Genome-scale models are made of this

Julius Battjes[1], Jurgen R. Haanstra[1], Gioele Lazzari[2], Francesco Moro[1], Christoff Odendaal[1], Maaike Remeijer[1], Steven Wijnen[1], and Pranas Grigaitis[1,✉]

[1]Systems Biology Lab, A-LIFE & AIMMS, Vrije Universiteit Amsterdam, De Boelelaan 1085, 1081HV Amsterdam, the Netherlands
[2]Department of Biotechnology, University of Verona, Verona, Italy

Correspondence p.grigaitis@vu.nl
Version 13.02.2024

Contents

Introduction and workflow

Genome-scale metabolic models (GEMs) are a widely-used and - by now - constitute the most comprehensive modeling framework for large-scale metabolic networks. The main idea behind this framework is that most of the information needed to understand metabolic networks is stored in the *stoichiometry* (=ratios in which molecules are consumed or formed) of (bio)chemical reactions. By collecting the information for as many reactions as possible, we can obtain a comprehensive snapshot of the *metabolic potential* of an organism, meaning all the interconversions it *could* do (famously illustrated by the Biochemical Pathways poster). Then we can apply mathematical optimization, primarily linear programming, to obtain quantitative predictions of the fluxes through the metabolic network.

Here, at the Systems Biology Lab, we have a good track-record of successfully creating, curating, and using genome-scale models to test biological hypotheses. Technically, this history started with the *Lactiplantibacillus plantarum* - formerly *Lactobacillus plantarum* - model by Bas Teusink [1]). The models that we have created and/or used span a wide range of organisms, from microbial cells to - increasingly - higher eukaryotes. We have recently reviewed the potential applications of GEMs in the context of food microbiology [2] (Figure 1). The current document covers some practical aspects regarding the reconstruction and analysis of GEMs.

With the increasing number of available genome sequences, the motivation for using GEMs for hypothesis testing is at an all-time high. Yet, practice shows that extensive - in many cases manual - curation is essential for crafting high-quality reconstructions. There are automatic reconstruction pipelines that handle all the steps that are traditionally done either manually or semi-automatically, yet the resulting models should be treated as exploratory tool and not complete/high-confidence reconstructions.

In this document, we have tried to summarize the current genome-scale modeling practices in our lab (Figure 1). This involves several steps: model drafting, model curation, and model analysis. We focus on the first two steps as model analysis depends strongly on the biological question, and is often done using custom scripts. This text aims to aid users with little knowledge of the *actual* procedures behind the concepts to make their way up the steep, but rewarding, hill of learning how to work with GEMs. Happy modeling!

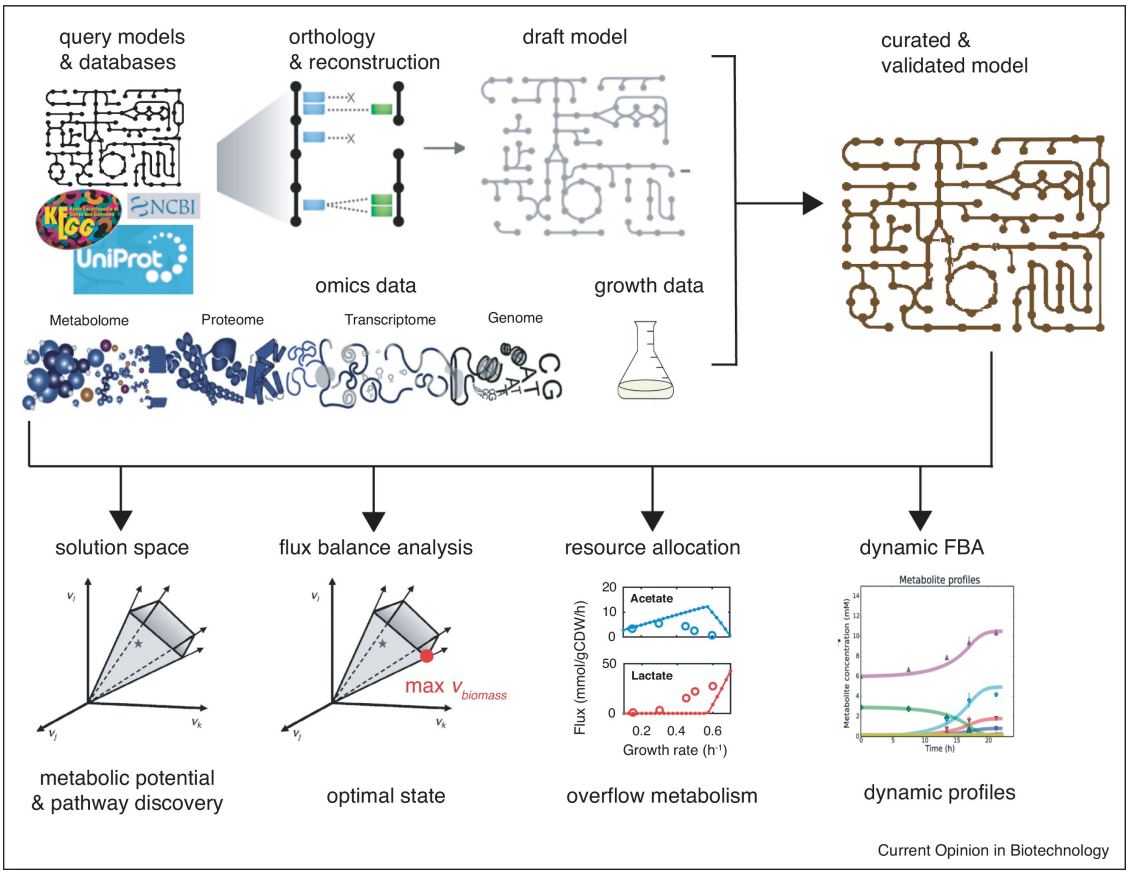

Figure 1: The overview of the workflow for reconstructing and analyzing GEMs. Figure taken from [2].

PRIMER TO *Flux Balance Analysis*[1] We can imagine a metabolic network as a highly-connected pipe system: matter enters the inlet pipe, flows from one pipe to another, connects another stream of matter, and flows away through the exhaust pipe. Thus the *flux* through the metabolic reaction ("pipe") is amount of matter per unit of time ($mol\ s^{-1}$ in SI units). We can transform GEMs into optimization (linear programming) problems, and the most popular method to analyze these models is Flux Balance Analysis (FBA) [4].

FBA assumes that the metabolic network is in a steady-state (=in balanced growth) ($N \times v = 0$, with the N being the stoichiometric matrix, and v - the flux vector, or the list of flux values through each reaction). We apply *constraints* to the metabolic network, so-called flux bounds, and then perform optimization with respect to the *objective function*: a flux that we want to optimize (minimize/maximize). The result of optimization is a so-called *optimal flux distribution*, the numerical values of the v which optimize the flux through the objective function and satisfy the constraints at steady-state.

In most cases, the objective function represents the formation of new biomass, and sometimes is called *biomass objective function* (BOF) instead. The BOF describes the quantity of each biomass component (proteins, RNA, DNA ...) that is required to produce 1 gram of dry weight (gDW) of cells. When FBA objective is to maximize the biomass production, FBA will output a solution with the highest biomass yield per limiting nutrient (see a graphical representation below, adapted from [5]). The growth rate corresponding to the highest yield solution then is the product of the yield and the nutrient influx into the cell.

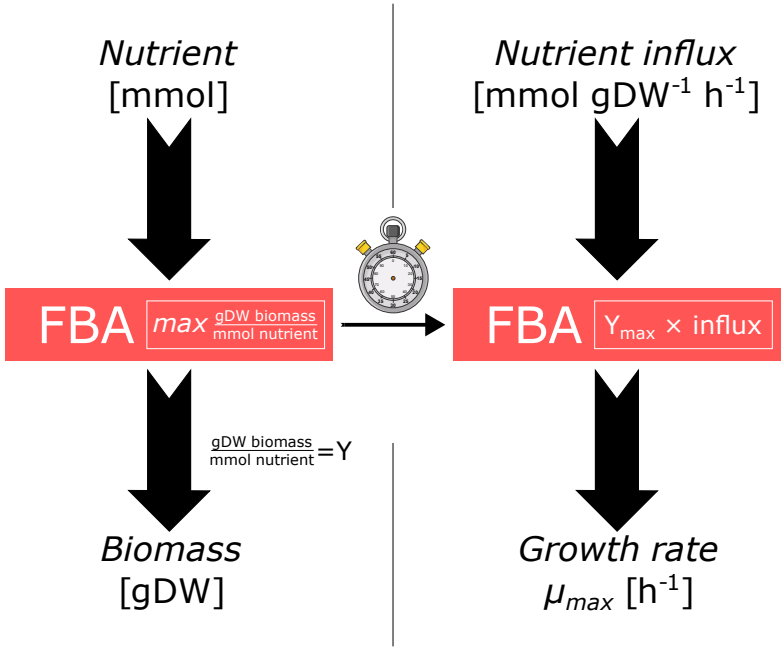

Figure 2: Schematic overview of flux balance analysis. Figure taken from [3].

A unit flux through the BOF would mean that 1 gram of new biomass is produced per time unit and the specific growth rate is 1. In genome-scale modeling, we usually handle the flux of matter in $mmol\ h^{-1}$, and scale fluxes proportional to the gDW of cell biomass (so, the typical dimension is $mmol\ \ gDW^{-1}\ h^{-1}$). As we input the amounts of cell biomass precursors in $mmol\ gDW^{-1}$, the dimension of the BOF flux is h^{-1} therefore. In GEMs, a unit flux through the BOF is equal to the specific growth rate $\mu = 1\ h^{-1}$.

1 SOFTWARE FOR HANDLING GEMS

The first steps, the data collection, usually does not use any specific software to execute. Before we continue with the actual modeling (reconstruction of a draft metabolic model), we first will cover the current options for the basic tools of handling GEMs. Over the years, a multitude of software packages has been developed for handling GEMs, mostly for use in Python- or MATLAB environments. GEMs are usually subjected to optimization, namely, linear programming (LP), thus, the choice of linear solver is a recurring issue - we will briefly discuss the most popular solvers.

[1] Material taken from [3]

Keep in mind that there are no predefined routines in any of these packages for many advanced applications, and you will have to write small original scripts yourself. However, having a good command of any of these tools will help you automatize many of such steps. Here we will not provide a comprehensive overview of *all* the packages, only the most popular ones.

1.1 PySCeS CBMPy

CBMPy ('PySCeS Constraint Based Modelling') [6] is a GEM manipulation package written in Python by our colleague Brett Olivier. Currently, CBMPy supports two LP solvers, CPLEX and GLPK (note that GLPK implementation is incomplete as for April 2023).

An advantage of the CBMPy philosophy (=how the models are interacted with) is that every variable in the model is an object - you can call individual compartments, genes, reactions, metabolites, and reaction reagents. For some, this might seem counter-intuitive, although it starts making sense as you become more proficient. Brett has extensively documented most of the package functions here and can help with more advanced issues himself. For most of the model handling, CBMPy is the lab's package of choice.

1.2 COBRA Toolbox

MATLAB-based (' COnstraint-Based Reconstruction and Analysis') COBRA Toolbox [7] is developed primarily in Bernhard Ø. Palsson's group at the University of California in San Diego. The COBRA Toolbox is probably the oldest multi-purpose GEM modeling package in active development. COBRA Toolbox's website has a lot of material for new users - tutorials, user manuals, case examples - all of which makes it easier to adopt. It works with all popular LP solvers: CPLEX, GLPK and Gurobi, as well as the internal MATLAB LP solver.

For a long time, the COBRA Toolbox used to be pretty much the only choice available (with a notable exception of CellNetAnalyzer). Many of its the functions are by now implemented in the RAVEN Toolbox as well [8].

Model handling in the COBRA Toolbox is primarily via function calling, object-based routines are rather rare - making its philosophy different from CBMPy and COBRApy (see below). Nowadays, a notable issue of using the COBRA Toolbox is its incomplete adherence of the latest Systems Biology Markup Language (SBML) standards (currently, the latest standard is SBML Flux Balance Constraints [FBC] v3.2 [9]).

1.3 COBRApy

With the adoption of Python in the life- and natural sciences (e.g. libraries such as pandas, numpy, and scipy), many scientific MATLAB packages were ported to Python. COBRApy [10] is the COBRA Toolbox counterpart in Python. COBRApy works with CPLEX, GLPK and Gurobi, and follows an object-based modeling philosophy, similar to CBMPy. A valuable functionality of COBRApy is the interoperability of different model file types: alongside SBML, COBRApy also supports YAML, JSON, and MATLAB-container models.

COBRApy is probably the most popular GEM toolbox in the scientific community right now, so many other (more niche) packages are compatible with COBRApy, such as StrainDesign [11] and Escher [12].

1.4 RAVEN Toolbox

The RAVEN Toolbox [8] is a multi-purpose toolbox for the generation and curation of genome-scale models, and is implemented in MATLAB. Model handling in the RAVEN Toolbox is similar to the COBRA Toolbox: most of the functionality has to be called as separate functions, rather than following an object-oriented approach (cf. CBMPy). Moreover, many of the recent methods for -omics data integration are implemented in RAVEN.

1.5 Handling GEMs in Julia

Julia is a programming language created to combine the ease of use and high-level abstractions of languages (like Python and MATLAB) with the performance of low-level languages (like C and Fortran). It can be a valuable option to consider when dealing with very large GEMs that require big clusters and high performance computing. There are Julia packages which allow handling and analysis of large constrained metabolic models, notably COBRA.jl (implementation of the COBRA toolbox in Julia) and COBREXA.jl.

1.6 Mostly used LP/MILP solvers

CPLEX, GLPK and Gurobi are the most popular solvers, but there are more solvers. Recently, they have been compared quite elaborately [13].

CPLEX IBM CPLEX is a commercial solver for linear programming- (LP) and mixed-integer linear programming- (MILP) problems which has long been considered the industry standard. For use with Python packages, there's a Python API. However, MATLAB support was discontinued from version 12 onwards. Non-academic users have to buy a license in order to use the solver, but a fully-equipped version is available for academic users, registered to IBM Academic Initiative (use your university e-mail and credentials).

Warning! There is also a trial free version of CPLEX solver (and academic *vs.* trial is easy to mix up), however, it is restricted to solving LP programs below 1000 variables. Be careful about which version you download.

Warning for Windows users! After installation on your computer, you need to connect CPLEX to Python (if you want to use it in Python). For this, you get a command on the last screen before you finish your installation. On Windows, you need to run this command in the command line *as an administrator* (Windows menu → e.g. Anaconda Prompt → on the right, click 'Run as administrator'). For more detailed installation instructions, see the GEM handling tutorial in Chapter 8.

Gurobi Gurobi is another leading commercial LP- and MILP-solver. While many Python-based GEM handling environments preferentially support CPLEX, Gurobi is easy to setup for MATLAB-based tools and, in general, for other MATLAB packages that perform mathematical optimization. A Python API is also available and is utilized by COBRApy, but Gurobi is currently not supported by CBMPy. Academic licenses for Gurobi are available for free by registering with your university e-mail. Every license has to be activated, see the instructions on the website for your operating system.

GLPK GNU Linear Programming Kit is an open-source - as all GNU things are - library for solving LPs and MILPs. GLPK has APIs/bindings to both Python and MATLAB environments, and can, in principle, be used with all the GEM handling packages listed above. There are a couple of issues with using GLPK as the primary solver: first, commercial solvers apply fast proprietary algorithms to solve LPs, and therefore GLPK does not exhibit comparable solving efficiency - especially for MILPs. Next, for (programming) beginners, compiling and setting up GLPK might be a challenging task.

2 Collecting the data

2.1 Start from scratch: annotating an unknown genome

To construct a GEM from scratch, a sequenced genome of the organism(s) of interest is required.

Often, the genomic information is stored in a FASTA file, containing contigs. Each contig is composed of a header (the line starting with '>'), followed by a DNA sequence. If your organism is a bacterium which has been deeply sequenced, then the assembler should give you a FASTA file with just 1 longer contig + n shorter contigs, where n is the number of plasmids. This is what is usually called a "complete" or "closed" genome. However, you don't necessarily need a closed genome to create a GEM. For example, common Illumina short read sequencing coupled to an established assembler like SPAdes [14] could give you a bacterial assembly of around 200 contigs. Even those fragmented assemblies should be enough to start building your GEM.

Given your genome assembly, you first need to extract genes from it. In particular, you are interested in predicting enzyme-coding sequences (CDS). Prodigal [15] is an established gene prediction tool worth trying. Remember that in this step you are only extracting sequences, and not assigning a function to them. Gene prediction tools like Prodigal give you the coordinates of the genes relative to their own contig, usually in generic feature format (`gff`) or Genbank-like format (`gbk`). Most importantly, they also give you the proteome of your organism, i.e. a FASTA file containing all the CDS sequences of your organism translated to amino acids (usually it has the `.faa` extension). In the next chapters, you will find out how you can use this proteome to build up a GEM.

An alternative way to get a reliable proteome out of your genome is to use the online tool, eggNOG-mapper (eggNOG-mapper). This tool automatically assigns open reading frames (ORFs) and maps them to known CDS in databases. The output is a complete proteome in FASTA format accompanied by an informative

sheet on the mapped proteins. The FASTA file can be readily used in further GEM construction. The default settings of the mapper are usually sufficient but can be manipulated if necessary.

2.2 Obtaining annotated proteomes

If your target organism's genome has already been sequenced, there is a good chance that the annotated proteome (and perhaps reference proteome as well) will be published in at least one of the following databases: UniProt or NCBI Genome. That said, the "reference proteome" from UniProt in many cases requires manual curation of the entries and the reference proteome might be incomplete. It is therefore recommended to use the NCBI Genome assembly instead: NCBI uses internally developed pipelines to annotate newly submitted genomes *de novo* (see the manual for the NCBI Eukaryotic Genome Annotation Pipeline). On the Assembly page, select "Download", click on "Protein FASTA (`.faa`)" and make sure that RefSeq is the only data source checked (uncheck GenBank).

2.3 Selection of template model(s) and proteome(s)

A good first step is to locate your target organism in the NCBI Taxonomy database. This interactive tool allows you to see the "neighboring" organisms in the taxonomy tree. These might be good candidates as your second most preferred template, if a metabolic reconstruction is available. Many tools allow you to rank models, i.e. prioritize obtaining reactions from Model #1, rather than Model #2 during the reconstruction process.

For all things GEM, the primary database is BiGG. There, you can also find many previously published GEMs, as well as a reaction and metabolite database. It is highly recommended, where possible, to use the models deposited on BiGG as template models to ensure compatibility.

3 Reconstructing the draft model

In this text, we will focus on template-based GEM reconstruction, which can be performed using several tools [8, 16]. In the process, a draft model is generated from the genome of an organism by homology prediction against one or more existing metabolic models ("templates"). Preferably, these models are ranked in the order of closeness to the organism of interest. An alternative to the template-based approach is *de novo* or *ab initio* reconstruction. *Ab initio* methods are conceptually similar to template-based approaches, but they mainly resort on information mining from metabolic pathway databases, such as KeGG or MetaCyc.

Rather than going fully purist on either *ab initio* or template-based approaches, these two are usually treated as complementary to each other. In such a case, template-based reconstruction would be performed to generate a draft model with a reliable core of well-annotated reactions from existing models. This first draft can then be combined with the *de novo* reconstruction to include organism-specific reactions that are absent in used template models.

3.1 MetaDraft

MetaDraft is a tool for automated GEM reconstruction based on BiGG models, which makes use of the CBMPy modelling framework. The tool uses an annotated proteome as input (`.faa`/`.fasta`). As every BiGG model comes with GPRs, the input proteome is compared to one or several BiGG proteomes. This comparison is done through Inparanoid in combination with an offline BLAST version. Inparanoid is an algorithm that is specialized in orthology selection.

When selecting multiple BiGG models as templates for draft reconstruction, the priority of the template models can be set. If one gene of the input proteome has multiple homology hits in different models, with differing reaction labels, the GPR homology from the model with highest priority is picked. You could argue that the priority of the template models be set according to phylogenetic distance. The Metadraft GUI makes it easy to select the preferred mapped reaction if desired. The same goes for homologous gene hits within one template model.

Another handy functionality of Metadraft is the option to upload any template model that you prefer. The model has to be loadable by the CBMPy function `cbmpy.loadModel()`. Beware that if the template model is not labeled in BiGG format, meaning that if multiple draft models are used, similar reactions with different labeling are added to the draft reconstruction.

3.2 RAVEN Toolbox

The RAVEN Toolbox (see Section 1 for details on the software itself) method `getModelFromHomology()` is used for generating draft models. The routine determines homologous enzymes by running a bidirectional protein BLAST (`getBlast()`). Then, the draft model is generated using `getModelFromHomology()`. Most of the parameters that one can tweak are related to the homology cut-offs for the bidirectional BLAST output: the minimal identity of the sequences (default: 40%), the minimal length of amino acid chain (default: 200), and so on.

The required inputs are:

- The annotated proteome (`.faa`/`.fasta`) file of the target organism
- Template models and their proteomes

Tip! How to generate the proteome of the template model: obtain the list of genes present in that model, preferably in some useful identifier schema (UniProt ID works the best but also consider other formats, like NCBI Gene, Ensembl, *etc.*). Then use the ID Mapping feature of the UniProt website to match your identifiers. Finally, in the Results window, select **Download > FASTA**.

Warning! Make sure your gene identifiers in the model (without the `G_` prefix) and in the `.fasta` file match, otherwise the algorithm will not detect them.

3.3 CarveMe

The traditional bottom-up approaches described above can be either replaced or complemented by the top-down approach implemented in CarveMe [17]. Instead of starting from the organism's genome and finding the reactions corresponding to its metabolic genes, CarveMe relies on a (semi-)manually curated universal draft model, which includes all the reactions available in BiGG. This universal model is then converted into an organism-specific model by removing reactions and metabolites unlikely to be present in the given organism, in light of its genome. Finally, the model undergoes a round of automatic gap-filling.

This approach allows for an automated and parallelizable reconstruction of ready-to-use (i.e. capable to grow) GEMs. These features make this tool really popular for microbial community modeling starting from metagenomic data. However, this strategy comes with several drawbacks, which make it not perfectly suited for generating high quality drafts.

CarveMe is affected by several problems, some of which are consequence of the CarveMe code itself, others of which are consequence of how the BiGG database is structured (CarveMe is based on BiGG). What follows refers to CarveMe v1.5.2.

(I) The generated GPRs are sub-optimal. Highly similar genes are not taken into account during the GPR generation. Moreover, the original protein complex definitions are never checked. You can find more information at Issue #180 and Issue #182. Both these issues depend on the CarveMe code itself.

(II) False positive reactions are included. CarveMe produces "FBA-ready" models, i.e. models that grow out-of-the-box. To accomplish this, a gap-filling step is performed internally, without the option to skip it. As a consequence, false positive reactions are included, with no associated GPR.

(III) Metabolites and reactions are not consistent. This is a problem inherited from the BiGG database. In BiGG, the same metabolite can appear in different models, with different IDs, different formulas, and different charges. For example, the cytosolic metabolite indole is encoded in the majority of models with ID indole, but the model iSynCJ816 encodes it as ind. Since CarveMe uses all the bacterial models available in BiGG to build its universe, your model in output could contain both `ind_c` and `indole_c`. Be aware.

4 Make it grow: gapfilling of the draft model

Now that the initial draft model is constructed it still contains gaps in the network that have to be filled. Gaps need to be filled to either consume or produce metabolites and biomass intermediates. There are several reasons why the initial draft model reconstruction pipeline did not pick up on the missing reactions. It is good to understand these reasons as it will help to fill the different gaps within the network. The reasons might be, but are not limited to, the following: (I) no gene is known for the missing reaction; (II) it is a non-catalyzed reaction; (III) the gene is already associated to another reaction. Keep in mind that there is

no need to fill all the gaps in the model. The most important gaps to fill are the ones that we require to get good predictions of growth and metabolite production or consumption.

4.1 Manual gapfilling strategies

To perform manual gapfilling, a good strategy is to pick a metabolite that you want the model to produce and then track the metabolic route back to where the first gap is found. To do so, the pathway has to be looked up, or known. This is particularly relevant for secondary metabolites that are organism-specific; in such a case, one might have to implement completely new pathways leading to the desired product. As an example, we could consider taurine synthesis pathway (Figure 3), which differs for mammals (shown in the Figure 3) vs. other classes of organisms.

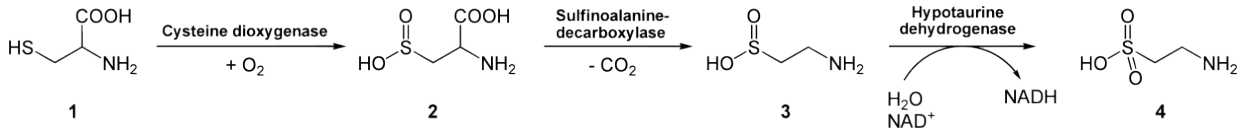

Figure 3: Mammalian taurine synthesis from cysteine. Picture taken from Wikimedia Commons.

Ideally, the biochemistry of these reactions (redox carriers, other cofactors) is known, otherwise one has to make an educated guess. For instance, for many biosynthetic processes, redox cofactors are $NADP^+$/NADPH, instead of NAD^+/NADH. For C_1 biochemistry, it's important to discriminate between the potential C_1 moiety donors etc.

A good source of data for this process is untargeted metabolomics: you can treat the annotated peaks as evidence that a metabolite can be transported and/or produced by the metabolic network. Based on the matrix (cell extracts *vs.* supernatants/excreted fluids), one might also deduce whether the metabolites can leave the cell (think of transport processes, see below). **Warning for multicellular organisms!** When looking at extracellular fluids, be aware that the compounds might be of microbial origin (local microbiota) or come from the food/complex diet these organisms receive (in case of sampling tissues that food passes). Consult literature to determine the origin of these metabolites.

4.2 (Semi-)automatic gapfilling strategies

As with manual strategies, the semi-automatic ones usually rely on custom scripts and imagination. Here we will consider a case study of model curation where we can speed up the process by setting a loop.

One of the typical issues of curation is cross-compartment transport. This happens largely for two reasons: first, metabolite exchange reactions (see Section 4.4 for more details) are not transferred from template models due to lack of GPRs. Similarly, in most models of bacteria, the transport of metabolites from the extracellular compartment (e) to the periplasm (p) is usually assumed to be passive diffusion (typical BiGG reaction identifiers end with `tex`). Next, transport processes usually form a large gap in our biochemical understanding since determining transport types (facilitated diffusion/active transport), stoichiometry (coupled transport and the direction of co-substrate transport) and, in many cases, transporter promiscuity is a headache for experimentalists and theoreticians alike. This translates into models having very incomplete information and, as a consequence, the model drafting algorithms do not pick up these reactions "to be transferred" into the new draft model.

To automate the process, one could set up a couple of loops: first, for each metabolite in extracellular compartment, add an exchange reaction. Then, loop over metabolites in a selected compartment which are present in another compartment, but do not share a reaction where the stoichiometric coefficients have opposite signs. In that case there are a couple of options:

- search for transport reaction in the template model or
- add passive transport reactions $S_{comp1} \longleftrightarrow S_{comp2}$

The next curation strategy is gapfilling to enable the new model to produce biomass from its "typical diet". One can write a relatively simple routine to do semi-automatic gap-filling to extract the fastest routes from the template model for individual biomass components to be produced. A good algorithm to do this

would be to create a sink for every biomass constituent in the template model and perform parsimonious FBA (pFBA) (see Section 6.1) to obtain a predicted pathway. This sort of gapfilling can be justified if one has a trustworthy template model, i.e. most of the reactions identified by pFBA are in the target model *already*.

4.3 Formulating (dummy) biomass equation

A common objective function for these models is the biomass equation. To make a biomass equation you need to know the composition of the cell. A biomass equation can have different levels of detail. More coarse-grained biomass equations (e.g. with classes of molecules rather than details) may need additional dummy reactions to make that biomass component. At the most basic level the macromolecular content of the cell is defined (i.e., weight fraction of DNA, RNA, protein, lipid, etc.) in combination with the metabolites that make up each macromolecular group. Keep in mind that if a compound is not part of the biomass equation there may be no need to produce it and the corresponding pathway may not be active in your simulation. As many cells are composed of the same macromolecules, the main challenge is to get the right associated weight fractions. For that reason you could start with a biomass equation from a related organism or cell-type and change the weight fractions for your context. It may also be good to know more about the physiology of your organism/cell. E.g. human keratinocytes make lots of keratine, which may mean high protein content and a skewing towards specific amino acids. Note that the fractions in the biomass equation may have a strong influence on the flux distributions. For more reading on the biomass equation, see here [18].

4.4 Adding exchange and transport reactions

Exchange reactions are needed for metabolites that need to enter or leave the cell from the external space. So far, we have explored the fundamental components of GEMs, consisting of curated reactions and their corresponding metabolites. Before we can compute and solve these networks, we need to define boundary conditions for the model. These boundary conditions encompass the fluxes and their associated metabolites that operate at the "edge" of our model.

One prevalent type of boundary reaction involves the provision or "creation" of metabolites in the extracellular space, known as an exchange reaction. This exchange reaction represents the flow of metabolites from inside our system to 'nothing'. The reaction is defined as follows:

$$S_e \iff \tag{1}$$

In the case of glucose, the equation will read:

$$Glucose_e \iff \tag{2}$$

The exchange reactions can carry either a negative or a positive flux, indicating the direction of metabolite flow into (negative) or out (positive) of our system. Both types of exchanges are essential, as the change in concentration of all metabolite, including external ones, should be zero to maintain steady state. If there is no exchange flux (in or out), the metabolite depletes or accumulates (i.e. its concentration changes).

In some (rare) cases, a compound is produced for which no evidence of further conversion or export is present. This can happen for example in essential reaction for biomass production, where such a compound is formed as a byproduct. In these cases, a so-called *sink reaction* can be added to avoid numerical problems. A sink reaction is essentially the same as an exchange reaction, but the compound is removed from the intracellular space.

In the SBML file the boundary condition is specified by a Boolean attribute on the `Species` object. This attribute determines whether the metabolite can cross the system boundary, i.e., whether it is involved in exchange reactions with the external environment. However, it is worth noting that sometimes the exchange reaction can be identified solely through the use of an *EX* prefix in the identifier. In these cases, the presence of the *EX* prefix signals that the corresponding reaction serves as an exchange reaction for the associated metabolite.

Alongside exchange reactions, another set of reactions has to be created for metabolite transport from the extracellular space to the cell. Again, using the example of glucose (reversible passive transport, like in *Saccharomyces cerevisiae* [19]):

$$Glucose_e \iff Glucose_c \tag{3}$$

Multi-compartment reconstructions will also need transport reactions to get metabolites from one compartment to another. Neither type of reaction is typically well-annotated for a given genome, so information is often obtained from biochemical or physiological evidence. It may be difficult to associate genes to these reactions (which would have an effect on gene-knockout analyses). This also means that, if constructed from template models, your draft reconstruction does not contain any exchange reactions and will lack unannotated transport reactions.

4.5 Representing the growth medium

The representation of the growth medium is an important aspect of genome-scale metabolic modeling. Like the biomass definition, an accurate medium definition is a prerequisite to get accurate quantitative predictions out of a GEM. The "environments", or growth media, are represented in GEMs using the exchange reactions (see Section 4.4). Setting the lower- and upper flux bounds of exchange reactions to non-zero values means that the corresponding metabolite can enter/leave the system. There are three ways to set the bounds for exchange reactions, and thus to represent the external environment of a cell. Here, we describe them in preferential order of use:

1. specifying the experimentally determined uptake and secretion rates;
2. calculating hypothetical maximal uptake fluxes from the substrate concentrations;
3. specifying the concentrations of the available chemical species.

4.5.1 Specifying experimentally determined uptake fluxes The exchange reactions describe specific rates expressed as $mmol/gDW/h$, and the maximisation of the biomass assembly will take units $1/h$, representing the maximum theoretical specific growth rate μ. As pointed out in the previous section, these fluxes can represent uptake or secretion, and some of them can be experimentally measured. While water and inorganic ions are often considered to be non-limiting, exchange reactions for nutrients like carbon sources require more attention. Given a compound, its uptake/secretion rate is usually organism- and context-specific, so the experimental conditions to model should be decided with care. Rates are determined at the steady-state, so the data from a chemostat are simpler to use: given a substrate and a bacteria growing at a dilution rate D, the specific uptake rate of the substrate can be computed as:

$$v_{uptake} = D(S_f - S_r)/X \tag{4}$$

where D is the dilution rate ($1/h$), S_f and S_r are the substrate concentrations respectively in the feed and in the bioreactor ($mmol/L$), and X is the biomass concentration in the bioreactor (gDW/L).

Often, especially when modelling higher eukaryotes, exchange rates are not measured. Then, we want to limit the capabilities of our model in a different way.

4.5.2 Approximating uptake fluxes from substrate concentrations If an experiment is performed in a batch culture, as is usually the case for surface-adherent mammalian cells, an estimate of a maximal uptake flux can also be made. To do this, we need to know, or be able to approximate, the molar amount (usually *mmol*; **NOTE: not a concentration!**) of the growth-medium components (S_i); the dry weight per cell (m_{cell}, usually in gram dry weight, or *gDW*) and the number of cells the culture ($n_{initial}$), or the total dry weight of the culture ($m_{cell} \times n_{initial}$), when the media was added ($t = 0$); and the time spent on the batch of medium before refreshing (τ, usually in *hours*).

Knowing this, we can construct an equation for converting molar amounts of available substrate into *lower bounds* for the exchange fluxes (LB_i) for each component in the culture:

$$LB_i = -1.0 \times \frac{S_i}{m_{cell} \times n_{initial}} \times \frac{1}{\tau} \tag{5}$$

In which the coefficient *-1.0* is added in line with the convention that uptake from the extracellular compartment has a negative flux value. This equation yields a lower flux bound for an exchange function with with units of $mmol/gDW/h$ (or similar). Note that the exercise of setting a **maximal** uptake flux requires two assumptions for each available substrate two main assumptions associated with this calculation:

1. The total cellular dry weight does not decrease over time (neither m_{cell} nor $n_{initial}$ decrease; cells don't die or lose weight)
    - If it did, the size of the denominator would decrease, thereby increasing the magnitude of the lower bound
    - Note that it is not a problem if the cells grow or multiply, as this would decrease the magnitude of the lower bound, meaning that the "actual" lower bound would still fall within our specified range (given the range goes up to at least 0)
2. The component in the growth medium is fully depleted after τ amount of time
    - This assumption is obviously false for most components, since we are interested in a hypotheticla maximal uptake flux, the maximum is reached when the component is fully consumed.

Also, the modeler should pay attention to which species are forming in solution, like the ions formed from dissolved salts. For example, if both $MgSO_4$ and $ZnSO_4$ salts are added to the medium, the bound through the exchange reaction for the sulphate ion SO_4^{2-} should account for both. We normally assume that the exchange bounds in GEMs come from chemically defined media, but in principle complex media could also be modeled, if it's possible to obtain their measured/approximated composition (for example: what is the amino acid content of 8 g/L of meat extract?). Please see [20] for further reading.

4.5.3 Specifying molar substrate amounts (advanced)

The unit of measure of the exchange reactions is *mmol*, and the objective value of the FBA - if the biomass assembly is set as objective - will be the amount of biomass produced in gDW. If the biomass yield ($gDW/mmol$) is the desired output, divide by the amount of substrate that was utilized. Since wet-lab liquid media recipes often have components expressed in g/L, it is necessary to convert them to molar amounts using the appropriate molecular weights (g/mol) and the volume of the medium (L). **Note that we are working with molar amounts, and not concentrations. We cannot use concentrations because the volume of the cell will determine the intracellular concentration, so the extracellular concentration cannot be transferred directly to the intracellular concentration.** Note, again, that all sources of the various ions should be considered and that complex media can also be modeled given certain assumptions, as mentioned above.

5 Model curation

5.1 Checks to run

When the model is (almost) finished, there are some 'sanity checks' that should be performed to rule out some common, but avoidable, mistakes. There are some tools that can help with this.

5.1.1 Blocked reactions

First, it does not make sense to have a reaction in your model that cannot carry a flux. We call this a 'blocked reaction'. Before you use your model, you should check whether you have blocked reactions. Some toolboxes, such as COBRApy, have specific functions to check this[2]. CBMPy has a function[3] that finds reactions with only a substrate or a product. If your preferred tool does not have a specific function, you can run Flux Variability Analysis (FVA, see Section 6.1) on all of your reactions and search for the ones with minimal and maximal flux of 0. It is good to be aware of these reactions, as they are either (i) a knowledge gap (unidentified reaction/enzyme missing) or (ii) an artifact. However, in the latter case we do not need to remove them as they do not carry flux anyways but could do so under other conditions.

5.1.2 Spontaneous ATP production

The next sanity check is for the spontaneous production of metabolites. The usual routine is as follows: without an external input, the model should not be able to phosphorylate ADP into ATP and then hydrolyze it (carry non-zero flux through ATP maintenance-like reaction). To implement this, one should add a hydrolysis reaction

$$ATP + H_2O \rightarrow ADP + P_i + H^+ \tag{6}$$

[2] cobra.flux_analysis.find_blocked_reactions()
[3] cbmpy.CBTools.findDeadEndReactions()

set it as optimization objective, and set all the exchange bounds to [0;1000]. If no ATP can be produced from *totally* nothing, another check could be opening an exchange reaction for oxygen uptake only (`EX_02_e` to [-1000;1000]. Some potentially spontaneous generation cycles do not need external carbon but do require oxygen.

5.1.3 Dead-end and orphan metabolites

Similarly, it does not make sense to have metabolites in your model that cannot be fully converted. If a metabolite can be produced, but not consumed it is called a 'dead-end metabolite'. If a metabolite can be consumed, but not produced, it is called an 'orphan metabolite'. CBMPy has a function[4] to find dead-end and orphan metabolites (both named dead-end metabolites).

5.1.4 Elemental and charge conservation

In addition, all reactions in the model should have elemental and charge conservation. Otherwise, mass could be created/lost in the model, allowing unrealistic/futile cycles to appear in your model simulations. An example of such a cycle could be an unbalanced reaction in the human genome-scale model Recon3D [21] which would spontaneously produce C_2H_4 moieties which then could be used for ATP production without external carbon- or energy source. Both COBRApy[5] and CBMPy[6] have functions to check this.

5.2 SBML features to store annotations

SBML standard specification has several tools to assist modelers to organize large amounts of information on different model objects. The completeness of annotations might influence the downstream applications (e.g. extraction of context-specific models, see Chapter 7), so compiling a comprehensive annotations set is highly advised. Also, as highlighted later (see Section 5.3), MEMOTE scoring heavily depends on the completeness of annotations. There are two SBML features to consider: the *SBML Groups*, and the MIRIAM annotations. MIRIAM ("Minimum Information Required In The Annotation of Models") is a community effort to encourage modelers to spend time and effort on the model annotation. Since MIRIAM was created with interoperability in mind, multiple identifiers (from different sources) can be assigned to the same object using this format.

For basic annotation of reactions, one can leverage the support of *SBML Groups*: every object can be assigned to multiple groups. This is mostly applied to reactions, where we group them in a coarse-grained manner according to the place of this reaction in the metabolic network (also called "Subsystems"). These descriptors can be really coarse, e.g. "Central carbon metabolism", or more fine-grained, "Lysine catabolism". Alongside metabolic functions, typical subsystems in the models are "Exchange/demand reactions" and "Transport reactions". The best practice here is to keep the subsystems relatively coarse-grained, as it helps you to group elements that you would like to inspect together. A moderate number of subsystems might simplify the visualization and interpretation of, for instance, new constraints introduced (# of active reactions/non-zero fluxes), or generation of context-specific models (omics-driven reduction of metabolic networks, see below).

The MIRIAM terms (e.g. alternative identifiers) can be added for any model object via `addMIRIAMannotation()` routine in CBMPy. It is good practice to label as many things as possible, although complete coverage of identifiers seems to be the most beneficial for genes and metabolites - and less so for reactions. For every gene, one should add at least one type of gene identifier: e.g., Ensembl Genome ID, UniProt KB accession code, NCBI Gene (formerly Entrez) identifier, or NCBI Protein ID. NCBI Transcript IDs are less encouraged, especially for organisms capable of alternative splicing. The most useful mapping schema for metabolites is ChEBI (Chemical Entities of Biological Interest), although alternatives are available.

Next, model objects can be assigned so-called Systems Biology Ontology (SBO) Terms (similar to the GO Terms), The standard form of these terms is, again, similar to GO Terms, e.g., `SBO:0000627`. They can be added to the model in the same way as other MIRIAM identifiers. Recently, a tool for automatic assignment of SBO Terms was published, SBOannotator [22], which detects the nature of the reaction (metabolic/transport/exchange etc.) and assigns SBO Terms accordingly. The presence of these annotations is also scored in the MEMOTE pipeline, as detailed below.

[4] cbmpy.CBTools.findDeadEndMetabolites()
[5] cobra.manipulation.validate.check_mass_balance()
[6] cbmpy.CBTools.checkReactionBalanceElemental()

5.3 MEMOTE

MEMOTE [23] is a tool to score how comprehensive the annotations of your GEM are. Since the web server seems to be discontinued (as of September 2023), the only option left is to install it either from the GitHub repository or install via `pip` directly. The method to use from the command line is `report`, and the command to run is as follows: `memote report snapshot <model.xml>`. The tool runs a multitude of tests to check the consistency of stoichiometric matrix, scores the (completeness of the) annotations, and does some other sanity checks and tests. However, only the first two items are scored in a final numerical output, ranging from 0 to 100%. The use of running MEMOTE is not to report its final score in the paper (although you are welcome to do so), but use it to identify gaps in annotation.

Often MEMOTE is mentioned in publications to showcase the performance of a model. However, the score of a model checked with MEMOTE consists mostly out of the presence or absence of annotations.

6 Running the model

6.1 Flux balance analysis

The most used application of GEMs is flux balance analysis (FBA). For a concise overview on the background of FBA, see [4] and the introduction of this text. Alongside the *vanilla* FBA, many different analyses based on the principles of FBA exist, and a couple of most used follow-up analyses are discussed below.

Parsimonious FBA Parsimonious FBA (pFBA) is a routine used in genome-scale modeling to obtain flux distributions with a reduced solution space. pFBA addresses the problem of obtaining a single flux distribution out of large solution space which might include fluxes that are not zero but cancel out other fluxes. The fundamental assumption is that cells minimize the amount of enzymes needed to sustain metabolism, and therefore carry only the fluxes that they **absolutely** need, while all other fluxes are kept at zero.

We run two sequential optimizations: first, the usual FBA to obtain the initial flux distribution and the optimal value of the optimization function. This value is then set as an additional constraint for the following optimization round: a new linear program (LP) is formed which minimizes the sum of absolute fluxes. Therefore, pFBA is sometimes also called "FBA with minimization of absolute fluxes".

Flux Variability Analysis Flux variability analysis (FVA) is used to determine the minimal and the maximal value of a flux under certain conditions at the optimal objective function value. This can be used to validate the uniqueness of your solution.

6.2 Software commands for running FBA

Most previously mentioned software packages have built-in functions for performing FBA.

CBMPy CBMPy has separate functions for the LP solvers it works with (GLPK[7] and CPLEX[8]). These functions take a `CBModel` object as an input. In these methods, the method of the solver can be set using the `method` argument. See the documentation for the different options. When this function is used, the `CBModel` object that was used as input is updated and now contains the information about the optimal fluxes generated. From this object, the stoichiometric matrix can be easily retrieved using the `buildStoichMatrix()` function.

COBRA and RAVEN The COBRA Toolbox uses the `solveCbModel()` wrapper to perform FBA, and parsimonious FBA can be run using the `pFBA()` command. For FVA, `fluxVariability()` wrapper is used. In a similar spirit to COBRA Toolbox, RAVEN command `solveLP()` will perform FBA (RAVEN currently contains no pFBA and FVA implementations).

COBRApy COBRApy performs 'normal' FBA when the `model.optimize()` function is called. The preferred solver can be set in general (using the functions in `cobra.Configuration()`). The optimization function returns a `cobra.core.solution` object, from which the fluxes, objective value, shadow prices and reduced costs can be obtained. It is more difficult to generate the stoichiometric matrix using COBRApy, which limits some analyses (e.g. matrix rank computation, see Section 7.2). Generally, COBRApy is user friendly to work with and can generate useful summaries that are well-readable in Jupyter Notebooks.

[7] cbmpy.CBCPLEX.cplx_analyzeModel()
[8] cbmpy.CBGLPK.glpk_analyzeModel()

6.3 Sensitivity analysis

Unlike the kinetic models, whose parameter space can be extensively analyzed in terms of the sensitivity of simulations to the parameter values, stoichiometric models have only a very limited set of follow-up analyses we can employ to infer the sensitivity of the solutions. These come from the very foundational theory of linear programming, and therefore bear names which can be easily associated with economics: (i) reduced costs-, and (ii) shadow prices analysis. In the following paragraphs we'll provide an intuitive explanation of these concepts.

Reduced costs A reduced cost describes how the change of the bound value affects the objective function value. Normally we consider the fluxes that are at one (or both, in case of equality constraints) of their bounds for this operation. For example, typically at least an exchange flux is hitting its lower (or upper) bound in your FBA solution. In such cases, we call the exchange flux at its bound an "active constraint", or, if the optimization objective is biomass formation, "growth-limiting". Why? To confirm the validity of this claim you can consider a thought experiment: if you change the flux bounds from the defaults inside the cell, it is very unlikely the flux distribution will change. Meanwhile, your FBA solution will contain at least one exchange flux at its bound, and increasing the module of the flux bound will affect your solution (see Figure 4, left panel). The reduced costs are actually outputted by the simplex algorithm for every reaction[9] and can be obtained alongside the FBA solution.

Mathematically, we define reduced costs as a ratio $r_i = \frac{\delta v_{obj}}{\delta b_i}$ between the change of the objective function value δv_{obj} and increment of the flux bound for the v_i, δb_i. The reduced costs could also be *scaled*, where we scale both terms by their original values: $r_{s,i} = \frac{\delta v_{obj}}{\delta b_i} \frac{b_i}{v_{obj}}$. If the $r_{s,i}$ equals 1, we can conclude that the flux v_i is the only one limiting growth (sanity check: the objective value increases by the same magnitude as the limiting flux value).

Shadow prices Shadow prices describe the influence of metabolite import on the objective function value. Imagine a situation where a metabolite which can be used to "ramp up" your objective function value becomes available in the environment "for free", e.g., adding an additional nutrient to existing growth media (media supplementation). Then we describe the shadow price (Figure 4, right panel) of the metabolite j as a ratio $s_j = \frac{\delta v_{obj}}{v_{j\ import}}$ between the change of the objective function value δv_{obj} and the flux value of the "free" import of metabolite j, $v_{j\ import}$.

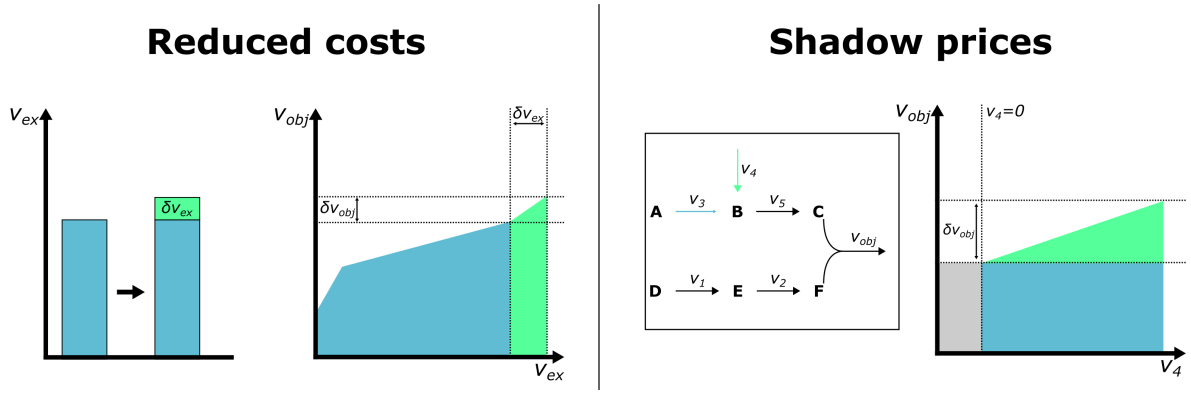

Figure 4: The visual interpretation of the reduced costs (left) and shadow prices (right). In the reduced costs scenario, the maximal flux through exchange reaction v_{ex} is increased by δv_{ex}, and the value of the objective flux v_{obj} increases by δv_{obj}. In the shadow prices scenario, the shadow price for metabolite **B** is computed by adding a source flux (v_4 in the example) and computing the ratio between the increase of objective flux δv_{obj} and the value of v_4.

[9] cobrapy: call solution object solution.to_frame(), the reduced costs is one of the columns of the corresponding *pandas DataFrame*; cbmpy: use cbmpy.CBCPLEX.getReducedCosts() or cbmpy.CBGLPK.getReducedCosts() method to obtain a dictionary of reduced costs.

7 Advanced model handling

7.1 Visualization of networks

Escher [12] is a web-based visualization tool for GEM solutions. In addition, a Python API is available for visualization, which works well with Jupyter Notebooks. Escher is compatible with COBRApy notations, while CBMPy results need to be converted first. Basically, Escher takes a combination (dictionary) of reaction (or gene) names and flux values, and maps it on a vector file with arrows for the correct reactions.

This vector file is the map that is your visualization. For model organisms, such as *E. coli*, *S. cerevisiae* and human, there are template maps available for the central carbon metabolism for some models (ecolicore, iJO1366, iMM904 and RECON1). These can be used as a starting point. However, if you use different models for the same organism, notation might be different for some reactions. Then, the map needs to be adapted. For different organisms, a new map needs to be created, or for a similar organism, one of the template maps can be adapted. This is quite a tedious job.

7.2 Verifying the numerical accuracy of the solution

The solution of FBA is determined by the constraints that were imposed, which are often the substrate uptake rate and the non-growth associated maintenance. Such a solution consists of a linear combination of a number of elementary flux modes (EFMs), where the number is equal to the amount of active constraints (for more details and mathematics, see [24]). To check the correctness of the solution, the rank of the active stoichiometric matrix (S) and the amount of columns can be checked according to Eq. 7. Note that this only works with the stoichiometric matrix where (i) all reversible reactions have been split into a forward and a reverse reaction and (ii) the reactions carrying zero flux are removed.

$$n_{columns,S} = rank(S) + n_{active\ constraints} \tag{7}$$

Using this approach, inconsistencies that are below the numerical tolerance bounds of the solver (which are usually 10^{-6} for CPLEX by default) can be determined. For example, in *E. coli* model iML1515 [25], the biotin uptake bounds are [0, 1000], which means that biotin can only be exported, while it is essential for growth. Because the stoichiometric coefficient for biotin in the biomass reaction is very low, the solution often remains feasible. Using this approach, a discrepancy in Eq. 7 will be found.

7.3 Setting a suitable solver method

On the background of GEM-handling software, a linear program is solved using a certain method. Most of the time, the (front-end) software used by genome scale modellers, such as cobrapy or cbmpy, is coupled to another software, such as CPLEX or GLPK (see Section 1.6), that solves the linear program. As linear programming has been used for decades for very complicated problems, very fast and efficient computing methods have been developed to solve them. The solution that is found is always a combination of rays (corresponding to thermodynamically infeasible irreversible cycles in a GEM), linealities (corresponding to reversible cycles) and vertices (corresponding to a route from a source to a sink metabolite) [26] (Figure 5). Visually, all possible solutions to a linear program form a polyhedral cone in the flux space. An optimal solution is always at a cornerpoint [27]; intuitively, if something is maximized (or minimized), it is pushed to a limit, which you can imagine visually as almost leaving a space, thus it ends up in a corner. The corners are spanned by the vertices (also see Figure 1 for an illustration).

However, a solution can still be optimal when a combination of vertices (routes from source to sink) meeting in a corner is used at the same time as a lineality (a cycle). An example of a lineality in a genome-scale model is back-and-forth transport of a molecule over a membrane without any transportation costs. This transport can occur with any value up until its bounds, without influencing the objective function value. However, this is a flux we are not interested in, as it has no effect whatsoever and will only create noise in our flux distributions. In addition, if you imagine that a cell has a limited amount of proteins it can express, and the proteins are needed to make such cycles run, it is unlikely that an 'optimal' cell has such cycles. Rays are even more irrelevant as they are thermodynamically infeasible.

So, what can we do practically? Different solvers use different methods to walk around the solution space toward the optimum. Interior point solvers move through the interior of the solution space, as the name suggests. This results in a solution that can contain linealities. On the other hand, a simplex solver moves

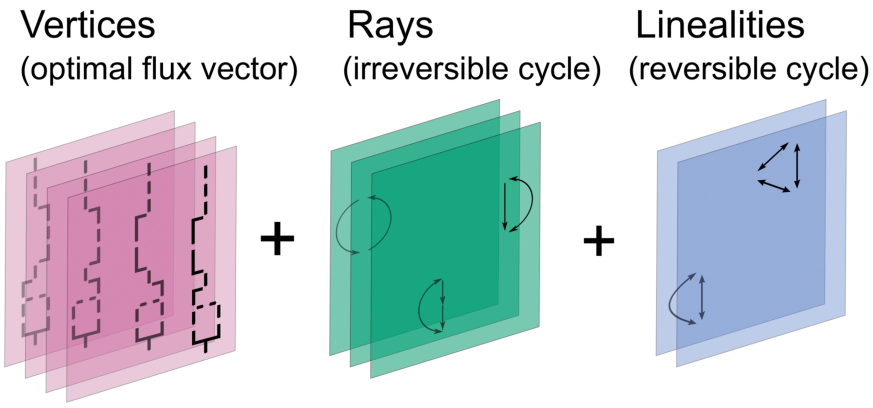

Figure 5: Visual representation of vertices, rays and linealities. Figure taken from [24].

along the vertices of the solution space. As a consequence, only a combination of vertices can be found as the optimal solution. Interior point methods are supposedly faster and more efficient than simplex models, however in practice, it does not make a big difference for GEMs of microbial metabolisms (≤5000 reactions).

CBMPy has the option to force CPLEX or GLPK to use a simplex algorithm. This can be done in the `analyzeModel()` functions or the `doFBA()` function, by setting the option method. When using GLPK, set method='s', which is also the default. When using CPLEX, there is an option for both a primal and a dual simplex solver (method='p' or 'd', respectively). However, this *does* result in a flux distribution with cycles (linealities), whereas using GLPK does not have this problem. Thus, it is recommended to use GLPK if a true simplex solution is required. A disadvantage is that GLPK is a bit slower than CPLEX, so when working with big models, this might become a problem. COBRApy currently does not have an easily accessible option to choose the solver that is used.

Note. Using a simplex method as a solver yields the same solution as using parsimonious FBA (see Section 6.1).

7.4 Context-specific models

Genome-scale reconstructions contain all of the reactions encoded for by the genome of an organism (as well as some non-genome-associated reactions). However, only subset of proteins is expressed in specific cells under a given condition, leading to the development of strategies to restrict GEMs only to the reactions that are expressed in the cell under study under the relevant conditions [28]. These are referred to as *context-specific models*. The process of removing non-relevant reactions while keeping in the relevant ones is called *model restriction* or *model extraction*. Different *model extraction methods* (MEMs) have been developed, all of them in the form of algorithms that start with a generic GEM and end with a restricted subset of that GEM meeting some criteria. The nature of the criteria according to which reactions are removed or retained is what differentiates different MEMs. Robaina Estevéz and Nikoloski [28] define three families of MEMs, each named after its first representative:

1. GIMME-like
2. iMAT-like
3. MBA-like

7.4.1 GIMME-like algorithms GIMME-like (*Gene inactivation moderated by metabolism and expression*) algorithms solve two LP problems. First, they optimise one (or a set of) Required Metabolic Function(s) (RMFs, a given production rate of lactate or a given biomass production flux, for instance). All reactions with insufficient evidence (e.g., expression levels below a set threshold) are removed. The model then tests whether it can perform its RMFs to a set degree (e.g. 90% of the optimised value). If it cannot, it systematically reinserts removed reactions. To decide which reactions to reinsert, GIMME calculates an *inconsistency score*, which is the product of the flux required through the reinserted reaction and the distance between the measured expression level and the previously set threshold (*threshold – measurement*). This

results in a weighted penalty which is minimised to find the most consistent set of reactions to reinsert. This is the second LP problem. A major advantage is that RMFs constrain the solution space further than the expression data alone, which has been shown to make for more accurate models [29]. A distinct feature is the use of flux- and evidence-weighted inconsistency scores, which the algorithm also provides as a report with the reduced model.

This family includes the original **GIMME algorithm** itself [30], as well as its descendants, **GIMMEp** [31], and **GIM^{3}E** [32]. GIMMEp allegedly allows for the easier integration of proteomics data, although its implementation is not well-described. GIM^{3}E allows for the integration of metabolomics data as well assigning penalty value to all reactions instead of only reactions with expression levels below the threshold. This does, however, render the second LP problem a MILP problem, which is more computationally expensive.

7.4.2 iMAT-LIKE ALGORITHMS This family, similarly to the GIMME-like family, tries to match the maximal number of reaction states (active or inactive) with expression data (expressed or not expressed). However, different to GIMME, the original members of this family *did not* require pre-set metabolic functions. It was argued that this is an advantage, as RMFs are optimised and optimisation objectives are tricky to define, especially in multicellular organisms. Instead, iMAT-like algorithms seek a flux distribution which maximises the number of reactions with evidence (expression above a threshold) that carry flux above a certain threshold while minimising the number of reactions *without* evidence that do. Reactions carrying flux are retained while those that do not, are discarded. Shlomi et al. [33], who made the first familty-member, **iMAT**, justify this intuitively by arguing that not all gene products (mRNA or protein) are necessarily detected, while detected proteins might be post-translationally inactivated (or detected mRNA might not form a functional enzyme). The mathematical formulation of this type of algorithm results in an MILP in which a binary variable (as opposed to GIMME's weighted inconsistency scores) denotes whether a reaction should be included or not. The most condordant flux distribution is sought.

As mentioned, the first member of this family was **iMAT** (integrative Metabolic Analysis Tool [33], named *post facto* after an online tool for performing the algorithm [34]). It was followed by **INIT** (Integrative Network Inference for Tissues, [35]) algorithms, as well as the descendants of INIT: the task-driven INIT (**tINIT**, [36]) and the fast task-driven INIT (**ftINIT**, [37]).

INIT expanded on the logic of **iMAT** by allowing the designation of metabolites that needed to be produced. This was done specifically with the multicellular organisms in mind, in which cells often do not simply optimise growth but rather use their metabolic machinery to produce metabolites for use elsewhere in the body. The production of these metabolites are not mandated, but positively weighted while the flux distribution most concordant with the data is sought. **tINIT** goes further by allowing the user to predefine a set of metabolic functions that need to be performed alongside the optimisation done by **INIT** e.g., "regardless of the data, x amount of biomass needs to be produced from y amount of substrate". **ftINIT** was developed to reduce the computational cost of **tINIT** by reformulating the MILP and splitting network minimisation into two substeps. This reduced evaluation time by more than an order of magnitude, e.g. by merging linearly dependent reactions in the optimisation. However, be wary that simplifications could lead to oversights. The inquisitive reader is referred to the orignal article [37]. INIT, tINIT, and ftINIT are integrated into the RAVEN toolbox [38], making them quite user-friendly.

7.4.3 MBA-LIKE ALGORITHMS MBA-like algorithms start with an *a priori* classification of reactions as either belonging to a *core*, or not. This *core* is the set of reactions for which positive evidence was found. This includes data (transcriptomics or proteomics) as well as literature. Once the core has been defined, these methods prune the GEM by eliminating non-core reactions while maintaining *flux consistency*. *Flux consistency* is attained when no reactions in the model are blocked (carrying no flux). Another major distinction of MBA-like algorithms is that they do not return a flux prediction, only a context-specific reconstruction. The advantage of this family of MEMs is that they allow for the integration or large amounts of evidence and expert input, which places very useful constraints on the solution space. Reactions for which overwhelming evidence is available, also from literature, can be forced into the final model. MBA-like MEMs also do not require RMFs, which means that the problem of specifying a metabolic objective is sidestepped. The family includes the original **MBA** (Model Building Algorithm, [39]), **mCADRE** (metabolic Context-specificity Assessed by Deterministic Reaction Evaluation, [40], **FASTCORE** [41], and **CORDA** Cost Optimisation Reaction Dependency Assessment, [42].

MBA allows the designation of high-likelihood (C_H) and moderate-likelihood (C_M) core reactions. C_H cannot be removed, while C_M reactions can be pruned at a penalty. The remaining reactions are non-core (N_C) and are removed at a benefit to the penalty score. The score is maximised while maintaining flux consistency. In MBA, the order in which reactions are pruned determines the possible outcomes, so the process is usually repeated (e.g. 1000 times) and the candidate models aggregated.

mCADRE includes a second type of evidence - *connectivity-based evidence* which includes proximity to reactions with high evidence as evidence in and of its self. This is particularly useful for scoring reactions without (known) gene associations. mCADRE also ranks non-core reactions based on evidence and removes them in order from least evidence to most. This provides an order to the pruning process, abrogating the need for multiple iterations. mCADRE also allows core reactions to be pruned (core is flexible) under certain, limited conditions, and also allows for the designation of key metabolites that must be made by the restricted model.

FASTCORE also relies on the predesignation of reactions as core or non-core, but then maximises cardinality through the core while minimising cardinality outside the core. The distinction is subtle, but important: maximising cardinality means that the number of reactions that can carry flux is maximised instead of maximising the flux through the set of core reactions. This is a softer objective and allows decreased computation time.

Finally, **CORDA** is unique in that it performs a *dependency assessment*: first, reactions are designated as high (HC), medium (MC) or negative confidence (NC), or other (OT, no evidence); a pseudometabolite is added as a product of every reaction in the model, which represents a cost (more undesirable reactions have higher stoichiometries for this metabolite); the model is initialised with HC reactions, and then MC and NC reactions are added back as required - all the while minimising the production the pseudometabolite - to allow the already-included reactions to carry flux; finally, OT reactions associated with any reaction in the model are added back. The strength of this approach is that is doesn't require a MILP to be solved, only FBAs, which reduces computational cost. Since the production of the pseudometabolite is minimised, the criteria is also softer: a reaction need not strictly speaking be necessary, but it reduces the flux through undesirable reactions.

7.5 Choosing the right MEM

The MEM employed by a user will depend on the genome-scale reconstruction to be restricted, the amount and quality of the available data, the biological expertise of the user, computational power and time available, and on the ease of defining RMFs. Molversmyr *et al.* [43] make some interesting observations on the performance of various algorithms in extracting a salmon liver model from an existing generic salmon GEM (SALARECON). They observe that iMAT, INIT, and GIMME reduce the model much more than FASTCORE, MBA, and mCADRE, indicating that the former three make "more context-specific" models. GIMME had the shortest computation time, followed by FASTCORE, then iMAT, then MBA and mCADRE, and finally INIT. Finally, it bears mentioning that GIMME was the only MEM to produce models with significant variance in principal component scores explained by life-stage. Based on the method, the following diagram from Robaina and Estevéz [28] is a useful summary of the different MEM families (Figure 6).

7.6 Strain-specific models

When working with bacteria, we know that different strains of the same species exhibit different phenotypes. They may have evolved differently due to geographic isolation, different selection pressures, horizontal gene transfer events, and so on. GEMs are usually bound to a genome, and thus they are specific for a particular strain (think about Bas' *Lactoplantibacillus* model which was mentioned during the Introduction: it was made for the strain WCFS1). Therefore, using a GEM built on a different strain from the one you are studying could yield misleading results.

The strain-specificity of your GEMs is even more important when you are trying to use modeling as a tool to explore the biodiversity of a species. In this kind of study, you typically have hundreds of strains of which to produce a GEM (see [44] for one of the earliest examples). This deck of GEMs can subsequently be used to predict strain-specific auxotrophies, growth-enabling sugars, and so on.

Strain-specific GEMs are usually produced *by subtraction*, in a way conceptually similar to the restriction procedures described above. In practical terms, you take a curated GEM as your reference, make a copy

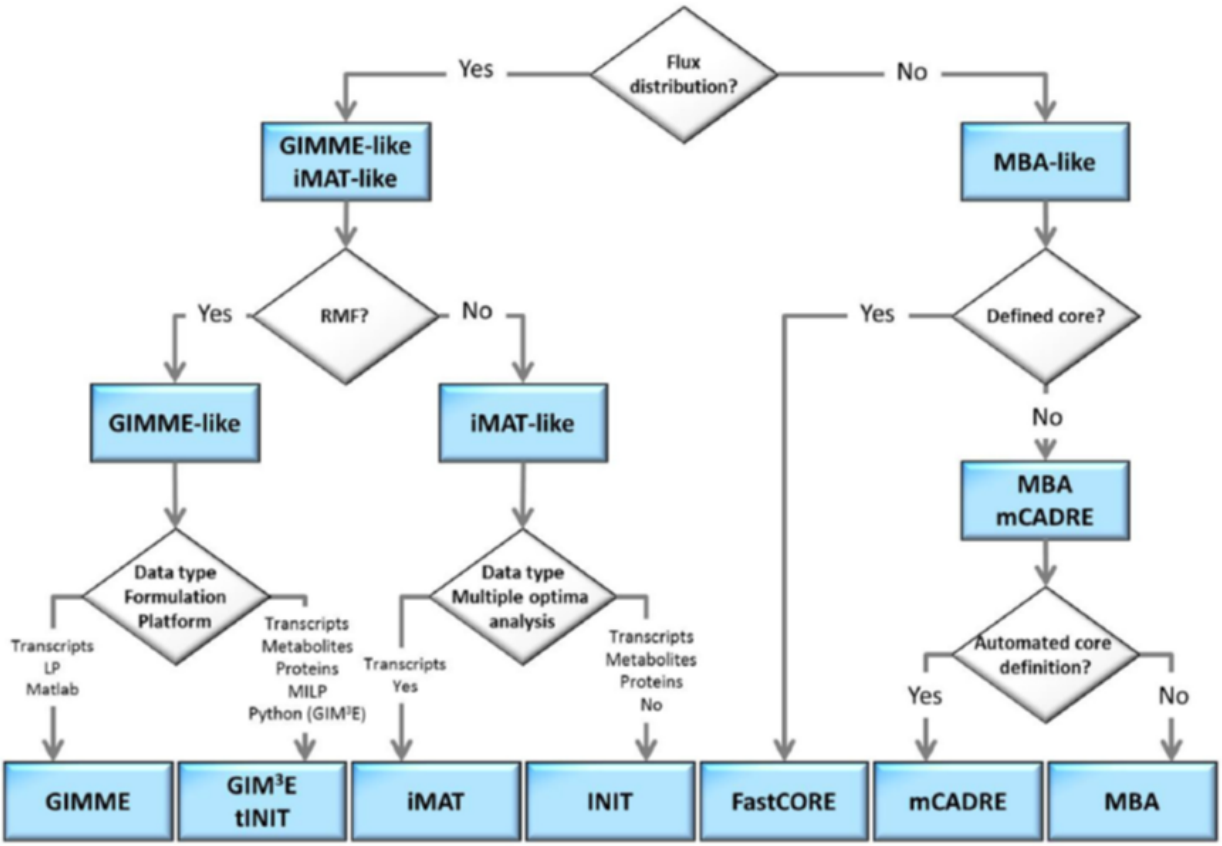

Figure 6: Optimal choice of methodologies when tackling a context-specific reconstruction problem. The choice can be made by answering a few questions, in a flowchart manner, related to: demand of model extraction and flux prediction, knowledge on a required metabolic functionality, the type of experimental data available or the computational platform. Figure taken from [28].

of it, and then you remove the genes (and thereby the reactions) that are not present in your strain, but are found in the reference strain. This reference GEM should be representative of the entire species and is usually built on a pan-genome instead of single genome.

Now, let's take a few steps back. First we collect some good quality genome assemblies of our strains. Then we apply the same method for CDS prediction as discussed in Section 2.2, obtaining a proteome for each strain. Next we compute the pan-genome of the species using a dedicated tool like the established Roary [45] or similar, providing as input the deck of proteomes. Such tools basically create groups or clusters of highly similar genes and then select one gene as representative sequence for that cluster (for example, the fructose-1,6-bisphosphate aldolases of all the strains are grouped together under the same `cluster_0123`, and the one coming from the WCFS1 strain is selected as representative). These tools usually return two important outputs: 1) the presence-absence matrix, a table indicating for each strain which clusters it harbours; 2) a FASTA file with the representative sequences. At this point, we use the representative sequences to build a GEM that we could call the pan-model, comprising of all the features of our species and encoding gene clusters instead of the usual strain-specific genes. Of course it must be curated like any other model, following the principles already discussed in Section 5.

Taking the pan-model as reference, we can create strain-specific GEMs *by subtraction* as described above, reading the presence-absence matrix to know which clusters to remove. Finally, we convert (rename) each gene cluster to the original strain-specific gene names. These last steps can now be automated by recently developed tools like Bactabolize [46], but be aware that you must provide the reference model yourself. For more details on strain-specific GEMs generation, refer to the 2019 Nature Protocol Extension [47].

7.7 Community models

In addition of single organisms, GEMs can also represent an entire microbial community. These community GEMs can be of two types:

- **bag-of-genes models**, where all reactions are grouped in the same compartment, regardless the organism they belong to.
- **compartmentalized community matrix**, where each organism is represented as a different compartment.

The first type of model can be generated as any single organism GEM, just using the metagenome as starting point. Despite their simplicity and high level of approximation, bag-of genes models can be useful to assess the overall metabolic capabilities of a microbial community.

On the other hand, compartmentalized community GEMs can capture more specific features of microbial communities, e.g. competition for substrates, cross-feeding, division of labour (subdivision of complex pathways into multiple organisms), etc. This type of model can be constructed by merging the individual GEMs of the individual organisms as different compartments in communication with the same extracellular space.

Constraint-based methods can then be applied to community models, but some additional aspects must be taken into account. Primarily, the type of objective function to use and the ratio in which each organism is present in the community need to be decided on.

There are different options for the objective function. One is to optimize the growth of each organism separately, but at the same time to allow the uptake of the byproducts secreted by the other organisms. This way can model competition and "costless" cross-feeding of byproducts (see for example [48]).

Another possibility is to set a community objective (typically the sum of the biomasses of all the organisms). This approach allows the modelling of "costly" cross-feeding interactions, but tends to unnaturally favor the organism with the highest yield on the limiting substrate. This issue can be solved using experimental data (see, for example, [49]), or assuming a balanced growth of the community, as done by cFBA [50] and SteadyCom [51].

Finally is possible to combine these two options in a bilevel optimization, optimizing individual fitness first, and then community fitness. This is done, for example, in d-OptCom [52].

8 Useful resources

The Jupyter notebooks are uploaded to a Google Drive, and you can use the Google Colab service to run them in cloud (press "Open in Google Colab"). Alternatively, download the notebooks to your local machine.

1. A tutorial on how to construct a toy FBA model in CBMPy (originally written for the *Basic Models of Biological Networks* course taught at the VU).
2. The GEM handling tutorial that is usually given to students starting their internships on genome-scale modeling.

9 Data types and index

Table 1: Data used for GEMs, their uses, and where to find information on the data type within this guide.

Data	Data type	General use	Chapter
sequenced genome of organism(s)	FASTA	Get open reading frames (ORF) Enzyme coding sequences (CDS)	3;7.6
ORF or CDS	gff, gbk, fasta	Reconstruction of metabolic model	3
GEM	xml	Metabolic modeling	3
untargeted metabolomics		Gap filling	4.1
biomass composition		Biomass equation	4.3
^{13}C metabolomics		Intracellular fluxes	
Expression profiling data (transcriptomics and proteomics)		Context specific model; model restriction or model extraction	7
Flux data (uptake and secretion rates)		Growth rate prediction	4.5
Yield data		Biomass concentration prediction	4.5
Media composition (concentrations)		Defining media composition	4.5

References

[1] Bas Teusink, Anne Wiersma, Douwe Molenaar, Christof Francke, Willem M De Vos, Roland J Siezen, and Eddy J Smid. Analysis of growth of lactobacillus plantarum wcfs1 on a complex medium using a genome-scale metabolic model. *Journal of Biological Chemistry*, 281(52):40041–40048, 2006.

[2] Vincent Somerville, Pranas Grigaitis, Julius Battjes, Francesco Moro, and Bas Teusink. Use and limitations of genome-scale metabolic models in food microbiology. *Current Opinion in Food Science*, 43: 225–231, 2022.

[3] Pranas Grigaitis. *Constrain and conquer: explaining metabolic strategies of microbial life through optimal resource allocation*. Phd-thesis - research and graduation internal, Vrije Universiteit Amsterdam, March 2023.

[4] Jeffrey D Orth, Ines Thiele, and Bernhard Ø Palsson. What is flux balance analysis? *Nature biotechnology*, 28(3):245–248, 2010.

[5] Bas Teusink, Anne Wiersma, Leo Jacobs, Richard A Notebaart, and Eddy J Smid. Understanding the adaptive growth strategy of lactobacillus plantarum by in silico optimisation. *PLoS computational biology*, 5(6):e1000410, 2009.

[6] Brett Olivier, willigott, Douwe Molenaar, and Bas Teusink. Cbmpy release 0.8.4, February 2023. URL `https://doi.org/10.5281/zenodo.7679232`.

[7] Laurent Heirendt, Sylvain Arreckx, Thomas Pfau, Sebastián N Mendoza, Anne Richelle, Almut Heinken, Hulda S Haraldsdóttir, Jacek Wachowiak, Sarah M Keating, Vanja Vlasov, et al. Creation and analysis of biochemical constraint-based models using the cobra toolbox v. 3.0. *Nature protocols*, 14(3):639–702, 2019.

[8] Hao Wang, Simonas Marcišauskas, Benjamín J Sánchez, Iván Domenzain, Daniel Hermansson, Rasmus Agren, Jens Nielsen, and Eduard J Kerkhoven. Raven 2.0: A versatile toolbox for metabolic network reconstruction and a case study on streptomyces coelicolor. *PLoS computational biology*, 14(10):e1006541, 2018.

[9] Brett G Olivier and Frank T Bergmann. Sbml level 3 package: flux balance constraints version 2. *Journal of integrative bioinformatics*, 15(1), 2018.

[10] Ali Ebrahim, Joshua A Lerman, Bernhard O Palsson, and Daniel R Hyduke. Cobrapy: constraints-based reconstruction and analysis for python. *BMC systems biology*, 7:1–6, 2013.

[11] Philipp Schneider, Pavlos Stephanos Bekiaris, Axel von Kamp, and Steffen Klamt. Straindesign: a comprehensive python package for computational design of metabolic networks. *Bioinformatics*, 38 (21):4981–4983, 2022.

[12] Zachary A King, Andreas Dräger, Ali Ebrahim, Nikolaus Sonnenschein, Nathan E Lewis, and Bernhard O Palsson. Escher: a web application for building, sharing, and embedding data-rich visualizations of biological pathways. *PLoS computational biology*, 11(8):e1004321, 2015.

[13] Daniel Machado. A benchmark of optimization solvers for genome-scale metabolic modeling of organisms and communities. *Msystems*, pages e00833–23, 2024.

[14] Andrey Prjibelski, Dmitry Antipov, Dmitry Meleshko, Alla Lapidus, and Anton Korobeynikov. Using SPAdes de novo assembler. *Current Protocols in Bioinformatics*, 70(1), June 2020. doi: 10.1002/cpbi.102. URL `https://doi.org/10.1002/cpbi.102`.

[15] Doug Hyatt, Gwo-Liang Chen, Philip F LoCascio, Miriam L Land, Frank W Larimer, and Loren J Hauser. Prodigal: prokaryotic gene recognition and translation initiation site identification. *BMC Bioinformatics*, 11(1), March 2010. doi: 10.1186/1471-2105-11-119. URL `https://doi.org/10.1186/1471-2105-11-119`.

[16] Sebastián N. Mendoza, Brett G. Olivier, Douwe Molenaar, and Bas Teusink. A systematic assessment of current genome-scale metabolic reconstruction tools. *Genome Biology*, 20(1):158, August 2019. ISSN 1474-760X. doi: 10.1186/s13059-019-1769-1. URL https://doi.org/10.1186/s13059-019-1769-1.

[17] Daniel Machado, Sergej Andrejev, Melanie Tramontano, and Kiran Raosaheb Patil. Fast automated reconstruction of genome-scale metabolic models for microbial species and communities. *Nucleic Acids Research*, 46(15):7542–7553, September 2018. ISSN 0305-1048. doi: 10.1093/nar/gky537. URL https://doi.org/10.1093/nar/gky537.

[18] Adam M Feist and Bernhard O Palsson. The biomass objective function. *Current opinion in microbiology*, 13(3):344–349, 2010.

[19] Bas Teusink, Jasper A Diderich, Hans V Westerhoff, Karel van Dam, and Michael C Walsh. Intracellular glucose concentration in derepressed yeast cells consuming glucose is high enough to reduce the glucose transport rate by 50%. *Journal of bacteriology*, 180(3):556–562, 1998.

[20] Georgios Marinos, Christoph Kaleta, and Silvio Waschina. Defining the nutritional input for genome-scale metabolic models: A roadmap. *PLOS ONE*, 15(8):1–17, 08 2020. doi: 10.1371/journal.pone.0236890. URL https://doi.org/10.1371/journal.pone.0236890.

[21] Elizabeth Brunk, Swagatika Sahoo, Daniel C Zielinski, Ali Altunkaya, Andreas Dräger, Nathan Mih, Francesco Gatto, Avlant Nilsson, German Andres Preciat Gonzalez, Maike Kathrin Aurich, et al. Recon3d enables a three-dimensional view of gene variation in human metabolism. *Nature biotechnology*, 36(3): 272–281, 2018.

[22] Nantia Leonidou, Elisabeth Fritze, Alina Renz, and Andreas Dräger. SBOannotator: a Python tool for the automated assignment of systems biology ontology terms. *Bioinformatics*, 39(7):btad437, 07 2023. ISSN 1367-4811. doi: 10.1093/bioinformatics/btad437. URL https://doi.org/10.1093/bioinformatics/btad437.

[23] Christian Lieven, Moritz E Beber, Brett G Olivier, Frank T Bergmann, Meric Ataman, Parizad Babaei, Jennifer A Bartell, Lars M Blank, Siddharth Chauhan, Kevin Correia, et al. Memote for standardized genome-scale metabolic model testing. *Nature biotechnology*, 38(3):272–276, 2020.

[24] Timo R. Maarleveld, Meike T. Wortel, Brett G. Olivier, Bas Teusink, and Frank J. Bruggeman. Interplay between constraints, objectives, and optimality for genome-scale stoichiometric models. *PLoS Computational Biology*, 11, 4 2015. ISSN 15537358. doi: 10.1371/journal.pcbi.1004166.

[25] Jonathan M. Monk, Colton J. Lloyd, Elizabeth Brunk, Nathan Mih, Anand Sastry, Zachary King, Rikiya Takeuchi, Wataru Nomura, Zhen Zhang, Hirotada Mori, Adam M. Feist, and Bernhard O. Palsson. iml1515, a knowledgebase that computes escherichia coli traits. *Nature Biotechnology 2017 35:10*, 35: 904–908, 10 2017. ISSN 1546-1696. doi: 10.1038/nbt.3956. URL https://www.nature.com/articles/nbt.3956.

[26] Steven M. Kelk, Brett G. Olivier, Leen Stougie, and Frank J. Bruggeman. Optimal flux spaces of genome-scale stoichiometric models are determined by a few subnetworks. *Scientific Reports 2012 2:1*, 2:1–7, 8 2012. ISSN 2045-2322. doi: 10.1038/srep00580. URL https://www.nature.com/articles/srep00580.

[27] Frank J Bruggeman, Maaike Remeijer, Maarten Droste, Luis Salinas, Meike Wortel, Robert Planqué, Herbert M Sauro, Bas Teusink, and Hans V Westerhoff. Whole-cell metabolic control analysis. *Biosystems*, 234:105067, 2023.

[28] Semidán Robaina Estévez and Zoran Nikoloski. Generalized framework for context-specific metabolic model extraction methods. *Frontiers in plant science*, 5:491, 2014.

[29] Daniel Machado and Markus Herrgård. Systematic evaluation of methods for integration of transcriptomic data into constraint-based models of metabolism. *PLoS computational biology*, 10(4):e1003580, 2014.

[30] Scott A Becker and Bernhard O Palsson. Context-specific metabolic networks are consistent with experiments. *PLoS computational biology*, 4(5):e1000082, 2008.

[31] Aarash Bordbar, Monica L Mo, Ernesto S Nakayasu, Alexandra C Schrimpe-Rutledge, Young-Mo Kim, Thomas O Metz, Marcus B Jones, Bryan C Frank, Richard D Smith, Scott N Peterson, et al. Model-driven multi-omic data analysis elucidates metabolic immunomodulators of macrophage activation. *Molecular systems biology*, 8(1):558, 2012.

[32] Brian J Schmidt, Ali Ebrahim, Thomas O Metz, Joshua N Adkins, Bernhard Ø Palsson, and Daniel R Hyduke. Gim3e: condition-specific models of cellular metabolism developed from metabolomics and expression data. *Bioinformatics*, 29(22):2900–2908, 2013.

[33] Tomer Shlomi, Moran N Cabili, Markus J Herrgård, Bernhard Ø Palsson, and Eytan Ruppin. Network-based prediction of human tissue-specific metabolism. *Nature biotechnology*, 26(9):1003–1010, 2008.

[34] Hadas Zur, Eytan Ruppin, and Tomer Shlomi. imat: an integrative metabolic analysis tool. *Bioinformatics*, 26(24):3140–3142, 2010.

[35] Rasmus Agren, Sergio Bordel, Adil Mardinoglu, Natapol Pornputtapong, Intawat Nookaew, and Jens Nielsen. Reconstruction of genome-scale active metabolic networks for 69 human cell types and 16 cancer types using init. *PLoS computational biology*, 8(5):e1002518, 2012.

[36] Rasmus Agren, Adil Mardinoglu, Anna Asplund, Caroline Kampf, Mathias Uhlen, and Jens Nielsen. Identification of anticancer drugs for hepatocellular carcinoma through personalized genome-scale metabolic modeling. *Molecular systems biology*, 10(3):721, 2014.

[37] Johan Gustafsson, Mihail Anton, Fariba Roshanzamir, Rebecka Jörnsten, Eduard J Kerkhoven, Jonathan L Robinson, and Jens Nielsen. Generation and analysis of context-specific genome-scale metabolic models derived from single-cell rna-seq data. *Proceedings of the National Academy of Sciences*, 120(6):e2217868120, 2023.

[38] Rasmus Agren, Liming Liu, Saeed Shoaie, Wanwipa Vongsangnak, Intawat Nookaew, and Jens Nielsen. The raven toolbox and its use for generating a genome-scale metabolic model for penicillium chrysogenum. *PLoS computational biology*, 9(3):e1002980, 2013.

[39] Livnat Jerby, Tomer Shlomi, and Eytan Ruppin. Computational reconstruction of tissue-specific metabolic models: application to human liver metabolism. *Molecular systems biology*, 6(1):401, 2010.

[40] Yuliang Wang, James A Eddy, and Nathan D Price. Reconstruction of genome-scale metabolic models for 126 human tissues using mcadre. *BMC systems biology*, 6(1):1–16, 2012.

[41] Nikos Vlassis, Maria Pires Pacheco, and Thomas Sauter. Fast reconstruction of compact context-specific metabolic network models. *PLoS computational biology*, 10(1):e1003424, 2014.

[42] André Schultz and Amina A Qutub. Reconstruction of tissue-specific metabolic networks using corda. *PLoS computational biology*, 12(3):e1004808, 2016.

[43] Håvard Molversmyr, Ove Øyås, Filip Rotnes, and Jon Olav Vik. Extracting functionally accurate context-specific models of atlantic salmon metabolism. *NPJ Systems Biology and Applications*, 9(1):19, 2023.

[44] Jonathan M. Monk, Pep Charusanti, Ramy K. Aziz, Joshua A. Lerman, Ned Premyodhin, Jeffrey D. Orth, Adam M. Feist, and Bernhard Ø. Palsson. Genome-scale metabolic reconstructions of multiple iescherichia coli/i strains highlight strain-specific adaptations to nutritional environments. *Proceedings of the National Academy of Sciences*, 110(50):20338–20343, November 2013. doi: 10.1073/pnas.1307797110. URL `https://doi.org/10.1073/pnas.1307797110`.

[45] Andrew J. Page, Carla A. Cummins, Martin Hunt, Vanessa K. Wong, Sandra Reuter, Matthew T.G. Holden, Maria Fookes, Daniel Falush, Jacqueline A. Keane, and Julian Parkhill. Roary: rapid large-scale prokaryote pan genome analysis. *Bioinformatics*, 31(22):3691–3693, July 2015. doi: 10.1093/bioinformatics/btv421. URL `https://doi.org/10.1093/bioinformatics/btv421`.

[46] Ben Vezina, Stephen C. Watts, Jane Hawkey, Helena B. Cooper, Louise M. Judd, Adam W. J. Jenney, Jonathan M. Monk, Kathryn E. Holt, and Kelly L. Wyres. Bactabolize: A tool for high-throughput generation of bacterial strain-specific metabolic models. 2023. doi: 10.1101/2023.02.26.530115. URL http://dx.doi.org/10.1101/2023.02.26.530115.

[47] Charles J. Norsigian, Xin Fang, Yara Seif, Jonathan M. Monk, and Bernhard O. Palsson. A workflow for generating multi-strain genome-scale metabolic models of prokaryotes. *Nature Protocols*, 15(1):1–14, December 2019. doi: 10.1038/s41596-019-0254-3. URL https://doi.org/10.1038/s41596-019-0254-3.

[48] Eleftheria Tzamali, Panayiota Poirazi, Ioannis G. Tollis, and Martin Reczko. A computational exploration of bacterial metabolic diversity identifying metabolic interactions and growth-efficient strain communities. *BMC Systems Biology*, 5(1):167, October 2011. ISSN 1752-0509. doi: 10.1186/1752-0509-5-167. URL https://doi.org/10.1186/1752-0509-5-167.

[49] Xiaolin Zhang and Jennifer L. Reed. Adaptive Evolution of Synthetic Cooperating Communities Improves Growth Performance. *PLOS ONE*, 9(10):e108297, October 2014. ISSN 1932-6203. doi: 10.1371/journal.pone.0108297. URL https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0108297. Publisher: Public Library of Science.

[50] Ruchir A. Khandelwal, Brett G. Olivier, Wilfred F. M. Röling, Bas Teusink, and Frank J. Bruggeman. Community Flux Balance Analysis for Microbial Consortia at Balanced Growth. *PLOS ONE*, 8(5):e64567, May 2013. ISSN 1932-6203. doi: 10.1371/journal.pone.0064567. URL https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0064567. Publisher: Public Library of Science.

[51] Siu Hung Joshua Chan, Margaret N. Simons, and Costas D. Maranas. SteadyCom: Predicting microbial abundances while ensuring community stability. *PLOS Computational Biology*, 13(5):e1005539, May 2017. ISSN 1553-7358. doi: 10.1371/journal.pcbi.1005539. URL https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1005539. Publisher: Public Library of Science.

[52] Ali R. Zomorrodi, Mohammad Mazharul Islam, and Costas D. Maranas. d-OptCom: Dynamic Multi-level and Multi-objective Metabolic Modeling of Microbial Communities. *ACS Synthetic Biology*, 3(4):247–257, April 2014. doi: 10.1021/sb4001307. URL https://doi.org/10.1021/sb4001307. Publisher: American Chemical Society.