TÉCNICO LISBOA

Development of Python library to integrate metabolic and regulatory networks

Sebastião Zoio Martins Ferreira dos Santos

Thesis to obtain the Master of Science Degree in

Computer Science and Engineering

Supervisors: Prof. Pedro Tiago Gonçalves Monteiro
Prof. Emanuel José Vieira Gonçalves

Examination Committee

Chairperson: Prof. Rui Filipe Fernandes Prada
Supervisor: Prof. Pedro Tiago Gonçalves Monteiro
Member of the Committee: Prof. Daniel Machado

October 2023

Declaration
I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

First and foremost, I would like to sincerely thank my supervisors, Prof. Emanuel Gonçalves and Prof. Pedro Monteiro, for your invaluable guidance, mentorship, availability and support. You were always available to answer my questions and to share your knowledge with me. I am truly grateful to have had the opportunity to develop my work with you. Thank you so much.

To my friends and colleagues at IST, who have been an essential part of my academic experience, I am profoundly thankful. Pedro Luis, Goncalo Carreira, Ruben Gualdino, and Simao Paiva. Our shared experiences have added immeasurable value to my time at the institute.

I owe an immeasurable debt of gratitude to my family. To my parents, whose unwavering support has been crucial throughout this academic journey. To my parents, sister and grandmother I am forever grateful.

To each and every one of you – Thank you.

Abstract

Traditionally, metabolic and regulatory networks have been modeled separately, leading to the development of specialized tools for each. However, the integration of these networks is essential for a comprehensive understanding of cellular behaviour and for making accurate biological predictions, as changes in one network can have downstream effects on the other. This thesis introduces a tool for integrating, simulating, and analyzing metabolic and regulatory networks. The tool extends the MEWpy metabolic modeling tool, focusing on its integrated models module. This extension combines metabolic model representations from external tools, such as COBRApy and ReFramed, with regulatory model representations in MEWpy. This framework enables users to work with integrated models and provides a wide range of methods for simulating and analyzing metabolic, regulatory, and integrated models. Incorporating an external tool to represent metabolic components enhances MEWpy's adaptability, allowing for seamless integration of updates and changes in metabolic techniques. It also provides access to additional functionalities and methods from the external tool. A set of unit tests were performed to validate the tool's functionality and compatibility. In addition, a use case was also conducted, in which the tool's advantages and relevance in practical applications is demonstrated. The use case includes the integration of a GECKO and a regulatory model from the *E. coli* organism. Taken together, the implemented tool enables the integration of two cellular networks, which improves accuracy in representation and simulations of real-life cellular behaviour. It brings together previously independent frameworks under a single architecture, leveraging the capabilities of both.

Keywords

Metabolic modelling · Regulatory networks · Optimization · Integrated tool

Resumo

Tradicionalmente, as redes metabólicas e regulatórias foram modeladas separadamente, resultando no desenvolvimento de ferramentas específicas para cada uma. No entanto, a integração destas redes é essencial para uma compreensão abrangente do comportamento celular e para obter previsões biológicas precisas, uma vez que as alterações em uma rede podem ter efeitos na outra. Esta tese apresenta o desenvolvimento de uma ferramenta para a integração, simulação e análise de redes metabólicas e regulatórias. A ferramenta desenvolvida extende as capacidades da ferramenta de modelação metabólica MEWpy, com um foco específico no módulo de modelos integrados. Esta extensão envolve a integração de modelos metabólicos de ferramentas externas consolidadas com modelos regulatórios do MEWpy. O resultado é uma *framework* que permite aos utilizadores representar modelos integrados e oferece um conjunto abrangente de métodos para simular e analisar modelos metabólicos, regulatórios e integrados. A incorporação de uma ferramenta externa para representar as componentes metabólicas aprimora a adaptabilidade do MEWpy e oferece funcionalidades adicionais ao disponibilizar acesso a funções da ferramenta externa. Foram realizados um conjunto de testes unitários para validar a funcionalidade e compatibilidade da ferramenta. Adicionalmente, foi realizado um estudo de caso, no qual são demonstradas as vantagens e relevância da ferramenta em aplicações práticas. O estudo de caso inclui a integração de um modelo GECKO e um modelo regulatório do organismo *E. coli*. Numa perspetiva geral, a ferramenta implementada permite a integração de duas redes celulares, melhorando a precisão na simulação do comportamento celular e reúne *frameworks* independentes sob a mesma arquitetura, permitindo usufruir de funcionalidades de ambas.

Palavras Chave

Modelação Metabólica; Redes Regulatórias; Optmização; Ferramenta Integrada;

Contents

List of Figures

List of Tables

Acronyms

COBRA	COnstraint-Based Reconstruction and Analysis
CSO	computational strain optimization
EA	Evolutionary Algorithm
EFM	Elementary Flux Mode
FBA	Flux Balance Analysis
FVA	Flux Variability Analysis
GECKO	Genome-scale metabolic models with Enzymatic Constraints using Kinetics and Omics
GEM	Genome-scale Model
GERM	Genome-scale Regulatory and Metabolic
GINsim	Gene Interaction Network Simulator
GPR	Gene-Protein-Reaction
iFBA	Integrated Flux Balance Analysis
iFVA	Integrated Flux Variability Analysis
KO	KnockOut
lMOMA	linear Minimization of Metabolic Adjustment
LP	Linear Programming
MOMA	Minimization of Metabolic Adjustment
MRS	Metabolic Regulatory Steady-state
ODE	Ordinary Differential Equation
pFBA	Parsimonious enzyme Flux Balance Analysis
PROM	Probabilistic Regulation of Metabolism
QP	Quadratic Programming

rFBA	Regulatory Flux Balance Analysis
SA	Simulated Annealing
SCC	Strongly Connected Component
SR-FBA	Steady-state regulatory Flux Balance Analysis
STG	State Transition Graph
TF	Transcription Factor
TRN	Transcriptional Regulatory Network

1

Introduction

Contents

Mathematical modelling has become a crucial tool in understanding cellular processes and systems. Some models are more detailed and complex than others, and the level of detail and complexity can depend on the specific goals of the modeling effort. Amongst the various models, logic and discrete models are particularly popular due to their simplicity when compared to more complex models. This simplicity allows them to be more scalable while still being able to replicate the expected behavior of biological systems. Metabolic networks and regulatory networks are both essential cellular systems that can be effectively represented through mathematical models. Metabolic models focus on quantifying the flow of metabolites through cellular pathways. These models rely on optimization techniques to predict how cells allocate resources and energy to different metabolic reactions. On the other hand, regulatory models investigate the complex network of gene regulation and signaling pathways within cells. These models aim to elucidate how genes are turned on or off in response to various signals and environmental conditions.

In recent years, there has been a growing interest in the integration of metabolic and regulatory networks

for simulation purposes. These networks are essential components of a cell, and their interactions play a critical role in determining the overall behavior of the cell. Integrating these networks allows for a better understanding of the complex interactions and relationships between them. For instance, changes in gene expression can have downstream effects on metabolism, which in turn can affect gene expression. Having integrated models enables a more accurate representation of the behavior of the cell, and ultimately, simulation results that more closely replicate real-life behavior.

Most models focus on specific subsystems or layers, such as metabolism or gene regulation. Some models and methods already exist that integrate metabolic and regulatory networks, however only few tools are able to support them.

1.1 Objective and Thesis outline

This thesis aims to develop a user-friendly tool that brings together metabolic and regulatory networks. This tool will enable researchers to run simulations, helping them understand how cells function. Ultimately, it will improve the ability to predict and control cellular behavior. The remainder of the thesis is organized as follows: Chapter 2 presents fundamental concepts that support the theme of this dissertation, including descriptions of model representations and simulation methods. Chapter 3 describes some existing tools for analysis and simulations on metabolic, regulatory and integrated metabolic and regulatory models. Chapter 4 details the solution implementation. Chapter 5 presents the results and evaluation of the solution and demonstrates its utility in a use case. Lastly chapter 6 summarizes the main conclusions of this work, and points to possible future work.

2

Background

Contents

Molecular biology held the promise that understanding the functions of the molecules within cells would provide profound insights into the workings of cells themselves. Today, we find ourselves equipped with a growing number of datasets detailing the composition of cells and organisms under specific conditions. The intricate chemical interactions among these components are now well-documented, leading to the construction of genome-scale biochemical reaction networks — an essential feature of molecular systems biology. The ultimate goal of (molecular) systems biology is to bridge the gap between the diverse chemical components within a cell, with their genetic foundations, and the resulting physiological functions. The advancement in the ability to identify biological components, delineate their interactions, and compile genome-scale data has given rise to a fundamental paradigm in systems biology, comprised of four key steps (Figure 2.1):

1. Initially, the biological components participating in a specific cellular process are meticulously defined and enumerated.

2. Next, begins the study of interactions between these components, essentially creating 'wiring diagrams' for genetic circuits. This process, known as biochemical reaction network reconstruction, unfolds step by step.

3. The reconstructed networks are then translated into a formal mathematical format, providing a structured representation of the biological knowledge they encapsulate. Computer models are subsequently generated for the analysis, interpretation, and prediction of biological functions arising from these networks.

4. Finally, these models are put to use in a forward-looking manner. This entails formulating specific hypotheses generated by the models, which can then be tested experimentally. An iterative process of refinement and improvement is employed for these *in silico* models of reconstructed networks.

This paradigm encapsulates the essence of systems biology, offering a systematic and holistic approach to deciphering the intricacies of life at the molecular level [1].

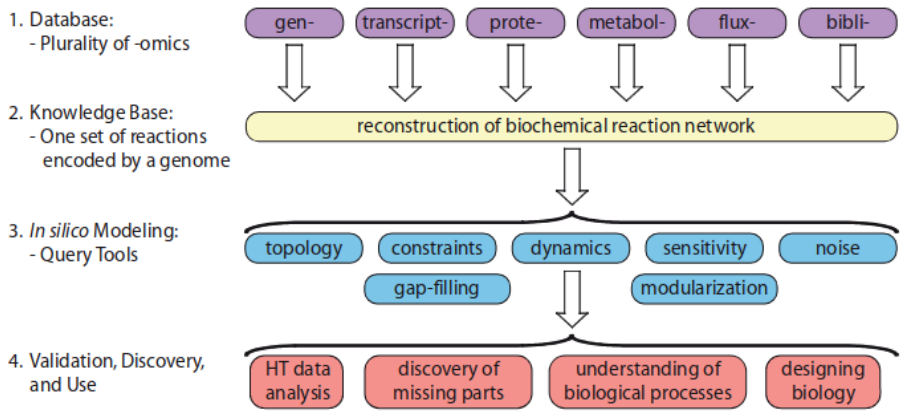

Figure 2.1: The four principal steps in the implementation of systems biology [1].

2.1 Metabolic Networks

Metabolic networks are fundamental in biology representing the intricate system of biochemical reactions that take place within the cells of an organism. These reactions are crucial for the maintenance of life and the execution of various cellular functions. Metabolic networks are central to the overall functioning of an organism and play a key role in processes such as energy production, the synthesis of biomolecules, and the regulation of metabolic pathways.

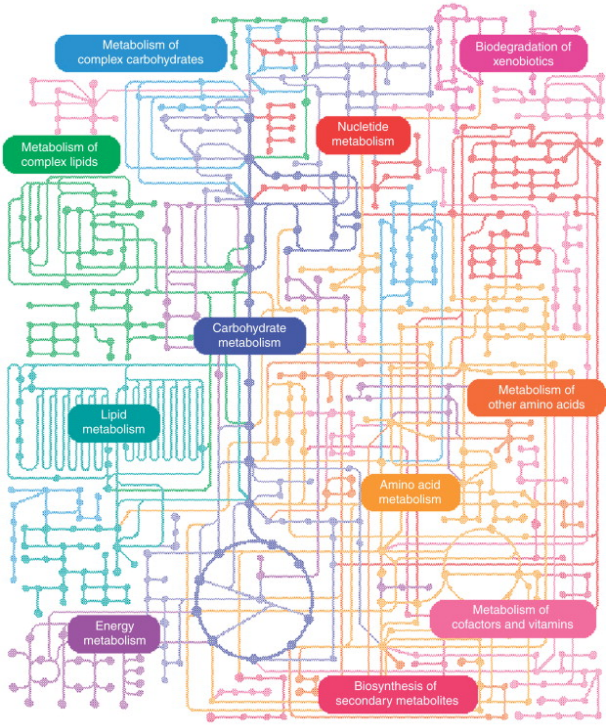

Figure 2.2: Schematic diagram of part of the metabolic network of *E. coli*. [2]

The process of network reconstruction is a meticulous and laborious endeavor, executed in a step-by-step fashion by identifying one reaction at a time. It entails the systematic construction of an extensive network by discerning individual reactions one step at a time. This reconstruction gathers a comprehensive array of biochemical, genetic, and genomic data pertaining to a specific cellular process of interest. Subsequently, it organizes this wealth of information into a formal and mathematically coherent framework that aligns with the fundamental chemical and genetic principles governing the process. Networks, in this context, consist of compounds (nodes) and the reactions that interconnect them (links). When multiple reactions are known to share common reactants and products, they can be visually interconnected in a graphical representation. As the scope of the network expands, more reactions can be added to this graphical depiction. For example, Figure 2.3-A illustrates the initial stages of glycolysis using this approach. Importantly, such a map can be translated into a precise mathematical representation (as depicted in Figure 2.3-B). This mathematical representation relies on stoichiometric coefficients, which enumerate the molecules consumed and produced by each biochemical reaction. All these coefficients for every reaction within the network can be organized in a matrix format—akin to a tabular structure. This information is both exact and quantitative, forming the foundation for the mathematical characterization and evaluation of the integrated biochemical properties of the network as a whole. It is worth noting that while a network map can be visually depicted in various ways, the mathematical representation remains unique and invariant.

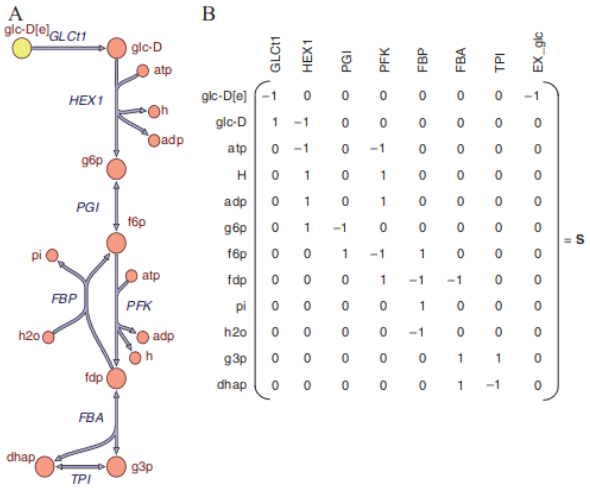

Figure 2.3: Stoichiometric representation of metabolic network from *Escherichia coli*. Panel (A) shows the first few reactions of glycolysis in a graphical form as a network of interacting reactions that share substrates and products. Panel (B) shows the stoichiometric matrix (S) corresponding to the reactions in panel (A). As indicated, each column corresponds to a particular reaction and each row to a particular metabolite. The last column, labeled 'EX glc,' is an exchange reaction for glucose that allows glucose to enter and leave the system. [1]

A genome-scale reconstruction aspires to encompass all the genetic and biochemical processes within an organism's genome comprehensively. Its goal is to provide a holistic understanding, contextualizing every genetic element in the target organism while systematically integrating all available knowledge. This entails the construction of a comprehensive metabolic model, following the steps outlined in Figure 2.1.

2.1.1 Genome-scale models

After the network reconstruction, it is possible to represent it into a computational Genome-scale Model (GEM). GEMs leverage the wealth of biochemical, genetic, and genomic data gathered during the reconstruction process to create sophisticated mathematical representations of entire cellular systems. They can predict how genes, proteins, and metabolites interact to influence cellular functions. This enables the simulation and analysis of the integrated biochemical properties of the entire network, enabling a deeper understanding of an organism's biology, metabolism, and responses to various conditions. In computational systems biology, metabolic models contain 3 main levels of information: metabolites, reactions and metabolic genes. The relation between metabolites and reactions can be described by a stoichiometric matrix and the one between reactions and genes is represented by a two-dimensional binary matrix [3]. Computational systems biology utilizes well-constructed metabolic models for in silico simulations of an organism's behavior, such as growth or compound production, under specific environmental conditions. These simulations are conducted using constraint-based modeling.

2.1.2 Stoichiometric Matrix

Since stoichiometric matrices are able to represent the relation between metabolites and reactions in a mathematical way, they are used to quantitatively analyze metabolic networks. To better understand the formulation of a stoichiometric model, a small metabolic network can be considered (Figure 2.4). This network considers five intracellular metabolites (A, B, C, D, E), six intracellular reactions ($v_1 : A \rightarrow B; v_2 : B \rightarrow C; v_3 : B \rightarrow D; v_4 : C \rightarrow D; v_5 : C \rightarrow E; v_6 : D \rightarrow E$), and four exchange reactions that link the extracellular environment to the intracellular space ($v_{E1} :\rightarrow A; v_{E2} : B \rightarrow; v_{E3} : C \rightarrow; v_{E4} : D \rightarrow$).

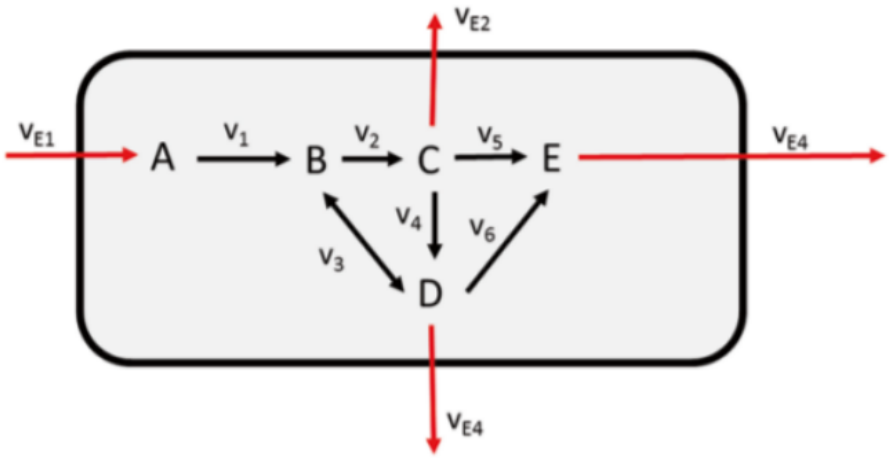

Figure 2.4: Illustrative example of a metabolic network considering five intracellular metabolites (A, B, C, D, E); six intracellular reaction rates ($v_1, v_2, v_3, v_4, v_5, v_6$); and four exchange reactions ($v_{E1}, v_{E2}, v_{E3}, v_{E4}$) [3].

Each metabolite's changes in concentration over time can be described by Ordinary Differential Equations (ODEs). For example, considering metabolite C, whose consumption and production are governed by Equation 2.1. In this context, producing reactions are accounted for as positive contributors to the conservation equation, while consuming reactions are represented as negative ones.

$$\frac{dC}{dt} = v_2 - v_4 - v_5 - v_{E2} \tag{2.1}$$

However, depicting an entire metabolic network using a system of ODEs is very complex. To simplify this process, we make the assumption that, for all intracellular metabolites, the rates of reactions producing a given metabolite are in equilibrium with the rates of reactions consuming that metabolite. This assumption allows us to establish what is known as a *steady state*, wherein metabolites are consumed at the same rate at which they are produced. This leads to the establishment of a mass balance within the metabolic network, where every incoming mass flux into the system must be meticulously balanced by an equivalent outflow. It is this maintained mass balance that produces a *steady state*. As a practical consequence, the rate of change of metabolite concentration, expressed as dC/dt, is set to zero. This simple step transforms the complex ODE system into a straightforward linear equation, ensuring that

mass balance is maintained for each metabolite in the network.

$$\frac{dC}{dt} = 0 \iff v_2 - v_4 - v_5 - v_{E2} \tag{2.2}$$

To describe the entire network, we establish a system of linear equations. The stoichiometry of reactions within the network is represented by a stoichiometric matrix (**S**), which is of size "**m** x **n**". The rows of the matrix correspond to each unique internal metabolite (in a system with **m** internal metabolites), while each column represents an individual reaction (**n** reactions in total). The entries within each column signify the stoichiometric coefficients of the metabolites involved in that specific reaction. A positive coefficient indicates metabolite production, whereas a negative coefficient signifies consumption. Importantly, many entries in the matrix are zeros because only a limited number of metabolites participate in each reaction. Equation 2.3 illustrates the stoichiometric matrix **S**, with its n columns representing the rates of reactions ($v_1, v_2, v_3, v_4, v_5, v_6, v_{E1}, v_{E2}, v_{E3}, v_{E4}$), and the m rows representing the various metabolites (A, B, C, D, E).

$$S = \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & -1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & -1 \end{bmatrix} \tag{2.3}$$

The mass balance is only complete with the reactions' flux values, defined in flux units: mmol/gDW/h. The fluxes of the intracellular reactions are represented by a vector **v** (length **n**). The system of mass balance equations at steady state is given by equation 2.4 [3].

$$S.v = 0 \tag{2.4}$$

The given equation 2.4, leads to a system of linear equations where the solution is vector *v*. **S** is a sparse matrix because most biochemical reactions involve only a few different metabolites. In any realistic large-scale metabolic model, there are more reactions than there are compounds (**n** $>$ **m**). In other words, there are more unknown variables than equations (under-determined equation system), so there is no unique solution to this system of equations (**S . v = 0**). That is where the constraint-based modeling comes in: constraints can be applied as inequalities that impose bounds on the system [4].

2.1.3 Constraints

Metabolic networks in biological systems are highly dynamic and adaptable. They can exist in numerous functional states, with the cell selecting a subset of these states based on various factors, including environmental conditions and evolutionary history (Figure 2.5). Unlike in traditional engineering or physics, where exact solutions are sought based on fundamental laws, biology operates differently. In biological

networks, multiple behaviors can achieve the same function, and the choice of which behavior to employ may vary over time due to evolutionary pressures [1].

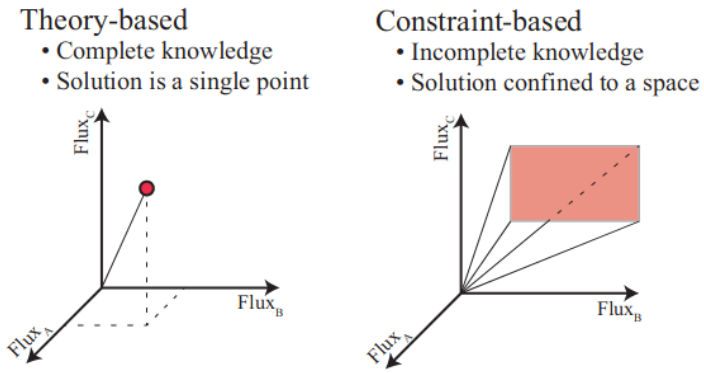

Figure 2.5: Theory- vs. constraint-based analysis. Illustration of finding an exact solution (a point) versus finding a range of allowable solutions (a solution space). [1]

The stoichiometry impose constraints on the flow of metabolites through the network. These constraints are expressed both as mathematical equations and as inequalities that govern the balance of inputs and outputs within metabolic reactions. This balance is essential for maintaining steady-state conditions within the network. The stoichiometric matrix defines these constraints, ensuring steady-state conditions where production equals consumption for every compound. Additionally, each reaction within the metabolic network can be assigned upper and lower bounds. These bounds act as regulatory limits, defining the maximum and minimum permissible rates at which reactions can occur. They are influenced by factors such as enzyme activity, substrate availability, and environmental conditions. Together, these balance equations and bounds delineate the space of allowable flux distributions within the system. This means they define the feasible rates at which each metabolite is either consumed or produced by each reaction [1, 4].

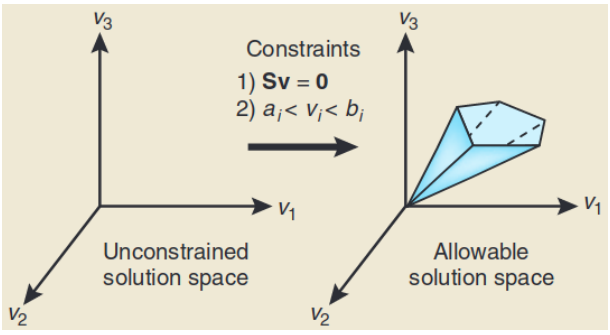

Figure 2.6: The conceptual basis of constraint-based modeling. With no constraints, the flux distribution of a biological network may lie at any point in a solution space. When mass balance constraints imposed by the stoichiometric matrix **S** (labeled 1) and capacity constraints imposed by the lower and upper bounds (a_i and b_i) (labeled 2) are applied to a network, it defines an allowable solution space. The network may acquire any flux distribution within this space, but points outside this space are denied by the constraints.

2.1.4 Gene-Protein-Reaction Rules

Advancements in genome sequencing technologies enabled the addition of another ”layer” of data to network models, and further constraint the solution space: the metabolic genes behind the reactions. This addition provides a more comprehensive readout of the model's functional state, resulting from the interplay between genome, biochemistry and environment. Metabolic reactions can be either enzymatic or non-enzymatic. Enzymatic reactions only occur when catalyzed by specific enzymes encoded by genes. The relationships between genes, proteins and reactions are defined through logical expressions referred to as Gene-Protein-Reaction (GPR) rules, or just gene-reaction rules Figure 2.7.

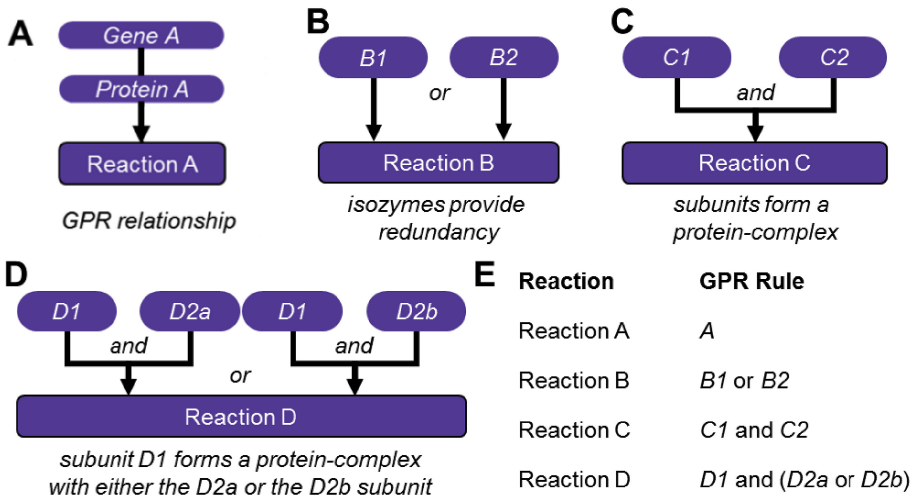

Figure 2.7: (A) Example of a GPR rule representing an enzymatic reaction catalyzed by the protein product of a gene. (B) Example of a GPR ”OR” rule where either protein B1 or protein B2 can independently catalyze the same reaction. (C) Example of an ”AND” GPR rule where both C1 and C2 are required for the reaction to occur - two non-redundant subunits that form a protein complex. (D) Example of a complex GPR where D1 can form a protein complex with either D2a or D2b. (E) Table summarizing genotype-phenotype relationships from A-D as Boolean GPR rules [5]

A GPR rule takes genes as variables and determines if the reaction catalyzed by the gene coded enzymes is active or not. The variables are represented as Boolean values, with each gene being assigned either 1 (active) or 0 (inactive). For instance, reaction C (Figure 2.7C) is given by the rule ”C1 AND C2”, so if either C1 or C2 take 0 as their value, reaction C will not be active. This information is gathered in the form of a Boolean matrix with all the model's metabolic genes taking the values of 1 or 0 and the information is propagated to the reactions. When a reaction is determined to be inactive, its lower and upper bounds, for the corresponding metabolic flux, are set to 0.

2.1.5 Simulating Metabolism

Constraints define a range of possible solutions within a given problem space. However, within this space, it is often essential to pinpoint and analyze specific points that hold particular significance. For

instance, it may be relevant to identify the point that corresponds to the maximum growth rate, optimal energy production, or the production of a specific metabolite, such as lactate, while adhering to a specific set of constraints.

Linear Programming (LP) approaches, including Flux Balance Analysis (FBA), Parsimonious enzyme Flux Balance Analysis (pFBA), and Flux Variability Analysis (FVA), are widely employed for precisely this purpose. These methodologies leverage stoichiometry and consider constraints imposed by GPR rules. Through these techniques, it becomes possible to predict crucial parameters such as an organism's growth rate or the rate of production of significant metabolites. This, in turn, allows us to simulate and gain insights into the metabolism of the organism [4].

In order to better understand the following methods, it is important to define Linear programming. LP is mathematical technique in which a given linear function (objective function) is maximized or minimized given a set of linear equality or inequality constraints. In mathematical terms, a LP problem finds values for **n** decision variables $\mathbf{v}_j$ (j = 1, 2,...,n) while maximizing/minimizing a linear function **z** (Equation 2.5), subject to a set of **m** constraints (Equation 2.6) and to $\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_n >= 0$ [6].

$$z = c_1 v_1 + c_2 v_2 + ... + c_n v_n \tag{2.5}$$

$$\begin{cases} a_{11} v_1 + a_{12} v_2 + ... + a_{1n} v_n = b_1 \\ a_{21} v_1 + a_{22} v_2 + ... + a_{2n} v_n = b_2 \\ ... \\ a_{m1} v_1 + a_{m2} v_2 + ... + a_{mn} v_n = b_m \end{cases} \tag{2.6}$$

To solve an LP problems, a specialized optimization solver such as GLPK[1], CPLEX (IBM Inc.)[2] and Gurobi[3] are generally used.

In many metabolic model simulation methods, LP plays a crucial role in determining desired solutions. These solutions are typically defined by the objective function (**z**), which can represent various metabolic goals such as optimizing growth rate, energy production, or maximizing the production of a specific metabolite. The constraints are established through a system of linear equations and inequalities, deriving from the model's stoichiometry and reaction flux bounds. The problem is then maximized (or minimized) subject to the linear constraints derived from the stoichiometry and reaction bounds. The output of this optimization is a flux distribution that achieves optimal value to the objective function while satisfying the imposed constraints.

A – Flux Balance Analysis FBA is the direct application of LP to metabolic models. The constraints (metabolic model's stoichiometry and reactions bounds) restrict the space of possible solutions for the

[1] https://www.gnu.org/software/glpk/
[2] https://www.ibm.com/analytics/cplex-optimizer
[3] https://www.gurobi.com/

flux distribution. The objective function is defined. And finally the objective function is maximized in order to get a flux distribution that gets its optimal value while satisfying the constraints. The output of a FBA is a flux distribution where the objective function is at its optimal value while adhering to the defined constraints. It is important to mention that, in many cases, there may be alternative solutions that also maximize the objective function, which means that there are multiple optimal solutions [4].

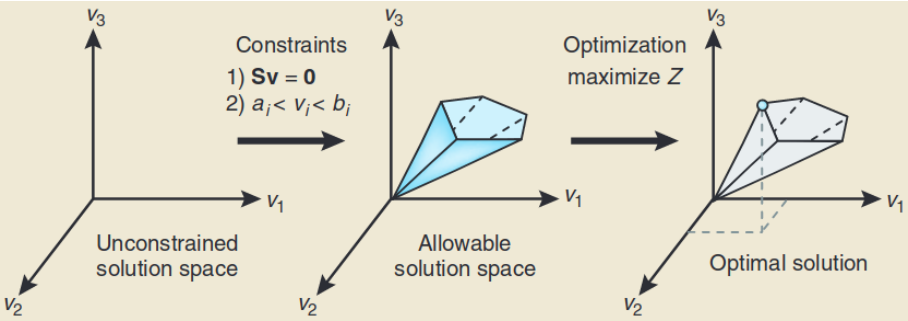

Figure 2.8: Through optimization of an objective function, FBA can identify a single optimal flux distribution that lies on the edge of the allowable solution space [4].

B – Parsimonious Enzyme Usage FBA Knowing that there may be multiple optimal solutions to a given objective function, it can be important to find the most biologically relevant solution among the multiple possible solutions. This means **min(v)**, where **v** is vector of fluxes with the same length as the number of reactions 2.4 [7]. To achieve this, alternative methods like the parsimonious FBA (pFBA) have been developed. The pFBA can select the most biologically relevant solution by minimizing the flux in the reactions, meaning that it finds the solution with the lowest flux values (the most parsimonious) among the multiple solutions that maximize the objective function.

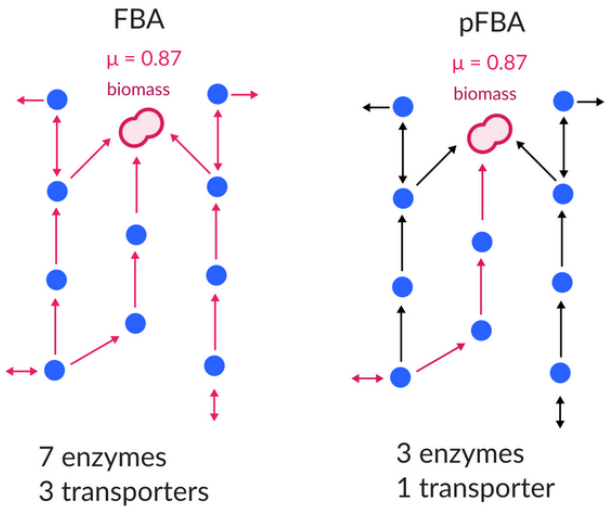

Figure 2.9: Comparison of Flux Distributions: Left - FBA with 7 Enzyme Utilization vs. Right - pFBA with 3 Enzyme Utilization, Both Yielding the Same Optimal Objective Function Value. [8]

C – Flux Variability Analysis In the domain of metabolic network analysis, when confronted with multiple flux distributions that yield an optimal value for the objective function, it may be relevant to ascertain the permissible range of fluxes for each individual reaction while maintaining this optimal outcome. This can be achieved through Flux Variability Analysis (FVA), a computational methodology that systematically explores and delineates the potential range of flux values for each reaction within the metabolic network. FVA is often used to analyze the robustness and flexibility of metabolic networks, and to identify reactions that are most important for maintaining the overall function of the network. It can be also used, for example, to detect ”deadend” reactions, which are reactions that unable to convey flux in any circumstance. These reactions can be relevant because they may indicate the presence of constraints in the metabolism of the cell [9]. It may be relevant to determine which reactions contribute more significantly to the optimal solution obtained through FBA. FVA can be used to evaluate the importance of a specific reaction by varying its flux value between its lower and upper bounds and determining the feasible range of flux values while maintaining the optimal solution (obtained from a FBA). For example, if one reaction has its flux value set to 0 or its maximum value, and the objective function keeps its optimal value in both cases, this could mean that that specific reaction is not crucial for that objective function. On the other hand, if the objective function is at its optimal value when a specific reaction has its maximum flux value, but is not once that flux value decreases, this could mean that the reaction is crucial for that objective function [9].

There are also methods to predict how the metabolism of an organism may change in response to genetic or environmental perturbations. Such as a gene KnockOut (KO), on a cell's metabolism. A gene KO is genetic technique in which one of the system's genes is made inoperative. Methods like Minimization of Metabolic Adjustment (MOMA) or linear Minimization of Metabolic Adjustment (lMOMA) are computational methods used in the field of systems biology and metabolic engineering aimed to do these type of predictions compared to a reference state. These methods assume the hypothesis that metabolic fluxes undergo a minimal redistribution of the flux path after a KO of one or more specific genes [10].

D – Minimization of Metabolic Adjustment MOMA follows the following key components:

1. MOMA starts with a reference metabolic flux distribution (typically obtained through a FBA simulation), which represents the baseline metabolic activity of an organism under normal conditions.
2. When there's a genetic or environmental perturbation (e.g., gene KO, nutrient limitation, or drug treatment), MOMA aims to predict the metabolic response by adjusting the flux distribution while minimizing changes in metabolic fluxes compared to the reference state.
3. MOMA uses a mathematical optimization approach, Quadratic Programming (QP) formulation, to find a new flux distribution that minimizes the Euclidean distance or a similar metric between the

altered state and the reference state. The objective function is typically designed to achieve this minimization.

The result of this simulation is a new flux path that is similar to the original one, but with minimal adjustments to the metabolic fluxes Computational simulations under the MOMA assumption have resulted in closer flux values to actual experiments [10].

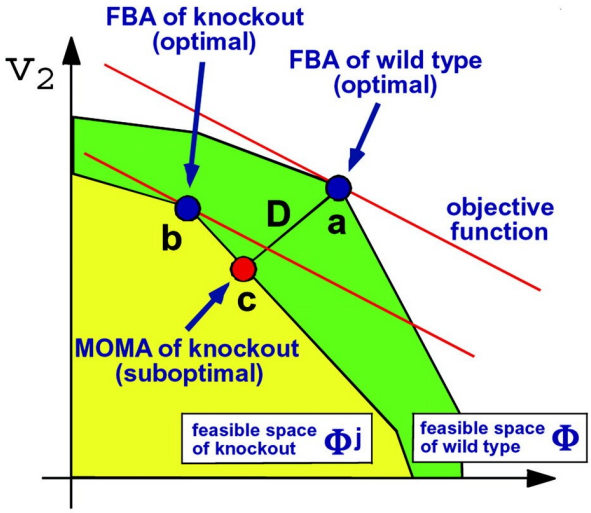

Figure 2.10: The optimization principles underlying FBA and MOMA. A schematic 2D depiction of the feasible space for the wild-type (θ^{WT}) and for mutant of flux j (θ^{j}) is represented by the green and superimposed yellow polygons. The coordinates are two arbitrary representative fluxes, an extremely simplified version of the multidimensional flux space. The solution to the FBA problem is the point that maximizes the objective function (red line). An optimal FBA prediction can be computed both for the wild-type (a) and for a knockout (b). The alternative MOMA knockout solution (c), calculated through QP, can be thought of as a projection of the FBA optimum onto the feasible space of the mutant (θ^{j}). The mutant FBA optimum and the corresponding MOMA solution are, in general, distinct [10].

E – linear Minimization of Metabolic Adjustment lMOMA is a variation of the MOMA method that simplifies the optimization problem by using a linear objective function. This linearization makes lMOMA computationally more efficient to implement. lMOMA is a valuable tool in systems biology for predicting metabolic responses to perturbations [10].

2.1.6 GECKO Models

The development of GEM models has been fundamental to calculate metabolic fluxes based on reactions' stoichiometry and metabolite balances. However, conventional GEMs tend to oversimplify the metabolic picture by making assumptions, such as assuming that the uptake rate of carbon sources, like glucose, limits production. This overlooks the fact that enzyme levels can also restrict metabolic fluxes. In response to this challenge, ME-Models were introduced. These models represent an innovative fusion of metabolic and gene product expression pathways, offering a more comprehensive

view of cellular metabolism [11]. While ME-Models provide a more holistic perspective, their complexity can make simulations challenging. The Genome-scale metabolic models with Enzymatic Constraints using Kinetics and Omics (GECKO) toolbox emerged to address these issues with an approach that enriches metabolic models by integrating enzyme constraints. GECKO extends its scope to incorporate enzyme-level considerations. This is achieved by including an extra entity for each metabolic reaction that represents enzyme usage. The entity is limited by the protein abundance, which can be provided as input to the model. Thus, it can conveniently simulate metabolism with constraints based on protein abundance measurements, correctly representing capacity constraints on fluxes. The method enhances a GEM with Enzymatic Constraints using Kinetic and Omics data and is referred to as GECKO. The resulting enzyme-constrained models have the same structure as any GEM, such that it can be used for any constrained-based analysis method (e.g., FBA, FVA). GECKO is well suited for direct integration of proteomics; by setting each enzyme abundance, we can run simulations with a more constrained solution space. This integration of proteomics data results in models that are not only more accurate but also more biologically relevant. Currently, there are some functional GECKO models for organisms like *S. cerevisiae* [12] (ecYeast7), *Yarrowia lipolytica* [13] (ec*i*Yali4), *Kluyveromyces marxianus* [14] (ec*i*SM996), *E. coli* [15] (ec*i*ML1515) and *H. sapiens* metabolism [16] (ecHuman1). This way, the GECKO models additionally constrain the flux variability within each reaction, as demonstrated in Figure 2.11.

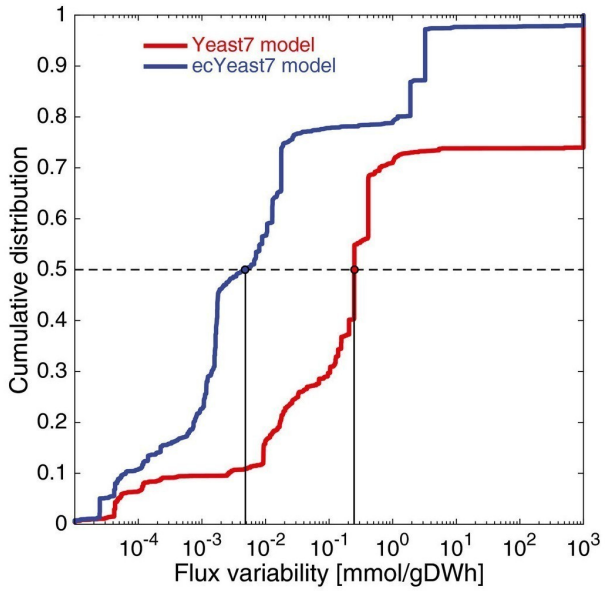

Figure 2.11: Flux variability for all non-zero variable fluxes from Yeast7 [12] (3,286 reactions, 66.1% of all reactions) and ecYeast7 (3,822 reactions, 56.7% of all reactions) [17].

The development of the GECKO toolbox has made advances in the field of metabolic modeling by addressing the limitations of traditional GEMs. The inclusion of enzyme constraints provides a more comprehensive representation of cellular metabolism, allowing for a more realistic simulation of fluxes

and responses to perturbations. This advancement improves the predictive power of constrained-based analysis techniques like FBA [17, 18].

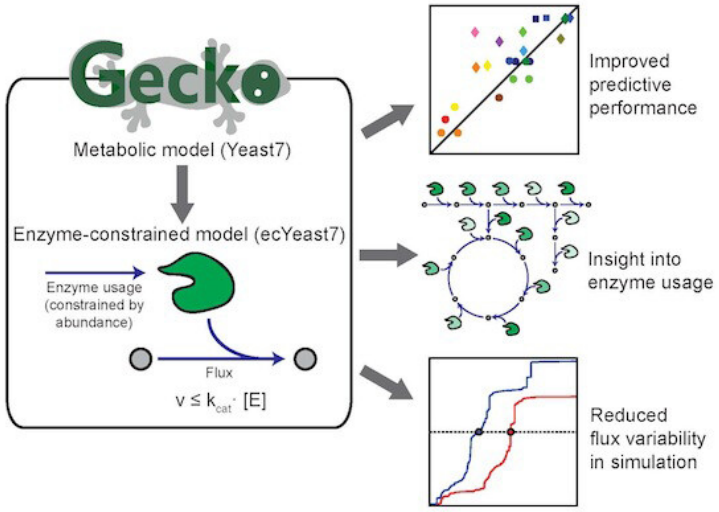

Figure 2.12: GECKO is a method that enhances a GEM with enzyme constraints, using both kinetic and omics data. [17].

2.2 Regulatory Networks

Regulatory networks are a way of representing the interactions between genes in a biological system. Genes can play different roles in these interactions, acting as either effector or target genes. Effector genes are those that activate or repress other genes, while target genes are those that are activated or repressed by other genes. However, it is important to note that a gene can play both roles in different interactions within the same regulatory network [19].

There are several different approaches to modeling gene regulatory networks, which can be broadly grouped into two categories: the continuous approach and the discrete approach. The continuous approach to modeling a gene regulatory network involves using mathematical models based on differential equations to describe the dynamic interactions between genes and the proteins they encode. This approach is useful because it can provide a detailed and accurate representation of the system dynamics. However, this approach entails some limitations. It requires a significant amount of data to accurately parameterize the model, and it can be computationally intensive to solve the equations. Additionally, the data available is typically not precise enough, which makes it difficult to estimate the parameters of the model and can lead to less accurate predictions.

On the other hand, the discrete approach to modeling gene regulatory networks uses a simpler and less data-intensive method. Typically, the expression of a gene can be either on or off, which is discrete.

Thus, this is the most typical approach because it is simpler and requires less precise data, being more scalable and handling larger systems. However, it is important to note that this approach only provides a trend, or an approximation/abstraction of the system dynamics and not a detailed representation of the system.

Overall, the choice of approach depends on the availability and quality of data, as well as the complexity of the system. The continuous approach is more accurate and detailed but requires more data and computational power, while the discrete approach is simpler and more scalable but provides less detailed and accurate representation of the system behaviour [20, 21].

2.2.1 Logical Modelling

Regulatory networks describe the relationships and interactions of the genes in a network. Regulatory network logical modeling describes the mathematical representation of those relationships and interactions. These models provide insight on how these genes behave, how they interact, and how they influence each other. A regulatory network logical model can be defined by:

- genes, each associated to an integer value in $[0, ..., max]$. Although Boolean discretization is usually enough (i.e., $max = 1$). The value is 1 if the gene is active and 0 if the gene is not active.
- interactions between genes. Each interaction has a source gene and a target gene. The interaction can activate or repress the target gene.
- a logical function function for each gene (regulatory rules) that define its value. Given a gene, its discrete functions take as arguments the current values of the regulators (effector genes) that interact with it (Equation 2.7).

Aside from representing the regulatory network as a directed graph, where the nodes are the genes and the directed edges are the interactions between the genes, regulatory network logical modeling has a logical function associated with each gene. In Equation 2.7 ($g_{i1}, g_{i2}, ..., g_{ik}$) are the effectors of g_i, and F is the logical function constructed according to the network. All these genes are "connected" through the interactions.

$$g_i(t+1) = F_i[g_{i1}(t), g_{i2}(t), ..., g_{ik}(t)] \tag{2.7}$$

The Boolean values assigned to each gene define the state of a regulatory network. The state of the system at a given instant $t+1$ can be inferred from the network's state at instant t. The logical functions define the possible transitions of the system. Meaning that, given a state, there are limited possible transitions to other states. There are as many states as there are possible transitions, this defines the state space of the system. This set of states and the possible transitions between them are represented

by State Transition Graphs (STGs), where the nodes denote the states and the directed edges represent the state transitions. In a STG, given a state, all its possible transitions are connected to it.

STGs can be generated using several updating modes. Two of the most commonly used are the asynchronous and synchronous (Figure 2.13). The synchronous update mode assumes that all the genes that can update their values, update them simultaneously. This generates, for each state of the system, at most a single successor. On the other hand, the asynchronous update mode assumes that each gene updates individually. Resulting in each state having as many successor states as genes that can change their value. Changing only one gene per successor. Since the number of states in the STG is finite, the asymptotic behavior of the system will always lead to at least one attractor. An attractor can be either a single stable state or a cyclic trajectory of states. In a stable state the values of the genes are fixed, and has no transitions are possible to other states. A cyclic trajectory of states is a maximal set of mutually reachable states, with no possible transitions leaving the set, where only a subset of of all genes are fixed. Cyclic attractors are defined in the context of the logical formalism as terminal Strongly Connected Components (SCCs) of the STG [20, 22].

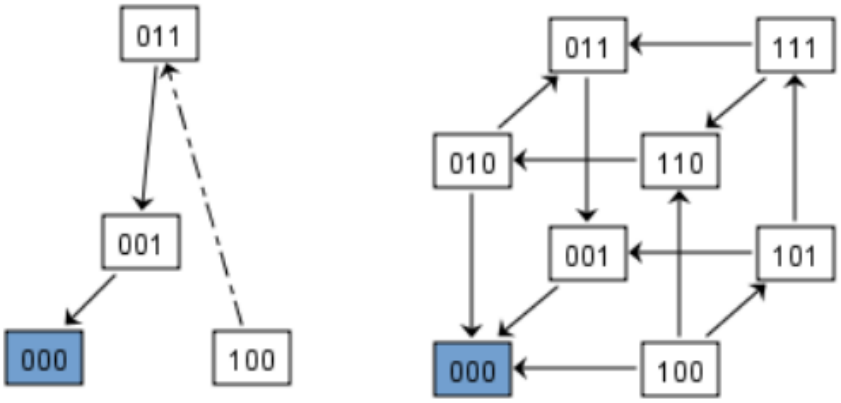

Figure 2.13: State transition graphs (obtained with the GINsim tool). Using the synchronous (left) and asynchronous (right) updating modes [20].

2.3 Metabolic and Regulatory Integration

Regulatory and metabolic networks are two separate layers of biological systems, each with their own set of interactions and mechanisms. The regulatory network involves enzymes and proteins that catalyze and regulate the reactions of metabolic networks, while the metabolic network is responsible for the chemical reactions that occur within a cell or organism. When these two networks are integrated, they can provide a more complete understanding of the interactions and mechanisms that occur within a biological system and enhance the precision and accuracy of computational models and simulations.

2.3.1 Bridging regulatory networks with metabolic pathways

The integration of regulatory and metabolic networks is important because the metabolic component can affect the regulatory component through the activation or inhibition of Transcription Factors (TFs) via the presence of certain metabolites. For example, the presence of certain metabolites may activate a TF, leading to the expression of certain genes, while the absence of certain metabolites may inhibit a TF, leading to the repression of certain genes.

In this way, by integrating these two networks, we can associate regulatory rules to metabolic reactions, and associate metabolites to certain TFs. This integrated approach allows for more accurate simulations of real-life systems. Additionally, it can help to identify key regulatory mechanisms and metabolic pathways that are important for maintaining the overall function of the network, and to identify constraints and limitations in the system [23, 24].

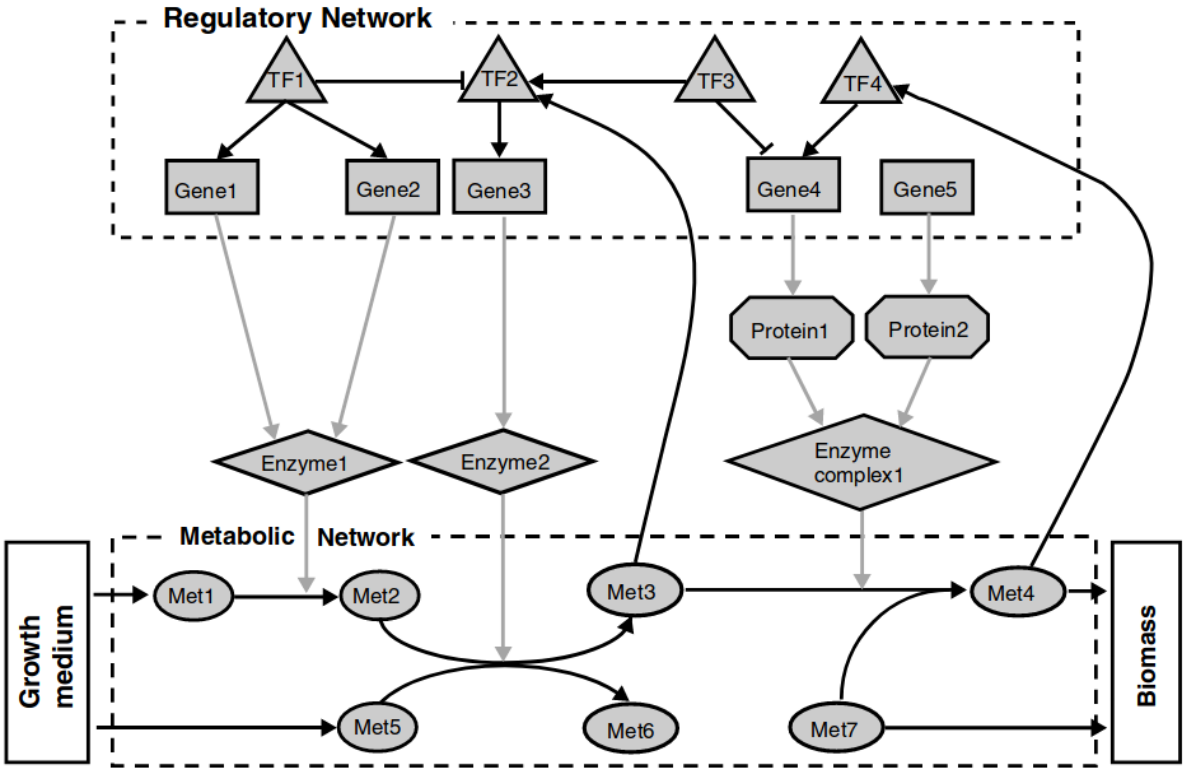

Figure 2.14: Schematic representation of an integrated metabolic and regulatory network. The regulatory network component consists of a set of interactions between TFs and other TFs and genes. The metabolic network component consists of a set of biochemical reactions between metabolites, with metabolites available from growth medium as input, and a pseudo-metabolite representing biomass production as output. The regulatory component affects the metabolic component through the expression of proteins that catalyze the biochemical reactions (downward pointing arrows). The metabolic component affects the regulatory component via the activation or inhibition of TF expression via the presence of specific metabolites (upwards arrows) [24].

2.3.2 Methods

Having a metabolic model, its stoichiometry and reaction flux bounds impose constraints on the solution space of flux distribution. By integrating regulatory data, such as information on gene expression or TF activity, the outputs of GPR rules change and consequently metabolic reactions can be further constrained. This additional constraints enable a more accurate representation of the cell and results in simulations are closer to real life cell behaviour. Computational methods like Regulatory Flux Balance Analysis (rFBA), Steady-state regulatory Flux Balance Analysis (SR-FBA) and Integrated Flux Balance Analysis (iFBA) implement this integration by grouping the regulatory data in logical rules that determine the activity of metabolic elements, such as enzymes or reactions. The resulting flux from LP optimizations, such as FBA, influences the activation or inhibition of TFs within the regulatory network. This, in turn, modifies the state of the regulatory network. Changes in the regulatory network cascade back to the metabolic elements, affecting their activity. This feedback loop continues [24–26].

A – Regulatory FBA rFBA describes the transcriptional regulatory data as Boolean logical functions (regulatory rules). In this method, the expression of a transcription unit is represented as 1 when it is being transcribed and 0 when it is not. Likewise, the presence of an enzyme, regulatory protein, or specific cellular conditions is expressed as 1 when they are present and 0 when they are absent. The regulatory rules representation includes well-known modifiers such as AND, OR, and NOT. These rules are then associated to metabolic reactions, depending on how the cellular network is structured. A logical function associated with a reaction determines whether that reaction is active or not at a specific time interval (t). If the logical function evaluates to 1, the reaction is considered active; otherwise, it is inactive. This integration of transcriptional regulatory data introduces additional constraints into the solution space, allowing rFBA to predict how cells behave under varying regulatory conditions. Much like traditional FBA, rFBA relies on LP. Once the constraints are defined and an objective function is set, the problem can be optimized to find an optimal distribution of fluxes. The resulting flux distribution informs us about which metabolites are being produced and consumed within the cell. This information can be used to deduce which TFs are either activated or inhibited. These TFs, in turn, influence the outcomes of the regulatory rules in the subsequent time interval ($t + 1$), which subsequently impacts metabolic reactions, creating a continuous feedback loop between the regulatory and metabolic networks. This loop continues until the optimal value repeats after two iterations [25].

B – Steady-state rFBA SR-FBA applies the same representation of transcriptional regulatory data and integrates it in the metabolic model in the same way as rFBA. However it does not operate on iterations. Instead, SR-FBA establishes a compound functional state called Metabolic Regulatory Steady-state (MRS). The MRS is by a pair of consistent metabolic and regulatory steady-states, which satisfy

both metabolic and regulatory constraints:

1. the metabolic functional state is represented by steady-state fluxes through metabolic reactions
2. the functional state of the transcriptional regulatory system at steady state is represented by a fixed, steady-state Boolean value for each gene, indicating whether it is expressed or not.

These Boolean values are determined using FVA. Through an optimal value derived from LP optimization and subsequent FVA analysis, SR-FBA explores various flux distributions. It then identifies which genes are consistently expressed and which are not, ultimately setting their Boolean values accordingly. This allows SR-FBA to consider a wider range of possible dynamic fluxes and generate more accurate and comprehensive representations of the metabolic network behavior. In contrast, rFBA operates on iterations and assumes an arbitrary optimal solution(flux distribution) for each iteration, leaving out a whole space of possible dynamic fluxes unconsidered [24].

C – Integrated FBA iFBA is similar to rFBA, but instead of using logic functions with discrete outputs (0 or 1) as regulatory rules, iFBA uses ODEs with continuous outputs. The key difference between rFBA and iFBA is that in rFBA, a given reaction is either activated or not based on the value of its associated logic function (regulatory rule), whereas in iFBA, the activation of a given reaction is based on the output of its associated ODE. The output of the ODE can be a continuous value, and the activation of a reaction can be determined by comparing the output of the ODE to a defined threshold. The use of ODEs in iFBA allows for more accurate and detailed simulation results compared to rFBA's approach, which relies on discrete values to represent the activation state of reactions. Additionally, iFBA also allows for the simulation of time-dependent changes in the regulatory and metabolic networks, which is not possible with rFBA. However, iFBA requires more complex mathematical models, as it uses ODEs as regulatory rules, whereas rFBA uses simpler logic functions. This can make iFBA more difficult to implement and computationally intensive to run. Also it requires more precise data to accurately parameterize the model, whereas rFBA can work with less precise data. Another disadvantage of iFBA is that it is not as scalable as rFBA, and it may not be as efficient to use on large metabolic networks. Lastly, iFBA may be more sensitive to the choice of the threshold value used to determine whether a given reaction is activated or not. This can make the results of iFBA less robust and less reliable than rFBA [26].

There are some limitations in these integrated modeling methods. One of the primary issues with these methods is their oversimplified representation of the connection between the transcriptome (gene expression) and the metabolome (metabolic reactions). These models often reduce this intricate relationship to a binary concept, where genes and metabolic reactions are either turned ”on” or ”off” within a population. However, the most significant obstacle to applying integrative modeling techniques across a wide range of species lies in the lack of an automated algorithm for establishing the Boolean rules

that define how a regulator (e.g., a gene or protein) relates to its target (e.g., a metabolic reaction). The current approach often involves manual intervention to construct these rules, which can indeed yield accurate metabolic regulatory models. However, this manual reconstruction process severely restricts the number of interactions that can be included in the model. Consequently, there are very few genome-scale metabolic-regulatory models.

D – Probabilistic Regulation of Metabolism In response to these challenges Probabilistic Regulation of Metabolism (PROM) has emerged. PROM offers a revolutionary approach to integrate transcriptional and metabolic networks for modeling purposes. Unlike previous methods, PROM eliminates the need for manually crafting Boolean rules by automatically quantifying interactions based on high-throughput data. This automation enhances the ability to construct genome-scale integrated models efficiently. The PROM algorithm operates by leveraging conditional probabilities to model transcriptional regulation. To build an integrated metabolic-regulatory network using PROM, several key components are required:

- a reconstructed GEM;
- a GRN structure that includes only TFs and their target genes;
- and gene expression data.

PROM introduces probabilities to represent the state of genes and the interactions between TFs and target genes. For example, $P(A = 1|B = 1)$ gives the probability of gene A being activated (ON) when the regulating TF B is activated (ON). Given some gene expression data, if 80% of the samples where TF B is activated, the target gene A is also activated, this would translate to $P(A = 1|B = 1) = 0.80$. These probabilities are then used to constrain the fluxes through the reactions controlled by the target genes. For example, if $P(A = 1|B = 1) = 0.80$, the maximum flux possible through the reaction controlled by gene A would be 0.8 $\times$ maximum velocity (Vmax). Vmax is estimated by FVA on the unregulated metabolic model. The probabilistic on/off formalism presents the advantage of gene states taking intermediate values, not only ON and OFF. This is in contrast to Boolean logic models, such as rFBA, where gene states are limited to binary values. The integration of regulatory networks in metabolic networks in Boolean logic models, like rFBA for example, is a manual process that requires extensive literature research. So, in the absence of existing integrated models, PROM can be a useful tool since it can generate approximate models in an automated process [27].

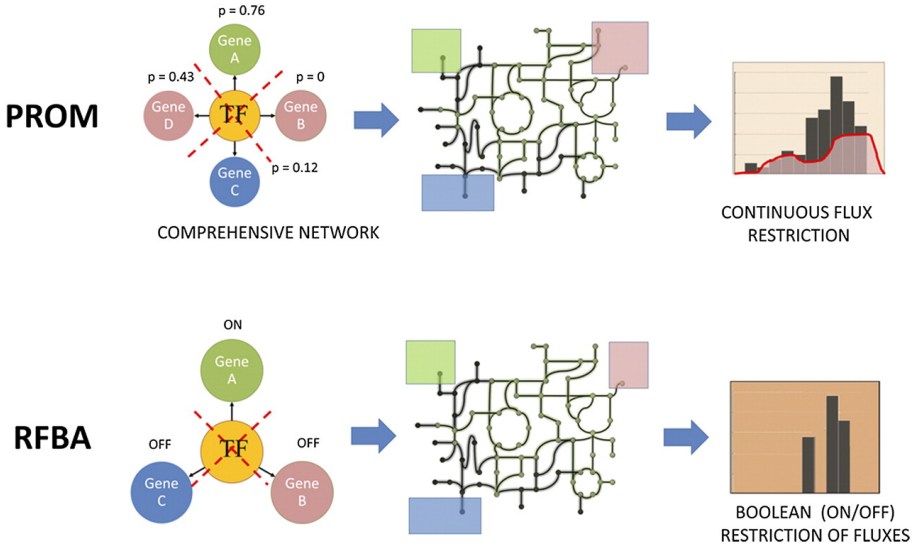

Figure 2.15: Comparison between PROM (Top) and rFBA (Bottom): a perturbation to a TF results in alteration in expression of its target genes. These are then mapped onto the metabolic network. Depending on the gene state, the fluxes through the reactions are constrained and the optimal growth rate is determined by using FBA. In PROM, the constraints based on gene expression (red) are used as cues to obtain the optimal flux state, whereas in rFBA, genes and fluxes can have only two states (on/off). Further, PROM's automated inference of interactions and probabilistic formalism enables it to create comprehensive models [27].

Several other methods, that integrate molecular data, specifically gene expression, have been developed and thoroughly compared. Further information on this topic can be found in this article [28].

3

Tools for regulatory and metabolic integration methods

Contents

3.1 Tools for regulatory network simulations

Currently there are some tools that enable simulation and analysis methods on regulatory network models. The core functionality of these tools is the ability to simulate the behaviour of regulatory models over time. They allow users to define, construct and specify regulatory models. Users can define the components, interactions, and rules governing the system of interest. Some examples of these tools include GINsim, GNA (focused on Piecewise-linear differential equations rather than Boolean models) [29], and The Cell Collective [30]. These tools support SBML-qual files, a file format specific for regulatory models.

3.1.1 GINsim

The Gene Interaction Network Simulator (GINsim) is a computer tool for the modeling and simple simulation of genetic regulatory networks. GINsim allows users to build regulatory network models by specifying the interactions between genes, and it can then simulate the behavior of the network under different conditions. This tool is developed in Java and provides a Graphical User Interface for modelers to view and analyse their models [31, 32]. GINsim encompasses three main components:

- a graphical user interface for regulatory graph definition;
- a simulation core (construction of state transition graphs);
- a graph analysis toolbox

GINsim offers a user-friendly interface with clickable icons and menus, making it easy for users to create regulatory graphs. In these graphs, users can define various aspects for each node:

1. Activity Levels: Users can specify a range of discrete activity levels for each node. This range represents the possible values the associated variable can take.

2. Logical Parameters: Users can define logical parameters for nodes. These parameters determine the target levels of activity based on sets of incoming interactions. By default, both logical parameters and variables are considered as Booleans (true or false), but users can refine them as they gain a better understanding of the logical formalism and the model's properties.

3. Defining Logical Parameters: Creating logical parameters is straightforward. Users can select interactions and associate them with specific discrete values. For each interaction, an interval can be set to specify the range of activity values for which that interaction is effective. By default, the interval encompasses the entire range of activity of the source node, except for zero.

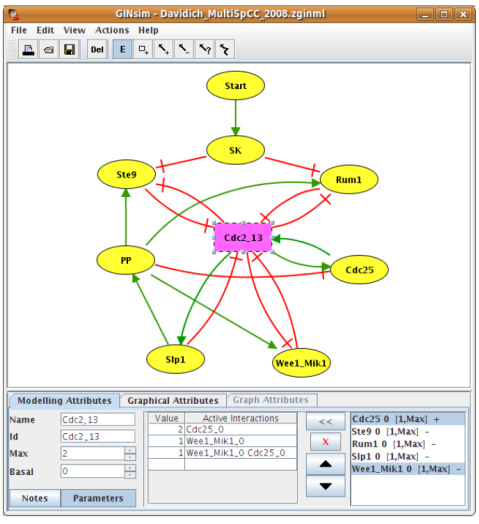

Figure 3.1: GINsim window displaying a regulatory graph [32]

On the basis of the definition of a regulatory graph and the associated logical parameters, GINsim computes the temporal evolution of the system, which is represented in terms of a STG. A graph where each node represents a state and each arrow represents a spontaneous transition. A simulation in GINsim implies:

1. defining a an initial state, or sets of initial states, for the system. This means specifying the initial values for each component in the regulatory network.
2. selecting an updating assumption. GINsim offers different updating modes. It is possible to choose between synchronous and asynchronous modes. There's also a mixed mode that uses priority classes to determine update order.
3. selecting perturbations (mutant conditions). This means blocking certain variable values within a specific range. For example, you can simulate a gene knockdown by blocking its activity to lower levels.
4. Optionally, you can set parameters like maximal depth, graph size, and exploration strategy (depth-first or breadth-first).

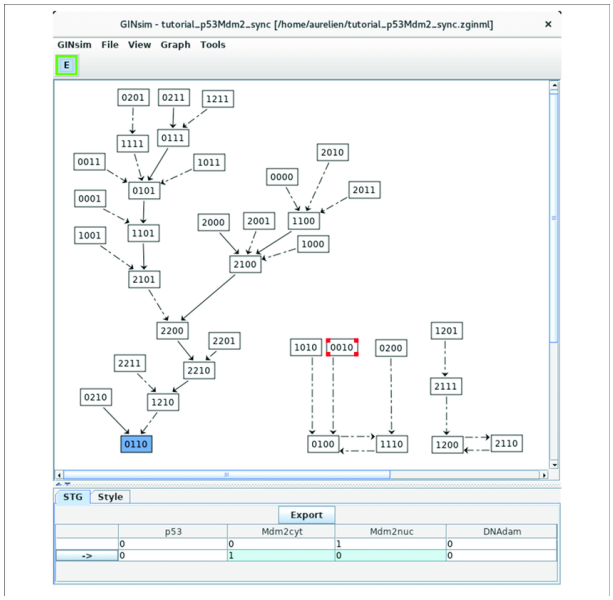

Figure 3.2: GINsim window displaying a STG [33]

Once a STG is computed users can choose a relevant trajectory from the STG and export it to a gnuplot file, allowing for the generation of time plots. Additionally, GINsim provides tools for analyzing and inspecting the STGs. This includes identifying stable states, computing strongly connected components, searching paths between states, and visualizing state transitions interactively.

3.2 Tools for metabolic network simulations

Metabolic network simulations help understanding and predicting the behavior of biochemical reactions within cells. Several tools and software packages are available for conducting metabolic network simulations, each with its own set of functionalities and purposes. Many tools allow constraint-based modelling and analysis of metabolic networks. They are valuable for studying metabolic pathways, predicting the effects of genetic perturbations, and optimizing metabolic engineering strategies. These tools usually allow users to build metabolic models or load them via several types of files (e.g., SBML, CSV). With key functionalities such as FBA, pFBA, FVA, and the identification of essential components, these tools provide users with a comprehensive toolkit for understanding and predicting cellular behavior at the molecular level.

3.2.1 COBRApy

The COnstraint-Based Reconstruction and Analysis (COBRA) is a versatile software suite used for the quantitative prediction of cellular and multicellular biochemical networks through constraint-based modeling. Initially developed as a set of tools for MATLAB, it has since expanded to include Python (COBRApy) and Julia-based modules [34].

COBRApy employs an object-oriented programming approach to represent various components of biochemical networks, including models, metabolites, reactions, and genes. These are represented as class objects with accessible attributes. It supports reading and writing of models in multiple formats, such as MAT-file (for MATLAB variables), JSON, YAML, and SBML. COBRApy provides access to common methods such as FBA, FVA, and gene deletion analysis. It supports GECKO models and represents enzyme-related data as metabolites or reactions, which can be used as constraints in solving constraint-based problems.

COBRApy provides an useful and efficient infrastructure for:

- creating and managing metabolic models
- accessing popular solvers
- analyzing models with methods such as FVA, FBA, pFBA, MOMA etc.
- inspecting models and drawing conclusions on gene essentiality, testing consequences of knockouts etc.

The core capabilities of COBRApy are enabled by a set of classes (3.3) that represent organisms (Model), biochemical reactions (Reaction), and biomolecules (Metabolite and Gene). The core code is accessible through either Python or Jython (Python for Java). COBRApy contains:

1. cobra.io: an input/output package for reading / writing SBML models and reading/writing COBRA Toolbox MATLAB structures.
2. cobra.flux_analysis: a package for performing common FBA operations, including gene deletion and flux variability analysis.
3. cobra.topology: a package for performing structural analysis.
4. cobra.test: a suite of unit tests and test data.
5. cobra.solvers: interfaces to linear optimization packages.
6. cobra.mlab: an interface to the COBRA Toolbox for MATLAB.

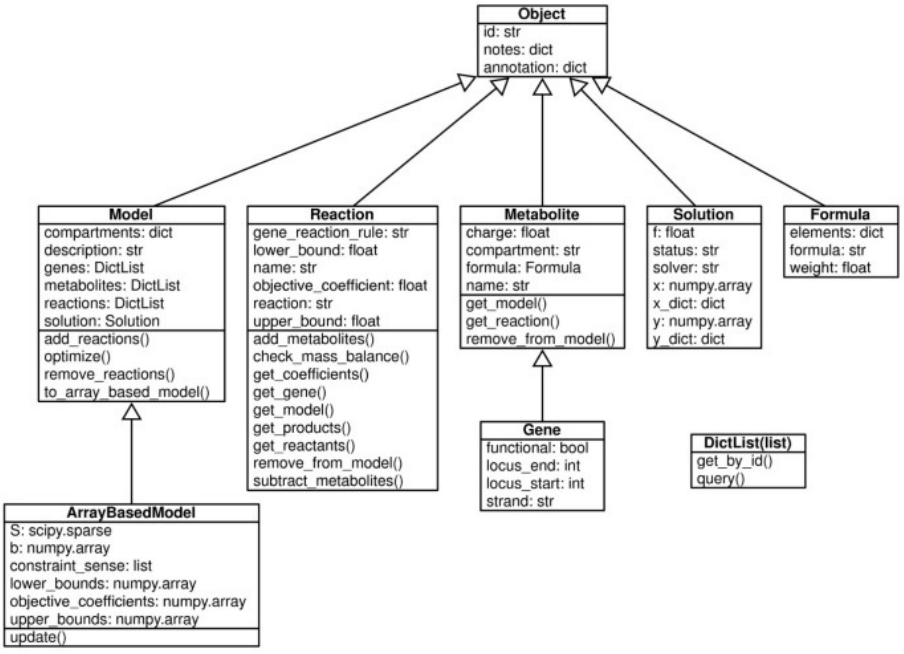

Figure 3.3: Core classes of COBRApy with key attributes and methods listed [35]

The primary classes within COBRApy are Model, Metabolite, Reaction, and Gene. The Model class serves as a container for chemical reactions, metabolites, and associated gene products (Figure 3.4-a). Metabolites are modified by reactions, which may be spontaneous or catalyzed by genes (Figure 3.4-b). Genetic requirements for reactions are defined as Boolean relationships, and the components are explicitly aware of each other within the model. For example, given a Metabolite, it is possible to use the get_reaction() method to determine in which Reactions this Metabolite participates. Then the genes associated with these Reactions may be accessed by the Reaction.get_gene() method [35]. COBRApy can be programmatically accessed and used with Python notebooks.

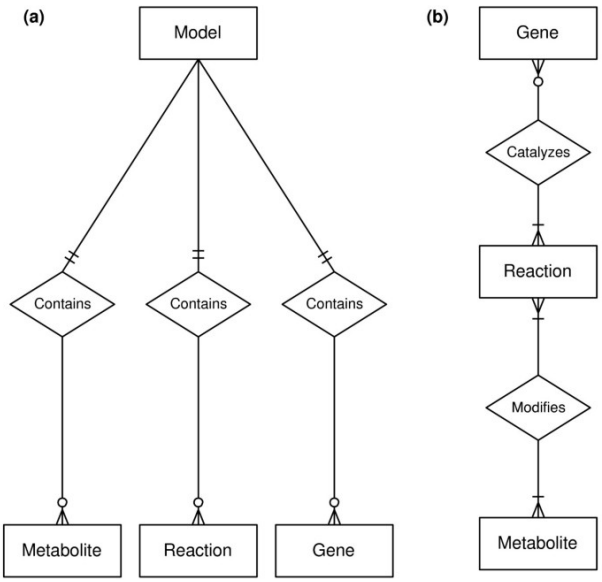

Figure 3.4: Entity relationship diagrams for core classes in COBRApy.

3.2.2 ReFramed

ReFramed[1] is a Python 3 library for metabolic model simulation. When ReFramed originally emerged, it aimed to fill a gap of a Python-based tool for metabolic modelling, other tools existed but ReFramed stood out for its robustness and completeness. It presents a different architecture approach than COBRApy and implements some additional methods . It currently supports 15 different constraint-based simulation methods (FBA variants), including knockout simulation, thermodynamic analysis, protein allocation, transcriptomics integration, and community simulation. Additionally it can be integrated with visualization tools. ReFramed employs an object-oriented programming approach to represent various components of biochemical networks, including models, metabolites, reactions, and genes. These are represented as class objects with accessible attributes. It can load and save models in the SBML format. ReFramed supports conversion of its models from and to COBRAPy models. This allows users to take full advantage of all the features present in both packages. This tool can be accessed and used programmatically with Python notebooks.

```
In [1]:  from reframed import FBA
         FBA(model)
```

```
Out[1]:  Objective: 0.8739215069684306
         Status: Optimal
```

[1] https://github.com/cdanielmachado/reframed

Name	Long name	Reference
FBA	Flux Balance Analysis	(Varma and Palsson, 1993)
FVA	Flux Variability Analysis	(Mahadevan and Schilling, 2003)
pFBA	Parsimonious FBA	(Lewis et al, 2010)
FBrAtio	FBA with flux ratios	(Yen et al, 2013)
CAFBA	Constrained Allocation FBA	(Mori et al, 2016)
MOMA	Minimization of Metabolic Adjustment	(Segre et al, 2002)
lMOMA	linear MOMA	(Segre et al, 2002)
ROOM	Regulatory On/Off Minimization	(Shlomi et al, 2005)
ll-FBA	loopless FBA	(Schellenberger et al, 2011)
TFA	Thermodynamic Flux Analysis	(Henry et al, 2007)
TVA	Thermodynamic Variability Analysis	(Henry et al, 2007)
NET	Network-embedded thermodynamic analysis	(Kummel et al 2006)
GIMME	Gene Inactivity Moderated by Met and Exp	(Becker and Palsson, 2008)
E-Flux	E-Flux	(Colijn et al, 2009)
SteadyCom	Community simulation	(Chan et al, 2017)

Table 3.1: ReFramed Methods

3.2.3 OptFlux

OptFlux is a user-friendly computational tool that can be used to optimize and analyze metabolic network models. It is fully implemented in Java and offers a modular architecture, allowing new features and services to be easily integrated [36]. The main modules of OptFlux include:

- **Modelling**: OptFlux enables users to build and manipulate stoichiometric metabolic models, including reactions, metabolites, equations, and gene-reaction associations. It supports various model formats such as SBML, Metatool, and flat files.

- **Simulation**: Users can perform simulations to understand how the metabolic system behaves under different environmental and genetic conditions. OptFlux provides different simulation methods, including FBA and MOMA, along with tools for FVA. It is also possible to find essengial genes and reactions using this tool.

- **Optimization**: OptFlux offers optimization capabilities to identify sets of genetic modifications (reactions or genes) that maximize specific objectives while considering microorganism growth. Optimization algorithms include OptKnock, Evolutionary Algorithms (EAs), and Simulated Anneal-

ing (SA).

- **Pathway Analysis**: The tool integrates the EFMTool for calculating Elementary Flux Modes (EFMs), essential for understanding metabolic pathways. OptFlux provides a user-friendly interface for filtering EFMs based on specific patterns.

The tool also allows for the integration of Boolean network-based regulatory models with metabolic models, enabling phenotype simulation and strain optimization methods like rFBA and SR-FBA.

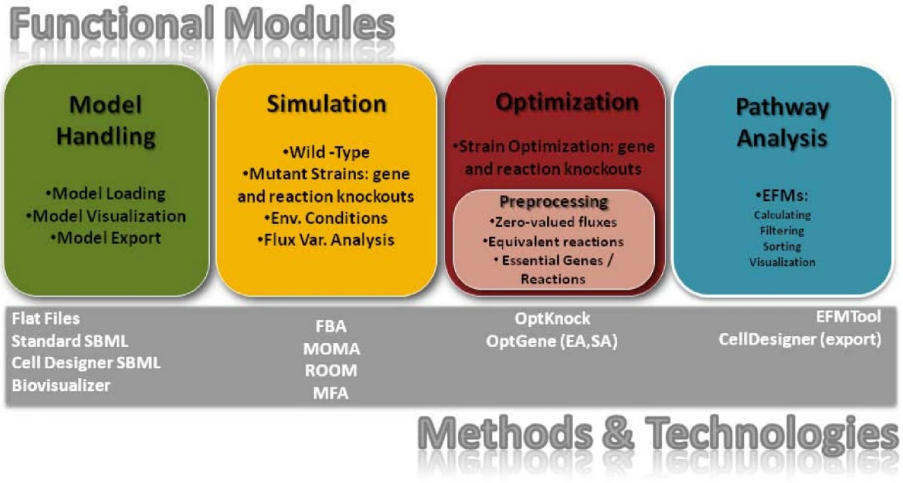

Figure 3.5: Functional modules of the OptFlux application.

3.3 Tools for metabolic and regulatory network simulations

Currently, numerous tools are available for metabolic and regulatory models, but they primarily concentrate on their respective layers, overlooking the relevant interconnection between these layers within a cell. There are few tools developed to integrate both layers, despite the significant potential they hold in accurately representing the cell and its behaviour. A couple of notable examples include MEWpy, which is implemented in Python and can be used programmatically through notebooks. And OptFlux, implemented in Java, this tool provides a user-friendly graphical interface.

3.3.1 MEWpy

MEWpy is a computational strain optimization (CSO) tool able to aggregate different types of constraint-based models and simulation approaches. MEWpy relies on EAs to identify a set of genetic modifications that can enhance and optimize a desired metabolic engineering outcome. The architecture of MEWpy encompasses three layers a problem definition layer, a phenotype simulation layer and an optimization layer:

1. Problem Definition Layer: Defines the CSO problem, including the modeling framework, modification targets (genes, reactions, proteins, regulatory variables), modification strategy, target product, and environmental conditions.

2. Phenotype Simulation Layer: Contains methods to evaluate mutant strains generated by the CSO algorithm, including various simulation techniques like FBA, rFBA and more.

3. Optimization Layer: Encompasses optimization algorithms used for strain optimization and objective functions to evaluate candidate solutions.

MEWpy offers different modelling approaches, they can account for the following types of constraints:

- Metabolic Constraints: MEWpy allows users to assess the effects of genetic and reaction modifications on cellular phenotypes. It does this by converting GPR rules into flux constraints. GPR rules, which describe gene-reaction relationships, are transformed into algebraic expressions, replacing Boolean operators (AND, OR) with mathematical functions (min, max), and gene identifiers are substituted with expression values. Users can manipulate reaction fluxes by adjusting their boundaries, enabling the study of over-expression, under-expression, and gene deletions.

- Enzymatic Constraints: In addition to metabolic constraints, MEWpy offers tools to incorporate enzymatic constraints into models. While GEMs capture metabolism and gene-reaction interactions, they often lack critical information like enzyme kinetics and abundance. MEWpy addresses this by allowing users to impose enzymatic constraints using models like GECKO or sMOMENT (short MetabOlic Modelling with ENzyme kineTics). These additional constraints enhance the accuracy of phenotype predictions and support strain optimization.

- Regulatory Constraints: MEWpy recognizes that the interplay between gene regulation and metabolism is complex and not fully represented by GEMs alone. Therefore, it provides tools to impose regulatory constraints on models. This includes algorithms like OptORF and OptRAM, which consider gene deletions, up-regulation, down-regulation, and other regulatory strategies. OptORF, for example, identifies gene deletions for metabolic genes and transcription factors, while OptRAM explores a broader range of strategies. MEWpy also supports multi-objective optimization, offering added flexibility and benefits for diverse optimization goals.

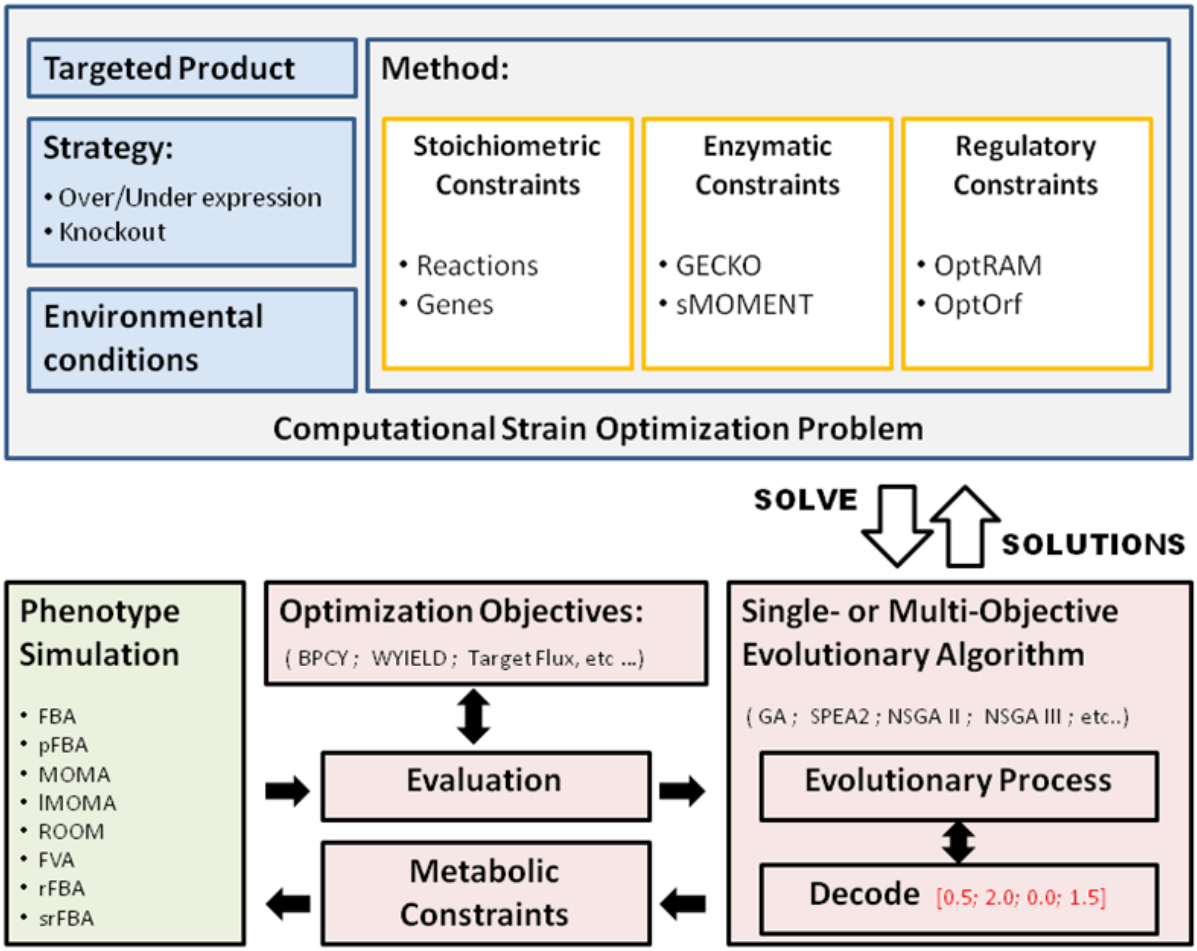

Figure 3.6: MEWpy architecture [37]

MEWpy currently supports ReFramed and COBRApy phenotype simulators integrating both in a common API which enables different simulation methods like FBA, pFBA, FVA or MOMA from external tools [37].

In order to one of the modelling approaches account for regulatory constraints, MEWpy has been designed to adeptly represent integrated models. MEWpy offers robust support for the integration of regulatory and metabolic models at the genome-scale. It provides a comprehensive suite of tools within its *mewpy.germ* module, which allows users to efficiently construct, simulate, and analyze Genome-scale Regulatory and Metabolic (GERM) models.

A GERM model includes a standard GEM model. The GEM model contains reactions (w/ GPRs), metabolites and genes. It also includes exchange reactions defining the environmental conditions of the system. In addition to the GEM, GERM models include Transcriptional Regulatory Networks (TRNs). A TRN consists of interactions, often described using boolean algebra expressions, connecting target genes to their respective regulators. These networks also account for external stimuli (effectors), regula-

tory metabolites, and regulatory reactions. GEM models and TRNs are often linked by the target genes of the TRN and the genes of the GEM model. MEWpy supports several methods to perform phenotype simulations using integrated GERM models [37]. The *mewpy.germ.analysis* module provides various methods, including FBA and pFBA, which can be employed with a metabolic model alone, as well as rFBA, SR-FBA, PROM, and CoRegFlux, which require the use of a Regulatory-Metabolic model. To assemble a GERM model the GEM data within the GERM model can be obtained from a variety of input sources, which include:

1. SBML file
2. JSON file
3. COBRApy model object
4. ReFramed model object

And the TRN data can be obtained from:

1. CSV file
2. SBML file

3.3.2 OptFlux

OptFlux goes beyond metabolic modeling by offering a comprehensive suite of tools for integrated analysis and simulation of metabolic and regulatory models. With the Reg4OptFlux plugin, users can seamlessly combine and analyze both types of models. OptFlux enables the process of loading regulatory models by supporting file formats such as Excel (.xls and .xlsx) and comma-separated values (.csv). By integrating these models, OptFlux enables simulation methods like rFBA and SR-FBA on the integrated model. Furthermore, OptFlux provides tools to identify essential genes within the regulatory model. Users can also perform gene knockout simulations, leverage core metabolic simulation methods, and access a selection of regulatory simulation tools.

Tools				
Method	**COBRApy**	**ReFramed**	**OptFlux**	**MEWpy**
pFBA	X	X	X	X
FVA	X	X	X	X
rFBA			X	X
SR-FBA			X	X
iFBA				
MOMA	X	X	X	X

Table 3.2: Methods available for each tool

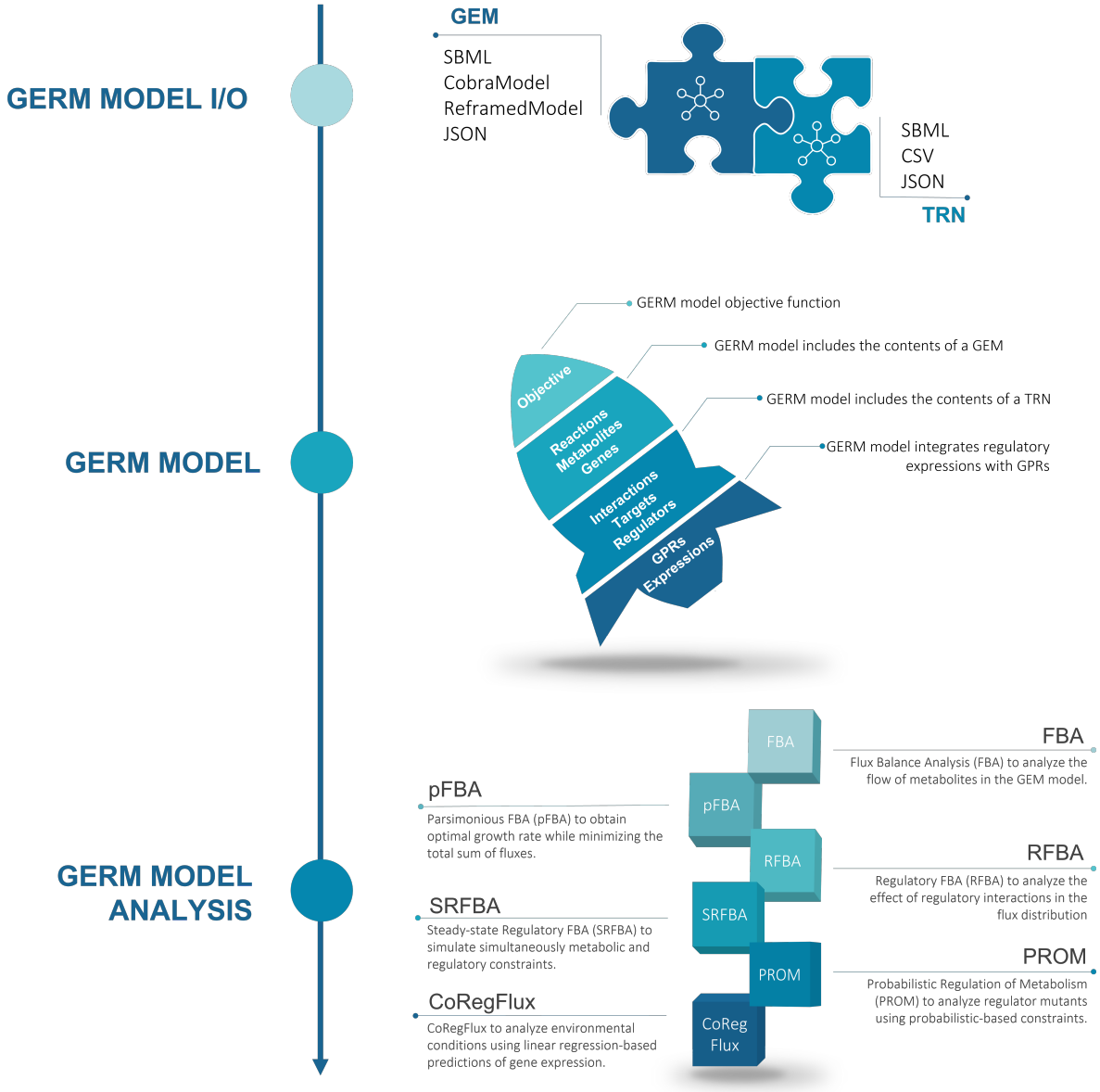

Figure 3.7: MEWpy GERM module overview [37]

4

Implemented Solution

Contents

Having thoroughly examined and assessed the available tools, including their functionalities and underlying architectures, the decision has been made to expand upon MEWpy as the solution for the integration of metabolic and regulatory networks. In particular, the focus will be on enhancing the capabilities of the *mewpy.germ* module for this purpose.

While MEWpy demonstrates remarkable functionality, particularly in its integration of metabolic and regulatory networks, it introduces a an issue by reimplementing the definition of metabolic components. MEWpy, specifically in the integrated models module (*mewpy.germ*), includes its own implementation of metabolic models, metabolic variables (reactions, metabolites, genes, etc.), and methods (FBA, FVA, etc.). This results in a certain redundancy because there are already well-established and widely used tools, such as COBRApy and ReFramed, that offer the same capabilities. This redundancy poses challenges in terms of adaptability and maintenance, especially in the face of evolving methods and tools.

MEWpy's current implementation entails a complete duplication of metabolic model representation, encompassing its own definitions of essential components such as reactions, genes, and metabolites.

MEWpy also reimplements essential methods like FBA and FVA. This practice of reimplementing established components and methods within MEWpy raises several concerns. Firstly, any changes or updates to these methods, whether due to improved algorithms or the introduction of new techniques, require a manual reimplementation within MEWpy. This not only hampers the software's agility but also places a substantial maintenance burden on developers. Additionally, it limits MEWpy's ability to incorporate novel functionalities that may emerge in the field.

To address these challenges and enhance the efficiency of MEWpy, it is advisable to consider integrating external tools, such as COBRApy and ReFramed, for managing the metabolic component of integrated models. These widely adopted tools already provide comprehensive support for metabolic modeling, offering well-established and continually updated methods. By adopting these external tools, MEWpy can align with industry standards and leverage the expertise and resources of the broader systems biology community.

Benefits of Integration:

1. **Efficiency**: Integrating external tools allows MEWpy to avoid redundant reimplementations of metabolic components and methods.

2. **Adaptability**: By relying on external tools, MEWpy can quickly incorporate updates and changes in metabolic modeling techniques, ensuring that it remains current and aligned with evolving best practices.

3. **Enhanced Functionality**: Integration with established tools enables MEWpy to access a broader range of functionalities and methods without the need for extensive in-house development.

In conclusion, MEWpy's current practice of reimplementing metabolic components and methods, although functional, poses significant challenges in terms of adaptability and maintenance. By integrating external tools like COBRApy and ReFramed, MEWpy can enhance its efficiency and stay up-to-date with the latest developments in the field. This approach not only mitigates redundancy but also paves the way for MEWpy to become a more versatile and sustainable tool for the integration of metabolic and regulatory networks.

4.1 Current Architecture of *mewpy.germ*

In order to clearly explain the developed solution for integrating external tools like COBRApy or ReFramed in the MEWpy's metabolic and regulatory integrated models module (*mewpy.germ*), it is necessary to explain how it is currently designed. A simplified overview of its classes can be depicted in Figure 4.1. The module *mewpy.germ* implements the integration of metabolic and regulatory models, represented by GERM models.

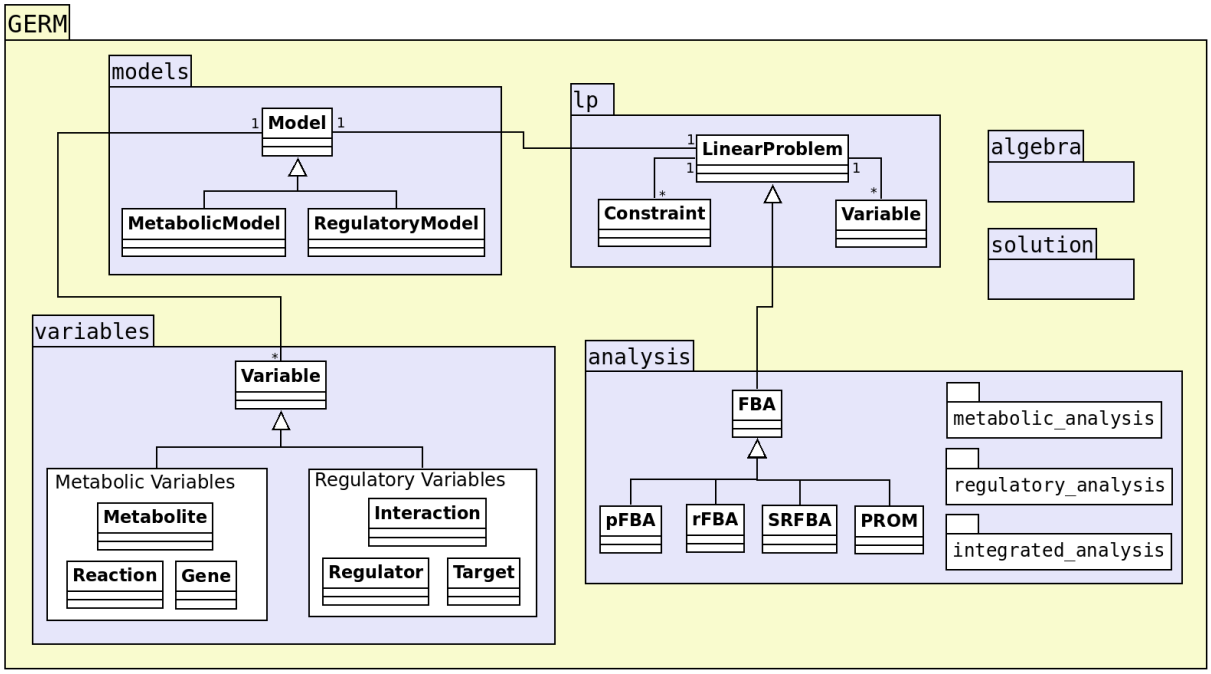

Figure 4.1: Simplified UML Diagram of *mewpy.germ* module

4.1.1 Folder Structure

This module is organized into six distinct folders:

1. **algebra**: This module serves as a repository for utility functions that are essential for solving linear problems within the module.

2. **analysis**: Within this module, you will find the definitions of simulation methods, such as FBA and SR-FBA. Additionally, it contains various analysis methods, including FVA and the identification of essential genes, for example.

3. **lp**: The lp module contains the definitions of linear problems, which are subsequently implemented in the simulation methods present in the analysis folder. This separation of concerns ensures a modular and efficient approach to solving complex problems.

4. **models**: In the models module, you will find the definitions of classes related to the models themselves. This includes the representation of metabolic models, regulatory models, and other relevant components.

5. **solution**: This module is dedicated to the definition of classes related to the solutions generated by simulation methods. It encapsulates the results of simulations and analysis performed within the module.

6. **variables**: The variables module plays a pivotal role in defining the entities within the models. It contains definitions for metabolites, reactions, genes (for metabolic models), and targets, regulators, and interactions (for regulatory models).

4.1.2 GERM Models Representation

A GERM model emerges from an integration between a metabolic model and a regulatory model. MEWpy has its own definition and implementation of these models.

A – Metabolic Model Metabolic models are structured around three primary components: genes, metabolites, and reactions. A metabolic model can be associated with an objective function for the analysis of the model. The model can be loaded with compartments, although these can be inferred from the available metabolites. It also can hold additional information such as demand reactions, exchange reactions, sink reactions, GPRs and external compartment. The *MetabolicModel* class and its attributes and methods are represented in Figure 4.2. The metabolic model, as with other models, provides a clean interface for manipulation with the add, remove and update methods. One can perform the following operations:

- Add reactions, metabolites and genes
- Remove reactions, metabolites and genes
- Update the objective function

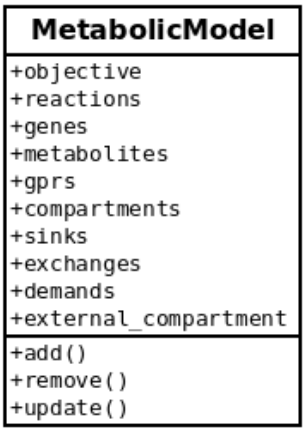

Figure 4.2: Metabolic Model class

B – Reaction Metabolic models primarily rely on reactions as their fundamental variables. These reactions serve as comprehensive containers of essential information, encompassing critical details

such as bounds, metabolite involvement, stoichiometry, and GPR rules. The GPR rules establish the connections between reactions, enzymes/proteins, and genes. Each reaction typically contains the stoichiometry of the metabolites that take part in the reaction. Furthermore, reactions encompass GPR rules or logic that outlines the genes associated with the reaction. In addition to these components, reactions also specify bounds, which define the permissible ranges of flux values during simulation. A reaction can infer from the previous attributes its related genes, metabolites, reactants, products, and reversibility. The *Reaction* class and its attributes and methods are represented in Figure 4.3. One can perform the following operations:

- Knockout, which sets the reaction bounds to zero
- Add metabolites or gpr to the reaction
- Remove metabolites or gpr from the reaction

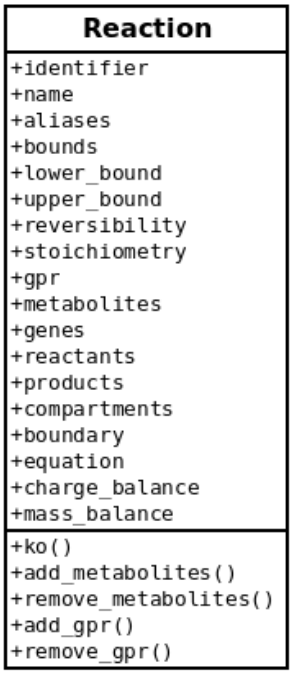

Figure 4.3: Reaction Class

C – Gene A metabolic gene is regularly associated with metabolic reactions. A gene contains multiple coefficients for GPR rule evaluation. It usually takes either 0 (inactive) or 1 (active). A gene object also holds information regarding the reactions that it is associated with. It can also perform the operation knockout, which sets its coefficients to 0. The *Gene* class and its attributes and methods are represented in Figure 4.4.

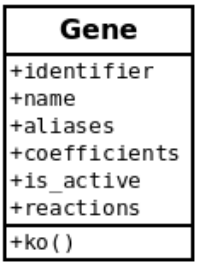

Figure 4.4: Gene class

D – Metabolite A metabolite is regularly associated with reactions. It holds information regarding the charge, compartment, formula and reactions to which is associated. The *Metabolite* class and its attributes and methods are represented in Figure 4.5.

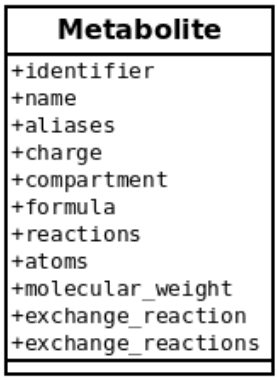

Figure 4.5: Metabolite class

E – Regulatory Model Regulatory models have as main attributes interactions, targets and regulators. The interactions are between the targets and the regulators. Each interaction contains a set of regulatory events that determine the state or coefficient of the target gene. A regulatory event consists of a boolean algebra expression and the corresponding coefficient. If the boolean expression is evaluated to True, the resulting coefficient is applied to the target gene. Otherwise, the coefficient is ignored. The regulatory model also contains a set of environmental stimuli, which are associated with interactions. The environmental stimuli can be used to define the initial state of the model. The *RegulatoryModel* class and its attributes and methods are represented in Figure 4.6. The regulatory model, as with other models, provides a clean interface for manipulation with the add, remove and update methods. One can perform the following operations:

- Add a new interaction, regulator or target
- Remove an interaction, regulator or target

- Update the compartments of the model

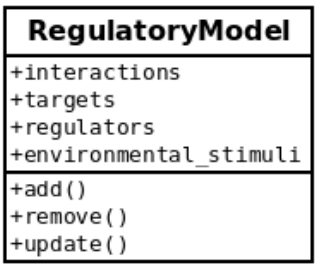

Figure 4.6: Regulatory Model Class

F – Interaction A regulatory interaction is frequently associated with a target and the regulatory events modelling the coefficients of this target variable. These events dictate the potential values that the target variable can assume, based on the underlying logic and conditions embedded within an expression object. This expression object must encompass all the relevant variables and contributors involved in the regulatory event/expression. The set of contributors can be inferred directly from these regulatory events. Also, these events are used to generate a table, which outlines the possible coefficients of the target for the default (or not) coefficients of the regulators. The *Interaction* class and its attributes and methods are represented in Figure 4.2. An interaction can perform the following operations:

- Add targets or regulatory events to the interaction. This removes the previous ones.
- Remove targets or regulatory events from the interaction.

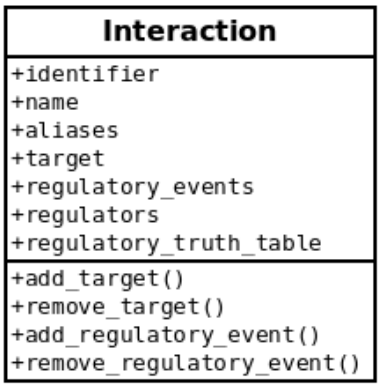

Figure 4.7: Interaction class

G – Regulator A regulator is commonly associated with interactions and can usually be available as target too. Regulators can control the gene expression of several target genes. It holds information regarding the coefficients that can take and the interactions to which is associated. It can also perform the operation knockout, which sets its coefficients to 0. The *Regulator* class and its attributes and methods are represented in Figure 4.8.

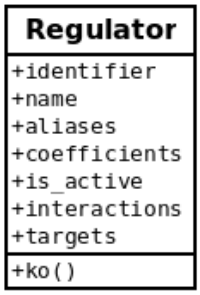

Figure 4.8: Regulator class

H – Target A target gene is associated with a single interaction but can be regulated by multiple regulators. The regulatory interaction establishes a relationship between a target gene and regulators. A target gene holds the coefficients that it can take. While 0 and 1 are added by default, these coefficients can be changed later. In regulatory models, a target gene can also be represented as a regulator gene, as it can be inferred from the regulatory rules defining regulatory interactions. It can also perform the operation knockout, which sets its coefficients to 0. The *Target* class and its attributes and methods are represented in Figure 4.9.

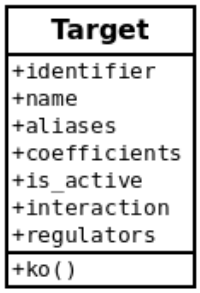

Figure 4.9: Target class

I – Factory design pattern MEWpy employs a factory design pattern to generate a specialized model known as a GERM Model. The key requirement for a GERM Model is that it must integrate both metabolic and regulatory information to ensure accuracy. When MEWpy receives data encompassing both metabolic and regulatory components for a model, it uses the factory design pattern to construct

an instance that combines the attributes and methods of both the MetabolicModel and RegulatoryModel classes. A GERMModel class and its attributes and methods is represented in Figure 4.10 This new instance type can take on two forms: MetabolicRegulatoryModel or RegulatoryMetabolicModel, and it possesses all the functionalities and characteristics of both the MetabolicModel and RegulatoryModel classes. Essentially, it seamlessly merges metabolic and regulatory information to create a comprehensive and unified model. To ensure comprehensive integration, this instance is equipped with two additional attributes: regulatory_metabolites and regulatory_reactions. These attributes store information related to the reactions and metabolites present in the regulatory component of the model.

Figure 4.10: MetabolicRegulatoryModel or RegulatoryMetabolicModel class

This same principle extends to the model variables. A variable within the model can have multiple roles, meaning it can function, for example, both as a metabolite and a regulator. This dual role is what bridges the gap between the two biological components, metabolic and regulatory. This mapping process is carried out through ID matching. When an ID is identified as having multiple roles, the factory design pattern, previously applied to the models, is also applied to the variables. In essence, to accurately represent variables with multiple roles, they need to encompass all relevant information from all of their roles. MEWpy achieves this by utilizing the factory design pattern to generate new instance types that possess all the attributes and methods required from the respective classes. For instance, a variable that serves as both a metabolite and a regulator will have its instance type defined as either *MetaboliteRegulator* or *RegulatorMetabolite*.

4.1.3 Reading GERM Models

To assemble an integrated GERM model, MEWpy provides the *mewpy.io.read_model* function. This function is capable of working with different ”engines,” which are essentially different methods or sources for reading data. The available engines in the MEWpy are listed in Table 4.1.

Engine	**IO**	**Model**
BooleanRegulatoryCSV	CSV File	Regulatory
CoExpressionRegulatoryCSV	CSV File	Regulatory
TargetRegulatorRegulatoryCSV	CSV File	Regulatory
RegulatorySBML	SBML File	Regulatory
MetabolicSBML	SBML File	Metabolic
CobraModel	cobra.core.model	Metabolic
ReframedModel	reframed.core.cbmodel	Metabolic
JSON	JSON File	Metabolic

Table 4.1: MEWpy engines

To use this function, it is necessary to create a *Reader* object. This *Reader* object requires two main arguments: an *engine* (which specifies how the data should be read) and an *io* (which is the input data source, either a file path or an object, like a COBRApy or ReFramed model). It is also possible to provide additional optional arguments as needed. This function takes as argument the information from the readers you specify and generates a model. This model can be one of three types: a *MetabolicModel*, a *RegulatoryModel*, or a *GERMModel*. This process is demonstrated in Figure 4.11).

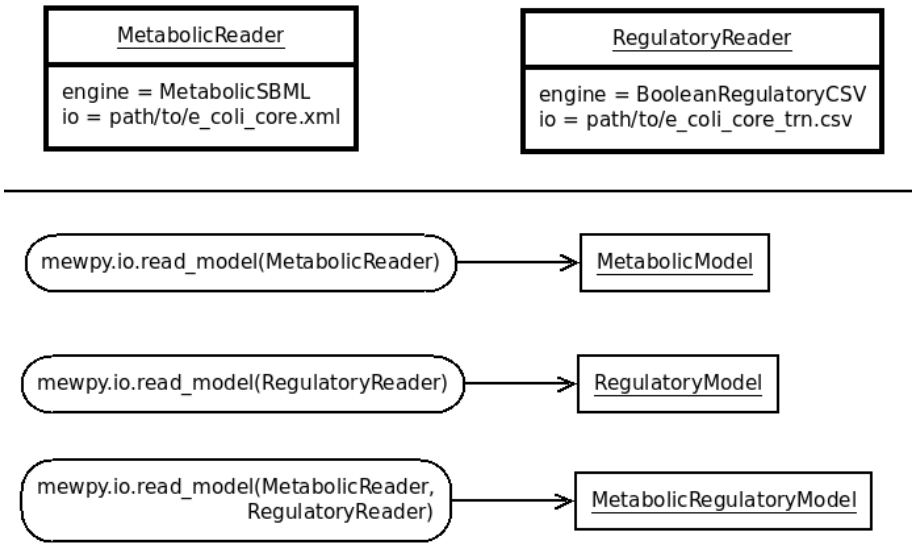

Figure 4.11: MEWpy reading GERM models examples

4.1.4 Simulations in GERM models

The simulation methods are defined in *mewpy.germ.analysis* module and they derive from the class *mewpy.germ.lp.LinearProblem* (Figure 4.12). These simulation methods are used for phenotype simulation and come with the following attributes and methods:

- Attributes:
  - method: This attribute stores the name of the simulation method.
  - model: It represents the model used to construct the linear problem.
  - solver: This attribute holds an instance of a MEWpy solver, which contains the linear programming implementation for variables and constraints. Available solver options include CPLEX, GUROBI, and OPTLANG.
  - constraints: It represents the mathematical representation of ordinary differential equations (ODE) that need to be implemented in the solver using linear programming.
  - variables: This attribute represents the system variables that need to be implemented in the solver using linear programming.
  - objective: It holds a linear representation of the objective function associated with the linear problem.
- Methods:

– build: The build method is responsible for gathering variables and constraints from a GERM model based on the mathematical formulation specific to each simulation method.

– optimize: The optimize method is responsible for solving the linear problem using linear programming or mixed-integer linear programming. This method can accept both method-specific arguments (such as initial state, dynamic settings, etc.) and solver-specific arguments (including linear, minimize, constraints, get_values, etc.). These arguments have the ability to temporarily override certain constraints or variables during the optimization process. The optimize method returns a ModelSolution object by default, which contains the objective value, value of each variable in the solution, among others.

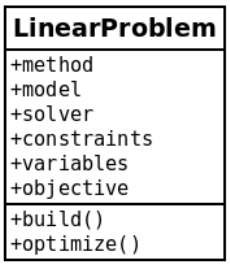

Figure 4.12: Linear Problem Class

A typical approach for employing simulation methods with MEWpy involves the following workflow:

In [1]:
```
model = read_model(reader1, reader2) # read the model
rfba = RFBA(model) # initialize the simulation method
rfba.build() # build the linear problem
solution = rfba.optimize() # perform the optimization
model.reactions['MY_REACTION'].bounds = (0, 0) # make changes to the model
solution = RFBA(model).build().optimize() # initialize, build and optimize
                                          # the simulation
```

The *mewpy.germ.analysis* module also includes FVA and other analysis methods. A typical approach for employing analysis methods with MEWpy involves the following workflow:

In [2]:
```
# reading the E. coli core model
core_gem_reader = Reader(Engines.MetabolicSBML, 'e_coli_core.xml')
model = read_model(core_gem_reader)

# FVA returns the DataFrame with minium and maximum values of each reaction
fva(model)
```

4.1.5 MEWpy phenotype simulations

MEWpy also offers provides wrappers for external models, such as COBRApy or ReFramed models. This wrappers, called simulators, provide a common interface to realize the main phenotype analysis tasks on these external models. The *simulator* interface remains the same regardless of how the model was loaded, using ReFramed or COBRApy. This simplify the use of both environments and ease the management of future changes and deprecation on their APIs. The *simulator* offers a wide API, and enable to perform basic tasks, such as, list metabolites, reactions, genes and compartments. Phenotype simulations are also run using the simulator instance using the simulate method.

In [1]:
```
# using REFRAMED
from reframed.io.sbml import load_cbmodel
model = load_cbmodel('iML1515.xml',flavor='cobra')

# using COBRApy
from cobra.io import read_sbml_model
model = read_sbml_model('iML1515.xml')
```

In [2]:
```
# build a phenotype simulator
from mewpy.simulation import get_simulator
simul = get_simulator(model)
```

In [3]:
```
# list 10 reactions
simul.reactions[:10]
```

Out[3]:
```
['CYTDK2',
 'XPPT',
 'HXPRT',
 'NDPK5',
 'SHK3Dr',
 'NDPK6',
 'NDPK8',
 'DHORTS',
 'OMPDC',
 'PYNP2r']
```

In [4]:
```
# FBA
result = simul.simulate()
# or
result = simul.simulate(method='FBA')
```

```
Out[4]:          objective: 0.8769972144269748
                 Status: OPTIMAL
```

4.2 Implemented solution architecture

The main objective is not to encompass the entirety of metabolic components within the GERM model but rather to maintain a reference to an external tool metabolic model through MEWpy's simulators. Consequently, every occasion requiring the retrieval of a metabolic variable from the GERM model will be accomplished via the simulator interface. A simplified overview of the implemented solution's classes can be depicted in Figure 4.13.

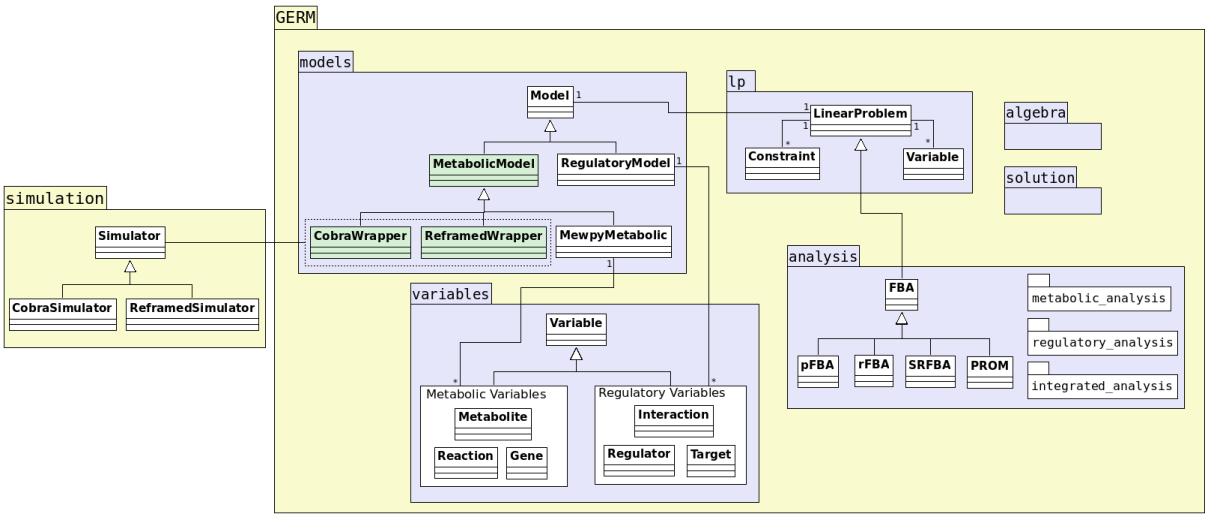

Figure 4.13: Simplified UML Diagram of the implemented solution on *mewpy.germ* module. Major differences are highlighted in green.

4.2.1 New Model Classes Hierarchy

To achieve this goal, a significant transformation of MEWpy's metabolic model implementation is required. In the revamped architecture, variables such as reactions or metabolites are no longer locally stored within MEWpy. Instead, they reside in external tool models, imposing access through a simulator.

To address these changes, *MetabolicModel* is now an abstract class. This abstract class serves as the foundation for various subclasses, including MEWpy's current *MetabolicModel* implementation, which is now renamed *MewpyMetabolicModel*. Keeping MEWpy's current representation of metabolic models ensures full backward compatibility with previous MEWpy implementations. Additionally, there are new separate classes for each supported external tool: *CobraModelWrapper* and *ReframedModelWrapper*. These classes and their relationships are represented in Figure 4.14. This restructuring

allows us to seamlessly integrate external tool models while preserving MEWpy's existing functionality, ensuring a smooth transition for MEWpy users.

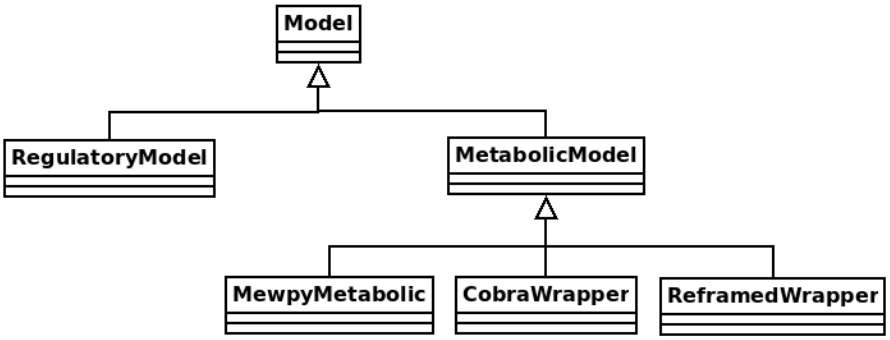

Figure 4.14: Model classes hierarchy

4.2.2 New Engines

To instantiate these new subclasses new engines were created. This change was required because all the previously existing engines instantiated the single metabolic model representation, which is now the *MewpyMetabolicModel*.

In response to these alterations, there are two new engines: the *CobraModelEngine* and the *ReframedModelEngine*. These engines are specifically designed to instantiate the *CobraModelWrapper* and *ReframedModelWrapper* metabolic models, respectively. They receive inputs in the form of COBRApy and ReFramed model instances. In MEWpy's current implementation of engines, which took COBRApy and ReFramed models as input, the workflow entailed copying every variable from the external tool metabolic model into a MEWpy metabolic model, essentially "translating" them into MEWpy's data structure (e.g. transforming a COBRApy reaction instance into a MEWpy reaction instance.) This is redundant and time consuming. With these new subclasses, this is no longer necessary because the external tool metabolic model is stored, and its variables are accessed only when required. Nonetheless, in situations where a variable exists in both regulatory and metabolic models, accessing its metabolic data becomes crucial at the moment of reading the models. This is imperative because, in such cases, a factory design pattern variable must be instantiated, such as *regulatory_metabolites* or *regulatory_reactions*. Rather than accessing and translating every variable from the external tool metabolic model, the implemented tool now retrieves only those variables that are also present in the regulatory model. The available engines in the implemented solution are listed in Table 4.2.

Engine	IO	Model
BooleanRegulatoryCSV	CSV File	Regulatory
CoExpressionRegulatoryCSV	CSV File	Regulatory
TargetRegulatorRegulatoryCSV	CSV File	Regulatory
RegulatorySBML	SBML File	Regulatory
MetabolicSBML	SBML File	MewpyMetabolic
CobraModelEngine	cobra.core.model	CobraWrapper
ReframedModelEngine	reframed.core.cbmodel	ReframedWrapper
JSON	JSON File	MewpyMetabolic

Table 4.2: Updated MEWpy engines

4.2.3 Working with variables

In these new subclasses, all metabolic variables remain accessible through the same interface as the current MEWpy metabolic model representation. This accessibility is made possible through the following workflow: The data from the metabolic variables is retrieved via a *simulator* (from *mewpy.simulation*) that accesses the external tool metabolic model. The data supplied by the simulator is structured in dictionary format. Subsequently, the data contained within the retrieved dictionary is harnessed to instantiate the corresponding variable in the MEWpy data structure. This process ensures that users can continue to interact with these variables using the familiar interface. Furthermore, it is worth noting that the methods *get()*, *add()*, and *remove()* continue to operate in the same manner as they did with the previous metabolic models, ensuring consistency and ease of use for MEWpy users.

4.2.4 Simulation and Analysis

Given that all variables remain accessible through the same interface as the previous metabolic model representation, it is noteworthy that all simulation and analysis methods within *mewpy.germ.analysis* continue to function seamlessly with the new wrapper classes. However, it is essential to recognize that numerous metabolic-related methods are offered by the external tool itself. In these instances, leveraging the external tool's methods proves highly advantageous. By utilizing the external tool methods, MEWpy gains increased adaptability. Relying on these external tools enables MEWpy to promptly integrate updates or modifications to these methods. It is important to highlight that the utilization of external tool methods can be facilitated through the *simulator* that these wrapper models have as attribute. When a method is to be executed in MEWpy, the system performs a check to ascertain whether the associated model has an external tool implementation of the method. If such an implementation exists, MEWpy employs the *simulator*'s capabilities to run the method via *simulator.simulate(method)*, which delegates the

processing of the method to the external tool. Subsequently, the output of this method is translated into a solution in the MEWpy data structure (*mewpy.solvers.solution.Solution*). However, in cases where the associated model lacks an external tool implementation of the method, MEWpy defaults to utilizing its local implementation and solver, ensuring a robust and versatile approach to metabolic modeling across a wide range of scenarios 4.15.

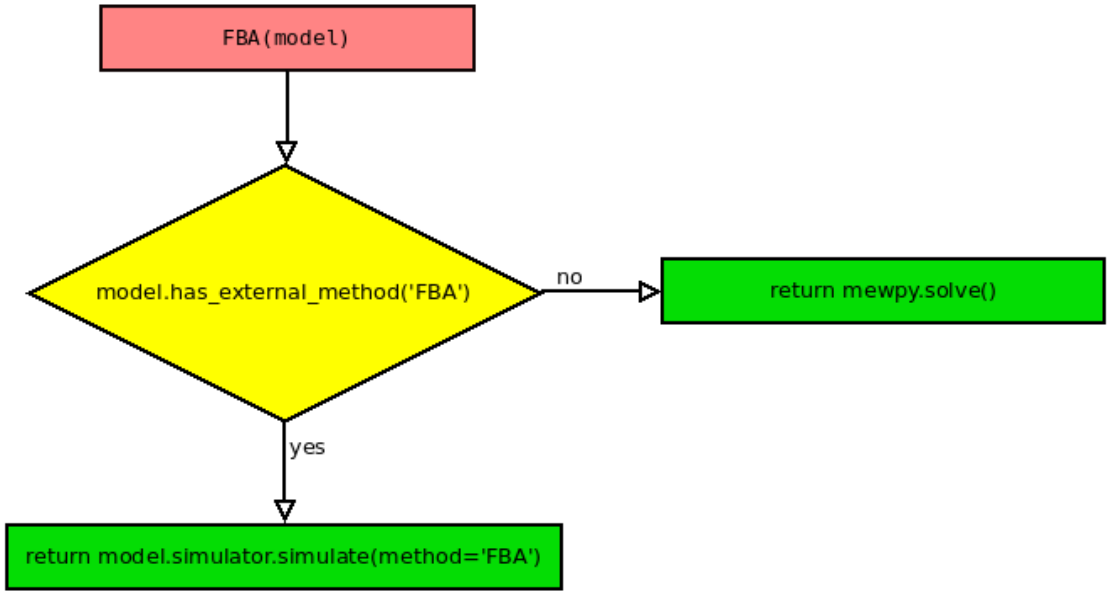

Figure 4.15: FBA simulation behaviour. The same behaviour applies to other simulation or analysis methods

4.2.5 Tutorial

To instantiate the implemented solution new models it is required to use one of the new engines

In [1]:
```
from mewpy.io import Reader, Engines

# using COBRApy
from cobra.io import read_sbml_model
ext_model = read_sbml_model("e_coli_core.xml")
reader = Reader(Engines.CobraModelEngine, ext_model)

# using ReFramed
from reframed import load\_cbmodel
ext_model = load_cbmodel("e_coli_core.xml")
reader = Reader(Engines.RefframedModelEngie, ext_model)
```

It is also necessary a regulatory model to construct a GERM model.

In [2]:
```
trn_model = "e_coli_core_trn.csv"
trn_reader = Reader(Engines.BooleanRegulatoryCSV,
                    trn_model,
                    sep=',',
                    id_col=0,
                    rule_col=2,
                    aliases_cols=[1],
                    header=0)
```

Read the GERM model, using both metabolic and regulatory related Readers.

In [3]:
```
from mewpy.io import read_model
germ_model = read_model(reader, trn_reader)
```

Having a GERM model, it is possible to list its variables.

In [4]:
```
germ_model.objective
```

Out[4]:
```
{'Biomass_Ecoli_core': Biomass_Ecoli_core || 1.496 3pg_c + 3.7478 accoa_c +
```

In [5]:
```
germ_model.reactions
```

Out[5]:
```
{'ACALD': ACALD || 1.0 acald_c + 1.0 coa_c + 1.0 nad_c <-> 1.0 accoa_c + 1.0
 '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,
 ...
 'TKT2': TKT2 || 1.0 e4p_c + 1.0 xu5p__D_c <-> 1.0 f6p_c + 1.0 g3p_c,
 'TPI': TPI || 1.0 dhap_c <-> 1.0 g3p_c}
```

In [6]:
```
germ_model.regulators
```

Out[6]:
```
{'surplusFDP': surplusFDP || (0.0, 1.0),
 'surplusPYR': surplusPYR || (0.0, 1.0),
 'b0113': b0113 || (0.0, 1.0),
 'b3261': b3261 || (0.0, 1.0),
 'b0400': b0400 || (0.0, 1.0),
 ...
 'LDH_D': LDH_D || 1.0 lac__D_c + 1.0 nad_c <-> 1.0 h_c + 1.0 nadh_c + 1.0 p
 'SUCCt2_2': SUCCt2_2 || 2.0 h_e + 1.0 succ_e -> 2.0 h_c + 1.0 succ_c}
```

In [7]:
```
germ_model.regulatory_reactions
```

Out[7]:
```
{'Biomass_Ecoli_core': Biomass_Ecoli_core || 1.496 3pg_c + 3.7478 accoa_c +
 'FBP': FBP || 1.0 fdp_c + 1.0 h2o_c -> 1.0 f6p_c + 1.0 pi_c,
 'TKT2': TKT2 || 1.0 e4p_c + 1.0 xu5p__D_c <-> 1.0 f6p_c + 1.0 g3p_c,
 'TALA': TALA || 1.0 g3p_c + 1.0 s7p_c <-> 1.0 e4p_c + 1.0 f6p_c,
 'PGI': PGI || 1.0 g6p_c <-> 1.0 f6p_c,
 ...
 'LDH_D': LDH_D || 1.0 lac__D_c + 1.0 nad_c <-> 1.0 h_c + 1.0 nadh_c + 1.0 p
 'SUCCt2_2': SUCCt2_2 || 2.0 h_e + 1.0 succ_e -> 2.0 h_c + 1.0 succ_c}
```

It is possible to run simulations with GERM models.

In [8]:
```
from mewpy.germ.analysis import  FBA, SRFBA, fva, ifva
```

In [9]:
```
FBA(germ_model).build().optimize()
```

Out[9]:
```
Method  FBA
Model   Model e_coli_core - E. coli core model - Orth et al 2010
Objective   Biomass_Ecoli_core
Objective value 0.8739215069684303
Status  optimal
```

In [10]:
```
SRFBA(germ_model).build().optimize()
```

Out[10]:
```
Method  SRFBA
Model   Model e_coli_core - E. coli core model - Orth et al 2010
Objective   Biomass_Ecoli_core
Objective value 0.8739215069684829
Status  optimal
```

In [11]:
```
ifva(germ_model)
```

Out[11]:

	minimum	maximum
ACALD	0.000000e+00	0.000000e+00
ACALDt	-2.273737e-13	-2.273737e-13
ACKr	-5.506706e-13	-5.506706e-13
ACONTa	6.007250e+00	6.007250e+00
ACONTb	6.007250e+00	6.007250e+00
...	...	...
TALA	1.496984e+00	1.496984e+00
THD2	0.000000e+00	1.394085e-11
TKT1	1.496984e+00	1.496984e+00
TKT2	1.181498e+00	1.181498e+00
TPI	7.477382e+00	7.477382e+00

5

Evaluation and Discussion

Contents

5.1 Evaluation

To assess the performance of the implemented solution, I subjected it to a set of unit tests and conducted an efficiency comparison with MEWpy's current implementation. In this evaluation, I chose to work with the extensively documented and studied organism, *Escherichia coli*. This selection was made because it offers access to published GEM models [38] and comprehensive whole-genome regulatory models [39].

5.1.1 Functionality

Unit tests are invaluable for evaluating a solution's functionality as they provide a granular examination of individual components or units of code. These tests are designed to isolate and scrutinize specific

parts of the software, ensuring that they perform as intended in isolation. To validate the functionality of the *mewpy.germ* module within the developed solution, I conducted comprehensive unit testing using two distinct test files: *test_cobra_wrapper.py* and *test_reframed_wrapper.py* Each of these test files corresponds to a specific wrapper class, namely CobraModelWrapper and ReframedModelWrapper, these wrapper classes are represented in Figure 4.14. During these unit tests, I compared the results obtained from the implemented solution metabolic model representation against the original MEWpy metabolic model representation results. For both of the unit test files, the reference model (original MEWpy metabolic model) utilized was a GERM model generated by the engines MetabolicSBML and BooleanRegulatoryCSV. In the test_cobra_wrapper file, a GERM model generated by the engines CobraModelEngine and BooleanRegulatoryCSV is tested. In the test_reframed_wrapper file, a GERM model generated by the engines ReframedModelEngine and BooleanRegulatoryCSV is tested, these engines are present in Table 4.2. The unit tests files contain the following tests:

- **Read Model**: This test validates if the process of loading a GERM model is accurate. It aims to confirm that the number of several variables aligns with the expected values. These variables encompass metabolic components, such as reactions and metabolites, regulatory elements, including interactions and regulators, and integrated variables, which are present in both metabolic and regulatory models, such as regulatory metabolites and regulatory reactions. This test also checks if all these variables have their attributes with the expected values. For example, it examines whether all reactions conform to the expected stoichiometry and GPR.
- **Simulations**: This test ensures the functionality of simulation techniques such as FBA, as well as integrated methods like rFBA and SR-FBA. Its goal is to ensure that both the reference model and the new solution model yield consistent results.
- **Analysis**: This test aims to confirm the consistency of analysis methods, such as FVA, by comparing their outcomes between the reference model and the new solution model.
- **Model Handling**: This test verifies the model handling methods like ”add” and ”remove” are producing the expected behaviour. It adds and removes both metabolic and regulatory variables.

To assess the quality of the unit test suite, the coverage was calculated. Test coverage is a ratio between the number of lines executed by at least one test case and the total number of lines of the code base. The results of this evaluation are detailed in Table 5.1.

Test File	Statements	Missing	Coverage
Cobra	10521	4945	47%
Reframed	11287	5636	44%

Table 5.1: Unit tests coverage results. Statements represent the total number of lines of the code base. Missing represents the lines of the code that the tests did not cover.

5.1.2 Performance

To assess the performance of the implemented solution, I conducted timing tests on various operations, comparing them to MEWpy's existing implementation. These operations included tasks such as reading a GERM model, executing simulation methods, and performing analysis methods.

A – Reading a GERM model To read a GERM model, it requires the use of two distinct Readers, one for the metabolic model and another for the regulatory network. Among the available engines, there are numerous combinations that can be employed to read a GERM model. To instantiate the implemented solution GERM model representation, it is imperative to employ the newly developed engines, namely the CobraModelEngine or ReframedModelEngine (listed in Table 4.2). These engines take as input instances of metabolic models generated by external tools. This evaluation aims to compare the new engines performance in the process of reading a GERM model against MEWpy's current implementation existing engines. From the available engines, listed in 4.1, the ones chosen for this evaluation were CobraModel and ReframedModel because these also take as input input instances of metabolic models generated by external tools. In this evaluation, I compared the time it takes to read a GERM model using MEWpy's CobraModel engine versus the implemented solution's CobraModelEngine. In both cases, the same engine was used to read the regulatory model. The results are depicted in Figure 5.1. A similar approach was followed for ReFramed. In this scenario, MEWpy's ReframedModel engine was employed, along with the implemented solution's ReframedModelEngine. The outcomes of this comparison can be observed in Figure 5.2.

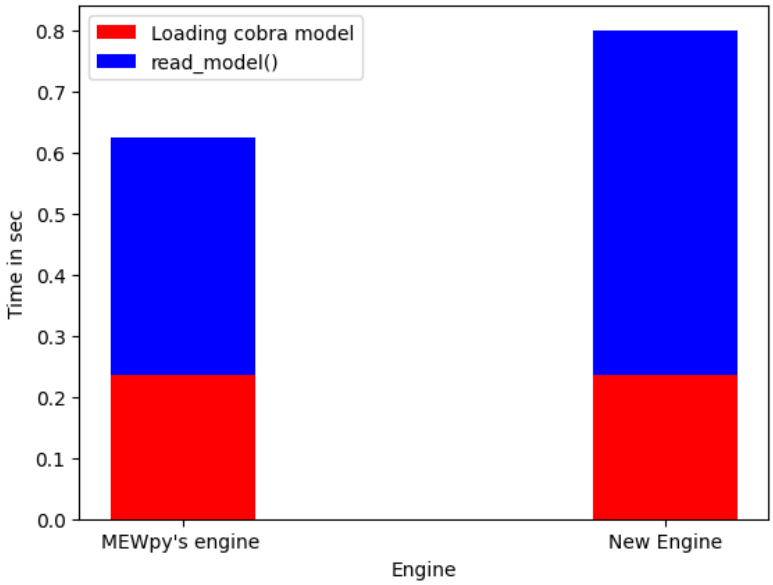

Figure 5.1: Time Comparison of reading a GERM model from a COBRA metabolic model

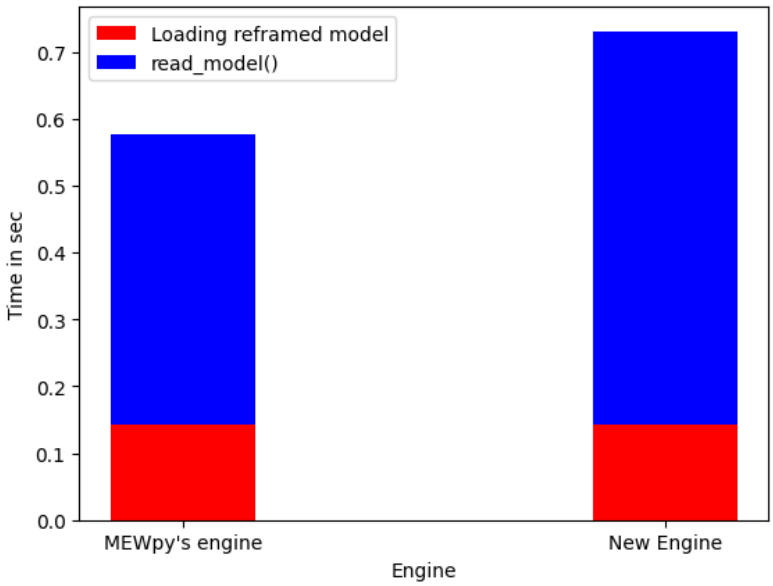

Figure 5.2: Computational Efficiency in reading a GERM model from a ReFramed metabolic model

Although the implemented solution differs from the current MEWpy approach by not needing the replication of every metabolic variable, such as reactions and metabolites, it does, however, require more time to read a GERM model. This added time is a result of the necessity to instantiate GERM variables, those that are present in both metabolic and regulatory models. To accomplish this, it is necessary to retrieve the metabolic-related data for these variables, which entails an often time-consuming process of searching for their IDs within the metabolic model.

B – Metabolic Simulations In the implemented solution, the metabolic simulations and analyses methods on GERM Models are now performed by an external tool associated to the model. This leads to conducting a comparative analysis, assessing the computation time of these tasks in the implemented solution against MEWpy's existing implementation, which employs its own solvers for the same processes. To perform a FBA in MEWpy you have to call the methods *build*, which set the variables and constraints of the linear problem, and *optimize* which optimizes the objective function and creates a flux solution.

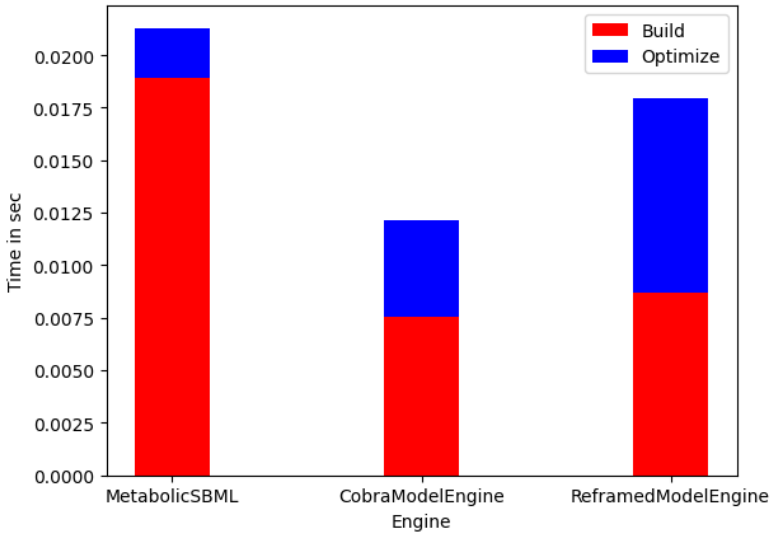

Figure 5.3: Computational Efficiency in FBA with a GERM model read using different engines. The leftmost bar symbolizes a GERM model with its metabolic component derived from the MetabolicSBML engine, the middle bar represents a GERM model read using a CobraModel engine, and the rightmost bar showcases a GERM model generated by ReframedModel engine.

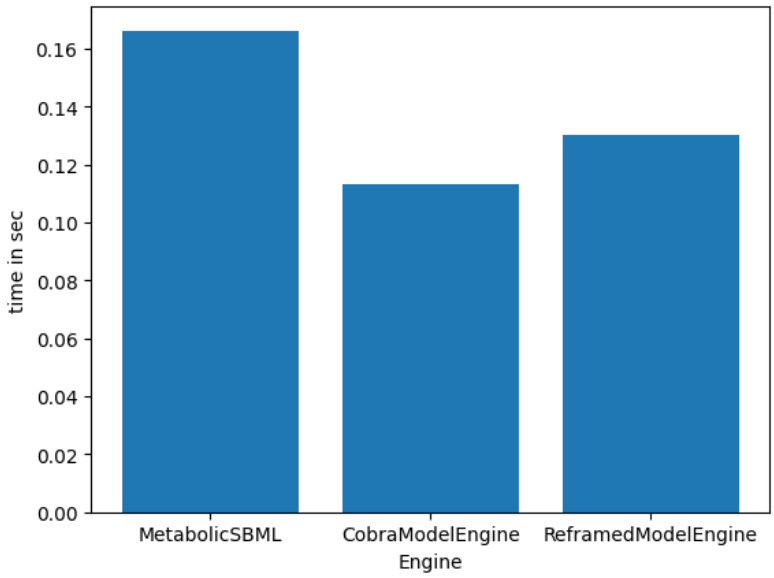

Figure 5.4: Computational Efficiency in FVA with a GERM model read using different engines. The leftmost bar symbolizes a GERM model with its metabolic component derived from the MetabolicSBML engine, the middle bar represents a GERM model read using a CobraModel engine, and the rightmost bar showcases a GERM model generated by ReframedModel engine.

As illustrated in Figure 5.3, using the new engines, which leads to the utilization of the new wrapper classes for representing metabolic models within a GERM model demonstrates a performance advantage over MEWpy's metabolic model representation, particularly a 40% decrease in *build* execution

time. This enhancement arises from the elimination of the need to set variables and constraints, as the linear problem optimization is delegated to an external tool. It is worth noting that this streamlined approach also results in a slightly extended duration for the *optimize* step. This is primarily because the *optimize* step essentially comprises aspects of both the *build* and *optimize* stages of the external tool. Additionally, it involves the translation of the solution data structure from the external tool into MEWpy's data structure. Similar to the FBA, the execution of FVA also exhibits a execution time decrease when utilizing the new wrapper classes to represent metabolic models within a GERM model, as illustrated in Figure 5.4.

C – Integrated Simulations In most cases, external tools do not incorporate integrated methods like SR-FBA or rFBA. As a result, the implemented solution relies on MEWpy's solvers to handle the execution of these methods. To enable MEWpy's solvers to establish constraints and variables for linear problems, they require access to metabolic data. Since the implemented solution stores metabolic data within an external tool's metabolic model representation, a necessary step involves translating the format of variables from the external tool to MEWpy's data structure. For instance, if a reaction is represented using COBRApy classes, it must be converted into MEWpy's *Reaction* class format.

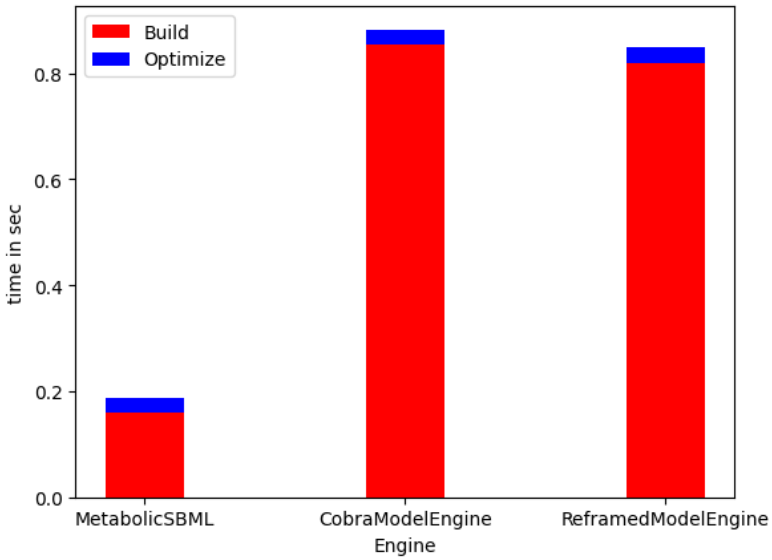

Figure 5.5: Computational Efficiency in SR-FBA with a GERM model read using different engines. The leftmost bar symbolizes a GERM model with its metabolic component derived from the MetabolicSBML engine, the middle bar represents a GERM model read using a CobraModel engine, and the rightmost bar showcases a GERM model generated by ReframedModel engine.

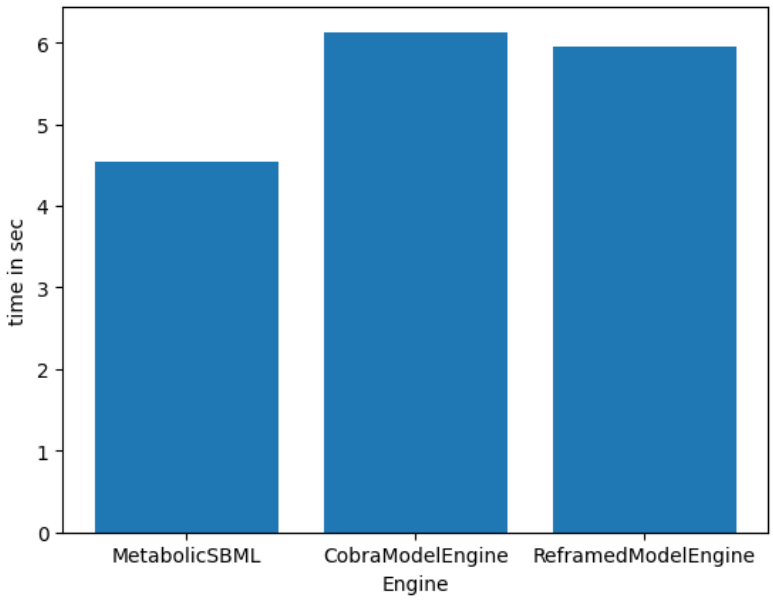

Figure 5.6: Computational Efficiency in iFVA with a GERM model read using different engines. The leftmost bar symbolizes a GERM model with its metabolic component derived from the MetabolicSBML engine, the middle bar represents a GERM model read using a CobraModel engine, and the rightmost bar showcases a GERM model generated by ReframedModel engine.

As depicted in Figure 5.5 (particularly in the *build* step) and Figure 5.6, it is evident that integrated simulations exhibit longer processing times when executed through the GERM model instances that include wrapper classes. This delay is primarily due to the additional step of translating the data required for these simulations.

5.2 Use case

Additionally, a use case was conducted to validate the successful integration of a regulatory model and a GECKO model, thereby achieving the objective of harnessing external tool functionalities. In the specific use case scenario, the widely studied bacterium *E. coli* was utilized [15]. In the use case, a GEM model (*iML1515*) and a GECKO model (*eciML1515*) are compared [40]. They are both integrated with a regulatory model of *E. coli*, generating two GERM models: *gem_model* and *ec_model*.

In [1]:
```
gem_model
```

Out[1]:

```
Model   e_coli_core
Name    model
Types   regulatory, metabolic
Compartments    e, c, p
Reactions   2712
Metabolites 1877
Genes   1516
Exchanges   331
Demands 6
Sinks   0
Objective   BIOMASS_Ec_iML1515_core_75p37M
Regulatory interactions 159
Targets 159
Regulators  45
Regulatory reactions    9
Regulatory metabolites  11
Environmental stimuli   3
```

In [2]:

```
ec_model
```

Out[2]:

```
Model   e_coli_core
Name    ecModel_batch of iML1515
Types   regulatory, metabolic
Compartments    e, c, p
Reactions   6084
Metabolites 3593
Genes   1505
Exchanges   662
Demands 12
Sinks   0
Objective   BIOMASS_Ec_iML1515_core_75p37M
Regulatory interactions 159
Targets 159
Regulators  45
Regulatory reactions    9
Regulatory metabolites  11
Environmental stimuli   3
```

GECKO models extend the scope to incorporate enzyme-level considerations. This is achieved by including an extra entity for each metabolic reaction that represents enzyme usage. By incorporating enzyme-related data, we introduce additional constraints that further refine the variability of flux within each reaction. To validate this, an integrated Integrated Flux Variability Analysis (iFVA) was conducted on each model, as depicted in Figure 5.7. Notably, the *ec_model* exhibits a more restricted range of

possible flux for its reactions compared to the *gem_model*.

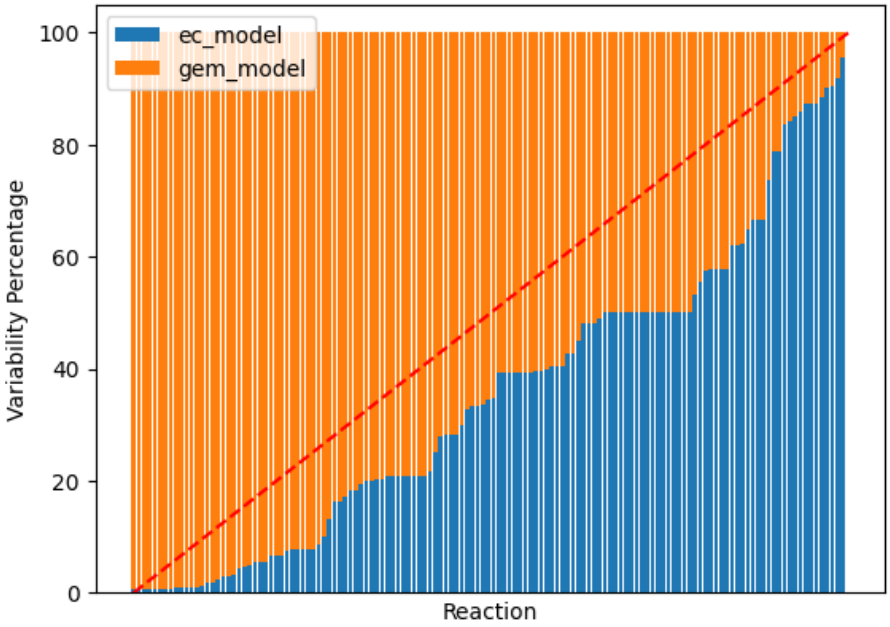

Figure 5.7: Comparative integrated Flux Variability Analysis of ec_model and gem_model. This stacked bar chart illustrates the comparative iFVA of ec_model and gem_model, with respect to a set non-null reactions (reactions with no flux). Each bar in the chart corresponds to an individual reaction and is divided into two segments: one representing the flux variability in the ec_model and the other in the gem_model. Flux variability, expressed as a percentage on the Y-axis, is a measure of the range within which the rate of chemical reactions can vary. The percentage indicates the relative contribution of each model to the total flux variability for a given reaction. The calculation involves computing the absolute difference between upper and lower bounds for each model and then expressing the percentage for the ec_model as a fraction of the total variability (ec_model and gem_model combined) for that reaction.

These additional constraints further constraint the solution space of possible flux distribution simulations of these models, enhancing accuracy and also biological relevance. Since GECKO models have the same structure as GEM models we can test these improved simulations.

```
In [3]:  FBA(gem_model).build().optimize()
```

```
Out[3]:  Method  FBA
         Model   Model e_coli_core - model
         Objective   BIOMASS_Ec_iML1515_core_75p37M
         Objective value 0.8769972144269684
         Status  optimal
```

```
In [4]:  FBA(ec_model).build().optimize()
```

Out[4]:

```
Method  FBA
Model   Model e_coli_core - ecModel_batch of iML1515
Objective   BIOMASS_Ec_iML1515_core_75p37M
Objective value 0.574364117726333
Status  optimal
```

To further enhance the precision and biological relevance of simulations, one can leverage MEWpy's integrated methods, including SR-FBA, to incorporate regulatory constraints.

In [1]:

```
SRFBA(gem_model).build().optimize()
```

Out[1]:

```
Method  SRFBA
Model   Model e_coli_core - model
Objective   BIOMASS_Ec_iML1515_core_75p37M
Objective value 0.8769972144281424
Status  optimal
```

In [2]:

```
SRFBA(ec_model).build().optimize()
```

Out[2]:

```
Method  SRFBA
Model   Model e_coli_core - ecModel_batch of iML1515
Objective   BIOMASS_Ec_iML1515_core_75p37M
Objective value 0.5699763017347708
Status  optimal
```

The use case presented here demonstrates the integration of GECKO models with regulatory models using the implemented solution. It successfully fulfills the objective of leveraging external tool functionalities, and it is a compelling example of how this integration enhances precision and biological relevance in the context of metabolic modeling.

5.3 System Limitations

In the current implementation of MEWpy, users can knock out specific metabolic variables, such as *Reactions* or *Genes*, by using the *ko()* method on instances of these variables. When you call the *ko()* method, it sets the bounds or coefficients associated with these variables to zero. This is significant because knocking out these variables introduces additional constraints which narrow the solution space within linear problem simulations. However, there is a key issue in the current implementation: metabolic variables are not stored within the metabolic model itself. Instead, they are accessed when needed from the external tool's metabolic model. When accessing these metabolic variables, the implemented

solution generates a temporary copy of the variable in MEWpy's data structure. These temporary copies can call the *ko()* method, but any changes made by this method only affect the copy and are not reflected in the actual variable stored in the external tool's model. This results in inconsistency within the metabolic model. Every time these variables need to be accessed again, a new copy will be generated with the original bounds or coefficients. Unfortunately, this limitation means that the *ko()* method is unable to achieve its intended functionality as it doesn't alter the original external tool's model variable. In addition, the current MEWpy GERM models are equipped with a history management feature. This means they support temporary modifications through the *with model* context manager and provide operations like *undo()*, *redo()*, *reset()* and *restore()* as depicted in the code below. However, as this functionality involves the *ko()* method as a model alteration, it was not integrated into the implemented solution and remains untested [37].

In [1]:
```
# make a temporary change to the model
pfk = model.get('PFK')

with model:
    model.remove(pfk)
    print('Does the model contain PFK?', 'PFK' in model.reactions)

print('Has PFK removal been reverted?', 'PFK' in model.reactions)
```

Out[1]:
```
Does the model contain PFK? False
Has PFK removal been reverted? True
```

6

Conclusion and Future Work

This thesis aimed to create a tool for integrating metabolic (GEM and GECKO) models with regulatory networks for more accurate and biologically relevant simulations. To achieve this, the implemented solution extended MEWpy, an existing metabolic modeling tool with the capability of integrating these metabolic and regulatory models. The developed extension incorporated external tools' metabolic representations into the integrated model functionality of MEWpy. This enhancement bolstered the robustness of MEWpy's integrated model functionality by outsourcing its metabolic component to well-established external tools like COBRApy or ReFramed. With this extension, MEWpy not only became more efficient, as it no longer needs to manage metabolic implementation and code maintenance, but it also became more adaptable, capable of readily incorporating recent changes and updates from external tools. Furthermore, this extension expanded MEWpy's functionality, allowing it to harness additional features offered by external tools. One example of a relevant additional functionality is COBRApy's ability to read GECKO models. GECKO models are an extension to traditional GEM models, they are more detailed because they enrich the models by integrating enzyme constraints. With the implemented solution, MEWpy can integrate a COBRApy GECKO model with a regulatory model. This integration grants a more comprehensive representation of cellular processes, ultimately culminating in simulation results that are more accurate and biologically pertinent. The implemented solution underwent a set of unit tests to ensure its functionality and compatibility. Furthermore, a comprehensive performance analysis

was conducted, which revealed that some tasks, such as importing models and conducting integrated method simulations, experienced longer execution times compared to MEWpy's current implementation. However, metabolic simulation methods showed an improvement in efficiency. This tradeoff was deemed advantageous, given the substantial additional functionality and biological relevance it brings to the tool. Some functionalities in MEWpy's current implementation, such as model history management and performing knockouts on metabolic variables are not fully functional in the implemented solution. To address these limitations and maximize the potential of the implemented solution, various strategies for future work are proposed. Firstly, implementing the integration of the *ko()* method and the history management feature would enhance the model's functionality and usability. Secondly, optimizing the process of translating external tool variables into MEWpy data structure could lead to increased efficiency, reducing runtime disparities between the implemented solution and MEWpy's current implementation. These proposed improvements and enhancements would ensure that MEWpy continues to evolve and meet the growing demands of metabolic modeling and analysis. The implemented tool is fully operational, capable of seamlessly integrating metabolic models, including both GEM and GECKO, with regulatory networks. It provides a programmatic interface, allowing users to easily simulate integrated methods and conduct analyses on this accurate cellular representation . The code has undergone thorough testing, and it follows lean principles while effectively applying design patterns. It is accessible to the public for usage and extension in a GitHub repository[1]. The tool is now in a state where it can be submitted as a pull request for integration into MEWpy's repository. The developed solution enables users to achieve a more precise representation of cell behavior, thereby enhancing the ability to predict and understand it. By integrating metabolic and regulatory networks and incorporating functionalities from external tools, this advancement offers a practical approach to improving the accuracy of our cell behavior predictions.

[1] https://github.com/SebastiaoZoio/MEWpy

Bibliography

[1] B. O. Palsson, *Systems biology: constraint-based reconstruction and analysis*. Cambridge University Press, Cambridge, UK2015, 2015.

[2] T. Lengauer and C. Hartmann, "Bioinformatics," in *Comprehensive Medicinal Chemistry II*. Elsevier, 2007, pp. 315–347. [Online]. Available: https://doi.org/10.1016/b0-08-045044-x/00088-2

[3] L. Kuepfer, "Stoichiometric modelling of microbial metabolism," in *Methods in Molecular Biology*. Springer New York, 2014, pp. 3–18.

[4] J. D. Orth, I. Thiele, and B. O. Palsson, "What is flux balance analysis?" *Nature Biotechnology*, vol. 28, no. 3, pp. 245–248, Mar. 2010.

[5] K. D. Rawls, B. V. Dougherty, E. M. Blais, E. Stancliffe, G. L. Kolling, K. Vinnakota, V. R. Pannala, A. Wallqvist, and J. A. Papin, "A simplified metabolic network reconstruction to promote understanding and development of flux balance analysis tools," *Computers in Biology and Medicine*, vol. 105, pp. 64–71, Feb. 2019. [Online]. Available: https://doi.org/10.1016/j.compbiomed.2018.12.010

[6] D. K. Fred Glover and N. V. Phillips, *Network Models in Optimization and Their Applications in Practice*. Wiley-Interscience, 1992, p. 249–259.

[7] N. E. Lewis, K. K. Hixson, T. M. Conrad, J. A. Lerman, P. Charusanti, A. D. Polpitiya, J. N. Adkins, G. Schramm, S. O. Purvine, D. Lopez-Ferrer, K. K. Weitz, R. Eils, R. König, R. D. Smith, and B. O. Palsson, "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, Jan. 2010.

[8] C. Diener and S. Gibbons, "Gibbons-lab/isb_course_2021," https://github.com/Gibbons-Lab/isb_course_2021, 2021.

[9] R. Mahadevan and C. Schilling, "The effects of alternate optimal solutions in constraint-based genome-scale metabolic models," *Metabolic Engineering*, vol. 5, no. 4, pp. 264–276, Oct. 2003.

[10] D. Segrè, D. Vitkup, and G. M. Church, “Analysis of optimality in natural and perturbed metabolic networks,” *Proceedings of the National Academy of Sciences*, vol. 99, no. 23, pp. 15 112–15 117, Nov. 2002.

[11] E. J. O'Brien, J. A. Lerman, R. L. Chang, D. R. Hyduke, and B. Ø. Palsson, “Genome-scale models of metabolism and gene expression extend and refine growth phenotype prediction,” *Molecular Systems Biology*, vol. 9, no. 1, Jan. 2013. [Online]. Available: https://doi.org/10.1038/msb.2013.52

[12] H. W. Aung, S. A. Henry, and L. P. Walker, “Revising the representation of fatty acid, glycerolipid, and glycerophospholipid metabolism in the consensus model of yeast metabolism,” *Industrial Biotechnology*, vol. 9, no. 4, pp. 215–228, Aug. 2013. [Online]. Available: https://doi.org/10.1089/ind.2013.0013

[13] E. J. Kerkhoven, K. R. Pomraning, S. E. Baker, and J. Nielsen, “Regulation of amino-acid metabolism controls flux to lipid accumulation in yarrowia lipolytica,” *NPJ systems biology and applications*, vol. 2, no. 1, pp. 1–7, 2016.

[14] S. Marcišauskas, B. Ji, and J. Nielsen, “Reconstruction and analysis of a kluyveromyces marxianus genome-scale metabolic model,” *BMC bioinformatics*, vol. 20, pp. 1–9, 2019.

[15] J. M. Monk, C. J. Lloyd, E. Brunk, N. Mih, A. Sastry, Z. King, R. Takeuchi, W. Nomura, Z. Zhang, H. Mori *et al.*, “i ml1515, a knowledgebase that computes escherichia coli traits,” *Nature biotechnology*, vol. 35, no. 10, pp. 904–908, 2017.

[16] J. L. Robinson, P. Kocabaş, H. Wang, P.-E. Cholley, D. Cook, A. Nilsson, M. Anton, R. Ferreira, I. Domenzain, V. Billa *et al.*, “An atlas of human metabolism,” *Science signaling*, vol. 13, no. 624, p. eaaz1482, 2020.

[17] 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, Aug. 2017. [Online]. Available: https://doi.org/10.15252/msb.20167411

[18] I. Domenzain, B. Sánchez, M. Anton, E. J. Kerkhoven, A. Millán-Oropeza, C. Henry, V. Siewers, J. P. Morrissey, N. Sonnenschein, and J. Nielsen, “Reconstruction of a catalogue of genome-scale metabolic models with enzymatic constraints using GECKO 2.0,” *Nature Communications*, vol. 13, no. 1, Jun. 2022. [Online]. Available: https://doi.org/10.1038/s41467-022-31421-1

[19] E. H. Davidson and I. S. Peter, “Genomic control process,” in *Gene Regulatory Networks*. Oxford Academic Press, 2015, ch. 2.

[20] W. Abou-Jaoudé, P. Traynard, P. T. Monteiro, J. Saez-Rodriguez, T. Helikar, D. Thieffry, and C. Chaouiya, "Logical modeling and dynamical analysis of cellular networks," *Frontiers in Genetics*, vol. 7, May 2016.

[21] H. de Jong, "Modeling and simulation of genetic regulatory systems: A literature review," *Journal of Computational Biology*, vol. 9, no. 1, pp. 67–103, Jan. 2002.

[22] W. Abou-Jaoudé, P. T. Monteiro, A. Naldi, M. Grandclaudon, V. Soumelis, C. Chaouiya, and D. Thieffry, "Model checking to assess t-helper cell plasticity," *Frontiers in Bioengineering and Biotechnology*, vol. 2, Jan. 2015.

[23] E. Gonçalves, J. Bucher, A. Ryll, J. Niklas, K. Mauch, S. Klamt, M. Rocha, and J. Saez-Rodriguez, "Bridging the layers: towards integration of signal transduction, regulation and metabolism into mathematical models," *Molecular BioSystems*, vol. 9, no. 7, p. 1576, 2013.

[24] 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, no. 1, p. 101, Jan. 2007.

[25] 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.

[26] M. W. Covert, N. Xiao, T. J. Chen, and J. R. Karr, "Integrating metabolic, transcriptional regulatory and signal transduction models in escherichia coli," *Bioinformatics*, vol. 24, no. 18, pp. 2044–2050, Jul. 2008.

[27] 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*, vol. 107, no. 41, pp. 17 845–17 850, Sep. 2010. [Online]. Available: https://doi.org/10.1073/pnas.1005139107

[28] M. H. Daniel Machado, "Systematic evaluation of methods for integration of transcriptomic data into constraint-based models of metabolism," *PLoS Computational Biology*, vol. 10, no. 10, p. e1003989, Oct. 2014.

[29] G. Batt, B. Besson, P.-E. Ciron, H. de Jong, E. Dumas, J. Geiselmann, R. Monte, P. T. Monteiro, M. Page, F. Rechenmann, and D. Ropers, "Genetic network analyzer: A tool for the qualitative modeling and simulation of bacterial regulatory networks," in *Bacterial Molecular Networks*. Springer New York, Oct. 2011, pp. 439–462. [Online]. Available: https://doi.org/10.1007/978-1-61779-361-5_22

[30] T. Helikar, B. Kowal, S. McClenathan, M. Bruckner, T. Rowley, A. Madrahimov, B. Wicks, M. Shrestha, K. Limbu, and J. A. Rogers, "The cell collective: Toward an open and collaborative approach to systems biology," *BMC Systems Biology*, vol. 6, no. 1, p. 96, 2012. [Online]. Available: https://doi.org/10.1186/1752-0509-6-96

[31] A. G. Gonzalez, A. Naldi, L. Sánchez, D. Thieffry, and C. Chaouiya, "GINsim: A software suite for the qualitative modelling, simulation and analysis of regulatory networks," *Biosystems*, vol. 84, no. 2, pp. 91–100, May 2006.

[32] A. Naldi, D. Berenguier, A. Fauré, F. Lopez, D. Thieffry, and C. Chaouiya, "Logical modelling of regulatory networks with GINsim 2.3," *Biosystems*, vol. 97, no. 2, pp. 134–139, Aug. 2009.

[33] A. Naldi, C. Hernandez, W. Abou-Jaoudé, P. T. Monteiro, C. Chaouiya, and D. Thieffry, "Logical modeling and analysis of cellular regulatory networks with GINsim 3.0," *Frontiers in Physiology*, vol. 9, Jun. 2018. [Online]. Available: https://doi.org/10.3389/fphys.2018.00646

[34] L. Heirendt, S. Arreckx, T. Pfau, S. N. Mendoza, A. Richelle, A. Heinken, H. S. Haraldsdóttir, J. Wachowiak, S. M. Keating, V. Vlasov, S. Magnusdóttir, C. Y. Ng, G. Preciat, A. Žagare, S. H. J. Chan, M. K. Aurich, C. M. Clancy, J. Modamio, J. T. Sauls, A. Noronha, A. Bordbar, B. Cousins, D. C. E. Assal, L. V. Valcarcel, I. A. Apaolaza, S. Ghaderi, M. Ahookhosh, M. B. Guebila, A. Kostromins, N. Sompairac, H. M. Le, D. Ma, Y. Sun, L. Wang, J. T. Yurkovich, M. A. P. Oliveira, P. T. Vuong, L. P. E. Assal, I. Kuperstein, A. Zinovyev, H. S. Hinton, W. A. Bryant, F. J. A. Artacho, F. J. Planes, E. Stalidzans, A. Maass, S. Vempala, M. Hucka, M. A. Saunders, C. D. Maranas, N. E. Lewis, T. Sauter, B. O. Palsson, I. Thiele, and R. M. T. Fleming, "Creation and analysis of biochemical constraint-based models using the COBRA toolbox v.3.0," *Nature Protocols*, vol. 14, no. 3, pp. 639–702, Feb. 2019.

[35] 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, Aug. 2013.

[36] I. Rocha, P. Maia, P. Evangelista, P. Vilaça, S. Soares, J. P. Pinto, J. Nielsen, K. R. Patil, E. C. Ferreira, and M. Rocha, "OptFlux: an open-source software platform for in silico metabolic engineering," *BMC Systems Biology*, vol. 4, no. 1, Apr. 2010.

[37] V. Pereira, F. Cruz, and M. Rocha, "MEWpy: a computational strain optimization workbench in python," *Bioinformatics*, vol. 37, no. 16, pp. 2494–2496, Jan. 2021. [Online]. Available: https://doi.org/10.1093/bioinformatics/btab013

[38] 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. [Online]. Available: https://doi.org/10.1128/ecosalplus.10.2.1

[39] V. H. Tierrafría, C. Rioualen, H. Salgado, P. Lara, S. Gama-Castro, P. Lally, L. Gómez-Romero, P. Peña-Loredo, A. G. López-Almazo, G. Alarcón-Carranza, F. Betancourt-Figueroa, S. Alquicira-Hernásndez, J. E. Polanco-Morelos, J. García-Sotelo, E. Gaytan-Nuñez, C.-F. Méndez-Cruz, L. J. Muñiz, C. Bonavides-Martínez, G. Moreno-Hagelsieb, J. E. Galagan, J. T. Wade, and J. Collado-Vides, "RegulonDB 11.0: Comprehensive high-throughput datasets on transcriptional regulation in escherichia coli k-12," *Microbial Genomics*, vol. 8, no. 5, May 2022. [Online]. Available: https://doi.org/10.1099/mgen.0.000833

[40] M. A. Iván Domenzain and B. Sánchez, "Sysbiochalmers/ecmodels," https://github.com/SysBioChalmers/ecModels, 2022.