology*, vol. 3, p. 154, Oct. 2015.

[37] L. Gerosa and U. Sauer, *Regulation and control of metabolic fluxes in microbes*, Aug. 2011.

[38] K. Kochanowski, U. Sauer, and V. Chubukov, *Somewhat in control-the role of transcription in regulating microbial metabolic fluxes*, Dec. 2013.

[39] P. Nicolas, U. Mäder, E. Dervyn, *et al.*, "Condition-dependent transcriptome reveals high-level regulatory architecture in Bacillus subtilis.", *Science (New York, N.Y.)*, vol. 335, no. 6072, pp. 1103–6, Mar. 2012.

[40] J. M. Buescher, W. Liebermeister, M. Jules, *et al.*, "Global network reorganization during dynamic adaptations of Bacillus subtilis metabolism.", *Science (New York, N.Y.)*, vol. 335, no. 6072, pp. 1099–103, Mar. 2012.

[41] Y. K. Wang, D. G. Hurley, S. Schnell, C. G. Print, and E. J. Crampin, "Integration of Steady-State and Temporal Gene Expression Data for the Inference of Gene Regulatory Networks", *PLoS ONE*, vol. 8, no. 8, S. D. Peddada, Ed., e72103, Aug. 2013.

[42] T. T. Perkins, R. A. Kingsley, M. C. Fookes, *et al.*, "A strand-specific RNA-seq analysis of the transcriptome of the typhoid bacillus Salmonella typhi", *PLoS Genetics*, 2009.

[43] K. C. Kao, Y. L. Yang, R. Boscolo, C. Sabatti, V. Roychowdhury, and J. C. Liao, "Transcriptome-based determination of multiple transcription regulator activities in Escherichia coli by using network component analysis", *Proceedings of the National Academy of Sciences of the United States of America*, vol. 101, no. 2, pp. 641–646, Jan. 2004.

[44] O. J. Shaw, C. Harwood, L. J. Steggles, and A. Wipat, "SARGE: A tool for creation of putative genetic networks", *Bioinformatics*, vol. 20, no. 18, pp. 3638–3640, Dec. 2004.

[45] W. A. Schmitt, R. M. Raab, and G. Stephanopoulos, "Elucidation of gene interaction networks through time-lagged correlation analysis of transcriptional data", *Genome Research*, vol. 14, no. 8, pp. 1654–1663, Aug. 2004.

[46] X. Fang, A. Sastry, N. Mih, *et al.*, "Global transcriptional regulatory network for Escherichia coli robustly connects gene expression to transcription factor activities.", *Proceedings of the National Academy of Sciences of the United States of America*, vol. 114, no. 38, pp. 10 286–10 291, Sep. 2017.

[47] S. Turkarslan, E. J. R. Peterson, T. R. Rustad, *et al.*, "A comprehensive map of genome-wide gene regulation in Mycobacterium tuberculosis", *Scientific Data*, vol. 2, p. 150 010, Mar. 2015.

[48] P. O. Brown and D. Botstein, "Exploring the new world of the genome with dna microarrays", *Nature Genetics*, vol. 21, no. 1S, p. 37, 1999.

[49] M. Eisenstein, *Microarrays: Quality control*, 2006.

[50] R. A. Young, "Biomedical Discovery with DNA Arrays", *Cell*, vol. 102, no. 1, pp. 9–15, Jul. 2000.

[51] R. Edgar, *Challenge of choosing right level of microarray detail [2]*, 2006.

[52] L. D. Poulsen and J. Vinther, "RNA-Seq for Bacterial Gene Expression", *Current Protocols in Nucleic Acid Chemistry*, 2018.

[53] K. James, S. J. Cockell, and N. Zenkin, "Deep sequencing approaches for the analysis of prokaryotic transcriptional boundaries and dynamics", *Methods*, vol. 120, pp. 76–84, May 2017.

[54] A. Conesa, P. Madrigal, S. Tarazona, *et al.*, "A survey of best practices for RNA-seq data analysis", *Genome Biology*, vol. 17, no. 1, p. 13, Dec. 2016.

[55] J. P. Creecy and T. Conway, *Quantitative bacterial transcriptomics with RNA-seq*, 2015.

[56] Z. Wang, M. Gerstein, and M. Snyder, *RNA-Seq: A revolutionary tool for transcriptomics*, 2009.

[57] B. Ren, F. Robert, J. J. Wyrick, *et al.*, "Genome-wide location and function of DNA binding proteins.", *Science (New York, N.Y.)*, vol. 290, no. 5500, pp. 2306–9, Dec. 2000.

[58] V. R. Iyer, C. E. Horak, C. S. Scafe, D. Botstein, M. Snyder, and P. O. Brown, "Genomic binding sites of the yeast cell-cycle transcription factors SBF and MBF", *Nature*, vol. 409, no. 6819, pp. 533–538, Jan. 2001.

[59] T. H. Kim and B. Ren, "Genome-Wide Analysis of Protein-DNA Interactions", *Annual Review of Genomics and Human Genetics*, 2006.

[60] J. Galagan, A. Lyubetskaya, and A. Gomes, "ChIP-Seq and the Complexity of Bacterial Transcriptional Regulation", in Springer, Berlin, Heidelberg, 2012, pp. 43–68.

[61] P. J. Park, *ChIP-seq: Advantages and challenges of a maturing technology*, Oct. 2009.

[62] C. Evans, J. Hardin, and D. M. Stoebel, "Selecting between-sample RNA-Seq normalization methods from the perspective of their assumptions", *Briefings in bioinformatics*, vol. 19, no. 5, pp. 776–792, Sep. 2018.

[63] E. R. Mardis, *The impact of next-generation sequencing technology on genetics*, Mar. 2008.

[64] A. Brazma, P. Hingamp, J. Quackenbush, *et al.*, "Minimum information about a microarray experiment (MIAME)—toward standards for microarray data", *Nature Genetics*, vol. 29, no. 4, pp. 365–371, Dec. 2001.

[65] T. Barrett, S. E. Wilhite, P. Ledoux, *et al.*, "NCBI GEO: archive for functional genomics data sets—update", *Nucleic Acids Research*, vol. 41, no. D1, pp. D991–D995, Nov. 2012.

[66] N. Kolesnikov, E. Hastings, M. Keays, *et al.*, "ArrayExpress update—simplifying data submissions", *Nucleic Acids Research*, vol. 43, no. D1, pp. D1113–D1116, Jan. 2015.

[67] A. Santos-Zavaleta, M. Sánchez-Pérez, H. Salgado, *et al.*, "A unified resource for transcriptional regulation in Escherichia coli K-12 incorporating high-throughput-generated binding data into RegulonDB version 10.0", *BMC Biology*, vol. 16, no. 1, p. 91, Dec. 2018.

[68] I. M. Keseler, A. Mackie, A. Santos-Zavaleta, *et al.*, "The EcoCyc database: reflecting new knowledge about <i>Escherichia coli</i> K-12", *Nucleic Acids Research*, vol. 45, no. D1, pp. D543–D550, Jan. 2017.

[69] N. Sierro, Y. Makita, M. de Hoon, and K. Nakai, "DBTBS: a database of transcriptional regulation in Bacillus subtilis containing upstream intergenic conservation information", *Nucleic Acids Research*, vol. 36, no. suppl_1, pp. D93–D96, Jan. 2008.

[70] P.-E. Jacques, A. L. Gervais, M. Cantin, *et al.*, “MtbRegList, a database dedicated to the analysis of transcriptional regulation in Mycobacterium tuberculosis”, *Bioinformatics*, vol. 21, no. 10, pp. 2563–2565, May 2005.

[71] M. T. D. Parise, D. Parise, R. B. Kato, *et al.*, “CoryneRegNet 7, the reference database and analysis platform for corynebacterial gene regulatory networks”, *Scientific Data 2020 7:1*, vol. 7, no. 1, pp. 1–9, May 2020.

[72] J. Wu, F. Zhao, S. Wang, *et al.*, “cTFbase: a database for comparative genomics of transcription factors in cyanobacteria”, *BMC Genomics*, vol. 8, no. 1, p. 104, Apr. 2007.

[73] S. Kılıç, D. M. Sagitova, S. Wolfish, *et al.*, “From data repositories to submission portals: rethinking the role of domain-specific databases in CollecTF”, *Database*, vol. 2016, baw055, Apr. 2016.

[74] P. S. Novichkov, A. E. Kazakov, D. A. Ravcheev, *et al.*, “RegPrecise 3.0 – A resource for genome-scale exploration of transcriptional regulation in bacteria”, *BMC Genomics*, vol. 14, no. 1, p. 745, Nov. 2013.

[75] D. Eckweiler, C.-A. Dudek, J. Hartlich, D. Brötje, and D. Jahn, “PRODORIC2: the bacterial gene regulation database in 2018”, *Nucleic Acids Research*, vol. 46, no. D1, pp. D320–D326, Jan. 2018.

[76] J. M. Escorcia-Rodríguez, A. Tauch, and J. A. Freyre-González, “Abasy Atlas v2.2: The most comprehensive and up-to-date inventory of meta-curated, historical, bacterial regulatory networks, their completeness and system-level characterization”, *Computational and Structural Biotechnology Journal*, vol. 18, pp. 1228–1237, Jan. 2020.

[77] M. Pachkov, I. Erb, N. Molina, and E. van Nimwegen, “SwissRegulon: a database of genome-wide annotations of regulatory sites”, *Nucleic Acids Research*, vol. 35, no. Database, pp. D127–D131, Jan. 2007.

[78] S. Okuda and A. C. Yoshizawa, “ODB: a database for operon organizations, 2011 update”, *Nucleic Acids Research*, vol. 39, no. Database, pp. D552–D555, Jan. 2011.

[79] A. Sebastian and B. Contreras-Moreira, “footprintDB: a database of transcription factors with annotated cis elements and binding interfaces”, *Bioinformatics*, vol. 30, no. 2, pp. 258–265, Jan. 2014.

[80] D. Szklarczyk, A. L. Gable, D. Lyon, *et al.*, “STRING v11: Protein-protein association networks with increased coverage, supporting functional discovery in genome-wide experimental datasets”, *Nucleic Acids Research*, vol. 47, no. D1, pp. D607–D613, Jan. 2019.

[81] M. Kanehisa, M. Furumichi, M. Tanabe, Y. Sato, and K. Morishima, “KEGG: new perspectives on genomes, pathways, diseases and drugs”, *Nucleic Acids Research*, vol. 45, no. D1, pp. D353–D361, Jan. 2017.

[82] R. Caspi, T. Altman, R. Billington, *et al.*, “The MetaCyc database of metabolic pathways and enzymes and the BioCyc collection of Pathway/Genome Databases”, *Nucleic Acids Research*, vol. 42, no. D1, pp. D459–D471, Jan. 2014.

[83] M. Moretto, P. Sonego, N. Dierckxsens, *et al.*, “COLOMBOS v3.0: leveraging gene expression compendia for cross-species analyses”, *Nucleic Acids Research*, vol. 44, no. D1, pp. D620–D623, Jan. 2016.

[84] J. J. Faith, M. E. Driscoll, V. A. Fusaro, *et al.*, “Many Microbe Microarrays Database: uniformly normalized Affymetrix compendia with structured experimental metadata”, *Nucleic Acids Research*, vol. 36, no. Database, pp. D866–D870, Dec. 2007.

[85] G. Sherlock, “The Stanford Microarray Database”, *Nucleic Acids Research*, vol. 29, no. 1, pp. 152–155, Jan. 2001.

[86] R. Leinonen, H. Sugawara, and M. Shumway, “The sequence read archive”, *Nucleic Acids Research*, vol. 39, no. SUPPL. 1, Jan. 2011.

[87] Y. Gao, J. T. Yurkovich, S. W. Seo, *et al.*, “Systematic discovery of uncharacterized transcription factors in Escherichia coli K-12 MG1655”, *Nucleic Acids Research*, vol. 46, no. 20, pp. 10 682–10 696, Aug. 2018.

[88] D. A. Rodionov, P. S. Novichkov, E. D. Stavrovskaya, *et al.*, “Comparative genomic reconstruction of transcriptional networks controlling central metabolism in the Shewanella genus”, *BMC Genomics*, 2011.

[89] A. de Jong, M. E. Hansen, O. P. Kuipers, M. Kilstrup, and J. Kok, “The Transcriptional and Gene Regulatory Network of Lactococcus lactis MG1363 during Growth in Milk”, *PLoS ONE*, vol. 8, no. 1, Jan. 2013.

[90] D. A. Rodionov, I. A. Rodionova, X. Li, *et al.*, “Transcriptional regulation of the carbohydrate utilization network in Thermotoga maritima”, *Frontiers in Microbiology*, vol. 4, no. AUG, 2013.

[91] H. Barzantny, J. Schröder, J. Strotmeier, E. Fredrich, I. Brune, and A. Tauch, “The transcriptional regulatory network of Corynebacterium jeikeium K411 and its interaction with metabolic routes contributing to human body odor formation”, *Journal of Biotechnology*, vol. 159, no. 3, pp. 235–248, Jun. 2012.

[92] K. Brinkrolf, I. Brune, and A. Tauch, “The transcriptional regulatory network of the amino acid producer Corynebacterium glutamicum”, *Journal of Biotechnology*, vol. 129, no. 2, pp. 191–211, Apr. 2007.

[93] D. A. Ravcheev, A. A. Best, N. V. Sernova, M. D. Kazanov, P. S. Novichkov, and D. A. Rodionov, “Genomic reconstruction of transcriptional regulatory networks in lactic acid bacteria”, *BMC Genomics*, vol. 14, no. 1, p. 94, Feb. 2013.

[94] J. J. Faith, B. Hayete, J. T. Thaden, *et al.*, "Large-scale mapping and validation of Escherichia coli transcriptional regulation from a compendium of expression profiles", *PLoS Biology*, 2007.

[95] R. Bonneau, M. T. Facciotti, D. J. Reiss, *et al.*, "A Predictive Model for Transcriptional Control of Physiology in a Free Living Cell", *Cell*, vol. 131, no. 7, pp. 1354–1365, Dec. 2007.

[96] J. K. Fredrickson, M. F. Romine, A. S. Beliaev, *et al.*, *Towards environmental systems biology of Shewanella*, Aug. 2008.

[97] E. Galán-Vásquez, B. Luna, and A. Martínez-Antonio, "The Regulatory Network of Pseudomonas aeruginosa", *Microbial Informatics and Experimentation*, vol. 1, no. 1, p. 3, 2011.

[98] T. UniProt Consortium, "UniProt: the universal protein knowledgebase", *Nucleic Acids Research*, vol. 46, no. 5, pp. 2699–2699, Mar. 2018.

[99] N. R. NCBI Resource Coordinators, "Database resources of the National Center for Biotechnology Information.", *Nucleic acids research*, vol. 44, no. D1, pp. 7–19, Jan. 2016.

[100] D. A. Benson, M. Cavanaugh, K. Clark, *et al.*, "GenBank", *Nucleic Acids Research*, vol. 41, no. D1, pp. D36–D42, Nov. 2012.

[101] T. Tatusova, S. Ciufo, B. Fedorov, K. O'Neill, and I. Tolstoy, "RefSeq microbial genomes database: New representation and annotation strategy", *Nucleic Acids Research*, vol. 42, no. D1, pp. 553–559, Jan. 2014.

[102] C. L. Schoch, S. Ciufo, M. Domrachev, *et al.*, "NCBI Taxonomy: a comprehensive update on curation, resources and tools", *Database : the journal of biological databases and curation*, vol. 2020, 2020.

[103] G. Zheng and T. Huang, "The reconstruction and analysis of gene regulatory networks", in *Methods in Molecular Biology*, vol. 1754, Humana Press Inc., 2018, pp. 137–154.

[104] D. Thompson, A. Regev, and S. Roy, "Comparative Analysis of Gene Regulatory Networks: From Network Reconstruction to Evolution", *Annual Review of Cell and Developmental Biology*, vol. 31, no. 1, pp. 399–428, Nov. 2015.

[105] R. Bonneau, D. J. Reiss, P. Shannon, *et al.*, "The inferelator: An algorithn for learning parsimonious regulatory networks from systems-biology data sets de novo", *Genome Biology*, 2006.

[106] J. P. Faria, R. Overbeek, R. C. Taylor, *et al.*, "Reconstruction of the Regulatory Network for Bacillus subtilis and Reconciliation with Gene Expression Data", *Frontiers in Microbiology*, vol. 7, p. 275, Mar. 2016.

[107] G. Karlebach and R. Shamir, "Modelling and analysis of gene regulatory networks", *Nature Reviews Molecular Cell Biology*, vol. 9, no. 10, pp. 770–780, Oct. 2008.

[108] D. A. Rodionov, "Comparative Genomic Reconstruction of Transcriptional Regulatory Networks in Bacteria", 2007.

[109] P. S. Novichkov, T. S. Brettin, E. S. Novichkova, *et al.*, "RegPrecise web services interface: programmatic access to the transcriptional regulatory interactions in bacteria reconstructed by comparative genomics", *Nucleic Acids Research*, vol. 40, no. W1, W604–W608, Jul. 2012.

[110] M. S. Gelfand, *Evolution of transcriptional regulatory networks in microbial genomes*, Jun. 2006.

[111] M. Madan Babu, S. A. Teichmann, and L. Aravind, "Evolutionary Dynamics of Prokaryotic Transcriptional Regulatory Networks", *Journal of Molecular Biology*, vol. 358, no. 2, pp. 614–633, Apr. 2006.

[112] M. Tompa, N. Li, T. L. Bailey, *et al.*, *Assessing computational tools for the discovery of transcription factor binding sites*, 2005.

[113] W. B. L. Alkema, B. Lenhard, and W. W. Wasserman, "Regulog analysis: Detection of conserved regulatory networks across bacteria: Application to Staphylococcus aureus", *Genome Research*, 2004.

[114] P. S. Novichkov, D. A. Rodionov, E. D. Stavrovskaya, *et al.*, "RegPredict: An integrated system for regulon inference in prokaryotes by comparative genomics approach", *Nucleic Acids Research*, 2010.

[115] R. Nicolle, F. Radvanyi, and M. Elati, "COREGNET: reconstruction and integrated analysis of co-regulatory networks", *Bioinformatics*, vol. 31, no. 18, pp. 3066–3068, Sep. 2015.

[116] G. Karlebach and R. Shamir, "Constructing logical models of gene regulatory networks by integrating transcription factor-DNA interactions with expression data: An entropy-based approach", *Journal of Computational Biology*, vol. 19, no. 1, pp. 30–41, Jan. 2012.

[117] V. A. Huynh-Thu, A. Irrthum, L. Wehenkel, and P. Geurts, "Inferring regulatory networks from expression data using tree-based methods", *PLoS ONE*, 2010.

[118] I. Gat-Viks and R. Shamir, "Refinement and expansion of signaling pathways: The osmotic response network in yeast", *Genome Research*, vol. 17, no. 3, pp. 358–367, Mar. 2007.

[119] I. Rocha, J. Förster, and J. Nielsen, "Design and Application of Genome-Scale Reconstructed Metabolic Models", *Methods in Molecular Biology, vol. 416: Microbial Gene Essentiality*, vol. 416, pp. 409–431, 2007.

[120] A. M. Feist, M. J. Herrgård, I. Thiele, J. L. Reed, and B. Palsson, "Reconstruction of biochemical networks in microorganisms", *Nature Reviews Microbiology*, vol. 7, no. 2, pp. 129–143, 2009.

[121] M. Hucka, A. Finney, H. M. Sauro, *et al.*, "The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models.", *Bioinformatics (Oxford, England)*, vol. 19, no. 4, pp. 524–31, Mar. 2003.

[122] O. Dias, M. Rocha, E. C. Ferreira, and I. Rocha, "Reconstructing genome-scale metabolic models with merlin", *Nucleic Acids Research*, vol. 43, no. 8, pp. 3899–3910, Apr. 2015.

[123] C. S. Henry, M. DeJongh, A. A. Best, P. M. Frybarger, B. Linsay, and R. L. Stevens, “High-throughput generation, optimization and analysis of genome-scale metabolic models.”, *Nature biotechnology*, vol. 28, no. 9, pp. 977–82, Sep. 2010.

[124] R. Agren, L. Liu, S. Shoaie, W. Vongsangnak, I. Nookaew, and J. Nielsen, “The RAVEN Toolbox and Its Use for Generating a Genome-scale Metabolic Model for Penicillium chrysogenum”, *PLoS Computational Biology*, vol. 9, no. 3, C. D. Maranas, Ed., e1002980, Mar. 2013.

[125] J. P. Faria, M. Rocha, I. Rocha, and C. S. Henry, “Methods for automated genome-scale metabolic model reconstruction”, *Biochemical Society Transactions*, vol. 46, no. 4, pp. 931–936, Aug. 2018.

[126] J. D. Orth, I. Thiele, and B. O. Palsson, “What is flux balance analysis?”, *Nature Publishing Group*, vol. 28, no. 3, pp. 245–248, 2010.

[127] N. E. Lewis, K. K. Hixson, T. M. Conrad, *et al.*, “Omic data from evolved E. coli are consistent with computed optimal growth from genome-scale models.”, *Molecular systems biology*, vol. 6, no. 1, p. 390, Jul. 2010.

[128] S. Gudmundsson and I. Thiele, “Computationally efficient flux variability analysis”, *BMC Bioinformatics*, vol. 11, no. 1, p. 489, Dec. 2010.

[129] L. Heirendt, S. Arreckx, T. Pfau, *et al.*, “Creation and analysis of biochemical constraint-based models using the COBRA Toolbox v.3.0”, *Nature Protocols*, vol. 14, no. 3, pp. 639–702, Mar. 2019.

[130] A. Ebrahim, J. A. Lerman, B. O. Palsson, and D. R. Hyduke, “COBRApy: COnstraints-Based Reconstruction and Analysis for Python”, *BMC Systems Biology*, vol. 7, no. 1, p. 74, Aug. 2013.

[131] P. Vilaça, I. Rocha, and M. Rocha, “A computational tool for the simulation and optimization of microbial strains accounting integrated metabolic/regulatory information”, *Biosystems*, vol. 103, no. 3, pp. 435–441, Mar. 2011.

[132] J. D. Orth, T. M. Conrad, J. Na, *et al.*, “A comprehensive genome￾scale reconstruction of Escherichia coli metabolism—2011”, *Molecular Systems Biology*, vol. 7, no. 1, p. 535, Jan. 2011.

[133] J. A. Lerman, D. R. Hyduke, H. Latif, *et al.*, “In silico method for modelling metabolism and gene product expression at genome scale”, *Nature Communications*, vol. 3, no. 1, p. 929, Jan. 2012.

[134] A. S. Blazier and J. A. Papin, “Integration of expression data in genome-scale metabolic network reconstructions”, *Frontiers in Physiology*, vol. 3, p. 299, Aug. 2012.

[135] J. Kim and J. L. Reed, “Refining metabolic models and accounting for regulatory effects”, *Current Opinion in Biotechnology*, vol. 29, pp. 34–38, Oct. 2014.

[136] S. Imam, S. Schäuble, A. N. Brooks, N. S. Baliga, and N. D. Price, “Data-driven integration of genome-scale regulatory and metabolic network models.”, *Frontiers in microbiology*, vol. 6, p. 409, 2015.

[137] E. J. O'Brien and B. O. Palsson, "Computing the functional proteome: recent progress and future prospects for genome-scale models", *Current Opinion in Biotechnology*, vol. 34, pp. 125–134, Aug. 2015.

[138] E. J. O'Brien, J. Utrilla, and B. O. Palsson, "Quantification and Classification of E. coli Proteome Utilization and Unused Protein Costs across Environments", *PLOS Computational Biology*, vol. 12, no. 6, C. D. Maranas, Ed., e1004998, Jun. 2016.

[139] S. Opdam, A. Richelle, B. Kellman, S. Li, D. C. Zielinski, and N. E. Lewis, "A Systematic Evaluation of Methods for Tailoring Genome-Scale Metabolic Models", *Cell Systems*, vol. 4, no. 3, pp. 318–329, Mar. 2017.

[140] T. Hao, D. Wu, L. Zhao, Q. Wang, E. Wang, and J. Sun, *The genome-scale integrated networks in microorganisms*, Feb. 2018.

[141] C. J. Lloyd, A. Ebrahim, L. Yang, *et al.*, "COBRAme: A computational framework for genome-scale models of metabolism and gene expression", *PLOS Computational Biology*, vol. 14, no. 7, A. E. Darling, Ed., e1006302, Jul. 2018.

[142] E. Brunk, R. L. Chang, J. Xia, *et al.*, "Characterizing posttranslational modifications in prokaryotic metabolism using a multiscale workflow", *Proceedings of the National Academy of Sciences*, vol. 115, no. 43, pp. 11 096–11 101, Oct. 2018.

[143] E. Gonçalves, M. Sciacovelli, A. S. Costa, *et al.*, "Post-translational regulation of metabolism in fumarate hydratase deficient cancer cells", *Metabolic Engineering*, vol. 45, pp. 149–157, Jan. 2018.

[144] M. W. Covert, E. M. Knight, J. L. Reed, M. J. Herrgard, and B. O. Palsson, "Integrating high-throughput and computational data elucidates bacterial networks", *Nature*, vol. 429, no. 6987, pp. 92–96, May 2004.

[145] T. Shlomi, Y. Eisenberg, R. Sharan, and E. Ruppin, "A genome-scale computational study of the interplay between transcriptional regulation and metabolism.", *Molecular systems biology*, vol. 3, p. 101, Apr. 2007.

[146] S. Chandrasekaran and N. D. Price, "Probabilistic integrative modeling of genome-scale metabolic and regulatory networks in Escherichia coli and Mycobacterium tuberculosis.", *Proceedings of the National Academy of Sciences of the United States of America*, vol. 107, no. 41, pp. 17 845–50, Oct. 2010.

[147] M. J. Herrgård, B.-S. Lee, V. Portnoy, and B. Ø. Palsson, "Integrated analysis of regulatory and metabolic networks reveals novel regulatory mechanisms in Saccharomyces cerevisiae.", *Genome research*, vol. 16, no. 5, pp. 627–35, May 2006.

[148] D. T. Banos, P. Trébulle, and M. Elati, "Integrating transcriptional activity in genome-scale models of metabolism", *BMC Systems Biology*, vol. 11, no. S7, p. 134, Dec. 2017.

[149] E. Motamedian, M. Mohammadi, S. A. Shojaosadati, and M. Heydari, "TRFBA: an algorithm to integrate genome-scale metabolic and transcriptional regulatory networks with incorporation of expression data", *Bioinformatics*, vol. 33, no. 7, btw772, Jan. 2017.

[150] Z. Wang, S. A. Danziger, B. D. Heavner, *et al.*, "Combining inferred regulatory and reconstructed metabolic networks enhances phenotype prediction in yeast", *PLOS Computational Biology*, vol. 13, no. 5, J. Nielsen, Ed., e1005489, May 2017.

[151] L. Marmiesse, R. Peyraud, and L. Cottret, "FlexFlux: combining metabolic flux and regulatory network analyses", *BMC Systems Biology*, vol. 9, no. 1, p. 93, Dec. 2015.

[152] P. A. Jensen, K. A. Lutz, and J. A. Papin, "TIGER: Toolbox for integrating genome-scale metabolic models, expression data, and transcriptional regulatory networks", *BMC Systems Biology*, vol. 5, no. 1, p. 147, Sep. 2011.

[153] M. L. Jenior, T. J. Moutinho, B. V. Dougherty, and J. A. Papin, "Transcriptome-guided parsimonious flux analysis improves predictions with metabolic networks in complex environments", *PLoS Computational Biology*, vol. 16, no. 4, e1007099, Apr. 2020.

[154] C. Angione and P. Lió, "Predictive analytics of environmental adaptability in multi-omic network models", *Scientific Reports*, vol. 5, Oct. 2015.

[155] C. Angione, M. Conway, and P. Lió, "Multiplex methods provide effective integration of multi-omic data in genome-scale models", *BMC Bioinformatics*, vol. 17, no. S4, p. 83, Feb. 2016.

[156] N. Töpfer, S. Jozefczuk, and Z. Nikoloski, "Integration of time-resolved transcriptomics data with flux-based methods reveals stress-induced metabolic adaptation in Escherichia coli", *BMC Systems Biology*, vol. 6, Nov. 2012.

[157] S. B. Collins, E. Reznik, and D. Segrè, "Temporal Expression-based Analysis of Metabolism", *PLoS Computational Biology*, vol. 8, no. 11, Nov. 2012.

[158] A. Navid and E. Almaas, "Genome-level transcription data of Yersinia pestis analyzed with a New metabolic constraint-based approach", *BMC Systems Biology*, vol. 6, Dec. 2012.

[159] X. Fang, A. Wallqvist, and J. Reifman, "Modeling Phenotypic Metabolic Adaptations of Mycobacterium tuberculosis H37Rv under Hypoxia", *PLoS Computational Biology*, vol. 8, no. 9, Sep. 2012.

[160] R. J. P. van Berlo, D. de Ridder, J.-M. Daran, P. A. S. Daran-Lapujade, B. Teusink, and M. J. T. Reinders, "Predicting Metabolic Fluxes Using Gene Expression Differences As Constraints", *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 8, no. 1, pp. 206–216, Jan. 2011.

[161] D. Lee, K. Smallbone, W. B. Dunn, *et al.*, "Improving metabolic flux predictions using absolute gene expression data", *BMC Systems Biology*, vol. 6, no. 1, p. 73, Jun. 2012.

[162] P. A. Jensen and J. A. Papin, "Functional integration of a metabolic network model and expression data without arbitrary thresholding", *Bioinformatics*, vol. 27, no. 4, pp. 541–547, Feb. 2011.

[163] M. Åkesson, J. Förster, and J. Nielsen, “Integration of gene expression data into genome-scale metabolic models”, *Metabolic Engineering*, vol. 6, no. 4, pp. 285–293, Oct. 2004.

[164] L. Jerby, T. Shlomi, and E. Ruppin, “Computational reconstruction of tissue-specific metabolic models: Application to human liver metabolism”, *Molecular Systems Biology*, vol. 6, 2010.

[165] T. Shlomi, M. N. Cabili, M. J. Herrgård, B. Palsson, and E. Ruppin, *Network-based prediction of human tissue-specific metabolism*, Sep. 2008.

[166] S. A. Becker and B. O. Palsson, “Context-specific metabolic networks are consistent with experiments”, *PLoS Computational Biology*, vol. 4, no. 5, May 2008.

[167] H. Zur, E. Ruppin, and T. Shlomi, “iMAT: An integrative metabolic analysis tool”, *Bioinformatics*, vol. 26, no. 24, pp. 3140–3142, Dec. 2010.

[168] R. Agren, S. Bordel, A. Mardinoglu, N. Pornputtapong, I. Nookaew, and J. Nielsen, “Reconstruction of genome-scale active metabolic networks for 69 human cell types and 16 cancer types using INIT”, *PLoS Computational Biology*, vol. 8, no. 5, May 2012.

[169] Y. Wang, J. A. Eddy, and N. D. Price, “Reconstruction of genome-scale metabolic models for 126 human tissues using mCADRE”, *BMC Systems Biology*, vol. 6, Dec. 2012.

[170] B. J. Schmidt, A. Ebrahim, T. O. Metz, J. N. Adkins, B. Ø. Palsson, and D. R. Hyduke, “GIM3E: condition-specific models of cellular metabolism developed from metabolomics and expression data”, *Bioinformatics*, vol. 29, no. 22, pp. 2900–2908, Nov. 2013.

[171] S. Rossell, M. A. Huynen, and R. A. Notebaart, “Inferring Metabolic States in Uncharacterized Environments Using Gene-Expression Measurements”, *PLoS Computational Biology*, vol. 9, no. 3, 2013.

[172] A. Schultz and A. A. Qutub, “Reconstruction of Tissue-Specific Metabolic Networks Using CORDA”, *PLoS Computational Biology*, vol. 12, no. 3, Mar. 2016.

[173] M. W. Covert and B. O. Palsson, “Constraints-based models: Regulation of Gene Expression Reduces the Steady-state Solution Space”, *Journal of Theoretical Biology*, vol. 221, no. 3, pp. 309–325, Apr. 2003.

[174] M. W. Covert and B. Ø. Palsson, “Transcriptional regulation in constraints-based metabolic models of Escherichia coli.”, *The Journal of biological chemistry*, vol. 277, no. 31, pp. 28 058–64, Aug. 2002.

[175] C. Chaouiya, D. Bérenguier, S. M. Keating, *et al.*, “SBML qualitative models: a model representation format and infrastructure to foster interactions between qualitative modelling formalisms and tools”, *BMC Systems Biology*, vol. 7, no. 1, p. 135, Dec. 2013.

[176] S. Ma, K. J. Minch, T. R. Rustad, *et al.*, “Integrated Modeling of Gene Regulatory and Metabolic Networks in Mycobacterium tuberculosis”, *PLOS Computational Biology*, vol. 11, no. 11, e1004543, Nov. 2015.

[177] A. N. Brooks, D. J. Reiss, A. Allard, *et al.*, “A system￾level model for the microbial regulatory genome”, *Molecular Systems Biology*, vol. 10, no. 7, p. 740, Jul. 2014.

[178] M. C. Teixeira, P. T. Monteiro, M. Palma, *et al.*, “YEASTRACT: An upgraded database for the analysis of transcription regulatory networks in Saccharomyces cerevisiae”, *Nucleic Acids Research*, vol. 46, no. D1, pp. D348–D353, Jan. 2018.

[179] R. Mahadevan, J. S. Edwards, and F. J. Doyle, “Dynamic Flux Balance Analysis of diauxic growth in Escherichia coli”, *Biophysical Journal*, vol. 83, no. 3, pp. 1331–1340, 2002.

[180] C. Colijn, A. Brandes, J. Zucker, *et al.*, “Interpreting Expression Data with Metabolic Flux Models: Predicting Mycobacterium tuberculosis Mycolic Acid Production”, *PLoS Computational Biology*, vol. 5, no. 8, J. A. Papin, Ed., e1000489, Aug. 2009.

[181] R. Mahadevan and C. H. Schilling, “The effects of alternate optimal solutions in constraint-based genome-scale metabolic models”, *Metabolic Engineering*, vol. 5, no. 4, pp. 264–276, 2003.

[182] S. Schuster, D. A. Fell, and T. Dandekar, “A general definition of metabolic pathways useful for systematic organization and analysis of complex metabolic networks”, *Nature Biotechnology*, vol. 18, no. 3, pp. 326–332, 2000.

[183] R. Vivek-Ananth and A. Samal, “Advances in the integration of transcriptional regulatory information into genome-scale metabolic models”, *Biosystems*, vol. 147, pp. 1–10, Sep. 2016.

[184] J. Kim and J. L. Reed, “No Title”, vol. 4, no. 1, p. 53, Apr. 2010.

[185] F. Shen, R. Sun, J. Yao, *et al.*, “OptRAM: In-silico strain design via integrative regulatory-metabolic network modeling”, *PLOS Computational Biology*, vol. 15, no. 3, C. A. Ouzounis, Ed., e1006835, Mar. 2019.

[186] J. Monk, J. Nogales, and B. O. Palsson, “Optimizing genome-scale network reconstructions”, *Nature Biotechnology*, vol. 32, no. 5, pp. 447–452, May 2014.

[187] S. Prigent, C. Frioux, S. M. Dittami, *et al.*, “Meneco, a Topology-Based Gap-Filling Tool Applicable to Degraded Genome-Wide Metabolic Networks”, *PLOS Computational Biology*, vol. 13, no. 1, C. Kaleta, Ed., e1005276, Jan. 2017.

[188] V. Satish Kumar, M. S. Dasika, and C. D. Maranas, “Optimization based automated curation of metabolic reconstructions”, *BMC Bioinformatics*, vol. 8, no. 1, p. 212, Jun. 2007.

[189] I. Thiele, N. Vlassis, and R. M. Fleming, “FASTGAPFILL: Efficient gap filling in metabolic networks”, *Bioinformatics*, vol. 30, no. 17, pp. 2529–2531, Sep. 2014.

[190] Z. Hosseini and S. A. Marashi, “Discovering missing reactions of metabolic networks by using gene co-expression data”, *Scientific Reports*, vol. 7, no. 1, pp. 1–12, Feb. 2017.

[191] J. L. Reed, T. R. Patel, K. H. Chen, *et al.*, “Systems approach to refining genome annotation”, *Proceedings of the National Academy of Sciences of the United States of America*, vol. 103, no. 46, pp. 17 480–17 484, Nov. 2006.

[192] E. Vitkin and T. Shlomi, “MIRAGE: a functional genomics-based approach for metabolic network model reconstruction and its application to cyanobacteria networks”, *Genome biology*, vol. 13, no. 11, R111, Nov. 2012.

[193] Z. A. King, J. Lu, A. Dräger, *et al.*, “BiGG Models: A platform for integrating, standardizing and sharing genome-scale models”, *Nucleic Acids Research*, vol. 44, no. D1, pp. D515–D522, Jan. 2016.

[194] K. S. Makarova, A. Slesarev, Y. Wolf, *et al.*, “Comparative genomics of the lactic acid bacteria.”, *Proceedings of the National Academy of Sciences of the United States of America*, vol. 103, no. 42, pp. 15 611–15 616, 2006.

[195] A. Bolotin, B. Quinquis, P. Renault, *et al.*, “Complete sequence and comparative genome analysis of the dairy bacterium Streptococcus thermophilus.”, *Nature biotechnology*, vol. 22, no. 12, pp. 1554–8, 2004.

[196] M. van de Guchte, S. Penaud, C. Grimaldi, *et al.*, “The complete genome sequence of Lactobacillus bulgaricus reveals extensive and ongoing reductive evolution.”, *Proceedings of the National Academy of Sciences of the United States of America*, vol. 103, no. 24, pp. 9274–9, 2006.

[197] K. A. Pagel, V. Pejaver, G. N. Lin, *et al.*, “When loss-of-function is loss of function: Assessing mutational signatures and impact of loss-of-function genetic variants”, in *Bioinformatics*, vol. 33, Oxford University Press, Jul. 2017, pp. i389–i398.

[198] M. Hucka, F. T. Bergmann, A. Dräger, *et al.*, “The Systems Biology Markup Language (SBML): Language Specification for Level 3 Version 2 Core”, *Journal of integrative bioinformatics*, vol. 15, no. 1, Mar. 2018.

[199] J. Capela, D. Lagoa, R. Rodrigues, *et al.*, “merlin, an improved framework for the reconstruction of high-quality genome-scale metabolic models”, *Nucleic Acids Research*, vol. 50, no. 11, pp. 6052–6066, Jun. 2022.

[200] T. Österlund, I. Nookaew, S. Bordel, and J. Nielsen, “Mapping condition-dependent regulation of metabolism in yeast through genome-scale modeling”, *BMC Systems Biology*, vol. 7, no. 1, p. 36, Apr. 2013.

[201] J. Hoskins, J. Alborn, J. Arnold, *et al.*, “Genome of the bacterium Streptococcus pneumoniae strain R6”, *Journal of Bacteriology*, vol. 183, no. 19, pp. 5709–5717, Oct. 2001.

[202] C. Goble and R. Stevens, “State of the nation in data integration for bioinformatics”, *Journal of Biomedical Informatics*, vol. 41, no. 5, pp. 687–693, Oct. 2008.

[203] K. Dolinski and O. G. Troyanskaya, “Implications of Big Data for cell biology”, *https://doi.org/10.1091/mbc.E13-12-0756*, vol. 26, no. 14, pp. 2575–2578, Oct. 2017.

[204] R. Kimball and M. Ross, *The data warehouse toolkit : the complete guide to dimensional modeling*, 2nd Edition. Wiley, 2002.

[205] Z. E. Akkaoui, E. Zimányi, J.-N. Mazón, and J. Trujillo, “A Model-Driven Framework for ETL Process Development”, *Proceedings of the ACM 14th international workshop on Data Warehousing and OLAP - DOLAP ’11*, 2011.

[206] F. Holzschuher and R. Peinl, “Performance of Graph Query Languages Comparison of Cypher, Gremlin and Native Access in Neo4j”, *Proceedings of the Joint EDBT/ICDT 2013 Workshops on - EDBT ’13*, 2013.

[207] X. Fang, A. Sastry, N. Mih, *et al.*, “Global transcriptional regulatory network for Escherichia coli robustly connects gene expression to transcription factor activities”, *Proceedings of the National Academy of Sciences of the United States of America*, vol. 114, no. 38, pp. 10 286–10 291, Sep. 2017.

[208] P. J. Cock, T. Antao, J. T. Chang, *et al.*, “Biopython: freely available Python tools for computational molecular biology and bioinformatics”, *Bioinformatics*, vol. 25, no. 11, pp. 1422–1423, Jun. 2009.

[209] E. W. Sayers, E. E. Bolton, J. R. Brister, *et al.*, “Database resources of the national center for biotechnology information”, *Nucleic acids research*, vol. 50, no. D1, pp. D20–D26, Jan. 2022.

[210] C. Lee and C. H. Huang, “LASAGNA: A novel algorithm for transcription factor binding site alignment”, *BMC Bioinformatics*, vol. 14, no. 1, pp. 1–13, Mar. 2013.

[211] N. Dedić and C. Stanier, “An evaluation of the challenges of multilingualism in data warehouse development”, *ICEIS 2016 - Proceedings of the 18th International Conference on Enterprise Information Systems*, vol. 1, pp. 196–206, 2016.

[212] I. A. Suvorova, Y. D. Korostelev, and M. S. Gelfand, “GntR Family of Bacterial Transcription Factors and Their DNA Binding Motifs: Structure, Positioning and Co-Evolution”, *PLOS ONE*, vol. 10, no. 7, e0132618, 2015.

[213] F. M. Camas, E. J. Alm, and J. F. Poyatos, “Local Gene Regulation Details a Recognition Code within the LacI Transcriptional Factor Family”, *PLOS Computational Biology*, vol. 6, no. 11, e1000989, Nov. 2010.

[214] A. L. Colclough, J. Scadden, and J. M. Blair, “TetR-family transcription factors in Gram-negative bacteria: Conservation, variation and implications for efflux-mediated antimicrobial resistance”, *BMC Genomics*, vol. 20, no. 1, pp. 1–12, Oct. 2019.

[215] B. Troxell and H. M. Hassan, “Transcriptional regulation by Ferric Uptake Regulator (Fur) in pathogenic bacteria”, *Frontiers in cellular and infection microbiology*, vol. 3, no. OCT, 2013.

[216] N. L. Brown, J. V. Stoyanov, S. P. Kidd, and J. L. Hobman, “The MerR family of transcriptional regulators”, *FEMS Microbiology Reviews*, vol. 27, no. 2-3, pp. 145–163, Jun. 2003.

[217] I. Rocha, P. Maia, P. Evangelista, *et al.*, “OptFlux: an open-source software platform for in silico metabolic engineering.”, *BMC systems biology*, vol. 4, no. 1, p. 45, 2010.

[218] B. J. Sánchez, C. Zhang, A. Nilsson, P.-J. Lahtvee, E. J. Kerkhoven, and J. Nielsen, "Improving the phenotype predictions of a yeast genome-scale metabolic model by incorporating enzymatic constraints.", *Molecular systems biology*, vol. 13, no. 8, p. 935, Aug. 2017.

[219] J. Kim and J. L. Reed, "OptORF: Optimal metabolic and regulatory perturbations for metabolic engineering of microbial strains.", *BMC systems biology*, vol. 4, no. 1, p. 53, Apr. 2010.

[220] J. G. Cardoso, K. Jensen, C. Lieven, *et al.*, "Cameo: A Python Library for Computer Aided Metabolic Engineering and Optimization of Cell Factories", *ACS Synthetic Biology*, vol. 7, no. 4, pp. 1163–1166, Apr. 2018.

[221] P. S. Bekiaris and S. Klamt, "Automatic construction of metabolic models with enzyme constraints", *BMC Bioinformatics*, vol. 21, no. 1, pp. 1–13, Jan. 2020.

[222] A. Tonda, "Inspyred: Bio-inspired algorithms in Python", *Genetic Programming and Evolvable Machines*, vol. 21, no. 1-2, pp. 269–272, Jun. 2020.

[223] A. Benítez-Hidalgo, A. J. Nebro, J. García-Nieto, I. Oregi, and J. Del Ser, "jMetalPy: a Python Framework for Multi-Objective Optimization with Metaheuristics", *Swarm and Evolutionary Computation*, vol. 51, Mar. 2019.

[224] J. D. Orth, R. M. T. Fleming, and B. Ø. Palsson, " Reconstruction and Use of Microbial Metabolic Networks: the Core Escherichia coli Metabolic Model as an Educational Guide ", *EcoSal Plus*, vol. 4, no. 1, Jan. 2010.

[225] J. L. Reed, T. D. Vo, C. H. Schilling, and B. O. Palsson, "An expanded genome-scale model of Escherichia coli K-12 (iJR904 GSM/GPR).", *Genome biology*, vol. 4, no. 9, pp. 1–12, Aug. 2003.

[226] N. Jamshidi and B. Palsson, "Investigating the metabolic capabilities of Mycobacterium tuberculosis H37Rv using the in silico strain iNJ661 and proposing alternative drug targets", *BMC systems biology*, vol. 1, Jun. 2007.

[227] G. Balázsi, A. P. Heath, L. Shi, and M. L. Gennaro, "The temporal response of the Mycobacterium tuberculosis gene regulatory network during growth arrest", *Molecular Systems Biology*, vol. 4, no. 1, p. 225, Nov. 2008.

[228] M. L. Mo, B. Palsson, and M. J. Herrgård, "Connecting extracellular metabolomic measurements to intracellular flux states in yeast", *BMC Systems Biology*, vol. 3, no. 1, pp. 1–17, Mar. 2009.

[229] M. J. Brauer, A. J. Saldanha, K. Dolinski, and D. Botstein, "Homeostatic adjustment and metabolic remodeling in glucose-limited yeast cultures", *Molecular Biology of the Cell*, vol. 16, no. 5, pp. 2503–2517, May 2005.

I

Supplementary Material

I.1 Supplementary material 1

Supplementary material 1 – File with the year of publication, availability of a tool with a user-friendly interface (namely a GUI without the requirement of coding skills), type of reaction constraint formulation, as well as the organism used for proof of concept in the methods for integrating TRNs into GEMs. Available at https://doi.org/10.1042/BST20190840.

I.2 Supplementary material 2

Supplementary material 2 – A comparison of BioISO with other state-of-the-art tools for gap-finding and gap-filling. Available at https://www.dropbox.com/s/kl35hexineyz377/supp_2.pdf?dl=1.

I.3 Supplementary material 3

Supplementary material 3 – A suvery of several state-of-the-art tools for gap-finding and gap-filling. Available at https://www.dropbox.com/s/rl4qsc4rsxufuy4/supp_3.xlsx?dl=1.

I.4 Supplementary material 4

Supplementary material 4 – A detailed description of BioISO's workflow, algorithm, and runtime analysis. Available at https://www.dropbox.com/s/m8hxbyoa7apmk4n/supp_4.pdf?dl=1.

I.5 Supplementary material 5

Supplementary material 5 – A detailed description of BioISO's validation methodology and materials. Available at https://www.dropbox.com/s/04y4m6hiop6t8ya/supp_5.pdf?dl=1.

I.6 Supplementary material 6

Supplementary material 6 – Results obtained in the BioISO's assessments: BioISO's depth analysis and exhaustive versus guided-searches. Available at https://www.dropbox.com/s/0m0j859cpce4bmg/supp_6.xlsx?dl=1.

I.7 Supplementary material 7

Supplementary material 7 – The detailed schema used to represent the graph store model of the CDS database. This schema includes all entities, relationships, and properties implemented in the CDS graph store model. Available at https://www.dropbox.com/s/8v1x9cz7iil2ts5/supp_7.xlsx?dl=1.

I.8 Supplementary material 8

Supplementary material 8 – Survey all objects and properties extracted from the resources of regulatory data. In addition to extracted data, this survey includes mapping between extracted and transformed objects integrated later into theCDS. Available at https://www.dropbox.com/s/nwwdreriyd8s7jn/supp_8.xlsx?dl=1.

I.9 Supplementary material 9

Supplementary material 9 – The integrated *E. coli* core model. This GERM model comprehends the *E. coli* core GEM model [224] and *E. coli* core TRN [224] . Available at MEWpy examples - e_coli_core.xml and MEWpy examples - e_coli_core_trn.csv.

I.10 Supplementary material 10

Supplementary material 10 – The integrated *E. coli* core model [224]. This GERM model comprehends the *E. coli* core GEM model [224] and *E. coli* core TRN [224]. Available at MEWpy examples - e_coli_core.xml and MEWpy examples - e_coli_core_trn.csv.

I.11 Supplementary material 11

Supplementary material 11 – The integrated iMC1010 model [144]. This GERM model comprehends the *E. coli* iJR904 GEM model [225] and *E. coli* iMC1010 TRN [144]. Available at MEWpy examples - iJR904.xml and MEWpy examples - iMC1010.csv.

I.12 Supplementary material 12

Supplementary material 12 – The integrated iNJ661 model [146]. This GERM model comprehends the *M. tuberculosis* iNJ661 GEM model and *M. tuberculosis* H37Rv TRN published by Balazsi *et al.* [227]. It also includes a gene expression dataset with 437 experiments (regulator deletions) [146]. Available at MEWpy examples - iNJ661.xml, MEWpy examples - iNJ661_trn.csv, and MEWpy examples - iNJ661_gene_expression.csv.

I.13 Supplementary material 13

Supplementary material 13 – The integrated iMM904 model [148]. This GERM model comprehends the *S. cerevisiae* iMM904 GEM model [228] and *S. cerevisiae* S288C TRN inferred by CoRegNet [115]. It also includes a gene expression dataset of 247 experiments [84], influence scores inferred by CoRegNet [115], and another gene expression dataset of 12-time points [229]. Available at MEWpy examples - iMM904.xml, MEWpy examples - iMM904_trn.csv, MEWpy examples - iMM904_gene_expression.csv, MEWpy examples - iMM904_influence.csv, and MEWpy examples - iMM904_experiments.csv.