Development of Python library to integrate metabolic and regulatory networks

SEBASTIÃO SANTOS, Instituto Superior Técnico, Portugal

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 work 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.

Additional Key Words and Phrases: Metabolic modelling, Regulatory networks, Optimization, Integrated tool

1 INTRODUCTION

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.

2 BACKGROUND

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 [Palsson 2015].

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.

2.1.1 Genome-scale models. Genome-scale models (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 [Kuepfer 2014]. 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 Simulating Metabolism. 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. 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 FBA (pFBA), and Flux Variability Analysis (FVA), are widely employed for precisely this purpose. These methodologies leverage stoichiometry and consider constraints imposed by Gene-Protein-Reaction (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 [Orth et al. 2010b].

2.1.3 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. 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) [Domenzain et al. 2022; Sánchez et al. 2017].

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 [Davidson and Peter 2015].

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 [Abou-Jaoudé et al. 2015, 2016]. 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.

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 [Gonçalves et al. 2013; Shlomi et al. 2007].

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.

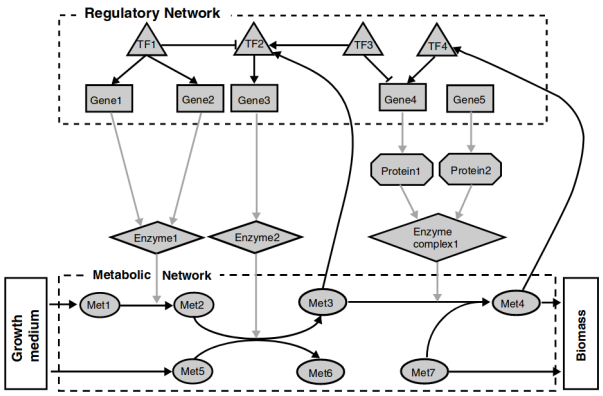

Fig. 1. Schematic representation of an integrated metabolic and regulatory network. The regulatory network component consists of a set of interactions between TFs and other TFs and genes. The metabolic network component consists of a set of biochemical reactions between metabolites, with metabolites 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) [Shlomi et al. 2007].

Computational methods like regulatory FBA (rFBA), Steady-state Regulatory FBA (SR-FBA) and Integrated FBA (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 [Covert et al. 2001, 2008; Shlomi et al. 2007].

3 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 [Gonzalez et al. 2006; Naldi et al. 2009], GNA (focused on Piecewise-linear differential equations rather than Boolean models) [Batt et al. 2011], and The Cell Collective [Helikar et al. 2012]. These tools support SBML-qual files, a file format specific for regulatory models.

4 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. Some examples of these tools include COBRApy [Ebrahim et al. 2013], ReFramed and OptFlux [Rocha et al. 2010].

5 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 [Pereira et al. 2021], 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.

6 IMPLEMENTED SOLUTION

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 critical 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.

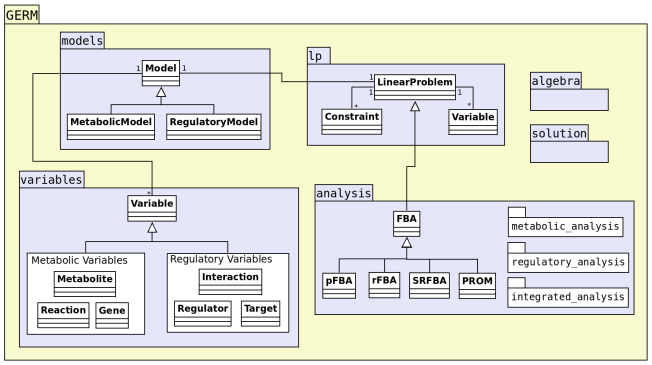

Fig. 2. Simplified UML Diagram of *mewpy.germ* module

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.

6.1 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 (Figure 2). The module *mewpy.germ* implements the integration of metabolic and regulatory models, represented by GERM models.

6.1.1 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.

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 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

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 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

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. 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. 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

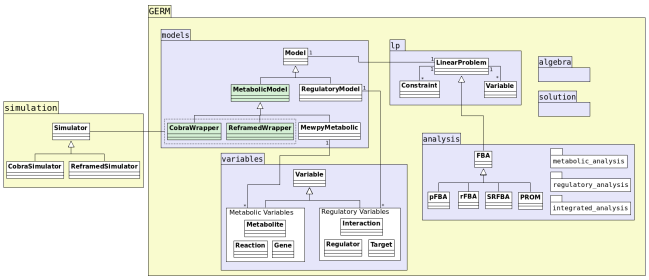

Fig. 3. Simplified UML Diagram of the implemented solution on *mewpy.germ* module. Major differences are highlighted in green.

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*.

6.1.2 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.

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.

6.2 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 3.

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

6.2.2 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.

6.2.3 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,

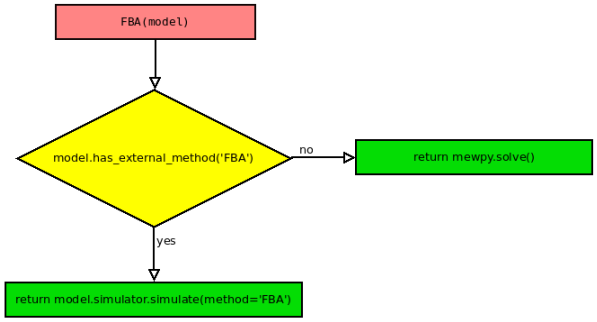

Fig. 4. FBA simulation behaviour. The same behaviour applies to other simulation or analysis methods

ensuring a robust and versatile approach to metabolic modeling across a wide range of scenarios 4.

7 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 [Orth et al. 2010a] and comprehensive whole-genome regulatory models [Tierrafría et al. 2022].

7.1 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. 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. 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.

7.2 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.

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. 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 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. 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 6.

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.

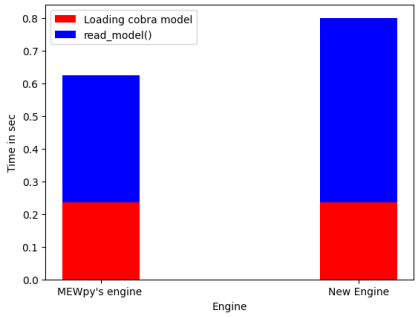

Fig. 5. Time Comparison of reading a GERM model from a COBRA metabolic model

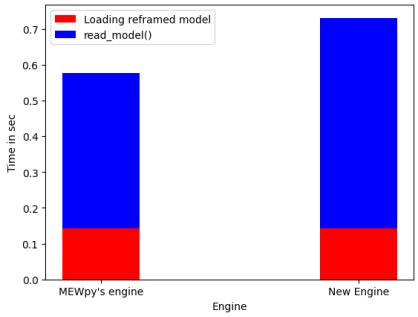

Fig. 6. Computational Efficiency in reading a GERM model from a ReFramed metabolic model

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.

As illustrated in Figure 7, 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

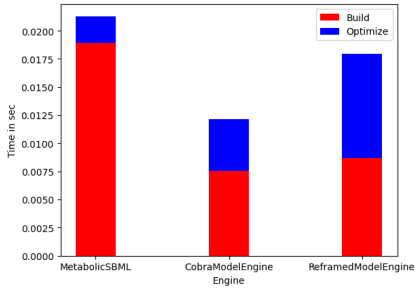

Fig. 7. Computational Efficiency in FBA with a GERM model read using different engines. The leftmost bar symbolizes a GERM model with its metabolic component derived from the MetabolicSBML engine, the middle bar represents a GERM model read using a CobraModel engine, and the rightmost bar showcases a GERM model generated by ReframedModel engine.

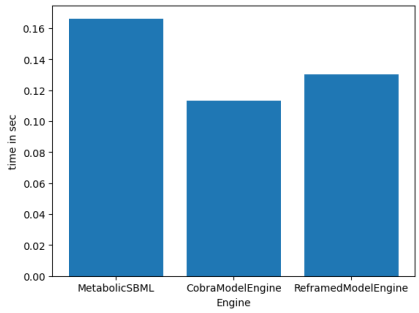

Fig. 8. Computational Efficiency in FVA with a GERM model read using different engines. The leftmost bar symbolizes a GERM model with its metabolic component derived from the MetabolicSBML engine, the middle bar represents a GERM model read using a CobraModel engine, and the rightmost bar showcases a GERM model generated by ReframedModel engine.

new wrapper classes to represent metabolic models within a GERM model, as illustrated in Figure 8.

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. As depicted in Figure 9 (particularly in the *build* step) and Figure 10, 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.

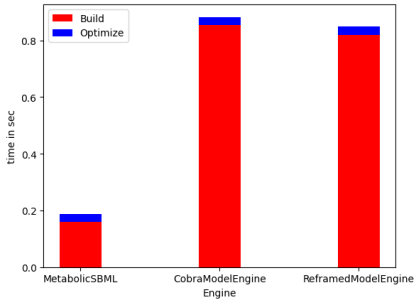

Fig. 9. Computational Efficiency in SR-FBA with a GERM model read using different engines. The leftmost bar symbolizes a GERM model with its metabolic component derived from the MetabolicSBML engine, the middle bar represents a GERM model read using a CobraModel engine, and the rightmost bar showcases a GERM model generated by ReframedModel engine.

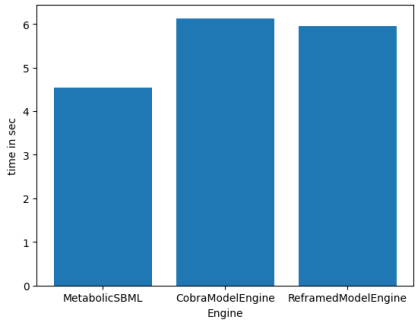

Fig. 10. Computational Efficiency in iFVA with a GERM model read using different engines. The leftmost bar symbolizes a GERM model with its metabolic component derived from the MetabolicSBML engine, the middle bar represents a GERM model read using a CobraModel engine, and the rightmost bar showcases a GERM model generated by ReframedModel engine.

8 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 [Monk et al. 2017]. In the use case, a GEM model (*iML1515*) and a GECKO model (*eciML1515*) are compared [Iván Domenzain and Sánchez 2022]. 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_75p37
           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_75p37
           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 FVA (iFVA) was conducted on each model, as depicted in Figure 11. Notably, the *ec_model* exhibits a more restricted range of possible flux for its reactions compared to the *gem_model*. 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_75p37
           Objective value 0.8769972144269684
           Status  optimal
```

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

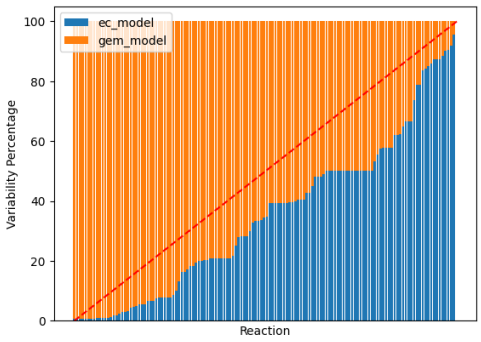

Fig. 11. Comparative integrated Flux Variability Analysis of ec_model and gem_model. This stacked bar chart illustrates the comparative iFVA of ec_model and gem_model, with respect to a set non-null reactions (reactions with no flux). Each bar in the chart corresponds to an individual reaction and is 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.

```
Out[4]:     Method   FBA
            Model    Model e_coli_core - ecModel_batch of
            Objective       BIOMASS_Ec_iML1515_core_75p37
            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 [5]:     SRFBA(gem_model).build().optimize()

Out[5]:     Method   SRFBA
            Model    Model e_coli_core - model
            Objective       BIOMASS_Ec_iML1515_core_75p37
            Objective value 0.8769972144281424
            Status   optimal

In [6]:     SRFBA(ec_model).build().optimize()

Out[6]:     Method   SRFBA
            Model    Model e_coli_core - ecModel_batch of
            Objective       BIOMASS_Ec_iML1515_core_75p37
            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.

9 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 [Pereira et al. 2021].

10 CONCLUSION AND FUTURE WORK

This work 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.

REFERENCES

Wassim Abou-Jaoudé, Pedro T. Monteiro, Aurélien Naldi, Maximilien Grandclaudon, Vassili Soumelis, Claudine Chaouiya, and Denis Thieffry. 2015. Model Checking to Assess T-Helper Cell Plasticity. *Frontiers in Bioengineering and Biotechnology* 2 (Jan. 2015). https://doi.org/10.3389/fbioe.2014.00086

Wassim Abou-Jaoudé, Pauline Traynard, Pedro T. Monteiro, Julio Saez-Rodriguez, Tomáš Helikar, Denis Thieffry, and Claudine Chaouiya. 2016. Logical Modeling and Dynamical Analysis of Cellular Networks. *Frontiers in Genetics* 7 (May 2016). https://doi.org/10.3389/fgene.2016.00094

Grégory Batt, Bruno Besson, Pierre-Emmanuel Ciron, Hidde de Jong, Estelle Dumas, Johannes Geiselmann, Regis Monte, Pedro T. Monteiro, Michel Page, François Rechenmann, and Delphine Ropers. 2011. Genetic Network Analyzer: A Tool for the Qualitative Modeling and Simulation of Bacterial Regulatory Networks. In *Bacterial Molecular Networks*. Springer New York, 439–462. https://doi.org/10.1007/978-1-61779-361-5_22

Markus W. Covert, Christophe H. Schilling, and Bernhard Palsson. 2001. Regulation of Gene Expression in Flux Balance Models of Metabolism. *Journal of Theoretical Biology* 213, 1 (Nov. 2001), 73–88. https://doi.org/10.1006/jtbi.2001.2405

Markus W. Covert, Nan Xiao, Tiffany J. Chen, and Jonathan R. Karr. 2008. Integrating metabolic, transcriptional regulatory and signal transduction models in Escherichia coli. *Bioinformatics* 24, 18 (July 2008), 2044–2050. https://doi.org/10.1093/bioinformatics/btn352

Eric H. Davidson and Isabelle S. Peter. 2015. Genomic Control Process. In *Gene Regulatory Networks*. Oxford Academic Press, Chapter 2.

Iván Domenzain, Benjamín Sánchez, Mihail Anton, Eduard J. Kerkhoven, Aarón Millán-Oropeza, Céline Henry, Verena Siewers, John P. Morrissey, Nikolaus Sonnenschein, and Jens Nielsen. 2022. Reconstruction of a catalogue of genome-scale metabolic models with enzymatic constraints using GECKO 2.0. *Nature Communications* 13, 1 (June 2022). https://doi.org/10.1038/s41467-022-31421-1

Ali Ebrahim, Joshua A Lerman, Bernhard O Palsson, and Daniel R Hyduke. 2013. COBRApy: COnstraints-Based Reconstruction and Analysis for Python. *BMC Systems Biology* 7, 1 (Aug. 2013). https://doi.org/10.1186/1752-0509-7-74

A. Gonzalez Gonzalez, A. Naldi, L. Sánchez, D. Thieffry, and C. Chaouiya. 2006. GINsim: A software suite for the qualitative modelling, simulation and analysis of regulatory networks. *Biosystems* 84, 2 (May 2006), 91–100. https://doi.org/10.1016/j.biosystems.2005.10.003

Emanuel Gonçalves, Joachim Bucher, Anke Ryll, Jens Niklas, Klaus Mauch, Steffen Klamt, Miguel Rocha, and Julio Saez-Rodriguez. 2013. Bridging the layers: towards integration of signal transduction, regulation and metabolism into mathematical models. *Molecular BioSystems* 9, 7 (2013), 1576. https://doi.org/10.1039/c3mb25489e

Tomás Helikar, Bryan Kowal, Sean McClenathan, Mitchell Bruckner, Thaine Rowley, Alex Madrahimov, Ben Wicks, Manish Shrestha, Kahani Limbu, and Jim A Rogers. 2012. The Cell Collective: Toward an open and collaborative approach to systems biology. *BMC Systems Biology* 6, 1 (2012), 96. https://doi.org/10.1186/1752-0509-6-96

Mihail Anton Iván Domenzain and Benjamín Sánchez. 2022. SysBioChalmers/ecModels. https://github.com/SysBioChalmers/ecModels.

Lars Kuepfer. 2014. Stoichiometric Modelling of Microbial Metabolism. In *Methods in Molecular Biology*. Springer New York, 3–18.

Jonathan M Monk, Colton J Lloyd, Elizabeth Brunk, Nathan Mih, Anand Sastry, Zachary King, Rikiya Takeuchi, Wataru Nomura, Zhen Zhang, Hirotada Mori, et al. 2017. i ML1515, a knowledgebase that computes Escherichia coli traits. *Nature biotechnology* 35, 10 (2017), 904–908.

A. Naldi, D. Berenguier, A. Fauré, F. Lopez, D. Thieffry, and C. Chaouiya. 2009. Logical modelling of regulatory networks with GINsim 2.3. *Biosystems* 97, 2 (Aug. 2009), 134–139. https://doi.org/10.1016/j.biosystems.2009.04.008

Jeffrey D. Orth, R. M. T. Fleming, and Bernhard Ø. Palsson. 2010a. Reconstruction and Use of Microbial Metabolic Networks: the Core Escherichia coli Metabolic Model as an Educational Guide. *EcoSal Plus* 4, 1 (Jan. 2010). https://doi.org/10.1128/ecosalplus.10.2.1

Jeffrey D Orth, Ines Thiele, and Bernhard O Palsson. 2010b. What is flux balance analysis? *Nature Biotechnology* 28, 3 (March 2010), 245–248. https://doi.org/10.1038/nbt.1614

Bernhard O. Palsson. 2015. Systems biology: constraint-based reconstruction and analysis.

Vítor Pereira, Fernando Cruz, and Miguel Rocha. 2021. MEWpy: a computational strain optimization workbench in Python. *Bioinformatics* 37, 16 (Jan. 2021), 2494–2496. https://doi.org/10.1093/bioinformatics/btab013

Isabel Rocha, Paulo Maia, Pedro Evangelista, Paulo Vilaça, Simão Soares, José P Pinto, Jens Nielsen, Kiran R Patil, Eugénio C Ferreira, and Miguel Rocha. 2010. OptFlux: an open-source software platform for in silico metabolic engineering. *BMC Systems Biology* 4, 1 (April 2010). https://doi.org/10.1186/1752-0509-4-45

Tomer Shlomi, Yariv Eisenberg, Roded Sharan, and Eytan Ruppin. 2007. A genome-scale computational study of the interplay between transcriptional regulation and metabolism. *Molecular Systems Biology* 3, 1 (Jan. 2007), 101. https://doi.org/10.1038/msb4100141

Benjamín J Sánchez, Cheng Zhang, Avlant Nilsson, Petri-Jaan Lahtvee, Eduard J Kerkhoven, and Jens Nielsen. 2017. Improving the phenotype predictions of a yeast genome-scale metabolic model by incorporating enzymatic constraints. *Molecular Systems Biology* 13, 8 (Aug. 2017). https://doi.org/10.15252/msb.20167411

Víctor H Tierrafría, Claire Rioualen, Heladia Salgado, Paloma Lara, Socorro Gama-Castro, Patrick Lally, Laura Gómez-Romero, Pablo Peña-Loredo, Andrés G. López-Almazo, Gabriel Alarcón-Carranza, Felipe Betancourt-Figueroa, Shirley Alquicira-Hernásndez, J. Enrique Polanco-Morelos, Jair García-Sotelo, Estefani Gaytan-Nuñez, Carlos-Francisco Méndez-Cruz, Luis J. Muñiz, César Bonavides-Martínez, Gabriel Moreno-Hagelsieb, James E. Galagan, Joseph T. Wade, and Julio Collado-Vides. 2022. RegulonDB 11.0: Comprehensive high-throughput datasets on transcriptional regulation in Escherichia coli K-12. *Microbial Genomics* 8, 5 (May 2022). https://doi.org/10.1099/mgen.0.000833

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